



Universidade Estadual de Campinas
Instituto de Computação



Carlos Felipe Estrada Solano

Traffic Engineering in Data Center Networks based on
Software-Defined Networking and Machine Learning

Engenharia de Tráfego em Redes de Centros de Dados
com base em Redes Definidas por Software e
Aprendizagem de Máquinas

CAMPINAS
2022

Carlos Felipe Estrada Solano

**Traffic Engineering in Data Center Networks based on
Software-Defined Networking and Machine Learning**

**Engenharia de Tráfego em Redes de Centros de Dados com base
em Redes Definidas por Software e Aprendizagem de Máquinas**

Tese apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação no âmbito do acordo de Cotutela firmado entre a Unicamp e a Unicauca.

Thesis presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Computer Science under the double-diploma agreement between Unicamp and Unicauca.

Supervisor/Orientador: Prof. Dr. Nelson Luis Saldanha da Fonseca
Co-supervisor/Coorientador: Prof. Dr. Oscar Mauricio Caicedo Rendón

Este exemplar corresponde à versão final da Tese defendida por Carlos Felipe Estrada Solano e orientada pelo Prof. Dr. Nelson Luis Saldanha da Fonseca.

CAMPINAS
2022

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Es88e Estrada Solano, Carlos Felipe, 1987-
Traffic engineering in data center networks based on software-defined networking and machine learning / Carlos Felipe Estrada Solano. – Campinas, SP : [s.n.], 2022.

Orientadores: Nelson Luis Saldanha da Fonseca e Oscar Mauricio Caicedo Rendón.

Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

Em cotutela com: Universidad del Cauca.

1. Redes definidas por software (Tecnologia de rede de computadores). 2. Aprendizagem supervisionada (Aprendizado do computador). 3. Centros de processamento de dados - Balanceamento de carga (Computação). 4. Roteamento (Administração de redes de computadores). 5. Roteamento multicaminho. I. Fonseca, Nelson Luis Saldanha da, 1961-. II. Caicedo Rendón, Oscar Mauricio. III. Universidade Estadual de Campinas. Instituto de Computação. V. Título.

Informações para Biblioteca Digital

Título em outro idioma: Engenharia de tráfego em redes de centros de dados com base em redes definidas por software e aprendizagem de máquinas

Palavras-chave em inglês:

Software-defined networking (Computer network technology)

Supervised learning (Machine learning)

Data processing service centers - Load balancing (Computing)

Routing (Computer network management)

Multipath routing

Área de concentração: Ciência da Computação

Titulação: Doutor em Ciência da Computação

Banca examinadora:

Nelson Luis Saldanha da Fonseca [Orientador]

Oscar Mauricio Caicedo Rendón

Wilmar Yesid Campo Muñoz

Jéferson Campos Nobre

Edmundo Roberto Mauro Madeira

Data de defesa: 07-03-2022

Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)

- ORCID do autor: <https://orcid.org/0000-0003-4797-9204>

- Currículo Lattes do autor: <http://lattes.cnpq.br/XXXXXXXXXXXXXXXXXX>



Universidade Estadual de Campinas
Instituto de Computação



Carlos Felipe Estrada Solano

**Traffic Engineering in Data Center Networks based on
Software-Defined Networking and Machine Learning**

**Engenharia de Tráfego em Redes de Centros de Dados com base
em Redes Definidas por Software e Aprendizagem de Máquinas**

Banca Examinadora:

- Prof. Dr. Nelson Luis Saldanha da Fonseca
IC/UNICAMP, Brasil
- Prof. Dr. Oscar Mauricio Caicedo Rendón
DTm/UNICAUCA, Colômbia
- Prof. Dr. Wilmar Yesid Campo Muñoz
UNIQUINDIO, Colômbia
- Prof. Dr. Jéferson Campos Nobre
INF/UFRGS, Brasil
- Prof. Dr. Edmundo Roberto Mauro Madeira
IC/UNICAMP, Brasil

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 07 de março de 2022

Dedication

I dedicate this thesis to everyone that I love, that supported and loved me through distance and time to come this far. Especially, I dedicate this thesis to my wife, Silvana, whose unconditional love and support have given me the strength to work every day. To my parents, Tina Adriana and Luis Carlos, whose dedication and sacrifice have been the base for my progress. And to my furry pets, Lucas and Chata (in memory), whose follies have reduced the stress and bad mood caused by research.

*It does not matter how slowly you go,
as long as you do not stop.*

(Confucius)

Acknowledgements

I want to thank my family, whose endless love and support have been with me every day, even in the distance and in adversity. To my wife, Silvana, my parents, Tina Adriana and Luis Carlos, and my mother-in-law, María Inés, for their good-natured forbearance with the process and for their pride in this accomplishment. It was a team effort.

I am very grateful to the distinguished faculty members, Professor Oscar Mauricio Caicedo Rendon and Professor Nelson Luis Saldanha da Fonseca. As my advisors, they shared a detailed guidance and encouragement throughout the course of preparing for and conducting the research. Their belief that it was, indeed, possible to finish kept me going. Certainly, today I am a better researcher because of their influence.

I thank the distinguished faculty member, Professor Raouf Boutaba, and his research team at the Cheriton School of Computer Science of the University of Waterloo, Canada, for their support, patience, encouragement, and insightful comments about my investigation during and after my research internship.

My thanks to the Department of Telematics of the University of Cauca, the Institute of Computing of the University of Campinas, and the Cheriton School of the University of Waterloo, to all the professors and employees who helped me and gave me the opportunity of developing my doctoral studies.

I am also grateful to the distinguished faculty members who served as evaluators of my thesis, Professor Wilmar Campo, Professor Jéferson Nobre, and Professor Edmundo Madeira. They used their valuable time and expertise to provide helpful comments on short notice and quickly continue with the review process of my research.

Thanks to all my friends in the 116 IPET room at the University of Cauca, in the Computing Research Laboratory at the University of Campinas, and in the Davis Centre at the University of Waterloo, for all those off-topic discussions, jokes, and laughs during the stressful days.

Finally, I thank to the Administrative Department of Science, Technology and Innovation of Colombia COLCIENCIAS under Grant 647-2014, the São Paulo Research Foundation FAPESP of Brazil under Grant #2019/04914-3 and Grant #2015/24494-8, and the Government of Canada under the ELAP scholarship program, for having financed my research.

Abstract

Data Centers Networks (DCN) represent the critical infrastructure for running Internet-based applications and services that demand colossal computing and storage resources. However, the most prevalent multipath routing mechanism in DCNs, Equal Cost Multiple-Path (ECMP), may degrade the performance of these applications and services while using low network capacity due to the traffic characteristics of flows in DCNs (mice and elephants). Novel multipath routing approaches tackle this problem by leveraging Software-Defined Networking (SDN) for detecting and rescheduling the elephant flows. Some SDN-based approaches have also incorporated Machine Learning (ML) techniques to improve elephant detection and predict elephant traffic characteristics. However, SDN-based multipath routing still requires finding the best trade-off between prompt elephant detection, traffic overhead, data collection accuracy, and network modifications. Moreover, SDN-based multipath routing algorithms call for finer granularity traffic characteristics of elephant flows for improving rescheduling decisions.

This thesis proposes a multipath routing mechanism that leverages both SDN and ML to improve the routing function in DCNs. Three major components form the proposed multipath routing mechanism. First, a flow detection method, called Network Elephant Learner and anaLYzer (NELLY), incorporates incremental learning at the server-side of SDN-based DCNs (SDDCN) to accurately and timely identify elephant flows at low traffic overhead while enabling continuous model adaptation under limited memory resources. Second, a Pseudo-MAC-based Multipath (PM2) routing algorithm supports transparent host migration across the whole network while reducing the number of rules installed on SDN switches, decreasing the delay introduced to flows (mainly mice) traversing the SDDCN. Third, a flow rescheduling method at the controller-side of SDDCNs, called intelligent Rescheduler of IDentified Elephants (iRIDE), improves network throughput and traffic completion time by using deep incremental learning to predict the rate and duration of elephants for computing and installing the best path across the network.

Results show that NELLY achieves high accuracy with a short classification time when using adaptive decision trees algorithms. Moreover, NELLY reduces traffic overhead, elephant detection time, and switch table occupancy compared to other ML-based flow detection methods. On the other hand, PM2 installs much fewer rules than other multipath routing algorithms that support transparent host migration across a large network area (other than the same switch). Finally, iRIDE achieves a low prediction error of the flow rate and flow duration when using deep neural networks with regularization and dropout layers. Moreover, iRIDE enables intelligent elephant rescheduling algorithms that efficiently use the available bandwidth, generating higher throughput and shorter traffic completion time than conventional ECMP.

Resumo

As Redes de Centros de Dados (DCN) representam a infraestrutura principal para a execução de aplicativos e serviços baseados na Internet que exigem recursos colossais de computação e memória. No entanto, o mecanismo de roteamento multipath mais prevalente em DCNs, Equal Cost Multiple-Path (ECMP), pode degradar o desempenho desses aplicativos e serviços quando usar baixa capacidade de rede devido às características de tráfego de fluxos em DCNs (ratos e elefantes). Novas abordagens de roteamento multicaminho resolvem esse problema aproveitando as Redes Definidas por Software (SDN) para detectar e reprogramar os elefantes. Algumas abordagens também incorporaram técnicas de Aprendizagem de Máquina (ML) para melhorar a detecção de elefantes e prever suas características do tráfego. No entanto, o roteamento multicaminho baseado em SDN ainda requer encontrar o melhor balance entre detecção oportuna de elefantes, sobrecarga de tráfego, precisão da coleta de dados e modificações de rede. Além disso, algoritmos de roteamento multicaminho baseados em SDN exigem características de tráfego de granularidade mais finas para melhorar as decisões de reprogramação.

Esta tese propõe um mecanismo de roteamento multicaminho que aproveita SDN e ML para melhorar a função de roteamento em DCNs. Três componentes principais formam o mecanismo proposto. Primeiro, um método de detecção de fluxo, chamado Aprendiz e Analisador de Elefantes de Rede (NELLY), incorpora aprendizagem incremental no lado do servidor de DCNs baseadas em SDN (SDDCN) para identificar com precisão e oportunamente elefantes, gerando baixa sobrecarga de tráfego e permitindo a adaptação contínua do modelo sob recursos limitados. Em segundo lugar, o algoritmo de roteamento Multicaminho baseado em Pseudo-MAC (PM2) suporta migração transparente de hosts em toda a rede enquanto reduz o número de regras instaladas em switches, diminuindo o atraso introduzido nos fluxos (principalmente ratos) que atravessam a SDDCN. Terceiro, um método de reprogramação de fluxo no lado do controlador, chamado Reprogramador Inteligente de Elefantes Identificados (iRIDE), melhora a taxa de transferência da rede e o tempo de conclusão do tráfego usando aprendizagem incremental profundo para prever a taxa e a duração dos elefantes e instalar o melhor caminho pela rede.

Os resultados mostram que o NELLY alcança alta precisão com um curto tempo de classificação ao usar algoritmos de árvores de decisão adaptativos. Além disso, o NELLY reduz a sobrecarga de tráfego, o tempo de detecção de elefantes e a ocupação das tabelas dos switches em comparação com outros métodos de detecção de fluxo baseados em ML. Por outro lado, o PM2 instala muito menos regras do que outros algoritmos de roteamento multicaminho que suportam migração transparente de hosts em uma grande área de rede (diferente do mesmo switch). Finalmente, o iRIDE alcança um baixo erro de previsão da taxa e da duração do fluxo ao usar redes neurais profundas com regularização e dropout. Além disso, o iRIDE permite algoritmos inteligentes de reprogramação de elefantes que usam eficientemente a largura de banda disponível, gerando maior taxa de transferência e menor tempo de conclusão do tráfego do que o ECMP convencional.

List of Figures

1.1	Thesis phases	24
2.1	High-level SDN architecture	26
2.2	SDN vertical management plane	28
2.3	Problem categories that benefit from ML paradigms	30
2.4	The constituents of ML-based solutions	31
2.5	The evolution of machine learning techniques with key milestones	33
2.6	Cognitive control loop for network management	42
2.7	Tree-based DCN with fourth degree and multipath routing	43
3.1	Architecture of NELLY	55
3.2	Structure of flow records in FlowRepo	58
3.3	Example of how flow records are created and updated in FlowRepo	58
3.4	Confusion matrix for binary classification	63
3.5	Classification accuracy of NELLY when using a single weight and inverse weights for the UNI1 and UNI2 datasets	65
3.6	Classification accuracy of NELLY with the ARF and AHOT algorithms when varying the labeling threshold (θ_L) for the UNI1 and UNI2 datasets	68
3.7	Classification accuracy of NELLY with the ARF and AHOT algorithms when varying the inverse weights of elephant flows (W_E) for the UNI1 and UNI2 datasets	69
3.8	Accuracy of NELLY with the ARF and AHOT algorithms when varying the labeling threshold (θ_L) for all the IPv4 flows in UNI1 and UNI2 data traces	72
3.9	Accuracy of NELLY with the ARF and AHOT algorithms when varying the inverse weights of elephant flows (W_E) for all the IPv4 flows in UNI1 and UNI2 data traces	73
4.1	RTT measurement experimental scenarios	77
4.2	Average RTT per number of flow rules installed in the switch of each experimental scenario with $T = 30$ and $N = 30$	78
4.3	PM2 process to install routing rules	79
4.4	ARP request interception in PM2	82
4.5	PMAC table in PM2	84
4.6	Implementation of PM2	86
4.7	Number of rules installed per switch for each layer in a fat-tree topology with size $k = 48$ and only one host in each server ($h_s = 1$)	90
4.8	Architecture of iRIDE	92
4.9	Transformations of flow rate in UNI1 and UNI2	98
4.10	Transformations of flow duration in UNI1 and UNI2	98

4.11	NN structure	101
4.12	Mean RMSE over a sliding window of instances for DNN regression models with different mini-batch sizes (n_B) in an incremental learning setting . . .	103
4.13	Mean RMSE over mini-batch sizes (n_B) for DNN regression models in an incremental learning setting	104
4.14	Mean prediction time per flow over mini-batch sizes (n_B) for DNN regres- sion models in an incremental learning setting	105
4.15	Implementation of iRIDE	108
4.16	PCAP parsing for the traffic generator in iRIDE	109
4.17	Throughput over time in the bisection links of a fat-tree topology of size $k = 4$ when using PM2 and iRIDE, with both LC and WF+BTF, for routing UNI1 traffic	110
4.18	Mean throughput in the bisection links of a fat-tree topology of size $k = 4$ and traffic completion time when using PM2 and iRIDE, with both LC and WF+BTF, for routing UNI1 traffic	111

List of Tables

2.1	Summary of the limitations of distributed multipath routing	47
2.2	Summary of the limitations of SDN-based multipath routing	51
3.1	Symbols in the architecture of NELLY	56
3.2	Details of real packet traces and extracted IPv4 flows	61
3.3	Header types defined during feature engineering	62
3.4	Classification performance of NELLY with different incremental algorithms	66
3.5	Comparison of NELLY with related approaches	70
4.1	Fat-tree topology size	80
4.2	Comparison of PM2 routing with related approaches	89
4.3	Symbols in the architecture of iRIDE	92
4.4	Learning algorithm and data preprocessing with the best accuracy performance in a batch learning setting	99
4.5	Data preprocessing with the best accuracy performance for MLP in a batch learning setting	100
4.6	Accuracy performance of MLP using feasible incremental data preprocessing techniques in a batch learning setting	101
4.7	DNN structures with the lowest training errors in a batch learning setting .	102
4.8	DNN structures and regularization methods with the lowest validation errors in a batch learning setting	103

List of Algorithms

- 3.1 NELLY Analyzer 57
- 3.2 NELLY Learner 60
- 4.1 Handling intercepted ARP messages in PM2 83

Acronyms

<i>AHOT</i>	Adaptive Hoeffding Option Tree
<i>AI</i>	Artificial Intelligence
<i>ARF</i>	Adaptive Random Forest
<i>ARP</i>	Address Resolution Protocol
<i>AS</i>	Autonomous System
<i>CIM</i>	Common Information Model
<i>CVFDT</i>	Concept-adapting Very Fast Decision Tree
<i>DCN</i>	Data Center Network
<i>DL</i>	Deep Learning
<i>DNN</i>	Deep Neural Network
<i>DSCP</i>	Differentiated Services Code Point
<i>ECMP</i>	Equal Cost Multiple-Path
<i>FCAPS</i>	Fault, Configuration, Accounting, Performance, and Security
<i>FPR</i>	False Positive Rate
<i>HTN</i>	Hierarchical Task Network
<i>IAT</i>	Inter-Arrival Time
<i>IIoT</i>	Industrial Internet of Things
<i>iRIDE</i>	intelligent Rescheduler of IDentified Elephants
<i>kNN</i>	<i>k</i> -Nearest Neighbors
<i>MAC</i>	Media Access Control
<i>MCC</i>	Matthews Correlation Coefficient
<i>ML</i>	Machine Learning
<i>MLP</i>	Multi-Layer Perceptron
<i>MOA</i>	Massive Online Analysis

<i>NB</i>	Naïve Bayes
<i>NELLY</i>	Network Elephant Learner and anaLYzer
<i>NFV</i>	Network Function Virtualization
<i>NN</i>	Neural Network
<i>OAUE</i>	Online Accuracy Updated Ensemble
<i>PA</i>	Passive-Aggressive
<i>PAW</i>	Probabilistic Adaptive Windowing
<i>PC</i>	Physical Computer
<i>PCP</i>	Priority Code Point
<i>PM2</i>	Pseudo-MAC-based Multipath
<i>PMAC</i>	Pseudo-MAC
<i>RL</i>	Reinforcement learning
<i>RTT</i>	Round-Trip Time
<i>SDDCN</i>	Software-Defined Data Center Network
<i>SDN</i>	Software-Defined Networking
<i>SGD</i>	Stochastic Gradient Descent
<i>SMOGN</i>	Synthetic Minority Over-sampling technique for regression with Gaussian Noise
<i>SVM</i>	Support Vector Machines
<i>ToR</i>	Top-of-Rack
<i>TPR</i>	True Positive Rate
<i>VFDR</i>	Very Fast Decision Rules
<i>VFDT</i>	Very Fast Decision Tree
<i>VM</i>	Virtual Machine
<i>YANG</i>	Yet Another Next Generation

Contents

1	Introduction	18
1.1	Problem statement	18
1.2	Hypothesis	20
1.3	Objectives	20
1.3.1	General objective	20
1.3.2	Specific objectives	20
1.4	Contributions	21
1.5	Scientific production	22
1.6	Methodology and organization	23
2	Background and state-of-the-art	25
2.1	Software-defined networking	25
2.1.1	SDN Architecture	26
2.1.2	Management plane	27
2.2	Machine learning for networking	29
2.2.1	Incremental learning	31
2.2.2	Evolution of machine learning techniques	33
2.2.3	Cognitive networking	41
2.3	Traffic engineering in data center networks	42
2.3.1	Data center network	43
2.3.2	Traffic engineering	44
2.3.3	Multipath routing in data center networks	45
2.4	Final remarks	51
3	Flow detection using online incremental learning at the server-side of software-defined data center networks	53
3.1	Architecture of NELLY	54
3.1.1	Analyzer	55
3.1.2	Learner	59
3.2	Evaluation	60
3.2.1	Datasets	61
3.2.2	Accuracy metrics	62
3.2.3	Experiment setup	64
3.2.4	Performance analysis	64
3.3	Comparative analysis	69
3.4	Final remarks	74

4	Multipath routing based on software-defined networking and machine learning for data center networks	75
4.1	Pseudo-MAC-based multipath routing in software-defined data center networks	75
4.1.1	Motivation	76
4.1.2	Pseudo-MAC-based multipath routing	78
4.1.3	Implementation	86
4.1.4	Analytic evaluation	87
4.2	Rescheduling of elephants in software-defined data center networks using deep incremental learning	90
4.2.1	Architecture of iRIDE	91
4.2.2	Prediction	94
4.2.3	Rescheduling	105
4.2.4	Implementation	107
4.2.5	Evaluation	109
4.3	Final remarks	111
5	Conclusions	112
5.1	Answers for the fundamental questions	113
5.2	Future work	114
	Bibliography	116
A	Scientific Production	141

Chapter 1

Introduction

1.1 Problem statement

Nowadays, the ever-growing Internet-based applications and services demand a huge amount of computing and storage resources. Data centers represent the key infrastructure that supports and provides such resources as a large number of servers interconnected by a specially designed network, called Data Center Network (DCN) [1]. The goal of DCN is to provide significant bandwidth capacity in order to achieve high throughput¹ and low-latency².

Everyday, DCN managers are looking for solutions that allow optimizing these performance requirements (i.e., high bandwidth, high throughput, and low latency) without the need to add more capacity to the network. Traffic engineering represents a great opportunity in this realm. Particularly, load-balancing is a desirable feature for reducing network congestion while improving network resource availability and application performance [2, 3]. A well-known technique for implementing load-balancing in DCNs is multipath routing, which distributes traffic over multiple concurrent paths such that all the links are optimally loaded [4]. Hereinafter, when this dissertation mentions routing in DCN, it is particularly referring to intra-DCN routing.

The most prevalent multipath routing solution in DCNs is Equal Cost Multiple-Path (ECMP) [5, 6]. Usually, ECMP uses a hash function in every switch to assign each incoming flow to one of the equal-cost forwarding paths maintained by the switch for reaching a destination [7]. However, traffic in DCNs presents a broad distribution of flow sizes: from small, short-lived flows (i.e., mice) to large, long-lived flows (i.e., elephants) [8–11]. This wide dispersion of flow sizes causes *hot-spots* in DCNs based on ECMP routing, i.e., some links are highly utilized while others are underutilized.

For example, if two mouse flows and two elephant flows arrive at the same ECMP-enabled switch, it is possible that this switch assigns the two mouse flows to one of the forwarding paths and the two elephant flows to one of the other forwarding paths. Therefore, the link transporting the two mouse flows is going to present less load and be free much faster than the link transporting the two elephant flows, causing an underusing

¹Total number of packets processed per second.

²Average processing time used for a single packet.

and overloading of links, respectively. Facebook’s Altoona [12] propose to prevent the degrading of elephant flows by making the network multi-speed. However, this is not an efficient approach. Rather than adding more capacity, the issue is selecting a routing mechanism for drawing traffic effectively.

For this reason, recent multipath routing mechanisms have been proposed for improving ECMP. Broadly, these mechanisms can be categorized as distributed and centralized multipath routing. Distributed multipath routing maintains routing decisions at switches or servers and tackles ECMP limitations by (i) using different levels of granularity for traffic splitting (i.e., packet-level [13, 14] and sub-flow-level [15]), (ii) adding weights to the paths [15, 16]; or (iii) incorporating congestion information for making routing decisions [17]. Distributed multipath routing mechanisms that combine sub-flow-level traffic splitting and congestion-awareness—local congestion [18, 19] or global congestion [20–22]—have provided great results for load-balancing in DCNs. However, these solutions require specialized hardware implementation, potentially cause packet reordering, and lack a global view of traffic for making routing decisions.

Centralized multipath routing have leveraged Software-Defined Networking (SDN) to face the ECMP limitations; DCNs using SDN are referred to as Software-Defined Data Center Networks (SDDCNs). SDN allows a logically centralized controller to dynamically make and install routing decisions on the basis of a global view of the network [23, 24]. Hereinafter, this dissertation uses the term SDN-based multipath routing to refer to centralized mechanisms. SDN-based multipath routing reschedules elephant flows, while handling mouse flows by employing default routing, such as ECMP. Early SDN-based mechanisms proposed reactive flow detection methods to discriminate elephants from mice by using static thresholds either at the controller-side [25, 26], switch-side [27], or server-side [28, 29] of SDDCNs. However, reactive methods are not suitable for SDDCNs since hot-spots may occur before the elephant flows are detected.

Novel SDN-based multipath routing have incorporated Machine Learning (ML)-based flow detection methods for proactively identifying elephants. However, ML-based methods train their classification models at the controller-side of SDDCNs, requiring the central collection of either per-flow data [30–32] or sampling-based data [33–35]. The central collection of per-flow data, however, causes problems such as heavy traffic overhead and poor scalability. Sampling-based data, on the other hand, tends to provide delayed and inaccurate flow information. Moreover, sampling techniques that mitigate the problem rely on non-standard SDN specifications.

Another gap in multipath routing is that SDN-based mechanisms, both with reactive and ML-based flow detection methods, merely identify elephant flows (i.e., binary classification) and lack fine-grained information for making routing decisions. Therefore, their routing algorithm reschedules all the elephants using the same approach regardless of the different traffic characteristics that elephant flows exhibit in DCNs. This elephant-oblivious routing, however, may cause hot-spots in SDDCNs, reducing the performance of the network. Only a few SDN-based mechanisms introduce ML-based methods that classify flows into more than two categories (i.e., multiclass classification) [31, 33]. However, the routing algorithms in such SDN-based mechanisms use only a part of the information given by their flow detection methods. Moreover, such flow detection methods employ a

reduced number of classification categories (up to five) that still fall short to cover the broad distribution of elephant flows in DCNs.

Based on these statements, SDN-based multipath routing still requires finding the best trade-off between prompt elephant detection, traffic overhead, data collection accuracy, and network modifications. Besides, SDN-based multipath routing algorithms call for finer granularity traffic characteristics of elephant flows for improving rescheduling decisions. Therefore, this thesis project focused on solving the following research question:

How to carry out multipath routing in DCNs for enabling high throughput and low delay while maintaining efficient use of resources?

1.2 Hypothesis

To address the research question, this thesis raised the following hypothesis: **using ML for fine-granularity prediction of flow characteristics and SDN for dynamic control of flow scheduling would allow building a multipath routing mechanism for DCNs that improves³ the routing function.**

The following fundamental questions, associated with the hypothesis, guided the investigation conducted in this thesis.

- What is the accuracy and efficiency, in terms of time and memory, of ML techniques for predicting flow characteristics of network traffic from DCNs?
- Does incorporating ML techniques to an SDN-based multipath routing mechanism improve network traffic routing, in terms of throughput and delay, in DCNs?

1.3 Objectives

1.3.1 General objective

To develop a multipath routing mechanism based on ML and SDN for improving the routing function in DCNs.

1.3.2 Specific objectives

- To design a multipath routing reference architecture that incorporates the capabilities of ML and SDN for improving the routing function in DCNs.
- To construct and evaluate⁴ a mechanism based on ML that predicts, in a fine-granularity way, flow characteristics in DCNs.
- To construct and evaluate⁵ a routing mechanism based on SDN that uses predicted flow characteristics for improving the routing function in DCNs.

³In terms of high throughput and low delay while efficient use of resources

⁴In terms of accuracy (e.g., true/false positives/negatives) and processing requirements (e.g., training data, training time, run-time).

⁵In terms of traffic performance (e.g., throughput, delay) and resource utilization (e.g., links load)

1.4 Contributions

The scientific research process conducted during this thesis led to building a multipath routing mechanism that leverages ML techniques for predicting traffic flow characteristics and SDN capabilities for differentiated and dynamic control of traffic flows aiming to improve the routing function in DCNs. Three major components form our multipath routing mechanism.

- A flow detection method that incorporates incremental learning at the server-side of SDDCNs to accurately and timely identify elephant flows while generating low traffic overhead and adapting to varying traffic characteristics under limited memory resources.
- A Pseudo-MAC (PMAC)-based multipath routing algorithm for steering traffic flows (mainly, mice) in SDDCNs that supports transparent host migration across the whole network while reducing the number of rules installed on SDN switches, decreasing the delay introduced to flows traversing the network.
- A flow rescheduling method at the controller-side of SDDCNs that applies deep incremental learning for predicting traffic characteristics of elephant flows to compute and install the best path per elephant flow across the network.

Moreover, collaborations with other researchers (i.e., student advisory and research internships) during this thesis led to the following contributions.

- An SDN management architecture based on Hierarchical Task Network (HTN) and Network Function Virtualization (NFV) that provides an automated, workable, and flexible approach for monitoring, configuring, and controlling SDN resources.
- A vertical Management Plane for SDN that considers management tasks involving more than one Autonomous System (AS).
- A set of data models for the SDN architecture based on the Yet Another Next Generation (YANG) language to support integrated management in a technology-agnostic and heterogeneous SDN environment.

These three contributions are built on the reference architecture for SDN integrated management and the Common Information Model (CIM)-based information model proposed in the author's master thesis [36].

- A cognitive control loop framework for autonomic network management that incorporates ML at every function of the closed-loop and each of the Fault, Configuration, Accounting, Performance, and Security (FCAPS) management areas. A discussion about the opportunities and challenges pertaining to using ML to manage autonomic networks complements this cognitive framework.
- A comprehensive body of knowledge on ML techniques in support of networking. Particularly, this body of knowledge comprehends the following contributions.

- A generic approach for designing ML-based solutions in networking.
- A brief history of ML focused on the techniques that have been applied in networking.
- A literature review about the advances made in the application of ML in different networking areas, including traffic prediction, classification, and routing, which are fundamental in traffic engineering for optimizing network performance.
- Prominent challenges and open research opportunities on the feasibility and practicality of ML in current and future networks.

1.5 Scientific production

Two published papers (one in a highly ranked journal and one in a renowned conference) and one journal paper in construction report to the scientific community the major contributions achieved during this thesis.

- “NELLY: Flow Detection Using Incremental Learning at the Server Side of SDN-based Data Centers,” published in IEEE Transactions on Industrial Informatics, 2020 [37]. Ranking: JCR Q1, SJR Q1, Publindex A1, Qualis A1. Contribution: the elephant flow detection method using incremental learning.
- “An Efficient Mice Flow Routing Algorithm for Data Centers based on Software-Defined Networking,” published in the proceedings of 2019 IEEE International Conference on Communications (ICC) [38]. Ranking: H5-index 56, Qualis A1, CORE B. Contribution: the PMAC-based multipath routing algorithm for SDDCNs.
- “iRIDE: Rescheduling of Elephant Flows in SDN-based Data Centers Using Incremental Deep Learning to Predict Traffic Characteristics,” in construction. Contribution: the flow rescheduling method using deep incremental learning.

Furthermore, six papers published in renowned journals and conferences report to the scientific community the contributions achieved in collaboration with other researchers. These papers are listed in chronological order.

- “A Framework for SDN Integrated Management based on a CIM Model and a Vertical Management Plane,” published in Computer Communications, 2017 [39]. Ranking: JCR⁶ Q2, SJR⁷ Q2, Publindex⁸ A1, Qualis⁹ A2. Contribution: the reference architecture for the SDN integrated management and the CIM-based information model.

⁶Quartile from Journal Citation Reports (JCR)

⁷Quartile from SCImago Journal Rank (SJR)

⁸Bibliographic index from COLCIENCIAS, Colombia

⁹Bibliographic index from CAPES, Brazil

- “SDN Management Based on Hierarchical Task Network and Network Functions Virtualization,” published in the proceedings of the 2017 IEEE Symposium on Computers and Communications (ISCC) [40]. Ranking: H5-index¹⁰ 20, Qualis A3, CORE¹¹ B. Contribution: the HTN- and NFV-based architecture for managing SDN.
- “A YANG Model for a vertical SDN Management Plane,” published in the proceedings of the 2017 IEEE Colombian Conference on Communications and Computing (COLCOM) [41]. Ranking: H5-index 9. Contribution: the YANG data model for the SDN Management Plane.
- “Machine Learning for Cognitive Network Management,” published in IEEE Communications Magazine, 2018 [42]. Ranking: JCR Q1, SJR Q1, Publindex A1, Qualis A1. Contribution: the cognitive control loop framework for autonomic network management.
- “A comprehensive survey on machine learning for networking: evolution, applications and research opportunities,” published in Journal of Internet Services and Applications, 2018 [43]. Ranking: SJR Q2, Publindex A2, Qualis A2. Contribution: the body of knowledge on ML in networking.
- “An Approach based on YANG for SDN Management,” published in International Journal of Communication Systems, 2021 [44]. Ranking: SJR Q2, Publindex A2, Qualis A3. Contribution: the Management Plane with multiple ASs support and the YANG data models for the SDN architecture.

Appendix A lists the eight published papers in chronological order.

1.6 Methodology and organization

The research process that guided the development of this thesis is based on a typical scheme of the scientific method [45]. Figure 1.1 depicts the phases that form this research process: Problem Statement, Hypothesis Construction, Experimentation, Conclusion, and Publication. Problem Statement, for identifying and establishing the research question. Hypothesis Construction, for formulating the hypothesis and the associated fundamental questions. In addition, this phase aimed to define and carry out the conceptual and technological approaches. Experimentation, for testing the hypothesis and analyzing the evaluation results. Conclusion, for outlining conclusions and future works. Note that Hypothesis Construction had feedback from Experimentation and Conclusion. Publication, for submitting and publishing papers for renowned conferences and journals. The writing of this dissertation document also belongs to this last phase.

The organization of this document reflects the phases of the methodology.

- This introductory chapter presents the problem statement, delineates the hypothesis, exposes the objectives, summarizes the contributions, lists the scientific production, and describes the overall structure of this dissertation.

¹⁰H-index from Google Scholar Metrics

¹¹Rank from COmputing Research and Education (CORE), Australia

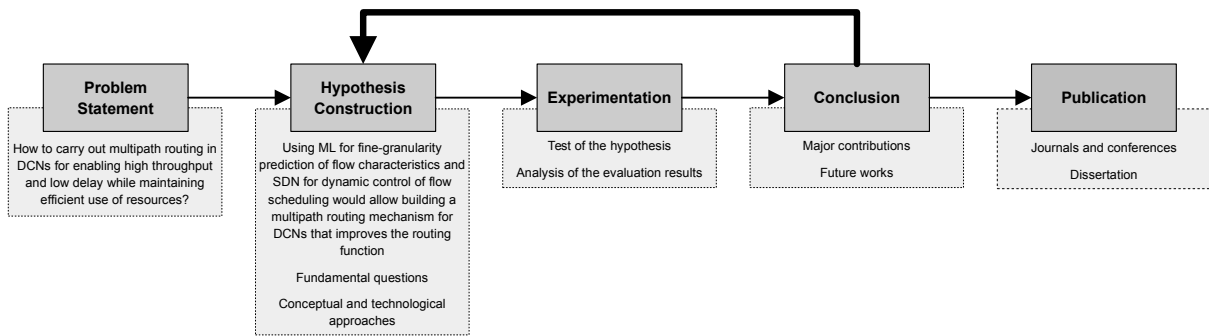


Figure 1.1: Thesis phases

- **Chapter 2** reviews the main concepts and research related to SDN management, ML for networking, and traffic engineering in DCNs.
- **Chapter 3** introduces the server-side flow detection method for SDDCNs based on incremental learning.
- **Chapter 4** details the multipath routing mechanism based on ML and SDN for DCNs. Section 4.1 describes the multipath routing algorithm for steering mice in SDDCNs.
- **Chapter 5** presents conclusions about the hypothesis and the fundamental questions as well as research directions.

Chapter 2

Background and state-of-the-art

This chapter presents the background of the main research topics encompassed in this thesis. In this way, the first section introduces a bottom-up description of the typical SDN architecture followed by a detailed explanation of our view of a management plane for SDN. The second section provides a primer of ML for networking, discussing different categories of ML-based techniques, their essential constituents, and their evolution. Moreover, this section reviews the notion of cognitive networking, focusing on our proposal for realizing a cognitive control loop for autonomic networking. Finally, this chapter contextualize the concepts of DCN and traffic engineering, and provides a literature review about multipath routing for load-balancing in DCNs, focusing on the seminal works that use SDN and ML for addressing such a challenge.

2.1 Software-defined networking

SDN represents one of the most accepted and attractive trends, in research and industry, for defining the architecture of future networks [46, 47]. From a general aspect, SDN decouples the control and forwarding planes for enabling a simpler network operation from a logically centralized software program, usually known as the *controller* [24]. The *control plane* (i.e., the controller) compiles decision policies and enforces them on the *data plane* (i.e., switches and routers) through a vendor independent protocol. OpenFlow [48] is the most well-known open SDN protocol and a *de facto* standard because of its widespread use by vendors and research.

SDN provides four major advantages for operating networks [49]: *(i)* a centralized global view about the network state (e.g., resource capabilities and dynamic status) and the deployed applications (e.g., QoS requirements), *(ii)* a dynamic programmability of multiple forwarding devices (e.g., allocating resources to prevent congestion and improve performance), *(iii)* open interfaces for handling the forwarding plane (e.g., OpenFlow) and for developing the applications (e.g., Application Programming Interfaces (APIs) based on protocols and programming languages); and *(iv)* a flexible flow management (e.g., multiple flow tables in OpenFlow). These unique features lead the SDN architecture to emerge as a promising scenario for efficiently and intelligently implementing management techniques, particularly for traffic engineering.

2.1.1 SDN Architecture

Multiple standardization bodies, such as the Linux Foundation [50] and Open Network Foundation (ONF) [51], focus on encouraging and normalizing open SDN frameworks. Also, various private networking vendors, such as Cisco [52] and Juniper [53], offer proprietary SDN deployments. In turn, several research surveys [23, 54] work on improving architectural aspects of SDN. These open, proprietary, and research proposals establish a typical SDN architecture composed of three horizontal planes (i.e., data, control, and application) and three interfaces (i.e., southbound, northbound, and east/westbound), as depicted in Figure 2.1.

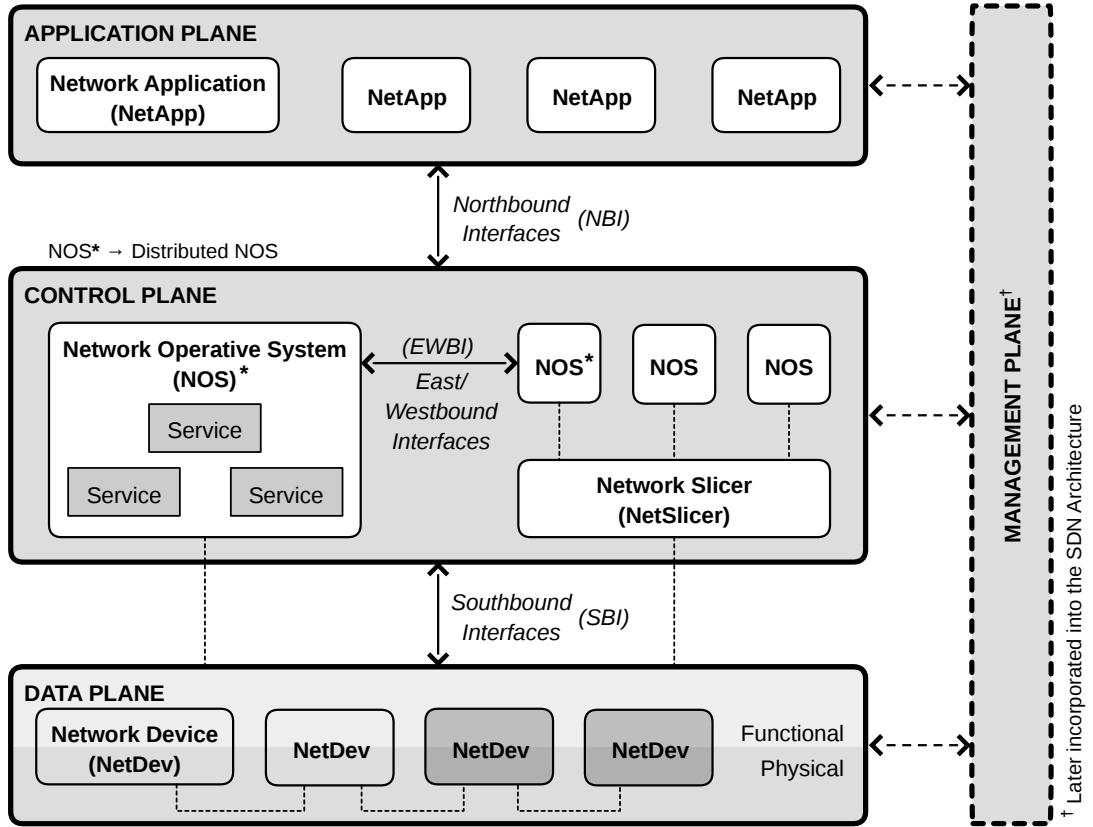


Figure 2.1: High-level SDN architecture

At the bottom of the SDN architecture, the *data plane* (a.k.a. forwarding plane) deploys the network infrastructure formed by interconnected Network Devices (NetDevs), such as switches and routers, that perform forwarding operations. A NetDev consists of a *physical* and a *functional* part. The former comprises hardware elements, such as ports, storage, processor, and memory. The latter defines a collection of software-based forwarding functions executed by NetDevs. Regarding this functional part, a NetDev ranges from *dumb* to *custom*. A dumb NetDev merely carries out simple forwarding functions, such as longest prefix match. For example, OpenFlow-only switches [55] just forward packets using the rules installed in their flow tables—updated by an OpenFlow controller. On the other hand, a custom NetDev relies on programmable platforms [56]—e.g., Protocol-Independent Switch Architecture (PISA) and Field-Programmable Gate Array (FPGA)—to integrate more complex forwarding functions, such as load balanc-

ing [57] and in-band network telemetry [58]. For example, P4 [59] provides a target- and protocol-independent language that allows programming packet processing functionality.

In the middle, the *control plane* compiles the network logic and enforces decision policies on the data plane through SouthBound Interfaces (SBIs). Each SBI defines the set of instructions and the communication protocols to allow the interaction between components in the control and data planes. The OpenFlow protocol [48] is the most well-known open standard SBI because its widespread use by vendors and research [46]. Other SBI proposals are Forwarding and Control Element Separation (ForCES) [60], Protocol-Oblivious Forwarding (POF) [61], and P4Runtime [62].

The control plane comprises Network Slicers (NetSlicers) and Network Operating Systems (NOSs). A NetSlicer divides the underlying network infrastructure into several isolated logical network instances (a.k.a. slices), assigning their control to specific NOSs. NetSlicers may employ SBIs to communicate with NOSs. For example, FlowVisor [63] acts as an OpenFlow proxy between switches and controllers, redirecting messages according to flow parameters, such as TCP ports and IP addresses. An NOS instructs the underlying data plane and provides generic services (e.g., topology discovering and host tracking) and NorthBound Interfaces (NBIs) to the *application plane*, facilitating to integrate custom Network Applications (NetApps). The possibility to add these NetApps in an easier way is the key advantage of SDN to encourage innovation on the Internet. OpenFlow controllers [55] and ForCES Control Element (CE) [60] represent NOS instances. It is important to highlight that a lot of frameworks exist to develop and deploy OpenFlow controllers, including open source projects like NOX [64] for C++, POX [65] and Ryu [66] for Python, Floodlight [67] and OpenDaylight [68] for Java, and Trema [69] for Ruby. Also, the control plane defines East/WestBound Interfaces (EWBIs) to deploy distributed NOSs. For example, SDNi [70] and ForCES CE-CE interface [60].

At the top of the SDN architecture, the application plane contains NetApps that deploy and orchestrate business logic and high-level network functions, such as routing policies and access control. As aforementioned, NetApps communicate with the control plane through NBIs provided by NOSs. NBIs encompass common APIs based on protocols (e.g., Floodlight REST API [71]), programming languages (e.g., ad-hoc, Pyretic [72], and Procera [73]), file systems (e.g., YANC [74]), among others. NetApps run either locally or remotely regarding NOSs. Local NetApps prefer NBIs based on programming languages, whereas remote NetApps usually employ protocol-based NBIs.

2.1.2 Management plane

As depicted in Section 2.1.1, the traditional SDN architecture lacked an integrated and standardized framework for managing the virtual¹, dynamic², and heterogeneous³ SDN environment. Later SDN approaches [75–78] considered a vertical *management plane* in the SDN architecture (see Figure 2.1) for carrying out different Operation, Administration, and Maintenance (OAM) functions. For example, assigning the data plane resources to

¹Capability for sharing network resources from a same physical infrastructure among several virtual network instances.

²Flexibility for adding, modifying, migrating, and removing network resources.

³Independence of the technology deployed by network resources.

the corresponding control components, and configuring the policies and Service Level Agreements (SLAs) of the control and application planes. Although NOSs may implement some OAM functions, flooding the control plane with a lot of managing tasks may cause low network performance.

These SDN approaches exposed a very high-level of their management component. Therefore, we extended and detailed such management plane aiming to facilitate integrated control and monitoring of heterogeneous SDNs [39]. This approach originally lacked inter-domain communication management; hence, we later incorporated *inter-AS* elements (i.e., repository, adapter, interface, and agent) for supporting management tasks involving more than one domain [41, 44]. Figure 2.2 depicts the proposed vertical management plane comprising different elements: *data repositories*, *managers*, *adapters*, *agents*, and *management interfaces*.

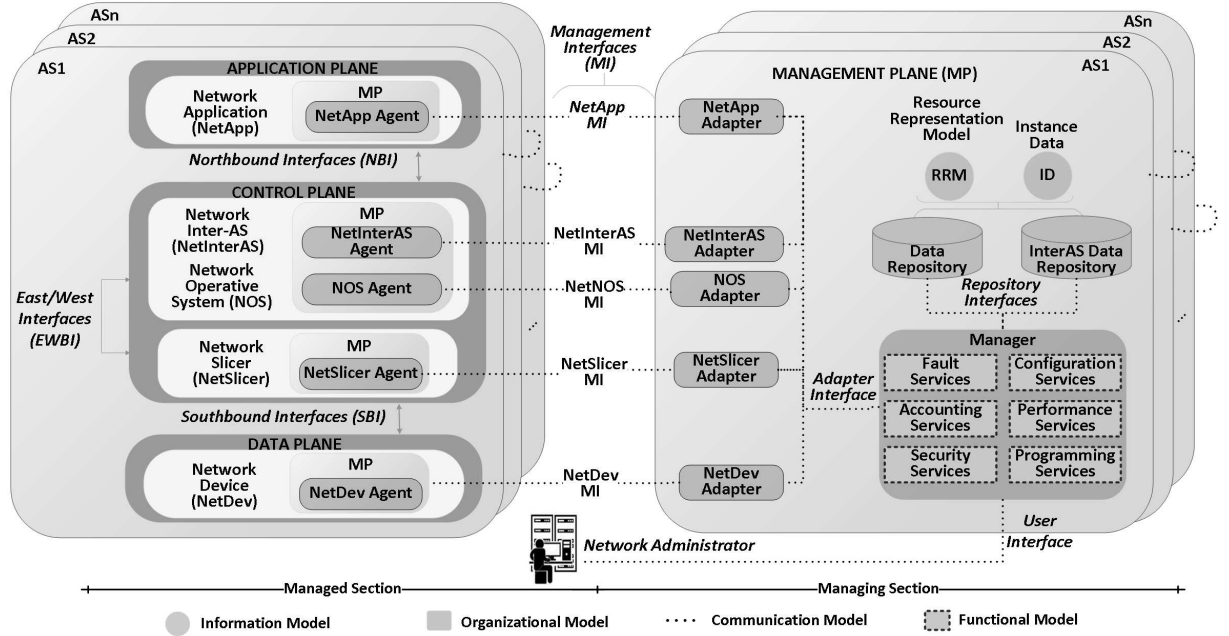


Figure 2.2: SDN vertical management plane

Two data repositories coexist, each holding a Resource Representation Model (RRM) that handles metadata to provide an abstract, technology-neutral characterization of SDN resources. Data repositories also serve managers for storing *instance data*, which represents execution-specific data whose structure follows such an RRM. Particularly, the inter-AS data repository focuses on information relevant from other ASs for enabling management tasks in an inter-domain environment. On the other hand, managers orchestrate and deploy *management services* to carry out different SDN management functions. These management services expose user interfaces to enable *network administrators* to interact with managers. Managers also interact with agents via adapters, which enable a protocol-agnostic communication using data type rendering, protocol translation, and well-defined management interfaces. Note each management interface connects an adapter with its corresponding agent. Finally, agents inside managed SDN resources act on behalf of managers.

Figure 2.2 also describes our management plane referencing the four Open System

Interconnection (OSI) network management submodels [79]. First, the *organizational model* specifies roles and collaboration forms of the managing entities (i.e., managers and adapters) and managed entities (i.e., agents). Second, the *communication model* delineates the exchange of management data (e.g., operations, queries, events) and the enabling technologies for the user and adapter interfaces (e.g., JSON [80] over HTTP [81]), repository interfaces (e.g., XML [82] over HTTP), and management interfaces (e.g., OVSDB [83], NETCONF [84], SNMP [85]). Third, the *functional model* structures the management services referencing the five OSI management functional areas (i.e., FCAPS [86]) along with a novel programmability function (i.e., FCAPS+P) introduced by SDN. Fourth, the *information model* establishes a shared abstraction of SDN resources for achieving an integrated and technology-independent management.

The whole operation of our management plane is based on RRM, which implements the information model and originally leveraged CIM⁴ for representing the SDN architecture as a conceptual model [39]. We later implemented the CIM model using YANG data models⁵ since the latter provides a more human-readable and easier-to-learn language than the former, while also being technology-agnostic [41, 44]. Note that different SDN solutions from the industry and academia increasingly use YANG to build new management solutions [89–91]. This is because YANG emerged from a widely adopted network management protocol (i.e., NETCONF) [92, 93] and structures data under a hierarchical tree topology using modules and submodules that allow description easily network devices and their relationships [94].

We also defined our management plane in the context of network automation by introducing an SDN management architecture based on HTN and NFV [40]. This architecture provides an automated, workable, and flexible approach for monitoring, configuring, and controlling SDN resources. To achieve this goal, our management plane instantiates the three NFV MANagement and Orchestrator (MANO) functional blocks [95]. The two managers (i.e., virtual function and virtualized infrastructure) enable the communication between the managing and managed entities. Whereas, the orchestrator leverages the automated planning capability from HTN [96] to facilitate composing network management tasks. This allows overcoming low automation management tasks, such as reconfiguring a broken connection, with minimal human intervention.

2.2 Machine learning for networking

Machine learning is a branch of artificial intelligence whose foundational concepts were acquired over the years from contributions in the areas of computer science, mathematics, philosophy, economics, neuroscience, psychology, control theory, and more [97]. In 1959, Arthur Samuel coined the term “Machine Learning”, as “*the field of study that gives*

⁴CIM is an open standard aimed at assisting the management of devices, services, and computer networks by facilitating their modeling [87].

⁵Information models represent managed objects at a conceptual level, independent of any specific protocols used to transport the data. Whereas, data models define managed objects at a lower level of abstraction, including implementation- and protocol-specific details. Multiple data models can be derived from a single information model [88].

computers the ability to learn without being explicitly programmed.” However, ML goes beyond simply learning or extracting knowledge, to utilizing and improving knowledge over time and with experience [98]. Broadly, ML can be divided into three paradigms, based on how the *learning* is achieved [97,99]: *supervised*, *unsupervised*, and *reinforcement* learning.

Supervised learning uses labeled training datasets to create models that map inputs to their corresponding outputs. Then, this learning approach requires labeling methods for establishing the *ground truth* in datasets and “learns” to identify patterns or behaviors in the “known” training datasets. Typically, supervised learning solves *classification* and *regression* problems that pertain to predicting discrete or continuous valued outcomes, respectively (see Figures 2.3(a) and 2.3(b)). For example, a classification problem can be to identify mice and elephant flows. Whereas, a regression problem can be to predict the size of each flow.

Unsupervised learning uses unlabeled training datasets to create models that find dominating structure or patterns in the data. This approach is most suited for *clustering* problems (see Figure 2.3(c)). For instance, outliers detection and density estimation problems in networking, can pertain to grouping different instances of attacks based on their similarities. Between supervised and unsupervised learning resides *semi-supervised* learning to face partial knowledge. That is, having incomplete labels or missing labels for training data.

Reinforcement learning (RL) uses an agent that interacts with the external world to learn by exploring the environment and exploiting the knowledge. The actions are rewarded or penalized. Therefore, the training data constitutes a set of state-action pairs and rewards (or penalties). The agent uses feedback from the environment to learn the best sequence of actions or “policy” to optimize a cumulative reward. Hence, this learning approach is best suited for making cognitive choices, such as decision making, planning, and scheduling [100]. For example, rule extraction from the data that is statistically supported and not predicted (see Figure 2.3(d)).

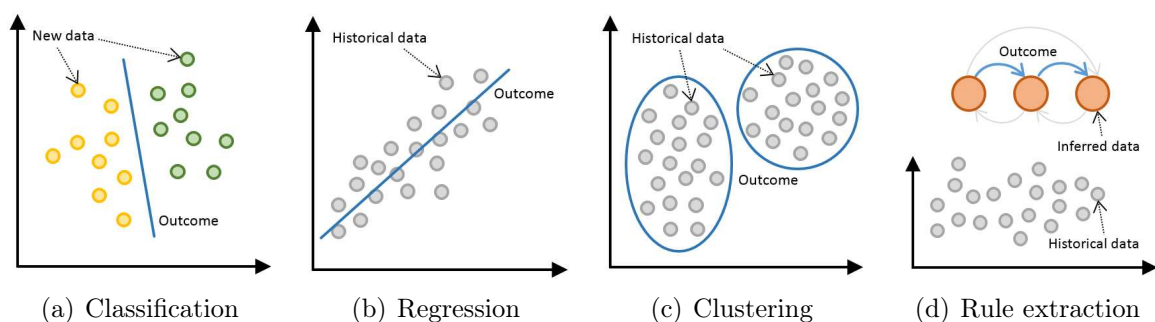


Figure 2.3: Problem categories that benefit from ML paradigms

Though there are different categories of problems that enjoy the benefits of ML, there is a generic approach to building ML-based solutions. Figure 2.4 illustrates the key constituents in designing ML-based solutions for networking. *Data collection* pertains to gathering, generating, and defining the set of data and the set of classes of interest. Depending on the applied ML paradigm, the collected data might be labeled for establishing

the *ground truth*. *Feature engineering* is used to reduce dimensionality in data and identify discriminating features that reduce computational overhead and increase accuracy. *Model learning* refers to training one or multiple models using ML techniques, which carefully analyze the complex inter- and intra-relationships in data to yield the outcome. Finally, *model validation* regards defining the accuracy metrics that measure the performance of the trained models.

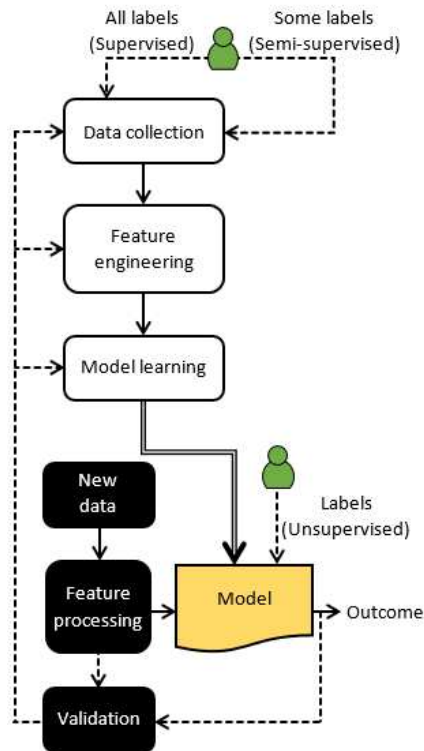


Figure 2.4: The constituents of ML-based solutions

2.2.1 Incremental learning

Most ML approaches, mainly based on supervised and unsupervised learning, implement the classical *batch* learning: all data needed to generate an inference is collected before training and is simultaneously accessed [101]. Usually, batch learning divides the data into *training*, *validation* (also called development), and *test* sets [102]. The training set is leveraged to fit the parameters of an ML model (e.g., weights). Whereas, the validation set is used to choose the suitable hyperparameters of an ML model (e.g., architecture, learning rate, regularization), or choose a model from a pool of ML models. Finally, the test set is used to assess the unbiased performance of the selected model. Note, batch learning considers that both the data and its underlying structure are static.

A common batch setting decomposition of the dataset can conform to 60/20/20% among training, validation, and test datasets, or 70/30% in case validation is not required [102]. These rule-of-thumb decompositions are reasonable for datasets that are not very large. However, in the era of big data, where a dataset can have millions of entries, other extreme decompositions, such as 98/1/1% or 99.8/0.1/0.1%, are also valid.

Several ML studies and practitioners consider that validation and test sets with sizes on the order of tens of thousands of instances are sufficient. In batch learning, validation and testing usually follows one of two methods⁶ [103]: holdout or k -fold cross-validation. In the holdout method, part of the available dataset is set aside and used as a validation (or testing) set. Whereas, in the k -fold cross-validation, the available dataset is randomly divided into k equal subsets. Validation (or testing) process is repeated k times, with $k - 1$ unique subsets for training and the remaining subset for validating (or testing) the model, and the outcomes are averaged over the rounds.

Batch learning has also incorporated some big data upgrades, such as external storage and data subsets, that enable processing large but static datasets [103]. However, the more data available, the less performance to simply output a final static model. Moreover, batch learning fails to handle a continuous supply of changing data (i.e., sequential data or data stream), which is characteristic of the networks and its high dynamicity. Since static models cannot continuously integrate new information, they have to be constantly reconstructed to ensure their validity over time. However, re-training a model from scratch is computationally expensive, time-consuming, and leads to potentially outdated models [104]. An interesting research direction is to achieve *incremental* learning, where the model is re-trained with only the new data.

Incremental learning⁷ refers to continuously updating a model using sequential data (i.e., constantly arriving data with no specific order) without re-processing the data already used [105]. In fact, many datasets, although static, are so large that they would be dealt with as sequential data. Since sequential data can become endless in some domains, including computer networks, incremental learning aims for bounding model complexity and processing time, enabling lifelong learning and a constantly updated model under limited memory resources. Therefore, four fundamental aspects characterize an incremental learning algorithm [103]: (i) it processes an instance at a time in the order that arrives, inspecting it as input only once⁸, (ii) it uses a limited amount of memory, even when processing data much more extensive than available memory, (iii) it restricts the runtime to a limit, which is particularly pivotal for algorithms aimed at real-time applications (e.g., networking); and (iv) it can provide a prediction anytime, no matter the number of instances used for training.

The evaluation of incremental learning algorithms follows one of two methods [103]: holdout and interleaved test-then-train. Holdout represents a natural extension from batch learning, where a single, independent, and sufficiently large (i.e., tens of thousands of instances) test set can provide a valid accuracy measurement. The model can be evaluated on the test set either after the last training or periodically to track its performance over time. In scenarios where the data statistics change over time (a.k.a. concept change),

⁶Other evaluation practices for batch learning exist, such as leave-one-out and bootstrap, but the ML community warns about using them.

⁷The definition of incremental learning is not always consistent in the literature and involves certain ambiguity regarding related terms, including online learning, data stream mining, stream learning, and incremental online learning. The definition presented in this thesis aims to cover the commonalities among such related concepts.

⁸An algorithm may store some instances internally in the short term. However, at some point, it needs to discard some of them to meet memory and time limits.

the test set cannot be static but must be constantly collected using new instances not yet used for training. On the other hand, interleaved test-then-train (a.k.a. prequential) refers to using each individual instance for testing the model before training. Note this method does not need a holdout set, making maximum use of the collected data for both testing and training. In interleaved test-then-train, early mistakes from a poorly trained model (a.k.a. cold start) punish the actual accuracy that incremental algorithms might achieve. Although this effect diminishes over time, evaluation techniques like pretraining and sliding windows help to correct the accuracy measurement. Incremental learning algorithms are usually evaluated using interleaved test-then-train as it easily handles a potentially infinite sequence of instances arriving one after another.

2.2.2 Evolution of machine learning techniques

Research efforts during the last 75 years have given rise to a plethora of ML techniques [97–99, 106]. This section provides a brief history of ML, focusing on the techniques that have been particularly applied in the area of computer networks, including this thesis (see Figure 2.5).

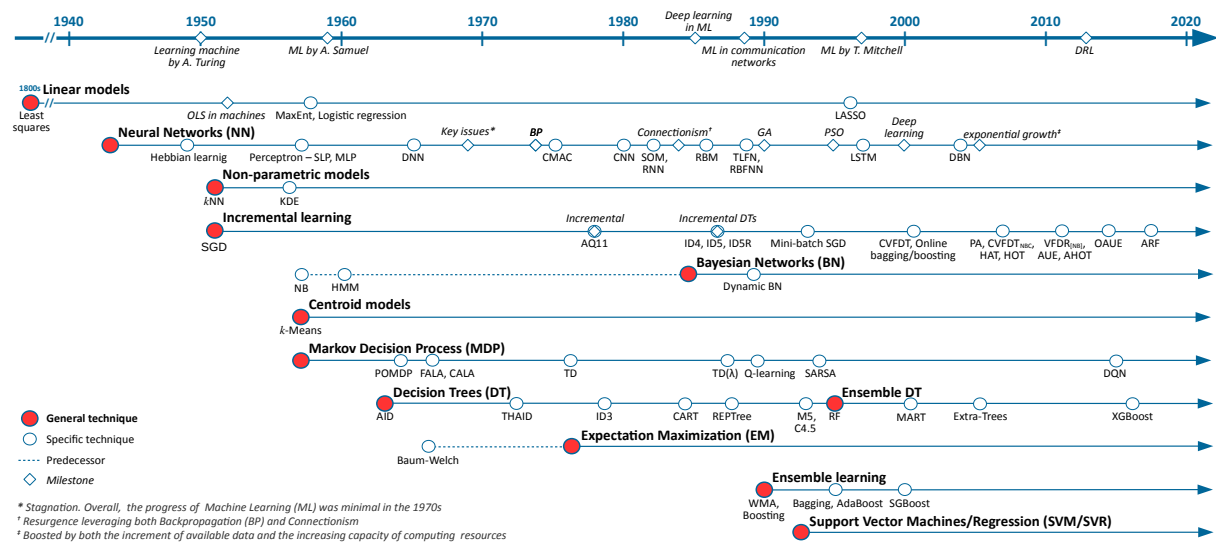


Figure 2.5: The evolution of machine learning techniques with key milestones

The beginning of ML dates back to 1943, when the first mathematical model of Neural Network (NN) for computers was proposed by McCulloch [107]. This model introduced a basic unit called artificial neuron that has been at the center of NN development to this day. However, this early model required to manually establish the correct weights of the connections between neurons. This limitation was addressed in 1949 by Hebbian learning [108], a simple rule-based algorithm for updating the connection weights of the early NN model. Like the neuron unit, Hebbian learning greatly influenced the progress of NN. These two concepts led to the construction of the first NN computer in 1950, called SNARC (Stochastic Neural Analog Reinforcement Computer) [97]. In the same year, Alan Turing proposed a test—where a computer tries to fool a human into believing it is also human—to determine if a computer is capable of showing intelligent behavior.

He described the challenges underlying his idea of a “*learning machine*” in [109]. These developments encouraged many researchers to work on similar approaches, resulting in two decades of enthusiastic and prolific research in the ML area.

In the 1950s, the simplest linear regression model called Ordinary Least Squares (OLS)—derived from the least squares method [110, 111] developed around the 1800s—was used to calculate linear regressions in electro-mechanical desk calculators [112]. To the best of our knowledge, this is the first evidence of using OLS in computing machines. Following this trend, two linear models for conducting classification were introduced: Maximum Entropy (MaxEnt) [113, 114] and logistic regression [115].

Different ML techniques derived from pattern recognition and root-finding problems appeared during this decade too. Research trends centered on pattern recognition exposed two non-parametric models (i.e., not restricted to a bounded set of parameters) capable of performing regression and classification: k -Nearest Neighbors (kNN) [116, 117] and Kernel Density Estimation (KDE) [118], also known as Parzen density [119]. The former uses a distance metric to analyze the data, while the latter applies a kernel function (usually, Gaussian) to estimate the probability density function of the data. On the other hand, research on root-finding optimization introduced the stochastic approximation algorithm [120, 121], which uses successive approximations to find the unique root of a regression function. This algorithm was later referred to as Stochastic Gradient Descent (SGD) [122] since it approximates the true gradient (computed using the whole dataset) by iteratively adding a scaled gradient estimate over every single instance of the dataset—a later proposed variant that iterate over instance subsets, called *mini-batches*, can improve performance and convergence [123]. SGD has become a well-known incremental (a.k.a. online) learning method for training various ML models, such as linear regression and NNs, facing large-scale datasets [124].

The 1950s also witnessed the first applications of the Naïve Bayes (NB) classifier in the fields of pattern recognition [125] and information retrieval [126]. NB, whose foundations date back to the 18th and 19th centuries [127, 128], is a simple probabilistic classifier that applies Bayes’ theorem on features with strong independence assumptions. NB was later generalized using KDE, also known as NB with Kernel Estimation (NBKE), to estimate the conditional probabilities of the features. In the area of clustering, Steinhaus [129] was the first to propose a continuous version of the to be called k -Means algorithm [130], to partition a heterogeneous solid with a given internal mass distribution into k subsets. The proposed centroid model employs a distance metric to partition the data into clusters where the distance to the centroid is minimized.

By the end of the 1950s, the Markov model [131, 132] (elaborated 50 years earlier) was leveraged to construct a process based on discrete-time state transitions and action rewards, named Markov Decision Process (MDP), which formalizes sequential decision-making problems in a fully observable, controlled environment [133]. MDP has been essential for the development of prevailing RL techniques [106]. Research efforts building on the initial NN model flourished too: the modern concept of perceptron was introduced as the first NN model that could learn the weights from input examples [134]. This model describes two NN classes according to the number of layers: Single-Layer Perceptron (SLP), an NN with one input layer and one output layer, and Multi-Layer

Perceptron (MLP), an NN with one or more hidden layers between the input and the output layers. The perceptron model is also known as feedforward NN since the nodes from each layer exhibit directed connections only to the nodes of the next layer. Finally, the term “*Machine Learning*” was coined and defined for the first time by Arthur Samuel (see Section 2.2), who also developed a checkers-playing game that is recognized as the earliest self-learning program [135].

ML research continued to flourish in the 1960s, giving rise to a novel statistical class of the Markov model, named Hidden Markov Model (HMM) [136]. An HMM describes the conditional probabilities between hidden states and visible outputs in a partially observable, autonomous environment. The Baum-Welch algorithm [137] was proposed in the mid-1960s to learn those conditional probabilities. At the same time, MDP continued to instigate various research efforts. The Partially Observable MDP (POMDP) approach to finding optimal or near-optimal control strategies for partially observable stochastic environments, given a complete model of the environment, was first proposed by Cassandra *et al.* [138] in 1965, while the algorithm to find the optimal solution was only devised five years later [139]. Another development in MDP was the learning automata—officially published in 1973 [140]—an RL technique that continuously updates the probabilities of taking actions in an observed environment, according to given rewards. Depending on the nature of the action set, the learning automata is classified as Finite Action-set Learning Automata (FALA) or Continuous Action-set Learning Automata (CALA) [141].

In 1963, Morgan and Sonquis published Automatic Interaction Detection (AID) [142], the first regression tree algorithm that seeks sequential partitioning of an observation set into a series of mutually exclusive subsets, whose means reduces the error in predicting the dependent variable. AID marked the beginning of the first generation of Decision Tree (DT) models. However, the application of DTs to classification problems was only initiated a decade later by Morgan and Messenger’s THeta AID (THAID) [143] algorithm.

In the meantime, the first algorithm for training MLP-NNs with many layers⁹—also known as Deep NN (DNN) in today’s jargon—was published by Ivakhnenko and Lapa in 1965 [145]. This algorithm marked the commencement of the Deep Learning (DL) discipline, though the term only started to be used in the 1980s in the general context of ML, and in the year 2000 in the specific context of NNs [146]. By the end of the 1960s, Minsky and Papertkey’s *Perceptrons* book [147] drew the limitations of perceptrons-based NN through mathematical analysis, marking a historical turn in Artificial Intelligence (AI) and ML in particular, and significantly reducing the research interest for this area over the next several years [97].

Although ML research was progressing slower than projected in the 1970s [97], this decade was marked by milestones that greatly shaped the evolution of ML, and contributed to its success in the following years. These include the Back-Propagation (BP) algorithm [148], the Cerebellar Model Articulation Controller (CMAC) NN model [149], the Expectation Maximization (EM) algorithm [150], the to-be-referred-to as Temporal Difference (TD) learning [151], the Iterative Dichotomiser 3 (ID3) algorithm [152], and the AQ11 learning system [153].

⁹By 1971, the learning algorithm Group Method of Data Handling was capable of training an 8-layer MLP [144]

Werbos’s application of BP—originally a control theory algorithm from the 1960s [154–156]—to train NNs [148] resurrected the research in the area. BP is to date the most popular NN training algorithm and comes in different variants [157], such as SGD (the *de facto* standard algorithm), conjugate gradient, one step secant, Levenberg-Marquardt, and resilient BP. Though, BP is widely used in training NNs, its efficiency depends on the choice of initial weights. In particular, BP has been shown to have slower speed of convergence and to fall into local optima. Over the years, global optimization methods have been proposed to replace BP, including Genetic Algorithms (GA), simulated annealing, and ant colony algorithms [158]. In 1975, Albus proposed CMAC, a new type of NN as an alternative to MLP [149]. Although CMAC was primarily designed as a function modeler for robotic controllers, it has been extensively used in RL and classification problems for its faster learning compared to MLP.

In 1977, in the area of statistical learning, Dempster *et al.* proposed EM, a generalization of the previous iterative, unsupervised methods, such as the Baum-Welch algorithm, for learning the unknown parameters of statistical HMM models [150]. At the same time, Witten developed an RL approach to solve MDPs, inspired by animal behavior and learning theories [151], that was later referred to as TD in Sutton’s work [159, 160]. In this approach, the learning process is driven by the changes, or differences, in predictions over successive time steps, such that the prediction at any given time step is updated to bring it closer to the prediction of the same quantity at the next time step. Towards the end of the 1970s, the second generation of DT models emerged as the ID3 algorithm was released. The algorithm, developed by Quinlan [152], relies on a novel concept for attribute selection based on entropy¹⁰ maximization. ID3 is a precursor to the popular and widely used C4.5 and C5.0 DT algorithms. In addition, Michalski and Larson developed AQ11 [153], a learning system that incrementally generates new rules using the existing ones and new training instances, later discarded after learning. AQ11 is the first evidence of using the term “incremental” in an ML technique, to the best of our knowledge.

The 1980s witnessed a renewed interest in ML research, and in particular in NNs. In the early 1980s, three new classes of NN models emerged, namely Convolutional Neural Network (CNN) [161], Self-Organizing Map (SOM) [162], and Hopfield network [163]. CNN is a feedforward NN specifically designed to be applied to visual imagery analysis and classification, and thus require minimal image preprocessing. Connectivity between neurons in CNNs is inspired by the organization of the animal visual cortex—modeled by Hubel in the 1960s [164, 165]—where the visual field is divided between neurons, each responding to stimuli only in its corresponding region. Similarly to CNN, SOM was also designed for a specific application domain; dimensionality reduction [162]. SOMs employ an unsupervised competitive learning approach, unlike traditional NNs that apply error-correction learning (such as BP with gradient descent).

In 1982, the first form of Recurrent Neural Network (RNN) was introduced by Hopfield. Named after the inventor, Hopfield network is an RNN where the weights connecting the neurons are bidirectional. The modern definition of RNN, as a network where connections between neurons exhibit one or more than one cycle, was introduced by Jordan in 1986 [166]. Cycles provide a structure for internal states or memory allowing RNNs

¹⁰Measure of the uncertainty about a source of messages

to process arbitrary sequences of inputs. As such, RNN are found particularly useful in time series forecasting, handwriting recognition, and speech recognition.

Several key concepts emerged from the 1980s' *connectionism movement*, one of which is the concept of *distributed representation* [167]. Introduced by Hinton in 1986, this concept supports the idea that a system should be represented by many features and that each feature may have different values. Distributed representation establishes a many-to-many relationship between neurons and *(feature,value)* pairs for improved efficiency, such that a *(feature,value)* input is represented by a pattern of activity across neurons as opposed to being locally represented by a single neuron. The second half of the 1980s also witnessed the increase in popularity of the BP algorithm and its successful application in training DNNs [168,169], as well as the emergence of new classes of NNs, such as Restricted Boltzmann Machines (RBM) [170], Time-Lagged Feedforward Network (TLFN) [171], and Radial Basis Function (RBF) NN (RBFNN) [172].

Originally named Harmonium by Smolensky, RBM is a variant of Boltzmann machines [173] with the restriction that there are no connections within any of the network layers, whether it is visible or hidden. Therefore, neurons in RBMs form a bipartite graph. This restriction allows for more efficient and simpler learning compared to traditional Boltzmann machines. RBMs are found useful in a variety of application domains such as dimensionality reduction, feature learning, and classification, as they can be trained in both supervised and unsupervised ways. The popularity of RBMs and the extent of their applicability significantly increased after the mid-2000s as Hinton introduced in 2006 a faster learning method for Boltzmann machines, called Contrastive Divergence [174], making RBMs even more attractive for DL [175]. Interestingly, although the use of the term “deep learning” in the ML community dates back to 1986 [176], it did not apply to NNs at that time.

As aforementioned, TLFN—an MLP that incorporates the time dimension into the model for conducting time series forecasting [171]—and RBFNN—an NN with a weighted set of RBF kernels trained in supervised or unsupervised ways [172]—joined the growing list of NN classes. Indeed, any of these NNs can be employed in a DL architecture, either by implementing a larger number of hidden layers or by stacking multiple simple NNs.

In addition to NNs, several other ML techniques thrived during the 1980s. Among these techniques, Bayesian Network (BN) arose as a Directed Acyclic Graph (DAG) representation for the statistical models in use [177], such as NB and HMM—the latter considered as the simplest dynamic BN [178,179]. Two DT learning algorithms, similar to ID3 but developed independently, referred to as Classification And Regression Trees (CART) [180], were proposed to model classification and regression problems. Another DT algorithm, under the name of Reduced Error Pruning Tree (REPTree), was also introduced for classification. REPTree aimed at building faster and simpler tree models using information gain for splitting, along with reduced-error pruning [181]. DT also experienced its earliest incremental learning algorithms built upon batch models, mainly ID3 (e.g., ID4 [182], ID5 [183], and ID5R [184]), which greatly influenced the incremental DTs in the new millennium.

Towards the end of the 1980s, two TD approaches were proposed for RL: TD(λ) [160] and Q-learning [185]. TD(λ) adds a discount factor ($0 \leq \lambda \leq 1$) that determines to

what extent estimates of previous state-values are eligible for updating based on current errors, in the policy evaluation process. For example, TD(0) only updates the estimate of the value of the state preceding the current state. Q-learning, however, replaces the traditional state-value function of TD by an action-value function (i.e., Q-value) that estimates the utility of taking a specific action in specific states. As of today, Q-learning is the most well-studied and widely-used model-free RL algorithm. By the end of the decade, the application domains of ML started expanding to the operation and management of communication networks [186–188].

In the 1990s, significant advances were realized in ML research, focusing primarily on NNs and DTs. Bio-inspired optimization algorithms, such as GA and Particle Swarm Optimization (PSO), received increasing attention and were used to train NNs for improved performance over the traditional BP-based learning [189, 190]. Probably one of the most important achievements in NNs was the work on Long Short-Term Memory (LSTM), an RNN capable of learning long-term dependencies for solving DL tasks that involve long input sequences [191]. Today, LSTM is widely used in speech recognition as well as natural language processing. In DT research, Quinlan published the M5 algorithm in 1992 [192] to construct tree-based multivariate linear models analogous to piecewise linear functions. One well-known variant of the M5 algorithm is M5P, which aims at building trees for regression models. A year later, Quinlan published C4.5 [193], that builds on and extends ID3 to address most of its practical shortcomings, including data overfitting and training with missing values. C4.5 is to date one of the most important and widely used algorithms in ML and data mining.

Several techniques other than NNs and DTs also prospered in the 1990s. Research on regression analysis propounded the Least Absolute Selection and Shrinkage Operator (LASSO), which performs variable selection and regularization for higher prediction accuracy [194]. Another well-known ML technique introduced in the 1990s was Support Vector Machines (SVM). SVM enables plugging different kernel functions (e.g., linear, polynomial, RBF) to find the optimal solution in high-dimensional feature spaces. SVM-based classifiers find a hyperplane to discriminate between categories. A single-class SVM is a binary classifier that deduces the hyperplane to differentiate between the data belonging to the class against the rest of the data, that is, *one-vs-rest*. A multi-class approach in SVM can be formulated as a series of single class classifiers, where the data is assigned to the class that maximizes an output function. SVM has been widely used primarily for classification, although a regression variant exists, known as Support Vector Regression (SVR) [195]. In addition, SVM can learn incrementally using SGD, though this applies only to models using a linear kernel function. By 1999, an incremental learning variant of SVM appeared for handling perceived and real concept drifts [196].

In the area of RL, State-Action-Reward-State-Action (SARSA) was introduced as a more realistic, however less practical, Q-learning variation [197]. Unlike Q-learning, SARSA does not update the Q-value of an action based on the maximum action-value of the next state, but instead it uses the Q-value of the action chosen in the next state.

A new emerging concept called *ensemble learning* demonstrated that the predictive performance of a single learning model can be improved when combined with other models [97]. As a result, the poor performance of a single predictor or classifier can

be compensated with ensemble learning at the price of (significantly) extra computation. Indeed the results from ensemble learning must be aggregated, and a variety of techniques have been proposed in this matter. The first instances of ensemble learning include Weighted Majority Algorithm (WMA) [198], boosting [199], bootstrap aggregating (or bagging) [200], and Random Forest (RF) [201]. RF focused explicitly on tree models and marked the beginning of a new generation of ensemble DT. In addition, some variants of the original boosting algorithm were also developed, such as Adaptive Boosting (AdaBoost) [202] and Stochastic Gradient Boosting (SGBost) [203].

These advances in ML facilitated the successful deployment of major use cases in the 1990s, particularly, handwriting recognition [204] and data mining [205]. The latter represented a great shift to data-driven ML, and since then it has been applied in many areas (e.g., , retail, finance, manufacturing, medicine, science) for processing huge amounts of data to build models with valuable use [98]. Furthermore, from a conceptual perspective, Tom Mitchell formally defined ML: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ” [206].

The 21st century began with a new wave of increasing interest in SVM and ensemble learning, and in particular ensemble DT. Research efforts in the field generated some of the the most widely used implementations of ensemble DT as of today: Multiple Additive Regression Trees (MART) [207], extra-trees [208], and eXtreme Gradient Boosting (XGBoost) [209]. MART and XGBoost are respectively a commercial and open source implementation of Friedman’s Gradient Boosting Decision Tree (GBDT) algorithm; an ensemble DT algorithm based on gradient boosting [203, 207]. Extra-trees stands for *extremely randomized trees*, an ensemble DT algorithm that builds random trees based on k randomly chosen features. However instead to computing an optimal split-point for each one of the k features at each node as in RF, extra-trees selects a split-point randomly for reduced computational complexity.

At the same time, the popularity of DL increased significantly after the term “deep learning” was first introduced in the context of NNs in 2000 [146]. However, the attractiveness of DNN started decreasing shortly after due to the experienced difficulty of training DNNs using BP (e.g., vanishing gradient problem), in addition to the increasing competitiveness of other ML techniques (e.g., SVM) [98]. Hinton’s work on Deep Belief Networks (DBN), published in 2006 [210], gave a new breath and strength to research in DNNs. DBN introduced an efficient training strategy for DL models, which was further used successfully in different classes of DNNs [211, 212]. The development in ML (particularly, in DNNs) grew exponentially with advances in storage capacity and large-scale data processing (i.e., big data) [98]. This wave of popularity in DL has continued to this day, yielding major research advances over the years. One approach that has recently received tremendous attention is Deep RL (DRL), which incorporates DL models into RL for solving complex problems. For example, Deep Q-Networks (DQN)—a combination of DNN and Q-learning—was proposed for mastering video games [213]. Although the term DRL was coined recently, this concept was already discussed and applied 25 years ago [214, 215].

The 2000s have also been fruitful for the incremental learning research community.

Although well-known ML techniques, such as NN and kNN, fit naturally to incremental learning, several algorithms emerged during the last 20 years, particularly for incremental DT. In 2001, the Very Fast Decision Tree (VFDT) algorithm [216] (a.k.a. Hoeffding tree) exposed an incremental DT that selects a root node from the first training instances and grows the tree from the leaf nodes. However, VFDT aimed for static concepts; hence, an extension of this algorithm, called Concept-adapting VFDT (CVFDT) [217], addresses concept drift by maintaining a set of alternate DTs and storing instances over a window of time. CVFDT experienced further refinements during the subsequent years. CVFDT_{NBC} [218] incorporated NB classifiers in the leaf nodes of the tree, which demonstrated better accuracy than its predecessor. Hoeffding Adaptive Tree (HAT) [219] introduced ADaptive WINdowing (ADWIN) for monitoring the accuracy of the DT branches, which are replaced with more accurate branches when their performance decreases. Hoeffding Option Tree (HOT) [220] refers to an ensemble method that uses additional option nodes to build a single structure with multiple CVFDTs as separate paths. Adaptive Hoeffding Option Tree (AHOT) [103] extends HOT by storing in each leaf the error estimation, computed using Exponentially Weighted Moving Average (EWMA). Finally, Adaptive Random Forest (ARF) [221] provides another incremental ensemble DT, which adapts the widely used RF ensemble. In contrast to RF, ARF comes with a drift and warning detector per base DT that enables selective resets and tree replacements—when the warning becomes a drift.

Other non-strictly-DT incremental learning techniques also appeared during and after the 2000s. In 2001, Oza and Russell [222] developed incremental variants of the bagging and boosting ensemble algorithms, simply referred to as online bagging and online boosting, respectively. Both incremental ensembles allow using different learners as the base model, such as NB and any incremental DT, though recent implementations commonly use CVFDT_{NBC}. Five years later, the online Passive-Aggressive (PA) algorithms [223] characterized a family of margin-based incremental learning techniques for solving classification and regression problems. These algorithms (*passively*) keep the same weights when no loss is present but (*aggressively*) update them when the loss is positive. In 2011, two new approaches emerged: Very Fast Decision Rules (VFDR) [224] and Accuracy Updated Ensemble (AUE) [225]. VFDR introduced a rule-based incremental algorithm aiming at more interpretability and flexibility than DTs. This single-pass algorithm continuously learns ordered and unordered rule sets and classifies using the majority class strategy. A variant called VFDR_{NB} [224] incorporates NB classifiers that improve the accuracy. On the other hand, AUE presented an adaptive block-based ensemble for incremental learning, which selects and updates the base classifiers regarding the current distribution in a block series. An improvement to AUE, called Online Accuracy Updated Ensemble (OAUE) [226], introduced a new error-based weighting function to incrementally train and evaluate the base classifiers. Similar to online bagging and boosting, recent implementations of both AUE and OAUE commonly use CVFDT_{NBC} as the base learner.

It is important to mention that the evolution in ML research has enabled improved learning capabilities which were found useful in several application domains, ranging from games, image and speech recognition, network operation and management, to self-driving cars [227].

2.2.3 Cognitive networking

In theory, ML can be used for automating network operations and management as it allows extracting knowledge from data. However, application of ML for incorporating intelligence and autonomy in networking is a non-trivial task. Prohibiting factors include the distributed control and vendor-specific nature of legacy network devices, lack of available data, and cost of compute and storage resources. Several technological advances have been made in the last decade to overcome these limitations. The advent of network softwarization and programmability through SDN and NFV offers centralized control and alleviates vendor lock-in. The advances in ML along with the proliferation of new sources of data and big data analytics platforms provide abundant data and extract knowledge from them. Furthermore, the availability of seemingly *infinite* storage and compute resources through the cloud overcomes the cost of resources. These together provide the environment to realize the vision of cognitive networks.

Knowledge-Defined Networking (KDN) [228] depicts a recent initiative for cognitive networking. This approach revisits the inclusion of a *knowledge plane* proposed initially 15 years before for the Internet [229]. KDN defines the knowledge plane at the top of the SDN architecture (see Section 2.1), replacing the application plane; hence, it can interact with the management and control planes to obtain a rich view and control over the network. The knowledge plane enables learning from the network behavior by processing operation and management data collected by the other SDN planes. Via ML techniques, such data become knowledge aimed at providing recommendations and making network decisions—either automated or human intervention.

In the context of autonomic systems and networks, IBM’s autonomic computing architecture [230] is to date the most influential reference model. It comprises several layers of autonomic managers. The behavior of each manager is governed by the MAPE control loop that consists of four functions; *Monitor*, *Analyze*, *Plan*, and *Execute*. As shown in Figure 2.6, the *Knowledge* source is orthogonal to every MAPE function. Functions can retrieve data from and/or log created knowledge to the Knowledge source. For example, the *Analyze* function obtains information about the historical behavior of a managed resource and stores the ML models and the analytics it generates in the Knowledge source.

In [230], we observe that cognition has been restricted to the *Analyze* function, which inhibits the ability to achieve closed-loop cognitive network management. In [42], we proposed to incorporate cognition at every function in the loop. For example, the *Monitor* function should be able to determine the what, when, and where to monitor. ML can be leveraged to build this cognition in every function and allow each function to operate in full autonomy. Therefore, we extend IBM’s MAPE control loop into a cognitive control loop that we denote as C-MAPE. As illustrated in Figure 2.6, cognition is achieved by introducing learning and inference in every function.

- *C-Monitor* function refers to the cognitive monitor that performs intelligent probing. For instance, when the network is overloaded, the C-Monitor function may decide to reduce the probing rate and instead perform regression for data prediction.
- *C-Analyze* function is responsible for detecting or predicting changes in the network environment (e.g., faults, policy violations, frauds, performance degradation, and

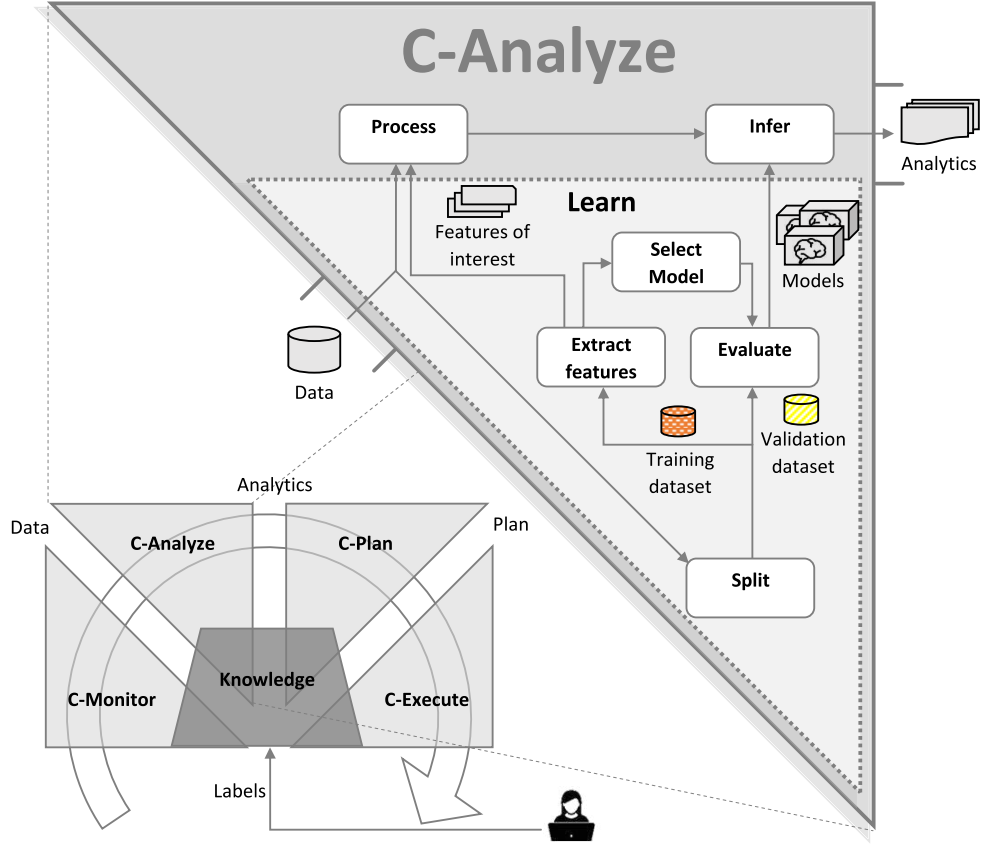


Figure 2.6: Cognitive control loop for network management

attacks). ML has been leveraged to address some of these challenges in each of the FCAPS management areas, as discussed in [42].

- *C-Plan* function can leverage ML to develop an intelligent Automated Planning (AP) engine that reacts to changes in the network by selecting or composing a change plan. In the last decade, AP systems have been applied to real-world problems and have been relying on ML for automating the extraction and organization of knowledge (e.g., plans, execution traces), and for decision making [231].
- *C-Execute* function can use ML to schedule the generated plans and determine the course of action should the execution of a plan fail. These tasks lend themselves naturally to RL where the C-Execute agent could *exploit* past successful experiences to generate optimal execution policies, and *explore* new actions should the execution plan fail.

Closing the control loop is achieved by monitoring the state of the network to measure the impact of the change plan.

2.3 Traffic engineering in data center networks

This section first reviews the concepts of DCN and traffic engineering to contextualize the approach of this dissertation. Then, it provides a literature review about multipath

routing for load-balancing in DCNs, focusing on the seminal works that use SDN and ML for addressing such a challenge.

2.3.1 Data center network

DCN encompasses the communication infrastructure of data centers that aim to interconnect a large number of servers with significant bandwidth capacity in order to achieve high throughput and low-latency [1]. Different DCN topologies have been designed to meet such performance requirements. In general, these DCN topologies can be classified into three categories based on the routing and switching equipment used to forward or process network traffic [232–234]: *switch-centric*, *server-centric*, and *hybrid*. Switch-centric topologies consider switches as the only relay nodes (i.e., routing decisions) and servers as mere endpoints, such as VL2 [235] and Jellyfish [236]. Server-centric topologies define servers as both endpoints and relaying nodes, such as BCube [237] and DCell [238]. Hybrid topologies use a combination of electrical, optical, and/or wireless equipment to add extra bandwidth capacity to the DCN, such as FireFly [239] and Helios [240].

Currently, most of the DCN deployments follow the switch-centric topology, particularly the tree-based design, such as Facebook’s Altoona [8] and Google’s Jupiter [241]. In switch-centric, the switches are interconnected in a hierarchical model with multiple layers and the servers are connected to the switches of the lowest layer, known as *edge* or Top-of-Rack (ToR). The tree-based design is an instance of the Clos network with a degree defined according to the network scale indicating the number of layers. For example, VL2 [235] and Fat-tree [242] arrange low-cost commodity switches in a tree-based topology with fourth-degree, namely, from bottom-up, one layer of servers and three layers of switches: *edge*, *aggregation*, and *core* (see Figure 2.7). Differing from this tree-based design, Jellyfish [236] uses a random regular graph to interconnect the switches at the edge layer. Although Jellyfish provides some benefits over the tree-based topologies, such as higher capacity, shorter paths, and more resilience to failures and wiring errors, it is more challenging and complex in terms of cabling, management, and routing.

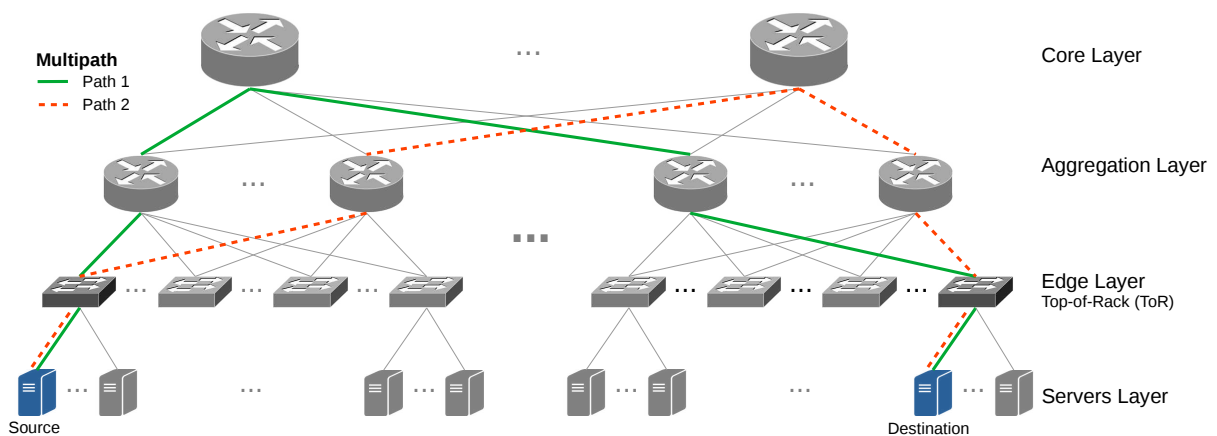


Figure 2.7: Tree-based DCN with fourth degree and multipath routing

In addition, several research works have analyzed the traffic characteristics of switch-centric tree-based DCNs, converging to similar patterns [8, 9, 11]: (i) 10% to 25% of

links are hot-spots, varying over time; (ii) less than 25% of capacity is utilized, even with over-subscription; (iii) around 80% of flows (i.e., mice) carry less than 10% of total bytes, last less than 10 seconds, and transmit less than 10KB, while 20% of flows (i.e., elephants) carry almost 80% of total bytes, last up to 500 seconds, and transmit up to 5GB; (iv) almost every flow at the ToR switch presents an inter-arrival time less than 100 milliseconds; and (v) the number of active flows can be up to 10,000 flows per second.

2.3.2 Traffic engineering

Traffic engineering involves the methods for measuring and managing network traffic to optimize the performance of the network [3, 243]. This optimization requires providing appropriate traffic requirements (e.g., throughput, delay, packet loss) while efficiently—in terms of cost and reliability—utilizing network resources (e.g., bandwidth).

Traffic engineering encompasses four main dimensions [49]: flow management, fault-tolerance, topology update, and traffic analysis. Flow management is about mapping and controlling the traffic flows in the network for optimizing the routing function to steer traffic (from ingress nodes to egress nodes) in the most effective way. Fault-tolerance refers to ensuring network reliability by providing mechanisms that enhance network integrity and by embracing policies emphasizing network survivability. Topology update involves managing the capacity of the network in order to carry out planned changes, such as network policy modifications. Traffic analysis deals with monitoring the performance of the network and verify the compliance with network performance goals to evaluate and debug the effectiveness of the applied traffic engineering methods. Each dimension may operate at multiple levels of temporal resolution, ranging from a few nanoseconds to possibly years. For example, topology update works at very coarse temporal levels, from days to years, while flow management operates at finer levels of temporal resolution, from microseconds to hours.

Load-balancing is one of the most well-known traffic engineering methods. As part of the flow management dimension, load-balancing controls and optimizes the routing function to minimize the maximum load across the links [2]. The goal is mapping traffic flows from the heavily loaded paths to the lightly loaded paths for avoiding congestion (i.e., hot-spots) and increasing network throughput and resource utilization. *Multipath routing* has shown to effectively achieve load-balancing by distributing traffic over multiple concurrent paths such that all the links are optimally loaded [4]. Figure 2.7 depicts two disjoint¹¹ paths in a tree-based DCN. In practice, multipath routing protocols may split the traffic at different levels of granularity, such as per-flow, per-sub-flow, and per-packet. In addition, these protocols may run at distinct layers of the TCP/IP model. For example, ECMP [7] and valiant load balancing [244] work at the network layer, while Multi-Path TCP (MPTCP) [245] operates at the transport layer.

¹¹No common nodes or links except for source and destination.

2.3.3 Multipath routing in data center networks

From a general perspective, multipath routing in DCNs divides into two categories: distributed and SDN-based (i.e., centralized). Initially, distributed approaches were the standard for multipath routing. However, with the emergence of centralized network programmability, SDN-based multipath routing has gained the same level of attention. Recent investigations that have adopted the SDN-based approach aim to optimize the routing function by using ML techniques for predicting flow traffic characteristics.

Distributed multipath routing

Distributed multipath routing mechanisms place routing decisions at either the switches or servers of a DCN. These routing decisions can be oblivious to traffic conditions or based on feedback information from the network (e.g., congestion). In addition, traffic splitting in distributed multipath routing is conducted at different levels of granularity (e.g., flow, packet, sub-flow). The following paragraphs describe the seminal works focused on distributed multipath routing.

ECMP [6,7] represents the state-of-the-art distributed mechanism for multipath routing in DCNs. ECMP is oblivious to traffic conditions and splits the traffic at the level of flows. Generally, ECMP applies a hash function at every switch on selected packet headers to assign each incoming flow to an output port. The output port belongs to one of the several equal-cost forwarding paths maintained by the switch for reaching a destination. A key limitation is that the broad distribution of flow sizes in DCNs (i.e., mice and elephants) cause hot-spots in ECMP-based routing [9,11]. This is because two or more elephant flows can collide on their hash and end up on the same output port. Since ECMP does not account for either current network utilization or flow size, the resulting collisions overwhelm switch buffers and degrade overall switch and link utilization. In addition, the performance of ECMP degrades significantly during asymmetric topologies caused by link failures.

Despite ECMP limitations, it remains as the standard multipath routing mechanism for load-balancing in today's DCNs [5]. For example, Facebook's DCN in Altoona [12] employs BGP to populate the routing tables and ECMP for routing through the equal-cost paths.

Motivated by the drawbacks of ECMP's flow splitting, Random Packet Spraying [13] and Digit Reversal Bouncing (DRB) [14] split the traffic at the level of packets through the different equal-cost paths. However, these works rely on an ideal symmetry of the network to avoid the adverse effects of packet reordering¹² on TCP. To deal with the asymmetry caused by failures, DRB includes a topology update mechanism, though the distributed nature of this mechanism is not suitable for large-scale DCNs.

Other approaches have also addressed asymmetry by proposing topology-dependent weighing of paths and using distinct granularity levels for splitting traffic. For example, Weighted Cost Multiple-Path (WCMP) [16] extends ECMP to support weighted flow-level splits at switches by repeating the same next-hop multiple times. Whereas, Presto [15]

¹²Packet reordering can cause TCP to unnecessarily reduce the sender's rate, leading to severe throughput inefficiencies.

implements weights at the servers and splits flows at the level of TCP Segmentation Offload (TCO) units, termed as *flowcells*. However, WCMP and Presto rely on static weights that are generally sub-optimal with asymmetry, particularly for dynamic traffic workloads. In terms of traffic splitting, WCMP inherits the elephant collision problem from ECMP, while the sub-flow splitting defined by Presto is prone to packet reordering. Moreover, neither of these distributed multipath routing mechanisms is aware of traffic conditions, causing degraded performance during link failures.

In light of these gaps, some works have used knowledge of congestion on different paths to make routing decisions. LocalFlow [18] and Flare [19] rely on local measurements of traffic congestion to balance the load on the switch ports. However, they lack taking global congestion information into account, yielding sub-optimal results for high varying traffic. Therefore, further approaches use feedback from the network to gather path-wise congestion information and shift traffic to less-congested paths. The congestion feedback can be collected either at the servers (e.g., FlowBender [17], LetFlow [20], MPTCP [245]) or switches (e.g., HULA [21], CONGA [22], DeTail [246]) of the network. These global congestion-aware mechanisms achieve better throughput and delay results in DCNs, yet at the cost of significant implementation complexity or specialized hardware support—software implementation produces sub-optimal results—at the servers or switches of the network.

On the other hand, the level of granularity for splitting traffic in congestion-aware multipath routing is variable. FlowBender works at flow-level, DeTail at packet-level, and the rest at sub-flow-level. Particularly, LocalFlow conducts a *spatial* flow splitting based on TCP sequence numbers, while MPTCP splits a TCP flow into multiple sub-flows by varying port numbers. Flare, CONGA, HULA, and LetFlow rely on the concept of *flowlets* [247], defined as a burst of packets from a flow separated by enough time gaps. As discussed before, the flow splitting carries the elephant collision problem, while the packet and sub-flow splitting potentially cause packet reordering. The latter because sub-flow splitting lack an exhaustive study about the appropriate space or time between sub-flows for achieving the best performance without causing packet reordering.

Table 2.1 outlines the limitations of distributed multipath routing. To summarize, the distributed multipath routing mechanisms based on global congestion-awareness and sub-flow splitting (particularly, flowlets) provide great results for load-balancing traffic in DCNs. However, they require the implementation of specialized hardware in the network, increasing the capital and operational expenditure for deploying a DCN. Moreover, these distributed multipath routing mechanisms rely on a static separation between sub-flows, which is not suitable for the varying traffic in DCNs.

Multipath routing based on software-defined networking

SDN-based multipath routing mechanisms leverage the potential of network programmability for enabling a centralized controller to make routing decisions in a DCN. The controller has a global view of the network status and instructs the switches for packet forwarding. The routing decisions rely on *flow detection* methods that classify flows, mostly into two classes: mice and elephants. Such classification occurs either at the

Table 2.1: Summary of the limitations of distributed multipath routing

Perspective	Approach	Gaps
Traffic conditions awareness	Oblivious [7, 13–16]	- Performance degradation due to asymmetry of network topology caused by link failures - Routing decisions not based on traffic conditions
	Local congestion [18, 19]	- Routing decisions not based on global congestion data - Require specialized hardware implementation
	Global congestion [17, 20–22, 245, 246]	- Significant implementation complexity - Require specialized hardware implementation
Traffic splitting granularity	Flow-level [7, 16, 17]	- Performance degradation due to hot-spots caused by collisions of large, long lived flows (i.e., elephant flows)
	Packet-level [13, 14, 246]	- Potential packet reordering - Rely on the symmetry of the network for avoiding packet reordering
	Sub-flow-level [15, 18–22, 245]	- No guarantee that two distinct sub-flows take different paths - Potential packet reordering - Unclear about the spatial or time space between sub-flows for avoiding packet reordering

controller-side (e.g., by pulling or sampling traffic statistics), switch-side, or server-side of SDDCNs. Moreover, some flow detection methods have leveraged ML techniques to classify flows proactively (i.e., prediction). In the following, the seminal works centered on Software-Defined Networking-based multipath routing are discussed.

Hedera [26] represents the state-of-the-art Software-Defined Networking-based mechanism for multipath routing in DCNs. Hedera defines a centralized controller that periodically pulls flow statistics from ToR-switches to discriminate elephant flows from mouse flows (i.e., binary classification). By default, Hedera assumes all flows as mice, forwarding them using ECMP, until a pre-defined threshold rate (10% of bandwidth, 100Mbps in implementation) is reached. Hedera then executes a simple routing algorithm that dynamically computes a suitable path for the detected elephant flow and installs the corresponding rules on the switches along that path. PMCE [248] provides an enhanced routing algorithm for improving the performance of Hedera.

However, the short inter-arrival time of flows at ToR-switches ($< 100ms$) [8, 9, 11] requires a high rate of statistics pulling for achieving good performance (e.g., $< 500ms$ to perform better than ECMP [245]). This high rate along with the huge amount of active flows in ToR-switches (up to 10,000 flows) greatly increases traffic overhead and controller processing, negatively affecting the performance of the network traffic. Moreover, since Hedera installs a flow rule per each active flow in the ToR-switches, the limited memory capacity of switches restricts its scalability. Furthermore, Hedera’s binary classification is too simple for the wide distribution of flow sizes in DCNs (i.e., tens of bytes to hundreds of megabytes), causing the routing algorithms to make inaccurate decisions regarding the traffic volume across a forwarding path.

To address the traffic overhead shortcoming of Hedera, Devoflow [27] reduces the number of interactions between the controller and switches by introducing a rule cloning

action for wildcard OpenFlow rules. Each cloned wildcard rule includes a trigger (i.e., a counter and a comparator) based on a pre-defined threshold (128KB, 1MB, or 10MB) that enables the switches to identify elephant flows. The detected elephant flows are sent to the controller, which calculates the least congested path. The mouse flows are locally handled by the switch through multipath and rapid re-routing actions (included as *select* and *fast failover* group types in OpenFlow v1.5.1 [55], respectively). A key limitation is that the rule cloning action is not supported in OpenFlow, therefore, DevoFlow requires a specialized switch hardware implementation. In addition, although the latest version of OpenFlow (i.e., 1.5.1) support threshold-based triggers, setting a trigger for each flow would limit the scalability and impose a lot of processing to the switch. On the other hand, the limitations of conducting a simple binary classification of flow sizes remain in Devoflow.

Similarly, Mahout [29] tackles the traffic overhead shortcoming of Hedera by monitoring and detecting elephant flows at the server-side of SDDCNs through a shim layer in the operating system. When an elephant flow is detected, the server marks the subsequent packets of the detected elephant flow. ToR-switches recognize these marked packets and send the first of each flow to the controller. The controller then computes and installs a path for the marked packets. As in Hedera, the packets without marks (i.e., mice) are routed using ECMP. Mahout greatly reduces traffic overhead, however, at the cost of software modifications in the servers of SDDCNs. Moreover, inaccurate routing decisions also appear in Mahout since it relies on a simple binary classification of flow sizes. MiceTrap [28] employs the same mechanism of Mahout for identifying and handling elephant flows, while proposing an aggregation module for managing mice flows. This mice aggregation module reduces the number of rules installed on switches for handling mice flows. Nevertheless, the shortcomings of Mahout persist in MiceTrap.

The Elephant Sensitive Hierarchical Statistics Pulling (ESHSP) approach [249] proposes an iterative process to detect elephant flows by decomposing the flow space until an elephant flow is isolated from the others. ESHSP uses a combination of aggregate and individual statistics messages of OpenFlow to reduce the traffic overhead generated by Hedera-like approaches. However, the traffic overhead produced by ESHSP is significantly higher than switch-side and server-side approaches, though it does not require hardware or software modifications in SDDCNs. While ESHSP does not perform further actions with the detected elephant flows, the routing algorithms defined in works like Hedera and Mahout can be easily integrated to it. In addition, ESHSP also suffer from the problems of relying on a simple binary classification of flow sizes.

Sampling is another method that has been used for collecting data to detect elephant flows without requiring hardware or software modifications. Sampling-based DevoFlow [27] and TinyFlow [25] adopt the packet sampling mechanism from sFlow [250] to identify elephant flows in DCNs. SDEFIX [251] has also integrated sFlow for detecting elephant flows, though in Internet exchange points. In particular, DevoFlow exposes sampling as an alternative to the threshold-based triggers for cloned wildcard rules. However, this sampling-based DevoFlow rely on a flow-level simulation that does not simulate actual packets but assume a distribution of packets as reported by earlier works [252]. Therefore, the evaluation of the sampling-based DevoFlow is biased and might not represent

the limitations of using sampling for detecting and scheduling elephant flows.

In the same context of SDDCNs, TinyFlow routes all flows using ECMP by default until a pre-defined threshold is exceeded. Once an elephant flow is detected through sampling, the TinyFlow controller installs a new rule on the ToR-switch that monitors the byte count of that flow. When the byte count exceeds a pre-defined threshold, the switch resets the byte count and chooses a different egress port to route the elephant flow through a different equal-cost path. A key limitation of TinyFlow is that it imposes processing overhead on the switch for monitoring each elephant and changing the egress port accordingly, which was not evaluated. On the other hand, the elephant detection based on sampling generates a traffic overhead lower than pulling statistics, though still significantly higher than switch-side and server-side approaches. In addition, since only a small fraction of packets are sampled (typically, 1 in 1000), the elephant detection is not as accurate as in the other approaches.

A common problem of these Software-Defined Networking-based multipath routing mechanisms is that they can only detect elephant flows reactively when their traffic characteristics (e.g., flow size or flow rate) surpass a pre-defined static threshold. This is not ideal since hot-spots may occur until traffic characteristics are detected and new paths are chosen by the routing algorithms. For this reason, recent works have incorporated ML techniques at the controller-side of SDDCNs to predict traffic characteristics and to make routing decisions proactively.

Early ML-based approaches implement flow size prediction methods that learn from centralized data collected by periodically pulling flow statistics from switches (similar to Hedera). Xiao *et al.* [253] introduces a cost-sensitive C4.5 DT classifier. The accuracy of this classifier heavily depends on the choice of the costs for identifying the class of a flow (i.e., mouse or elephant). Experiments conducted on two real datasets, from a trans-Pacific line [254] and a private data center edge link, demonstrated that incorporating cost-sensitive to C4.5 improves its accuracy for elephant detection. Similarly, the Online Flow Size Prediction (OFSP) approach [32] explores different ML techniques for elephant flow detection. These include gaussian regression, BN, and NN. Experiments were performed on three public real datasets from two university DCNs [255] and an academic building at Dartmouth College [256] with over three million flows each. In contrast to the previous work, OFSP employs the predicted flow size for making routing decisions. Presumably mouse flows are routed through ECMP, whereas elephant flows are routed through the least congested path. However, both approaches assume that the data is centralized, requiring the centralized controller to periodically pull flow statistics from the switches. This collection of data increases traffic overhead, which exponentially grows as these approaches require per-packet headers. In addition, as other non-ML approaches, the problems of a static threshold for simple binary classification of flow sizes remain.

Other ML-based approaches applied similar learning algorithms but on data collected by sampling. The Efficient Sampling and Classification Approach (ESCA) [35] uses sampling for collecting data and proposes a two-phase elephant flow detection. In the first phase, the approach improves sampling efficiency by estimating the arrival interval of elephant flows and filtering out redundant samples using a filtering flow table, which requires modifications in the OpenFlow specification. In the second phase, the approach classifies

samples with a new supervised classification algorithm based on the C4.5 DT. Once classified a flow, a differentiated scheduling approach called DIFFERENCE [34] searches for the best path using a specific algorithm for each flow class: a blocking-island-based algorithm for elephants and a weighted multipath algorithm for mice. Extensive experiment results demonstrated that ESCA can provide accurate detection with less sampled packets and shorter detection time than sFlow-based approaches (e.g., DevoFlow and TinyFlow). However, the proposed sampling method depends on non-existing SDN specifications, hence, requiring custom-made switch hardware.

FlowSeer [33] also leverages sampling and DTs for performing a two-phase cooperative classification for elephant detection. In the first phase, the controller applies a simple DT algorithm (i.e., C4.5) for identifying potential elephant flows. The statistics for this phase are collected by an unsupervised flow sampling method based on wildcard rules. Then, the switches are instructed to forward the headers of the first five packets of the potential elephant flows to the controller. In the second phase, the controller use an incremental DT algorithm (i.e., Hoeffding tree) to detect true elephant flows from the potential ones and further classify them into five categories regarding their size and duration. The median of the classified range is used by the routing algorithm as the predicted demand for computing the best path. Although a five-class classification is much better than a binary classification, a finer granularity prediction is desirable for improving the decisions of the routing algorithm. In addition, FlowSeer carries the limitations of sampling statistics collection, including inaccurate elephant detection and moderate traffic overhead.

All these mechanisms for detecting elephant flows are pre-configured with a fixed threshold value, which might cause high detection error rates when working with the frequently changing traffic of DCNs. Motivated by this shortcoming, Liu *et al.* [257] introduce an adaptive approach for elephant flow detection by adopting a dynamical traffic learning algorithm to configure the threshold value. Although the results show improvement compared to other methods, this work relies on pulling flow statistics, which generates high traffic overhead, and conducts a simple binary classification of flow sizes.

Rather than flow detection methods, some SDN-based multipath routing approaches have used the short-term and partial predictability of the traffic matrix to make routing decisions. MicroTE [258] proposes software modifications into each server (similar to Mahout) for incorporating a monitoring component that collects network traffic. Only one server per rack is responsible for aggregating, processing, and summarizing the network statistics for the entire rack. This designated server then determines the matrix traffic predictability and communicates this information to the controller. The controller computes paths for predictable traffic, while the unpredictable traffic is routed in a weighted form of ECMP. Similarly, Nie *et al.* [259] predicts and estimates the traffic matrix in DCNs but applying a DL technique (i.e., DBN). The authors demonstrate through simulations that the proposed method can capture the short time scale property of traffic flows faithfully. However, the bursty nature of the traffic in DCNs makes traffic matrix prediction questionable. Moreover, the short time scale predictability of the traffic matrix ($1 - 2s$) might not be enough for the whole process: sending information to the controller, making routing decisions, and installing the computed paths on the switches. This concern is more severe for the DBN approach than for MicroTE due to the time and

processing requirements for training DL techniques. Further investigation is required to clarify these questions.

Table 2.2 summarizes the limitations of the reviewed SDN-based multipath routing mechanisms. In general, SDN-based multipath routing have demonstrated improvements over the state-of-the-art multipath routing (i.e., ECMP). However, the existing mechanisms lack finding the best trade-off between traffic overhead, data collection accuracy, and network modifications. In addition, there is still room for improving the routing decisions of SDN-based multipath routing by analyzing finer traffic characteristics.

Table 2.2: Summary of the limitations of SDN-based multipath routing

Perspective	Approach	Gaps
Traffic analysis location	Centralized by pulling [26, 32, 248, 249, 253, 257, 259]	- High traffic overhead - High controller processing - Low resource scalability
	Centralized by sampling [25, 27, 33–35, 251]	- Moderate traffic overhead - Moderate controller processing - Inaccurate collection of traffic statistics
	Distributed at switches [27]	- Specialized hardware - Low resource scalability - High switch processing
	Distributed at servers [28, 29, 258]	- Software modifications
Traffic characteristics	Binary classification of flow sizes - <i>non-ML</i> [25–29, 248, 249, 251] - <i>ML-based</i> [32, 34, 35, 253]	- Static threshold to identify elephants - Coarse classification of flow sizes (two classes) - Reactive detection of traffic characteristics (<i>non-ML approaches</i>)
	Multiclass classification of flow sizes based on ML [33]	- Moderate classification of flow sizes (four to five classes)
	Adaptive binary classification of flow sizes based on ML [257]	- Coarse classification of flow sizes
	Prediction of traffic matrix - <i>non-ML</i> [258] - <i>ML-based</i> [259]	- Not suitable for the bursty traffic in DCNs - Short time predictability of traffic matrix

2.4 Final remarks

First, this chapter detailed the traditional three-plane architecture for deploying an SDN-based network as well as the later inclusion of the management plane in the architecture. We extended such a management plane for integrated control and monitoring of heterogeneous SDNs in one or more domains. Our management plane included an information model that leveraged either CIM or YANG to represent the SDN architecture. We also defined our management plane in the context of network automation by introducing an SDN management architecture based on HTN and NFV. Subsequently, the chapter provided a primer on ML, which discussed different categories of ML-based approaches, including

supervised, unsupervised, reinforcement, batch, and incremental learning. This primer also included a brief history of ML techniques used in networking and the vision of cognitive networks towards automating operations and management. The latter included our C-MAPE approach that incorporates cognition in every function of IBM's autonomic computing control loop. Finally, this chapter exposed the concepts and a literature review related to multipath routing for load-balancing in DCNs, focusing on the seminal works that use SDN and ML for addressing such a challenge. From the literature review, we can conclude that SDN-based multipath routing have demonstrated improvements over the prevalent ECMP. However, the existing SDN-based mechanisms lack finding the best trade-off between traffic overhead, data collection accuracy, and network modifications. Furthermore, finer traffic characteristics is desirable for improving the routing decisions of SDN-based multipath routing.

Chapter 3

Flow detection using online incremental learning at the server-side of software-defined data center networks

Data centers provide significant bandwidth capacity for a large number of servers interconnected by a specially designed network, called DCN [1, 260]. This bandwidth capacity can be optimized by using *multipath routing*, which distributes traffic over multiple concurrent paths [4]. Nowadays, ECMP is the default multipath routing mechanism for DCNs [5]. ECMP uses in every router a hash function on packet headers to assign each incoming flow to one of the equal-cost forwarding paths for reaching a destination. However, ECMP can degrade the performance of DCNs due to the coexistence of many small, short-lived flows (i.e., mice) and few large, long-lived flows (i.e., elephants), since ECMP can assign more elephant flows to the same path, generating *hot-spots* (i.e., some links overused while others underused). Flows traversing hot-spots suffer from low throughput and high latency.

Recent multipath routing mechanisms have leveraged SDN to face the ECMP limitations; DCNs using SDN are referred to as SDDCNs. SDN allows a logically centralized controller to dynamically make and install routing decisions on the basis of a global view of the network [24]. SDN-based multipath routing dynamically reschedules elephant flows, while handling mouse flows by employing default routing such as ECMP [5] and our pseudo-MAC-based approach (see Section 4.1). Reactive *flow detection* methods, which are at the heart of SDN-based mechanisms, discriminate elephants from mice by using static thresholds [26, 27, 29]. However, reactive methods are not suitable for SDDCNs since hot-spots may occur before the elephant flows are detected.

Novel SDN-based flow detection methods incorporate ML for proactively identifying elephant flows. However, ML-based methods operate at the *controller-side* of SDDCNs, requiring the central collection of either *per-flow* data [32] or *sampling-based* data [33, 35]. The central collection of per-flow data, however, causes problems such as heavy traffic overhead and poor scalability. Sampling-based data, on the other hand, tends to provide delayed and inaccurate flow information. Moreover, sampling techniques that mitigate the problem rely on non-standard SDN specifications. Using ML on either the *switch-side* or *server-side* represents a potential solution to the controller-side problems since

these locations enable prompt and per-flow data with low traffic overhead. Switch-side flow detection methods based on ML are impractical because they require specialized hardware and put a heavy processing load on the switches. Conversely, ML-based flow detection methods at the server-side require only software modifications in the servers; nonetheless, these methods have not been fully explored.

In this chapter, we propose a novel flow detection method denominated Network Elephant Learner and anaLYzer (NELLY), which applies online incremental learning at the server-side of SDDCNs for accurately and timely identifying elephant flows while generating low control overhead. Incremental learning allows NELLY to constantly train a flow size classification model from continuous and dynamic data streams (i.e., flows), providing a constantly updated model and reducing time and memory requirements. Thus, NELLY adapts to the variations in traffic characteristics and performs endless learning with limited memory resources. We extensively evaluate NELLY using datasets extracted from real packet traces and incremental learning algorithms. Quantitative evaluation demonstrates that NELLY is efficient in relation to accuracy and classification time when adaptive decision trees algorithms are used. Analytic evaluation corroborates that NELLY is scalable, causes low traffic overhead, and reduces detection time, yet it is in conformance with SDN standards.

The remainder of this chapter is as follows. Section 3.1 introduces the architecture of NELLY. Section 3.2 presents a quantitative evaluation of NELLY using incremental learning algorithms and real packet traces. Section 3.3 compares NELLY to other related work. Section 3.4 concludes the chapter.

3.1 Architecture of NELLY

Figure 3.1 introduces NELLY, a flow detection method that applies online incremental learning at the server-side of SDDCNs to identify elephant flows accurately in a reasonable time while generating low control overhead. NELLY operates as a software component either in the kernel of the host operating system or in the hypervisor of servers in the SDDCN with the aim of monitoring all packets sent by the applications, containers, and virtual machines. Since NELLY detects elephant flows at their origin, a small overhead is demanded.

The architecture of NELLY (see Figure 3.1) presents two subsystems: *Analyzer* and *Learner*. The Analyzer applies a flow size classification model for detecting and marking elephant flows on the fly. The Learner then applies an incremental learning algorithm for building and updating the flow size classification model. This model maps online features (i.e., features extracted from the first few packets of a flow) onto the corresponding class of flows (i.e., mice or elephants). The processes of the Analyzer and the Learner run concurrently as depicted in Algorithms 3.1 and 3.2, respectively. Moreover, for the sake of readability, Table 3.1 lists and describes the symbols defined in the architecture of NELLY.

NELLY is conceived for recognizing and handling elephant flows in real SDN implementations. NELLY can run on any host operating system or hypervisor. In the control

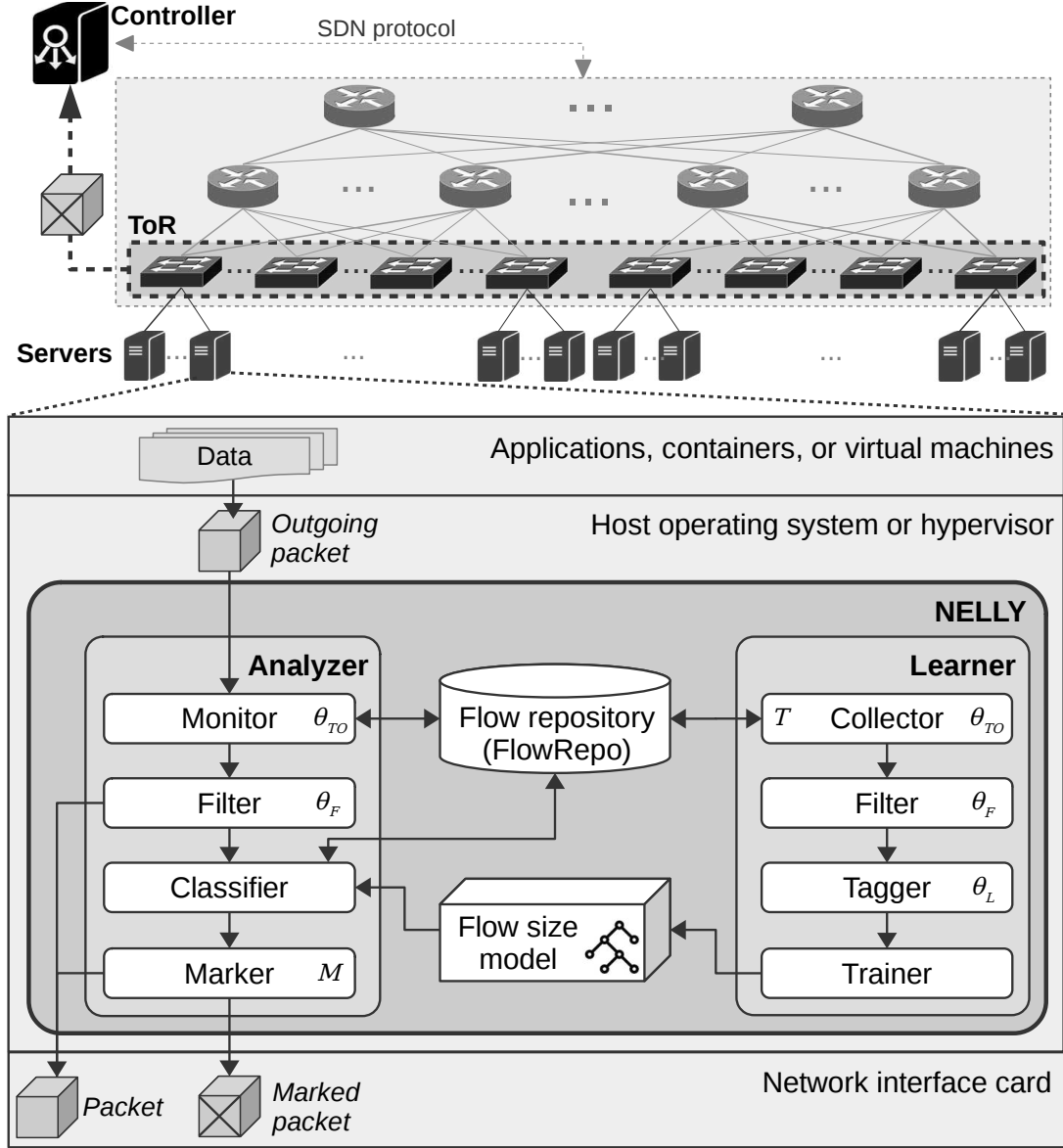


Figure 3.1: Architecture of NELLY

plane, any OpenFlow-compliant controller (e.g., OpenDaylight, ONOS, Ryu) can be used since NELLY operates at the server-side. In the data plane, OpenFlow-compliant switches (e.g., Open vSwitch) can be employed since NELLY requires only that the ToR switches include a pre-configured routing rule to forward elephant flows to the controller. The controller then can install specific routing rules per elephant flow based on the best path computed by a rescheduling algorithm, as discussed in Chapter 4.

3.1.1 Analyzer

As illustrated in Figure 3.1, the Analyzer consists of four modules: *Monitor*, *Filter*, *Classifier*, and *Marker*. The process of each module is detailed in Algorithm 3.1. As shown in lines 1–2, the Monitor keeps track of flows by extracting the header, size, and timestamp of each outgoing packet. A flow consists of subsequent packets sharing the same value

Table 3.1: Symbols in the architecture of NELLY

Symbol	Name	Description
θ_{TO}	Timeout threshold	Time limit above which both the Monitor and the Collector acknowledge a flow has terminated
θ_F	Filter threshold	Flow size limit below which either the Analyzer sends the packets without further processing or the Learner discards the flow records
M	Packet marking	Number of subsequent packets in an elephant flow to be marked
T	Collection rate	Time interval at which the Collector looks for terminated flows in FlowRepo
θ_L	Labeling threshold	Flow size limit above which the Tagger labels the flows as elephants

for certain header fields, and separated by a time-space shorter than a threshold timeout (θ_{TO}). NELLY enables a flexible configuration of these flow parameters, namely, flow header fields and θ_{TO} . For example, the flow header fields can be set as the well-known 5-tuple: source IP, source port, destination IP, destination port, and IP protocol. These flow header fields can also include MAC addresses and VLAN ID. On the other hand, the configuration of θ_{TO} is discussed in Section 3.1.2.

The Analyzer manages a flow record in the Flow Repository (FlowRepo) for each observed flow. As illustrated in Figures 3.2 and 3.3, the flow record includes the Flow Identifier (FlowID), start time, last-seen time, packet header (e.g., 5-tuple), flow size, the size and Inter-Arrival Time (IAT) of the first N packets, as well as the identified class (i.e., mice or elephants). Note that the IAT of the first packet is not included because it does not provide distinctive flow information (i.e., the IAT is always zero for the first packet of every flow).

As depicted in lines 3–13 in Algorithm 3.1, the Monitor then generates a FlowID from the flow header fields of each packet and checks to see if it exists in the FlowRepo. If this FlowID is missing (e.g., for packets 1 and 3 in Figure 3.3), or if the time since the last update of an existing record with this FlowID is longer than θ_{TO} (e.g., for packet 4), the Monitor creates a new record in the FlowRepo (Algorithm 3.1, lines 25–32). Otherwise, the Monitor fetches and updates the flow record (Algorithm 3.1, lines 33–43) using the FlowID stored in the FlowRepo (e.g., for packets 2 and 5 through 10 in Figure 3.3). When multiple flow records sharing the same FlowID exist in the FlowRepo, the Monitor always works with the most recent one (e.g., for packets 5 to 10).

Using the updated flow record, the Filter (Algorithm 3.1, line 14) avoids the introduction of a delay in the classification of a large number of mouse flows (usually latency-sensitive [27, 29]) by sending the packets of flows with a size below a certain threshold (θ_F) directly to the SDDCN without further processing (e.g., for packets 1 to 9 in Figure 3.3). The Filter also ensures that the Classifier receives all the required online features for making the classification. The online features refer to flow data extracted from the

Algorithm 3.1: NELLY Analyzer

input : outgoing packet p with header h_p , size s_p , and timestamp t_p , and flow size classification model m

output: either packet p or packet marked p^*

data : flow timeout threshold θ_{TO} , filtering flow size threshold θ_F , and number of first packets N

```

1 begin on receiving  $p$ 
  // Monitor
2   get  $h_p$ ,  $s_p$ , and  $t_p$  from  $p$ ;
3    $fid \leftarrow$  compute FlowID using the flow header fields from  $h_p$ ;
4   if  $fid \notin FlowRepo$  then
5      $f \leftarrow$  call CREATE_FLOW( $fid$ ,  $h_p$ ,  $s_p$ ,  $t_p$ )
6   else
7      $F \leftarrow$  fetch the last flow  $f \in FlowRepo$  such that  $f.id = fid$ ;
8     if  $(currentTime - f.lastSeenTime) > \theta_{TO}$  then
9        $f \leftarrow$  call CREATE_FLOW( $fid$ ,  $h_p$ ,  $s_p$ ,  $t_p$ )
10    else
11       $f \leftarrow$  call UPDATE_FLOW( $f$ ,  $s_p$ ,  $t_p$ )
12    end
13  end
  // Filter
14  if  $f.size < \theta_F$  then return  $p$ ;
  // Classifier
15  if  $\nexists f.class$  then
16     $f.class \leftarrow m.CLASSIFY(f)$ ;
17    update  $f \rightarrow FlowRepo$ ;
18  end
  // Marker
19  if  $f.class = \text{"Elephant"}$  then
20     $p^* \leftarrow$  mark  $p$ ;
21    return  $p^*$ ;
22  end
23  return  $p$ ;
24 end
25 function CREATE_FLOW( $fid$ ,  $h_p$ ,  $s_p$ ,  $t_p$ ):
26    $f \leftarrow$  initialize a new flow with FlowID  $fid$ ;
27    $f.headerFields[] \leftarrow$  array of flow header fields from  $h_p$ ;
28    $f.startTime \leftarrow f.lastSeenTime \leftarrow t_p$ ;
29    $f.size \leftarrow f.sizePackets[0] \leftarrow s_p$ ;
30   create  $f \rightarrow FlowRepo$ ;
31   return  $f$ 
32 end
33 function UPDATE_FLOW( $f$ ,  $s_p$ ,  $t_p$ ):
34    $n \leftarrow$  current number of packets of  $f$ ;
35   if  $n \leq N$  then
36      $f.sizePackets[n] \leftarrow s_p$ ;
37      $f.iatPackets[n] \leftarrow t_p - f.lastSeenTime$ ;
38   end
39    $f.size \leftarrow f.size + s_p$ ;
40    $f.lastSeenTime \leftarrow t_p$ ;
41   update  $f \rightarrow FlowRepo$ ;
42   return  $f$ 
43 end

```

first N packets of a flow. The Filter then guarantees the size and IAT of the first N

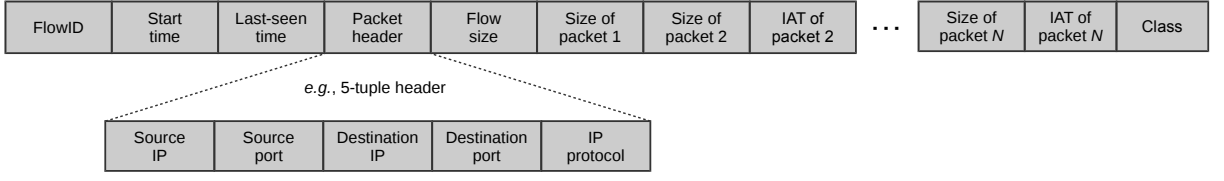


Figure 3.2: Structure of flow records in FlowRepo

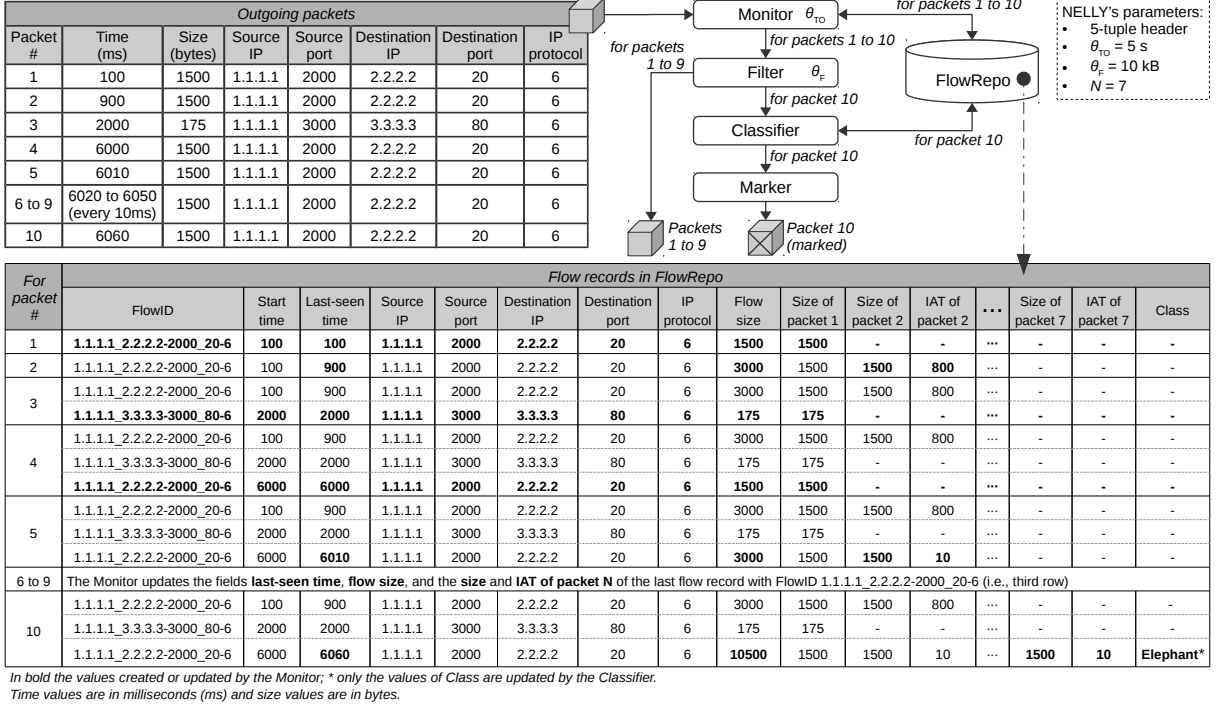


Figure 3.3: Example of how flow records are created and updated in FlowRepo

packets of a flow since the maximum value of N depends on θ_F . For example, $\theta_F = 10$ kB would require an $N \leq 7$ over Ethernet, otherwise, data from some packets would be missed. Consequently, the Classifier operates once the Monitor has processed packets that increment the size of flows over θ_F (e.g., for packet 10).

The Classifier (Algorithm 3.1, lines 15–18) applies the flow size classification model to the online features to identify flows as either mice or elephants. This model results from an incremental learning algorithm, which maps the online features to the corresponding class of flows used as training data. After applying the flow size classification model, the Classifier stores the identified class in the FlowRepo for each flow record with a flow size greater than θ_F (e.g., elephant for flow of packet 10 in Figure 3.3). Therefore, when processing a packet of a previously identified flow, the Classifier checks the fetched class from the FlowRepo to avoid any delay from the classification. The Classifier then reports to the Marker the class of the flow for each packet. We discuss in Section 3.1.2 how the Learner collects the training data for building and updating the flow size classification model.

The Marker (Algorithm 3.1, lines 19–23) forwards the packets of flows classified as mice without changes but marks those classified as elephants (e.g., packet 10 in Figure 3.3). To mark a packet, the Marker sets a predefined value in a code point header field supported

by SDN switches. For example, OpenFlow switches support matching in two code point header fields. The first of these is the 6-bit Differentiated Services Code Point (DSCP) field of the IPv4 header. This DSCP reserves a code point space for experimental and local usage (i.e., $****11$, where $*$ is 0 or 1). The second is the 3-bit 802.1Q Priority Code Point (PCP) field of the Ethernet header. In practice, NELLY can rely on either one of these fields, since it is improbable that a data center use both DSCP and PCP simultaneously [29].

The Marker can be extended by enabling a flexible configuration of the number of subsequent packets in an elephant flow to be marked (M), thus enabling a trade-off between reliability and latency. For instance, as M increases, the lesser the probability that the controller will miss elephant flows due to losses of marked packets in the SDDCN. However, a higher M introduces a delay in the Marker for a higher number of packets of elephant flows. Once the controller has installed a higher priority routing rule for handling a specific elephant flow across the SDDCN, the subsequent marked packets of this flow are not forwarded to the controller.

3.1.2 Learner

As depicted in Figure 3.1, the Learner consists of four modules: *Collector*, *Filter*, *Tagger*, and *Trainer*. The process of each module is detailed in Algorithm 3.2. As shown in lines 1–4, the Collector fetches terminated flows from the FlowRepo at every interval T . A flow is considered terminated if it remains idle for longer than θ_{TO} . Therefore, the Collector recognizes terminated flows by checking that a time longer than θ_{TO} has passed since the last-seen time of the FlowID records in the FlowRepo. Note that the Collector relies on the FlowID records updated by the Monitor for the recognition of the terminated flows, so their actual size can be obtained.

The Collector avoids increasing memory consumption in NELLY by removing terminated flows from the FlowRepo (Algorithm 3.2, line 5). The actual size of terminated flows can also be further used to provide fixed-memory probability distributions that support autonomous configuration of flow size thresholds [257]. Memory requirements in the FlowRepo thus depend on both T and θ_{TO} . T provides a trade-off between memory and processing. As T decreases, the Collector removes the terminated flows from the FlowRepo more quickly, consuming less memory, but leading to more processing. In turn, θ_{TO} directly affects the number of FlowID records stored in memory. As θ_{TO} increases, the FlowRepo retains FlowID records for a longer time. θ_{TO} is related to the *inactive timeout* configuration of flow rules in SDN-enabled switches, which provides a trade-off between flow table occupancy and miss-rate (i.e., when the packet IAT is greater than the timeout) [261].

The Filter of the Learner (Algorithm 3.2, line 6) receives the terminated flows from the Collector and reports to the Tagger only those with size greater than θ_F . The terminated flows are then used by the Trainer to build the flow size classification model. Since the Classifier operates only with flows of a size greater than θ_F , the Filter of the Learner prevents the introduction of noise to the model.

The Tagger (Algorithm 3.2, lines 7–8) compares the actual size of the filtered flows to

Algorithm 3.2: NELLY Learner

```

input : flow size classification model  $m$ 
output: either actual  $m$  or updated  $m$ 
data : learning time interval  $T$ , flow timeout threshold  $\theta_{TO}$ , filtering flow size threshold  $\theta_F$ ,
        and labeling flow size threshold  $\theta_L$ 
1 begin every  $T$ 
   // Collector
2    $F \leftarrow \text{fetch flows } f \in \text{FlowRepo};$ 
3   for  $f \in F$  do
4     if  $\text{currentTime} - f.\text{lastSeenTime} > \theta_{TO}$  then
5       delete  $f \rightarrow \text{FlowRepo};$ 
6       // Filter
7       if  $f.\text{size} \geq \theta_F$  then
8         // Tagger
9         if  $f.\text{size} \geq \theta_L$  then  $f.\text{class} \leftarrow \text{"Elephant"};$ 
10        else  $f.\text{class} \leftarrow \text{"Mouse"};$ 
11        // Trainer
12         $m \leftarrow m.\text{UPDATE}(f.\text{headerFields}[], f.\text{sizePackets}[], f.\text{iatPackets}[], f.\text{class});$ 
13      end
14    end
15  end
16  return  $m;$ 
17 end

```

a labeling threshold (θ_L) so that they can be tagged as either mice or elephants. θ_L will vary (e.g., 100 kB or 1 MB) as a function of the traffic characteristics and performance requirements of SDDCNs. Labeled flows provide the Trainer (Algorithm 3.2, line 9) with the *ground truth* for building a supervised learning model for flow size classification (see Section 2.2). This classification model maps online features (i.e., packet header, size, and IAT of the first N packets) onto the corresponding class (i.e., mice or elephants). Recall that the Classifier relies on the flow size classification model to identify elephant flows.

Since flows represent continuous and dynamic data streams, the Trainer uses an incremental learning algorithm (e.g., Hoeffding tree and online ensembles) for building the flow size classification model. Incremental learning enables updating the flow size classification model as the Trainer receives labeled flows over time, rather than retraining from the beginning (see Section 2.2.1). Therefore, NELLY adapts to varying traffic characteristics and performs continuous learning with limited memory resources. There is no need for the Trainer to maintain labeled flows in memory. This is an important characteristic of NELLY, since it helps to reduce the consumption of resources in all the servers of the SDDCN.

3.2 Evaluation

This section presents the evaluation of NELLY in relation to classification accuracy and time by using real packet traces and incremental learning algorithms. The generic approach for designing ML-based solutions in networking (see Figure 2.4) is used to describe and conduct the evaluation of NELLY: (i) data collection to gather the raw packet traces and generate the flow size datasets, (ii) feature engineering to extract and format the on-

line features of the flow size datasets, *(iii)* establishing the ground truth to label each flow in the datasets using the flow size and the classes of interest (i.e., mice and elephants), *iv* model validation to define the accuracy metrics that measure the performance of the trained models; and *(v)* model learning to train different models by using a variety of incremental learning algorithms.

3.2.1 Datasets

Two real packet traces, UNI1 and UNI2, captured in university data centers [255], were employed to evaluate NELLY (Table 3.2 summarizes their characteristics). These two traces are shorter than three hours long, but their mice and elephants distributions are similar to those found in non-public traces collected at different periods along the day [9, 11]. On the other hand, to the best of our knowledge, neither traces nor datasets of IPv6 traffic in DCNs are publicly available. In line with that, NELLY was evaluated using IPv4 traffic only which represents over 99% of the packets in UNI1 and UNI2.

Table 3.2: Details of real packet traces and extracted IPv4 flows

Packet traces [255]		UNI1		UNI2	
Duration		65 min		158 min	
Packets		19.85 M		100 M	
IPv4 % of total traffic		98.98% (<i>mostly TCP</i>)		99.9% (<i>mostly UDP</i>)	
IPv4 flows		1.02 M (<i>TCP and UDP</i>)		1.04 M (<i>mostly UDP</i>)	
Details of IPv4 flows	Flow size	% of IPv4 flows	% of IPv4 traffic	% of IPv4 flows	% of IPv4 traffic
	≥ 10 kB	7.16%	95.06%	5.91%	98.81%
	≥ 100 kB	0.83%	83.71%	1.93%	96.86%
	≥ 500 kB	0.14%	73.14%	0.76%	93.52%
	≥ 1 MB	0.07%	69.52%	0.48%	90.83%
	≥ 5 MB	0.01%	60.33%	0.17%	81.34%

IPv4 flows obtained using the 5-tuple header and a threshold timeout $\theta_{TO} = 5$ s

Only the following parameters needed to be defined to generate the datasets: the flow header fields, θ_{TO} , and N . Firstly, the 5-tuple header (i.e., source IP, destination IP, source port, destination port, and IP protocol) as the flow header fields since it sufficiently characterizes IPv4 flows; hereinafter, they are referred just as flows. Secondly, $\theta_{TO} = 5$ s was established on the basis of the break-even point analysis between the flow table occupancy and the miss-rate in OpenFlow switches for DCNs considered by [261]. Then, since the maximum value of N depends on θ_F , $N = 7$ was set as the maximum for $\theta_F = 10$ kB. As shown in Table 3.2, the selected θ_F encompasses all the potential elephants (i.e., flows carrying more than 95% of the traffic) and avoids the introduction of the classification delay to mice (for more than 93% of the flows). Using these parameters, the UNI1 and UNI2 data traces were processed to generate the corresponding flow size datasets, each containing somewhat more than a million flows (see Table 3.2). Since NELLY only classifies flows greater than θ_F , those smaller than $\theta_F = 10$ kB were removed from both datasets. Therefore, the UNI1 and UNI2 datasets consisted of approximately 70,000 and 60,000 flows, respectively.

The datasets [262] included the following flow information: start time, end time, 5-tuple header, size and IAT of the first 7 packets, as well as flow size. The start and end

times enabled a more realistic evaluation (see Section 3.2.3). The 5-tuple header and the size and IAT of the first 7 packets represented the online features for the flow size classification model. The flow size was compared to different θ_L (e.g., 100 kB, 500 kB, 1 MB, and 5 MB) to label the flows as mice or elephants (i.e., classes of interest). Unless otherwise stated, the datasets with $\theta_L = 100$ kB were used. Labeled flows represented the ground truth for learning and validating the flow size classification model.

For complementing feature engineering, various different data types were considered for the online features, particularly for the 5-tuple header. Certainly, the size and IAT of the first 7 packets (13 features, since the IAT of the first packet is not included) indicate a measurement, hence, numeric data, whereas the 5-tuple header contains two IP addresses in dotted-decimal notation (i.e., categorical data) and three numeric codes (i.e., nominal data). However, the huge set of possible categories for IP addresses (i.e., 2^{32}) hinders a real implementation. To address this problem, the IP addresses were divided into four octets, resulting in a total of 11 nominal features for the 5-tuple header. To handle these 11 nominal features as numeric data, a *Numeric* (Num) header type was defined. These features were then transformed into binary digits (bits), generating 104 features for the 5-tuple header. Considering these binary features, two more header types were defined: *Binary-Numeric* (BinNum) to treat binary features as numeric data (i.e., a value between zero and one) and *Binary-Nominal* (BinNom) to handle binary features as nominal data (i.e., zero or one). Table 3.3 illustrates the features included by each header type. Unless otherwise stated, the datasets with BinNom-header were used.

Table 3.3: Header types defined during feature engineering

Header type	Features of the header type (example)						# of features
5-tuple	Source/Destination IP				Source/Destination Port	IP Protocol	5
	41.177.26.55 244.3.160.248				80 43521	6	
Num	Source/Destination IP				Source/Destination Port	IP Protocol	11
	Octet 1	Octet 2	Octet 3	Octet 4			
	41 244	177 3	26 160	55 248	80 43521	6	
BinNum BinNom	Source/Destination IP				Source/Destination Port	IP Protocol	104*
	8 bits	8 bits	8 bits	8 bits	16 bits	8 bits	
	00101001 11110100	10110001 00000011	00011010 10100000	00110111 11111000	0000000001010000 1010101000000001	00000110	

* Each bit represents one feature

3.2.2 Accuracy metrics

Metrics derived from the confusion matrix (see Figure 3.4) were used, including the True Positive Rate (TPR) and the False Positive Rate (FPR), thus avoiding the over-optimism of the conventional accuracy metric caused by an imbalance of classes [263]. In the datasets, the imbalance between mice and elephants depends on θ_L . For example, assuming $\theta_L = 100$ kB, only 12% of flows above 10 kB in the UNI1 dataset represent the elephant class (see Table 3.2). The imbalance grows as θ_L increases.

		Actual instance	
		Positive (P)	Negative (N)
Predicted outcome	P	True Positive (TP)	False Positive (FP)
	N	False Negative (FN)	True Negative (TN)

Figure 3.4: Confusion matrix for binary classification

Consider that each row in the confusion matrix, illustrated in Figure 3.4, represents a predicted outcome and each column represents the actual instance. In this manner, True Positive (TP) is the intersection between correctly predicted outcomes for the actual positive instances. Similarly, True Negative (TN) is when the classification model correctly predicts an actual negative instance. Whereas, False Positive (FP) and False Negative (FN) describe incorrect predictions for negative and positive actual instances, respectively. Note, that TP and TN correspond to the true predictions for the positive and negative classes, respectively.

Recall that flows classified as elephants are forwarded to the controller for further processing, thus introducing transmission and processing delays. Therefore, NELLY aims at detecting as many elephants while negatively affecting as few latency-sensitive mice as possible. Considering elephants as the positive condition, the TPR describes the proportion of detected elephants (see Equation 3.1) whereas the FPR provides the ratio of negatively affected mice (see Equation 3.2). Both TPR and FPR range between 0 and 1. Furthermore, the Matthews Correlation Coefficient (MCC) was used to analyze the balance between the TPR and the FPR. The MCC takes all values from the confusion matrix to provide a measure between 1 and -1 (see Equation 3.3). As the MCC gets closer to 1, the difference of the TPR over the FPR increases, leading to a more accurate classifier. An MCC between 0 and -1 means that $TPR \leq FPR$, which would be less accurate than a random classifier. In our experiment, the MCC values were always greater than 0, hence, we use a range between 0 and 1 to plot TPR, FPR, and MCC in Figures 3.5, 3.6, and 3.7.

$$TPR = \frac{TP}{TP + FN} \quad (3.1)$$

$$FPR = \frac{FP}{FP + TN} \quad (3.2)$$

$$MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (3.3)$$

The MCC metric is employed in the performance analysis because it is recommended for imbalanced datasets (like UNI1 and UNI2) [264]. The MCC score is only high when the classification algorithms are doing well in both the positive and negative elements (i.e., elephants and mice, respectively). The ROC curve has also proven to be useful for imbalanced datasets but it is more appropriate to analyze classification algorithms that output a real value [265]. Thus, we preferred the MCC because the output of the incremental learning classification algorithms employed in this paper is a single class value (either mouse or elephant) rather than a real value.

3.2.3 Experiment setup

Incremental learning algorithms are commonly evaluated using the interleaved test-then-train approach [266]. This approach refers to going through each flow to classify it first by working only with the online features and then use its actual class for training the flow size classification model. However, since flows start and end over time, some order of the flows must be established. Moreover, under real conditions, some flows start before a classified flow ends, whereas others end before a new flow starts. Therefore, the flows are classified at the start time and the model is trained at the end time, so the performance evaluation will be based on more realistic conditions.

The imbalance of classes in the UNI1 and UNI2 datasets was addressed by training the flow size classification model using *inverse weights*, as in [32], i.e., weights (between 0 and 1) inversely proportional to the ratio of training instances previously encountered by the model for each class. If the model is trained with a *single weight* (i.e., 1 by default in the Massive Online Analysis (MOA) tool [266]), it would tend to classify all flows as mice due to the imbalance of classes.

To corroborate the weighting decision, the accuracy was evaluated when training the model with the single weight and inverse weights for three incremental learning algorithms available in MOA, namely, Adaptive Random Forest (ARF), Adaptive Hoeffding Option Tree (AHOT), and Hoeffding trees (a.k.a. CVFDT)—as later discussed in Section 3.2.4, NELLY achieves the best performance by using these algorithms. Figure 3.5 shows that the three algorithms achieve a higher elephant detection rate (i.e., TPR) for both datasets with the inverse weights than with the single weight. These gains in the TPR come at a sacrifice to the FPR; however, the three algorithms maintain a similar MCC. Therefore, in the performance evaluation, inverse weights were used to improve the TPR while maintaining the trade-off between the TPR and FPR.

3.2.4 Performance analysis

To determine the consideration for the best classification performance of NELLY, the UNI1 and UNI2 datasets were used with different header types (i.e., Num, BinNum, and BinNom), as well as 50 incremental learning classification algorithms available in MOA.

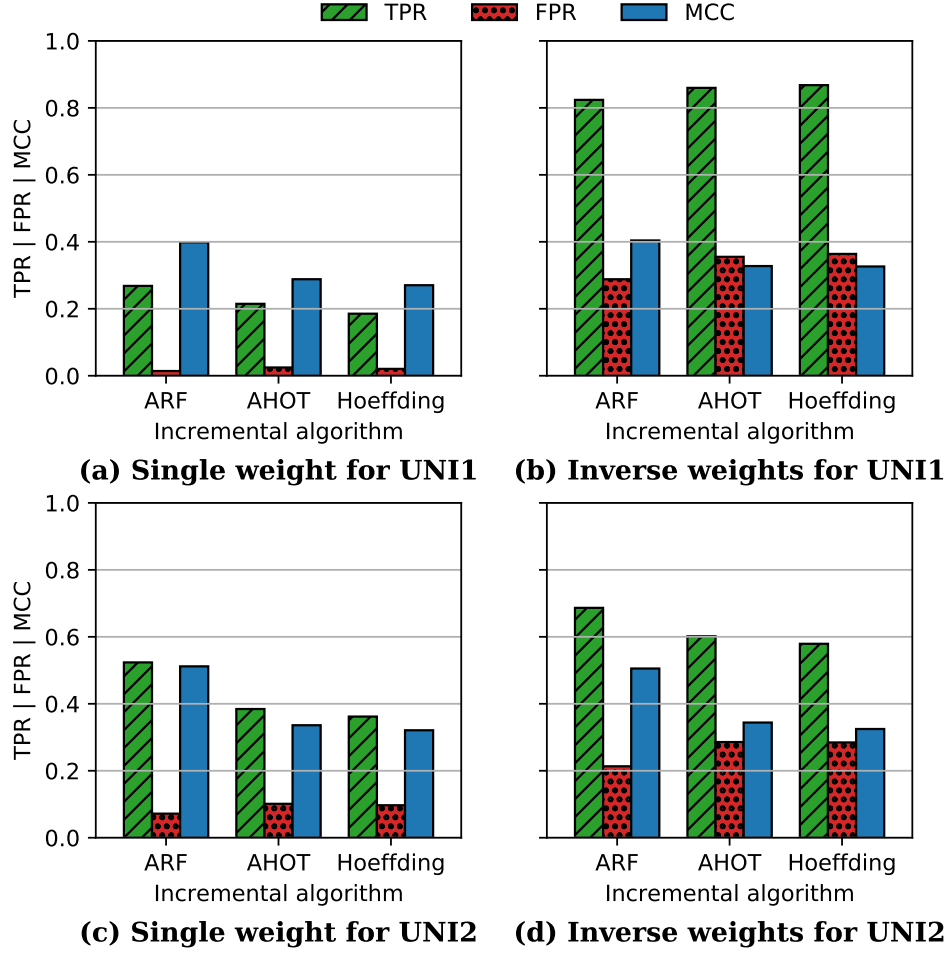


Figure 3.5: Classification accuracy of NELLY when using a single weight and inverse weights for the UNI1 and UNI2 datasets

The performance evaluation included the accuracy metrics (i.e., TPR, FPR, and MCC) and the classification time per flow (T_C). The algorithms were executed with their default settings (except for the training weights) and without previous model initialization.

For the sake of brevity, Table 3.4 presents ten algorithms, namely, AHOT, ARF, Hoeffding tree, k -Nearest Neighbors (kNN) with Probabilistic Adaptive Windowing (PAW), Naïve Bayes (NB), Online Accuracy Updated Ensemble (OAUE), online bagging, online boosting, Stochastic Gradient Descent (SGD) for Support Vector Machines (SVM), and Very Fast Decision Rules (VFDR) with NB classifiers (VFDR_{NB}). These algorithms were selected on the basis of the best performance results between algorithms with a similar learning approach. Furthermore, Table 3.4 includes only the best results of each algorithm, taking into account both accuracy and classification time for a specific header type. The BinNom headers were found to enable the best performance of the majority of the algorithms for the UNI1 and UNI2 datasets. This was due to the fact that most algorithms achieved greater accuracy using the BinNom headers than the Num headers for a comparable classification time. The use of the BinNum headers is strongly discouraged; although similar or slightly better accuracy results were obtained, there was a significant increase in the classification time (up to 4x).

Table 3.4: Classification performance of NELLY with different incremental algorithms

Algorithms	UNI1					UNI2				
	TPR (%)	FPR (%)	MCC	T_C (μ s)	Header type	TPR (%)	FPR (%)	MCC	T_C (μ s)	Header type
AHOT	85.97	35.52	0.327	4.07	BinNom	60.16	28.58	0.304	10.17	BinNom
ARF	82.39	28.82	0.359	12.01	BinNom	68.65	21.33	0.460	17.39	BinNom
Hoeffding tree	86.79	36.38	0.326	3.18	BinNom	57.92	28.46	0.284	4.64	BinNom
kNN-PAW	25.30	2.99	0.311	473.1	Num	40.29	10.22	0.302	454.1	Num
NB	74.76	35.69	0.254	4.76	BinNom	49.74	23.18	0.267	4.82	BinNom
OAUE	86.79	33.63	0.347	25.58	BinNom	63.28	28.65	0.332	33.06	BinNom
Online bagging	87.78	37.11	0.327	23.98	BinNom	64.17	31.13	0.314	36.61	BinNom
Online boosting	75.88	29.93	0.307	11.56	BinNom	64.62	32.22	0.307	16.82	BinNom
SGD-SVM	16.76	10.21	0.067	0.81	Num	38.69	30.99	0.076	0.8	Num
VFDR _{NB}	74.44	33.77	0.267	18.47	BinNom	54.83	29.15	0.248	18.59	BinNom

In bold the top five results of TPR and MCC, and the T_C results shorter than 17.5 μ s, for both UNI1 and UNI2

The accuracy results show that no single algorithm achieves the best values of the TPR and MCC for the UNI1 and UNI2 datasets. This is due to the fact that the flow size distribution and the features of the elephant and mouse flows were specific for each dataset. Therefore, the top five results were used to analyze the accuracy performance. Regarding the T_C , most algorithms introduced a classification delay per flow shorter than 17.5 μ s, but this represents only a small percentage (7%) of the Round-Trip Time (RTT) in DCNs (i.e., 250 μ s in the absence of queuing [267]).

Both Hoeffding tree and NB represent the state-of-the-art in incremental learning algorithms. Their simplicity and low computational cost enabled a very short delay ($T_C < 5 \mu$ s) that accounts for only 2% of the RTT in DCNs. However, only the Hoeffding tree represents a valid alternative for the traffic similar to that of UNI1 because its TPR and MCC were among the top five results for the UNI1 dataset. The Hoeffding tree in MOA uses NB classifiers on the leaves (i.e., CVFDT_{NBC}), which improves the accuracy without compromising the computational cost.

The ARF, OAUE, online bagging, and online boosting are ensemble-based algorithms that combine multiple Hoeffding trees (ten in our evaluation) for improving the accuracy at the expense of increasing the computational cost. As described in Section 2.2.2, ensemble learning aims at improving the accuracy performance of a single learning model by combining it with other models. As a result, the poor performance of a single classifier can be compensated with ensemble learning at the price of extra computation. Indeed the results from ensemble learning must be aggregated, and a variety of techniques have been proposed in this matter, including random forest, block-weighting, bagging, and boosting.

Therefore, the ARF and online boosting algorithms introduced a T_C shorter than 7% of the RTT in DCNs. ARF provided the best MCC and a TPR among the top five for the UNI1 and UNI2 datasets. Online boosting can be seen as an option for the traffic similar to that of UNI2 since it was in the top five accuracy results only for the UNI2 dataset. In contrast, although the OAUE and online bagging algorithms also provided good accuracy results (particularly for the UNI1 dataset), they introduced a T_C twice longer than the T_C of ARF and online boosting. This long T_C is because OAUE and online bagging

rely on ensemble methods (block-weighting and bagging, respectively) that demand more computation than those used by ARF and online boosting (random forest and boosting, respectively).

Similar to ARF, the AHOT algorithm figured in the top five accuracy results for both datasets. Moreover, AHOT only introduced a T_C shorter than 2% (5 μ s) and 4% (10 μ s) of the RTT in DCNs for the UNI1 and UNI2 datasets, respectively. AHOT is capable of improving the accuracy of the Hoeffding tree algorithm without demanding too much computation by providing an intermediate solution between a single Hoeffding tree and an ensemble of Hoeffding trees. AHOT uses additional option paths (five maximum in our evaluation) to build a single structure that efficiently represents multiple Hoeffding trees.

The implementations in MOA of the VFDR_{NB}, SGD-SVM, and kNN-PAW algorithms are strongly discouraged. The VFDR_{NB} presented accuracy results outside the top five for both datasets and a T_C slightly longer than 7% of the RTT in DCNs. This is because rule-based algorithms focus on building more interpretable models than does the Hoeffding tree algorithm, which increases the computational cost but not necessarily improves the accuracy. The SGD-SVM algorithm introduced the shortest classification delay ($T_C < 1 \mu$ s) but it produced the worst values in the TPR and MCC metrics. The reason for these values is that MOA implements a very simple SGD-SVM algorithm that uses a linear kernel which is not sufficient to model different patterns in flows of packets. The kNN-PAW provided the second-worst TPR for both datasets and a very long classification delay ($T_C > 450 \mu$ s), which increased up to 3,000 μ s with the BinNum and BinNom headers (i.e., 12x the RTT in DCNs). This long T_C value is a consequence of the computation of a distance metric by the algorithms based on kNN every time the classification is performed.

In conclusion, NELLY achieves the best classification performance by using the Bin-Nom headers along with the following incremental learning algorithms:

- The ARF is good for any type of traffic and if the RTT is flexible. It achieved the best MCC for the UNI1 and UNI2 datasets, and it was also the fifth- and second-best for the TPR while introduced a T_C lesser than 7.5% of the RTT in DCNs.
- The AHOT is good for any type of traffic and a strict RTT. The TPR and MCC ranked among the top five for both datasets while the T_C was shorter than that of the ARF, especially for the UNI1 dataset.
- The Hoeffding tree (CVFDT_{NBC}) is good for traffic similar to that of UNI1 and if the RTT is very strict. The TPR was the second-best and the MCC was the fifth (quite close to the AHOT) for the UNI1 dataset while introduced a very short T_C . When the RTT constraint takes precedence over the accuracy, this would be a good option for traffic similar to that of UNI2 because a very short T_C was maintained while provided the sixth-best TPR and MCC for such traffic.

The classification accuracy of NELLY was also evaluated with the ARF and AHOT algorithms for different values of θ_L since this threshold may vary as a function of traffic and routing requirements. Both ARF and AHOT ranked among the top five in accuracy for both datasets with a T_C shorter than 7.5% of the RTT in DCNs. As shown in

Figure 3.6, the MCC results of both algorithms were degraded as θ_L increased, especially for the UNI1 dataset. This is because the difference between the features of the elephants and mice becomes less significant as θ_L increases. In contrast, the TPR remained very similar as θ_L increased, except that the ARF suffered from a significant reduction in the TPR for the UNI1 dataset. Therefore, the AHOT was more robust to variations in θ_L for traffic similar to that of UNI1, although the performance of both algorithms was similar for the UNI2 dataset. Based on this summary, NELLY with the AHOT algorithm enables a flexible configuration of θ_L while providing great elephant flows detection in data centers regardless the type of traffic. For traffic similar to that of UNI2, both ARF and AHOT represent valid alternatives for the use of NELLY and the flexible configuration of θ_L is possible since they perform similarly in terms of elephants detection.

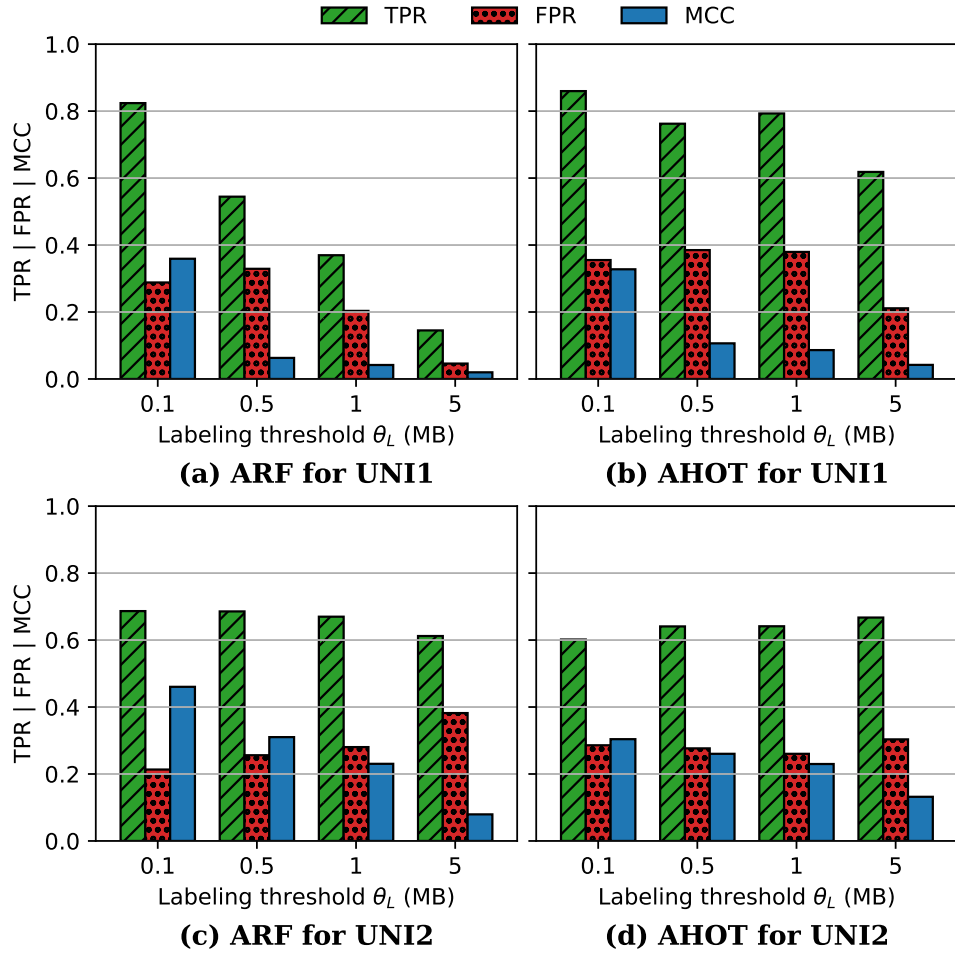


Figure 3.6: Classification accuracy of NELLY with the ARF and AHOT algorithms when varying the labeling threshold (θ_L) for the UNI1 and UNI2 datasets

Finally, the effect of the handling of different ranges of the inverse weights in the two classes on the classification accuracy of NELLY with the two algorithms (ARF and AHOT) was analyzed. The weights of the mice were maintained between 0 and 1, whereas the weights of the elephants ranged from 0 to W_E , where W_E varied from 1 to 5. Figure 3.7 shows that both the ARF and AHOT algorithms achieved a higher TPR for both datasets as W_E increased (up to 94% and 98% of elephants detection, respectively). These results

were expected since establishing greater weights for the elephant class makes the learning algorithms increment the influence of the features of the elephant flows in the classification model. Moreover, the trade-off between the TPR and FPR (i.e., MCC) remained quite similar for UNI1-type traffic whereas that of UNI2 was degraded as W_E increased. This is due to the greater differences between the elephants and mice for the UNI1 dataset than for UNI2 when $\theta_L = 100$ kB. Therefore, as W_E increased for the UNI2 dataset, the increment of mouse flows wrongly classified as elephants (i.e., FPR) was greater than that of elephant flows correctly classified (i.e., TPR). In conclusion, NELLY supports a flexible configuration of inverse weights for meeting different accuracy requirements.

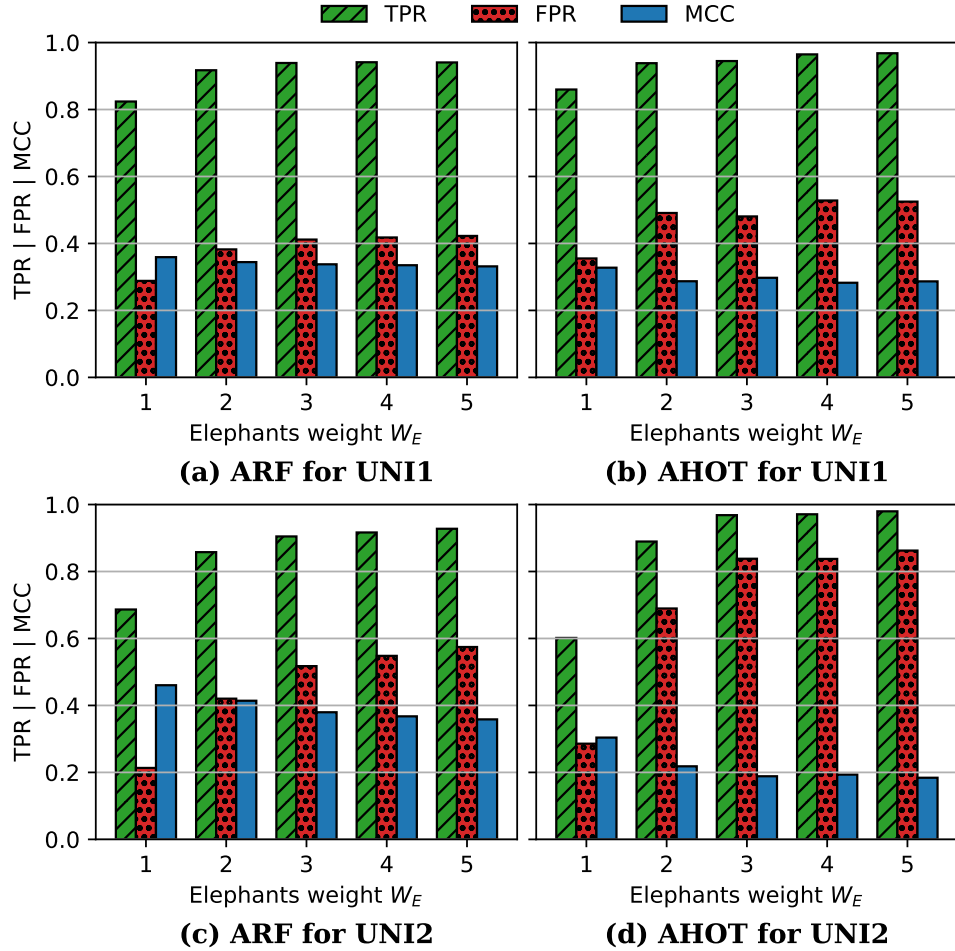


Figure 3.7: Classification accuracy of NELLY with the ARF and AHOT algorithms when varying the inverse weights of elephant flows (W_E) for the UNI1 and UNI2 datasets

3.3 Comparative analysis

NELLY was compared with OFSP [32], ESCA [35], FlowSeer [33], and Mahout [29]. OFSP, ESCA, and FlowSeer incorporate ML at the controller-side of SDDCNs for proactively detecting elephant flows, whereas Mahout performs reactive detection at the server-side. The results reported by each work for the UNI1 dataset were used to compare them in relation to: learning approach, elephants detection, false elephants, table occupancy,

control overhead, detection time, network modifications, and performance factors. The seminal works involving Hedera [26] and DevoFlow [27] were not considered. These approaches perform reactive flow detection and their limitations hinder real implementation. Hedera causes large control traffic overhead and has poor scalability, whereas Devoflow requires custom-made switch hardware and imposes a heavy burden on switches. Devoflow also presents an alternative based on sFlow [250], however, ESCA revealed and outperformed the inaccurate and late elephant detection suffered by the sFlow-based DevoFlow.

Table 3.5 summarizes the comparative analysis. Overall, NELLY achieved a better balance between the comparative features than the other approaches, namely, a very high elephant detection rate with a very short detection time, while significantly reducing traffic overhead, without demanding switch table occupancy, and only software modifications and resources in servers were required. The following paragraphs outline the comparison of NELLY with the other approaches.

Table 3.5: Comparison of NELLY with related approaches

FEATURE	OFSP [32]	ESCA [35]	FlowSeer [33]	Mahout [29]	NELLY
Learning approach	Incremental	Batch	Batch and incremental	None	Incremental
Elephants detection	Very high	High	Very high	Perfect	Very high
False elephants	Very low	Very low	High	None	Very low
Table occupancy	Very high	Medium	Low	None	None
Control overhead	High	Medium	Medium	Very low	Very low
Detection time	Very short	Medium	Medium	Long	Very short
Network modifications*	None	Hardware of switches	None	Software in servers	Software in servers
Performance factors	Controller and ToR	Controller and ToR	Controller and ToR	Servers	Servers

* Assuming OpenFlow-based networks [48]

Learning approach. ML algorithms used for detecting elephant flows can involve batch or incremental learning. Batch learning refers to the use of training models based on static datasets (i.e., all training data are simultaneously available). However, batch learning requires the storage of unprocessed data to cope with traffic variations in DCNs, so the models must repeatedly work from scratch. This is time-consuming and prone to outdated models. Conversely, incremental learning continuously adapts the ML models on the basis of streams of training data, enabling constantly updated models and reducing time and memory requirements (see Section 2.2.1). ESCA relies on batch learning whereas NELLY and OFSP rely on incremental learning for detecting elephant flows. FlowSeer is a mixed approach using batch learning for the identification of potential elephants and incremental learning for the classification of the potential ones. Mahout has no learning approach, since it performs reactive elephants detection.

Elephants detection. The main goal of flow detection methods is to identify elephant

flows (i.e., TPR). NELLY, OFSP, and FlowSeer all proactively detected more than 95% of elephant flows, whereas ESCA detected a maximum of 88.3%. Mahout provides perfect detection, although this is reactive.

False elephants. Mouse flows mistakenly identified as elephants (i.e., FPR) are needlessly forwarded to and processed by the controller. For achieving the highest elephants detection rate, FlowSeer informed the controller of 29% of mice as potential elephants, whereas OFSP and ESCA only reported around 2%. No mouse flow is reported to the controller by Mahout since detection is reactive. NELLY yielded an FPR of 40%, but this was computed using only 7% of the flows (i.e., $\theta_F = 10$ kB, in Section 3.2.1). NELLY sent the other 93% of the flows (corresponding to mice) directly to the SDDCN without further processing (see Section 3.1.1). NELLY thus forwards only 2.5% of mice to the controller for achieving the highest elephants detection rate (see Figure 3.9(b)).

Figures 3.8 and 3.9 depict the elephants detection and false elephants results that NELLY achieves when considering all the IPv4 flows from the UNI1 and UNI2 data traces, including the portion of mice sent directly to the SDDCN without further processing. Figure 3.8 shows the results for different values of θ_L (cf. Figure 3.6), while Figure 3.9 sets $\theta_L = 100$ kB and varies the range of the inverse weights of elephant flows (W_E) (cf. Figure 3.7).

Table occupancy. Controller-side flow detection methods install flow table entries in ToR switches for centrally collecting flow data. The smaller the number of flow table entries, the more efficient is the resource utilization. OFSP requires one entry per flow, thus constraining its scalability because of the limited memory in SDN switches. ESCA and FlowSeer install wildcard entries for sampling packets of flows. They reported 236 and 50 flow table entries, respectively, for achieving their highest detection rate in the UNI1 dataset. Conversely, NELLY and Mahout do not require flow table entries for collecting data since they operate at the server-side.

Control overhead. Flow detection methods require ToR switches to send control packets to the controller, either for the collection of flow data or for the reporting of detected elephant flows. The smaller the control overhead, the lower are the link utilization and the impact on the controller performance (since it has to process fewer control packets). The overhead of this control was computed by assuming no loss in the network and a control packet of 64 bytes. OFSP collects information from the first three packets of each flow, generating a control overhead of 402 kbps. FlowSeer collects information from the first five packets of sampled flows (i.e., 30% of the flow data) and potential elephants, yielding a control overhead of 288 kbps. ESCA reduces the control overhead to 215 kbps by using a sampling method that only reports information from the first packet. In contrast, NELLY and Mahout merely require that ToR switches send information of flows marked as elephants, greatly reducing the control message overhead to 4.4 kbps and 1.1 kbps, respectively.

Detection time. Timely detection of elephant flows enables the controller to make early decisions to improve routing. OFSP, ESCA, FlowSeer, and NELLY enable a short detection time by proactively detecting elephant flows. ESCA reported a detection time of 1.98 s for achieving the highest detection rate. OFSP and NELLY detect elephants in a shorter time since they rely on the first N packets. On average, the detection time was

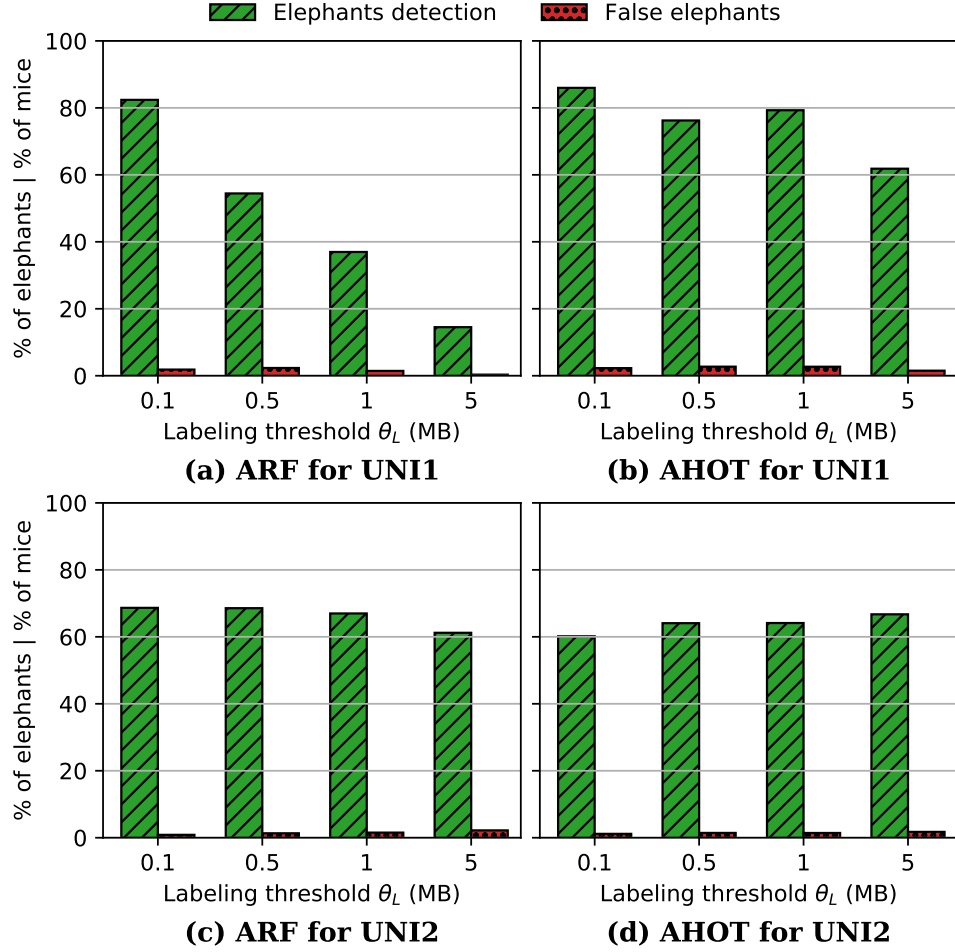


Figure 3.8: Accuracy of NELLY with the ARF and AHOT algorithms when varying the labeling threshold (θ_L) for all the IPv4 flows in UNI1 and UNI2 data traces

0.5 s for OFSP ($N = 3$) and 0.8 s for NELLY ($N = 7$). Further experimentation is needed to evaluate the detection time of FlowSeer. Nevertheless, the detection time of the latter would be slightly greater than for ESCA, since it is also based on sampling and considers the first five packets (ESCA considers only one packet). In contrast, Mahout relies on a reactive mechanism that detects elephant flows after their corresponding socket buffer in a server surpasses a certain threshold. Assuming a small threshold of 100 kB, the average detection time of Mahout is 3.8 s. However, unlike ML-based flow detection methods, the detection time of Mahout becomes longer as the threshold increases, which may cause hot-spots before the traffic carried by elephant flows reaches the threshold.

Network modifications. ESCA proposes a sampling method that depends on non-existing SDN specifications, hence, requiring custom-made switch hardware. In contrast, OFSP, FlowSeer, NELLY, and Mahout rely on OpenFlow [48], therefore enabling the use of commercial switches. Essentially, NELLY and Mahout require the installation of additional software in the servers, which need only to be done once with further configuration possible on the basis of a policy manager or autonomously. This installation can be carried out by using DevOps automation tools, such as Puppet and Chef, that support the distribution of software components to the operating systems of servers [268]. Moreover,

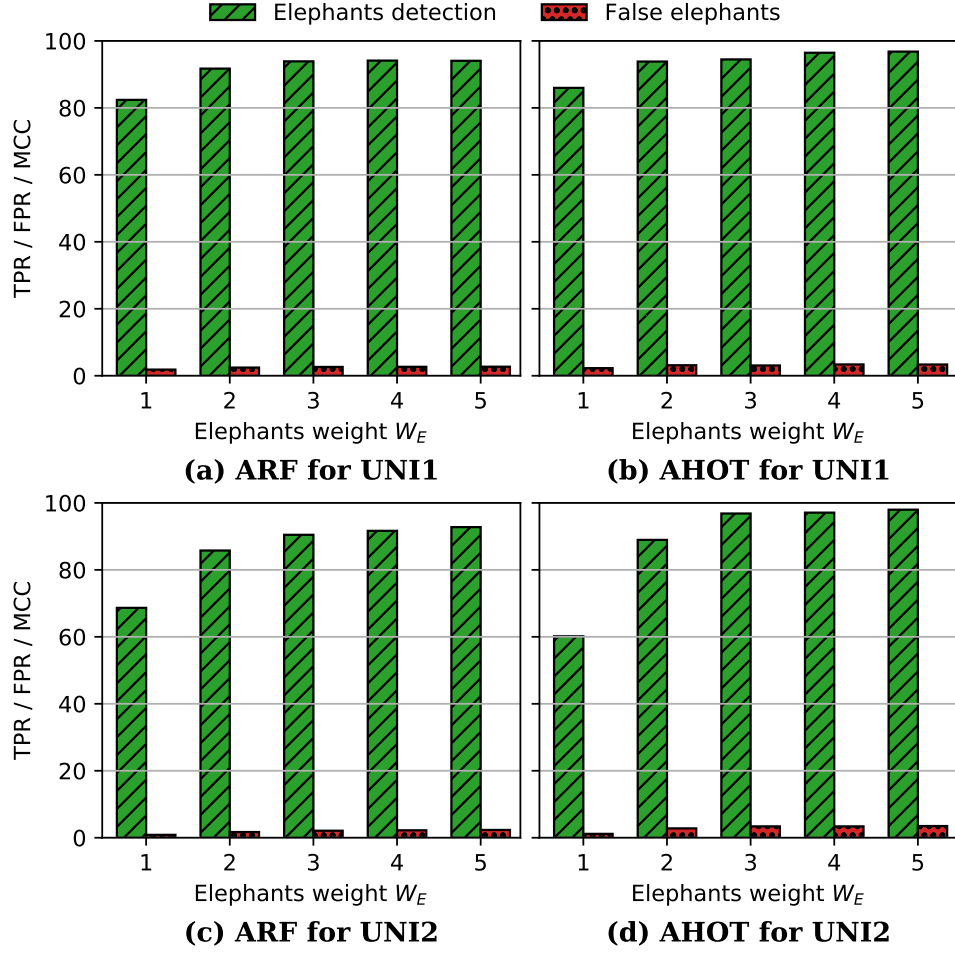


Figure 3.9: Accuracy of NELLY with the ARF and AHOT algorithms when varying the inverse weights of elephant flows (W_E) for all the IPv4 flows in UNI1 and UNI2 data traces

virtualization platforms, such as VMWare and Xen, support software distribution to the servers as updates to the hypervisor without interrupting running virtual machines (either by live-migration or live-updating) [269].

Performance factors. Depending on the location of the flow detection method, different factors may affect its performance. Controller-side methods (i.e., OFSP, ESCA, and FlowSeer) rely on the resources available at the controller and ToR switches. The controller should be powerful enough for detecting all the elephants and processing the control packets sent by the ToR switches in the DCN. Similarly, the ToR switches should have enough memory for installing the required flow table entries. Moreover, the accuracy of the controller-side methods can be negatively affected if the ToR switches drop some of the first packets of the elephant flows. On the other hand, NELLY and Mahout operate at the server-side, so they depend on servers resources. As NELLY is based on ML, it requires more resources than does Mahout. Both server-side methods detect the elephants generated by each server (i.e., distributed operation). Note that servers should be able to monitor the first packets of the elephant flows for avoiding a decrease in accuracy.

3.4 Final remarks

ECMP, which is the default routing technique in DCNs, can degrade the network performance when handling mouse and elephant flows. Novel techniques for scheduling the elephant flows can alleviate this problem. Recently, several approaches have incorporated ML techniques at the controller-side of SDDCNs to detect elephant flows. However, these approaches can produce heavy traffic overhead, low scalability, low accuracy, and high detection time. In this chapter, we introduced NELLY to deal with this limitations. NELLY performs continuous learning and requires limited memory resources by virtue of using incremental learning. An extensive evaluation based on real packet traces and various incremental learning algorithms demonstrated the high accuracy and speed of NELLY when used with the ARF and AHOT algorithms. Moreover, an analytical comparison to seminal related works corroborated the scalability of NELLY as well as its generation of low traffic overhead and the fact that no modifications in SDN standards are required.

Chapter 4

Multipath routing based on software-defined networking and machine learning for data center networks

The majority of the flows in DCNs are mice (i.e., , small, short-lived flows), whereas only very few are elephants (i.e., , large, long-lived flows) [8–11]. Mouse flows represent latency-sensitive and bursty network traffic, such as search results [260], and elephant flows depict massive data traffic, such as server migrations [270]. These traffic characteristics negatively impact the performance of mice in DCNs as elephants tend to utilize most bandwidth, introducing delay to mouse flows sharing the same links.

To tackle this problem, first, this chapter proposes a PMAC-based multipath routing algorithm for steering traffic flows (mainly, mice) in SDDCNs that supports transparent host migration across the whole network while reducing the number of rules installed on SDN switches, decreasing the delay introduced to flows traversing the network. Second, this chapter introduces a flow rescheduling method at the controller-side of SDDCNs that applies incremental deep learning for predicting traffic characteristics of elephant flows to compute and install the best path per elephant flow across the network.

4.1 Pseudo-MAC-based multipath routing in software-defined data center networks

As described in Section 2.3.3, several SDDCN (i.e., DCNs using SDN) multipath routing mechanisms focus on dynamically rescheduling paths for identified elephants from the controller while relying on a default multipath routing algorithm, like ECMP [7], for handling the rest of the flows (potentially, mice). However, most of these SDN-based mechanisms do not specify how to implement such a default multipath routing algorithm in an SDN environment. The simplest SDN implementation would dynamically compile and install the path for each flow from the controller. However, this implementation introduces

a delay (approximately, 10 ms¹) when edge switches (a.k.a. ToR) send the first packet of each flow to the controller [27, 271]. This delay negatively affects the latency-sensitive mouse flows [272]. Moreover, the massive number of flows in DCNs leads to scalability issues due to a large occupancy of the narrow flow tables in SDN switches and a significant overload to the controller, which increases the processing delay [273].

Implementing the default multipath routing algorithm (e.g., ECMP) by installing static flow rules overcomes the controller-related drawbacks (i.e., overload and first packets delay). However, this implementation usually relies on multipath routing algorithms that install a high number of source-destination (either Media Access Control (MAC) or IP) static flow rules. As aforementioned, an SDN switch with many flow rules installed creates scalability issues and increases the delay while the switch matches the rule for a specific flow. Some approaches have proposed algorithms for reducing the number of flow rules installed, particularly for mice [15, 28, 274]. However, these approaches do not address the complexity of continuously updating the static flow rules in case of dynamic changes in the network state. Therefore, an efficient implementation of the default multipath routing algorithm should avoid sending the first flow packet to the controller, install the less possible number of flow rules in switches, and update these rules as the network state changes.

This section introduces an SDN-based multipath routing algorithm, named Pseudo-MAC-based Multipath (PM2), which performs efficient routing of flows (mainly, mice) in DCNs following the fat-tree topology [242]. Unlike other proposals that use addresses (either MAC or IP) from hosts, PM2 identifies each switch's layer (i.e., edge, aggregation, core) and position for generating Pseudo-MAC (PMAC) prefixes, which allow installing routing flow rules with wildcards to save space in the flow tables. PM2 then intercepts Address Resolution Protocol (ARP) messages at the controller to generate a PMAC address for each host (virtual and physical) and install the corresponding flow rules in edge switches for both parsing MAC addresses and reaching destination. Results reveal that PM2 significantly reduces the number of rules installed in switches while supporting transparent host migration across the whole SDDCN.

The remainder of this section is as follows. Section 4.1.1 demonstrates the impact of the number of switch installed flow rules on the routing delay. Section 4.1.2 describes the algorithm and procedures of PM2. Section 4.1.3 presents a proof of concept of PM2 in an emulated scenario. Section 4.1.4 compares PM2 with feasible routing approaches in SDDCNs regarding the number of rules installed.

4.1.1 Motivation

Figure 4.1 shows three experimental scenarios that we use to explain how the number of flow rules installed in a switch table impact the delay of flows traversing that switch. All three scenarios deploy a Ryu controller [66] running an algorithm developed in Python that simply installs static flow rules into the flow table of the corresponding switch. The *emulated* scenario uses the network emulation tool Mininet 2.2.2 [275] to deploy

¹Assuming that only the first packet goes to the controller. If multiple packets arrive (i.e., bursty traffic) before installing the flow rule, more packets will bear the cost.

two virtual hosts² (h) and an OpenFlow switch in a single Virtual Machine (VM). The Ryu controller runs in the same Physical Computer (PC) hosting the VM. The *virtual* scenario deploys two virtual hosts and an Open vSwitch 2.5.4 [277] using different VMs interconnected through virtual Ethernet interfaces [278]. The Ryu controller and the three VMs run in the same PC. The *physical* scenario deploys an HP 2920 OpenFlow switch (a.k.a. Aruba 2920) [279] that interconnects two PCs (physical hosts) through OpenFlow VLAN interfaces. The Ryu controller runs in an independent PC. The PCs for all three scenarios shared the following system specifications: Lubuntu 16.04 operating system, 2.40 GHz Intel Core i5 processor, and 3 GB RAM.

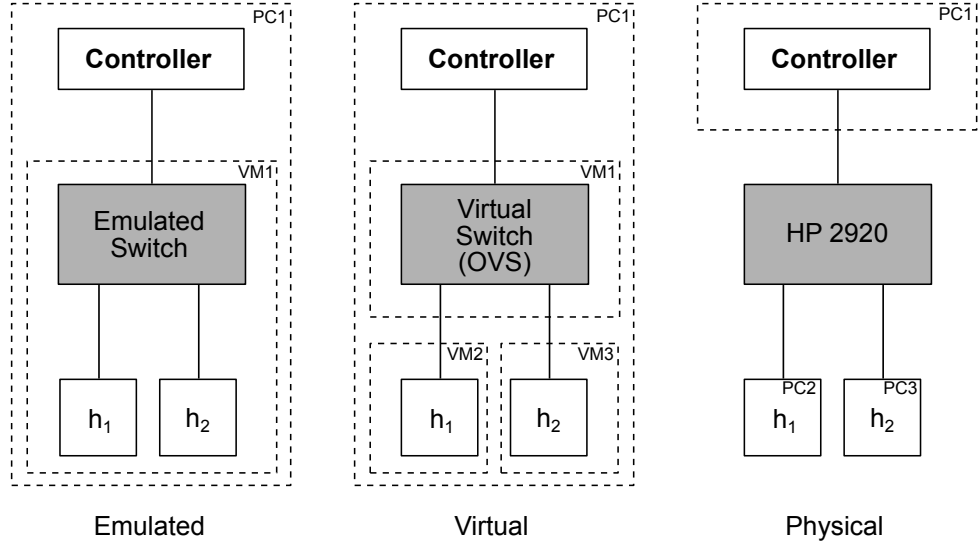


Figure 4.1: RTT measurement experimental scenarios

We measured the RTT for all three scenarios while varying the number and arrangement of flow rules installed in the corresponding switch. RTT represents the total time a packet takes to go to the destination and back to the source. The number of installed flow rules (r) ranged from 1000 to 16000 in steps of 1000 ($r \in R = \{1000, 2000, \dots, 16000\}$), where only two rules matched and forwarded the packets sent between the hosts (*routing rules*) while the remaining occupied the flow table memory (*filling rules*). For each $r \in R$, the arrangement of the flow rules varied by placing the routing rules at the beginning, middle, and end of the filling rules. We installed each flow rule arrangement T times in the switch and took the RTT N times for each $t \in T$. Each experiment was about sending a ping request from the first host to the second one and then measuring the RTT in the first host after receiving the corresponding ping reply from the second host.

Figure 4.2 depicts the results for each scenario with $T = 30$ and $N = 30$. These results reveal that the RTT for the emulated scenario remains around 0.1 ms regardless of the number of installed flow rules and the position of the routing rules. Similarly, the RTT for the virtual scenario persists as the number and arrangement of the installed flow rules vary, though it approximates to 1.3 ms. Such an increment in the RTT value is because the virtual scenario deploys more virtual Ethernet interfaces between the hosts than the emulated scenario. In contrast, the RTT for the physical scenario increases from 0.8 ms

²Mininet virtual hosts are processes running in their own Linux network namespace [276].

up to 2.1 ms as the number of installed flow rules grows and as the position of the routing rules moves to the end of the filling rules. For example, for routing rules placed at the beginning of the filling rules, the RTT grows from 0.8 ms for $r = 1000$ to 1.1 ms for $r = 16000$. Whereas, for $r = 16000$, the RTT increases to 1.6 ms and 2.1 ms when placing the routing rules in the middle and end of the filling rules, respectively.

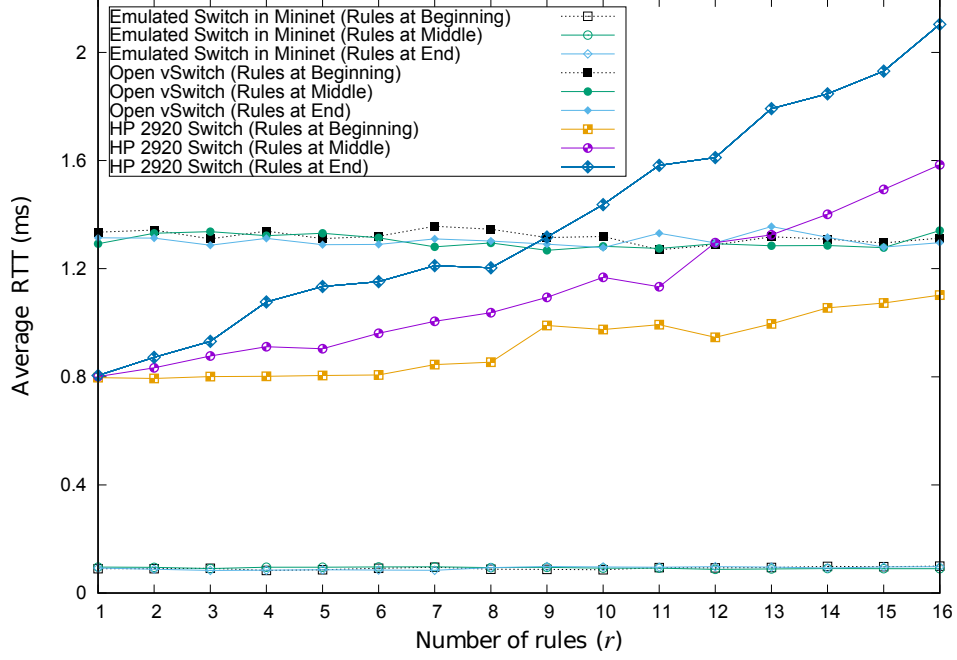


Figure 4.2: Average RTT per number of flow rules installed in the switch of each experimental scenario with $T = 30$ and $N = 30$

We learned from these experiments that the RTT in the emulated and virtual scenarios remains nearly constant while varying the number of installed flow rules. Second, in the physical scenario, the limited memory in switches causes the number of installed flow rules to impact the delay introduced to the packets of flows traversing the switch. Thus, reducing the number of flow rules installed in SDN switches represents an opportunity to lower the delay introduced to packets of flows (mainly, mice) traveling in SDDCNs.

4.1.2 Pseudo-MAC-based multipath routing

PM2 provides a multipath routing algorithm for steering flows (mainly, mice) in a fat-tree SDDCN. PM2 focuses on supporting transparent host migration across the whole SDDCN while reducing the number of rules installed in the switches, decreasing the delay introduced to the packets of flows traversing the network. To do so, PM2 reuses the key concept of Pseudo-MAC (PMAC), which defines a hierarchical MAC address assigned to the physical and virtual hosts in a fat-tree DCN topology [280].

PM2 generates and assigns 48-bit PMACs using the form *lia:pod:pos:port:vmid*, where *lia*, *pod*, *pos*, and *port* represent eight-bit fields, whereas *vmid* depicts a 16-bit field. *lia* defines PMACs as local-individual addresses (e.g., 0x0a or 00001010³). *pod* depicts the

³A MAC address with the second-least-significant bit of the first octet set to **one** (1) represents a

number of the pod containing the edge switch. *pos* describes the position of the edge switch within the pod. *port* details the switch port number connected with the physical host. *vmid* identifies different virtual hosts using a bridge adapter in the same physical host (0x00 if no virtual hosts using bridge adapters).

Figure 4.3 presents the process executed by PM2 for installing the routing rules. This process involves four main tasks: (i) generate PMAC addresses, (ii) define the set of routing rules based on the PMAC addresses, (iii) install the set of routing rules in the corresponding switches; and (iv) update the routing rules when network changes occur. Note the PM2 process runs after either discovering the topology or intercepting an ARP message.

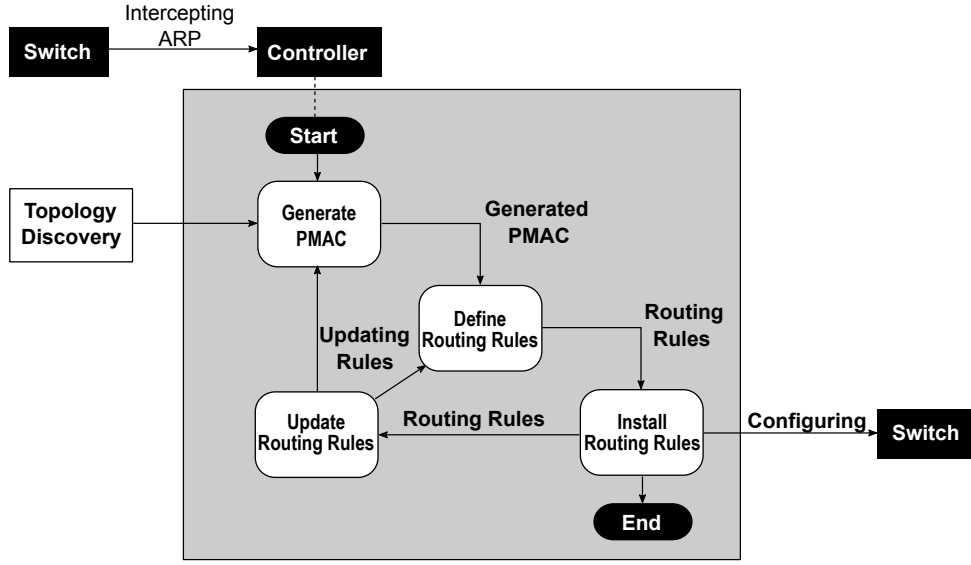


Figure 4.3: PM2 process to install routing rules

Topology discovery

PM2 requires an overview of the fat-tree DCN topology, including the position of the switches and their connections to the other switches, for generating the PMAC addresses and installing the required routing rules. The fat-tree topology represents a k -ary network that consists of k -port switches distributed into three layers: core, aggregation, and edge, top-down. There are k pods interconnected by $\frac{k^2}{4}$ core switches. Each pod contains $\frac{k}{2}$ aggregation switches (upper pod) and $\frac{k}{2}$ edge switches (lower pod). Aggregation switches connect to the core switches. Each aggregation switch connects to each edge switch in the same pod. Each edge switch connects to $\frac{k}{2}$ physical hosts. The fat-tree topology provides a high degree of available path diversity: between any source and destination host pair from different pods, $\frac{k^2}{4}$ equal-cost paths exist, each corresponding to a core switch, although the paths are not link-disjoint. Table 4.1 depicts the fat-tree topology size, in terms of hosts, switches, and links, based on the typical number of ports in switches.

locally administered address, whereas setting to **zero (0)** the least-significant bit of the first octet defines an individual address meant for unicast communication [281].

Table 4.1: Fat-tree topology size

Order* (k)	Pods	Hosts	Edge switches	Aggregation switches	Core switches	Total switches	Links
4	4	16	8	8	4	20	48
8	8	128	32	32	16	80	384
10	10	250	50	50	25	125	750
12	12	432	72	72	36	180	1296
16	16	1024	128	128	64	320	3072
24	24	3456	288	288	144	720	10368
28	28	5488	392	392	196	980	16464
48	48	27648	1152	1152	576	2880	82944
52	52	35152	1352	1352	676	3380	105456

*Based on the typical number of ports in switches

For *topology discovery*, PM2 relies on the Link Layer Discovery Protocol (LLDP) to obtain the connections among the switches. To do so, first, PM2 installs a routing rule on every switch for redirecting any received LLDP packet to the controller, and then frequently instructs every switch for sending LLDP packets through all the ports. The switch connections allow PM2 determining the position of the switches in the fat-tree topology. Assuming the network is fully connected, PM2 identifies as edge switches those ones that failed to receive LLDP packets from half of their ports (i.e., ports connected to physical hosts). In case the network is not fully connected, PM2 might rely on the switch identifier (ID) to identify edge switches. In OpenFlow, for example, the switch ID is a 64-bit field known as datapath ID: the 48 Least Significant Bits (LSB) correspond to the switch MAC, whereas the 16 Most Significant Bits (MSB) depend on the switch implementation (vary among models). Most OpenFlow switch implementations allow defining a custom MSB, enabling network administrators to set a specific value in the datapath ID of edge switches that PM2 identifies when the switches connect to the controller. Finally, switches connected to edge switches become aggregation switches, and those connected to aggregation switches become core switches.

Having determined the position of the switches in the fat-tree topology, PM2 generates PMAC prefixes for the pods and edge switches. PMAC prefixes for the pod and edge switches have the form $lia:pod:*$ and $lia:pod:pos:*$, respectively, where $*$ represents the remaining wildcard bits (32 bits and 24 bits, respectively). For example, the pod two in the fat-tree topology would have the PMAC prefix $0a:02:*$, whereas the edge switch in the position one of the pod two would have the PMAC prefix $0a:02:01:*$. Note that all the hosts (virtual and physical) under the same pod and edge switch share the same PMAC prefix.

After generating the PMAC prefixes, PM2 defines and installs the wildcard routing rules in the core and aggregation switches for enabling the top-down communication (i.e.,

from core to aggregation and from aggregation to edge). Core switches only need to match the PMAC prefix of a pod for sending a packet to a host that is under that pod. For example, if the port two of a core switch connects to the pod two, the core switch would have installed a wildcard routing rule that forwards through the port two every packet whose destination MAC address matches the PMAC prefix `0a:02:*`. Similarly, aggregation switches only need to match the PMAC prefix of an edge switch for sending a packet to a host that is connected to that edge switch. For example, if the port one of an aggregation switch in the pod two connects to the edge switch in position one, the aggregation switch would have installed a wildcard routing rule that forwards through the port one every packet whose destination MAC address matches the PMAC prefix `0a:02:01:*`. These wildcard routing rules based on PMAC prefixes enable reducing the number of flow rules installed on switches at the core and aggregation layers.

PM2 also defines and installs group routing rules at the edge and aggregation switches for enabling the bottom-up communication (i.e., from edge to aggregation and from aggregation to core). These group routing rules represent low priority rules that switches apply when none of the PMAC-based routing rules, with a higher priority, match the packet. Therefore, if a packet from a source host arrives at an edge switch but the destination host is not connected to the same edge switch, this switch applies a group routing rule to select one of the ports connected to the aggregation switches to forward the packet. Similarly, if a packet from an edge switch arrives at an aggregation switch but the destination host is not in the same pod, the aggregation switch applies a group routing rule select one of the ports connected to the core switches to forward the packet. These group routing rules enable multipath routing between the equal-cost paths in the fat-tree topology. For example, OpenFlow [55] provides a group table that contains group entries, each consisting of a list of action buckets (e.g., forward through port two, forward through port three, etc.). OpenFlow switches apply the actions in one or more action buckets depending on the group type. Particularly, the select group type executes one of the action buckets in the group. The selection of the action bucket in the group depends on the switch implementation. Open vSwitch, for instance, applies a hash function on the packet header for selecting the action bucket in the group.

The last routing rules that PM2 defines using the topology discovery and installs on the switches are for handling ARP messages. PM2 installs a routing rule on every edge switch for intercepting all the ARP requests and replies. To do so, the ARP intercepting routing rule matches all the packets with the EtherType field set to the ARP protocol (0x0806) and forwards them to the controller. At the core and aggregation layer, PM2 simply instructs the switches to drop all the ARP messages. Note the ARP routing rules present the highest priority among the installed routing rules.

Finally, PM2 triggers the update of routing rules when discovering changes in the topology, such as broken links or switches down. That way, PM2 avoids forwarding packets through disconnected ports, minimizing the rate of dropped packets. To do so, PM2 generates a new configuration of routing rules and compares it with the current configuration to enforce only the modifications. Note this updating task might imply generating new PMAC prefixes, installing new routing rules, and deleting existing routing rules.

ARP interception

PM2 also operates when intercepting ARP messages. As shown in Figure 4.4, let's assume that a source host *A*, using the IP 10.0.0.1, wants to communicate with a destination host *B*, using the IP 10.0.0.6. The source host *A* initially ignores the IP 10.0.0.6 is served by the destination host *B*, so host *A* sends an ARP request to ask “who has 10.0.0.6?” and “what is your MAC address?” in order to get the MAC address from host *B*. Since host *A* ignores the MAC address from host *B*, the ARP request uses the broadcast MAC address ($ff:ff:ff:ff:ff:ff$) for the destination MAC. This ARP request matches the intercepting routing rule at the edge switch, which forwards it to the controller.

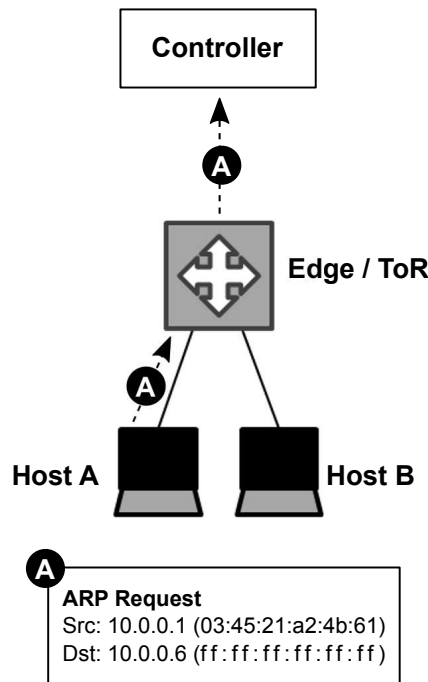


Figure 4.4: ARP request interception in PM2

As depicted in Algorithm 4.1, PM2 receives as input the intercepting edge switch data (*sw*) and the intercepted ARP message (*A*). The intercepting edge switch data includes the switch ID (*id*) and the input port number the edge switch received the ARP packet in (*in_port*). Note the intercepting edge switch data is not part of the ARP message but of the messages sent to the controller by SDN protocols (e.g., OpenFlow packet-in message). The intercepted ARP message contains the fields *oper*, *ip_src*, *ip_dst*, *eth_src*, and *eth_dst*, which represent the ARP operation code (e.g., request or reply), the source IP address, the destination IP address, the source MAC address, and the destination MAC address. In the example from Figure 4.4, the value of the *oper* field is 0x01 since the ARP message corresponds to an ARP request (this field is 0x02 for ARP reply).

Algorithm 4.1 shows that PM2, first, generates the PMAC address for the source host using the switch ID (*sw.id*), the input port (*sw.in_port*), the source IP (*A.ip_src*), and the source MAC (*A.eth_src*). Then, PM2 inserts the host generated PMAC into the PMAC table, associating it with the source IP address (*A.ip_src*). Note that the switch ID (*sw.id*), the input port (*sw.in_port*), and the source MAC (*A.eth_src*) are also stored

Algorithm 4.1: Handling intercepted ARP messages in PM2

```

data:
    sw   edge switch that intercepted ARP message
    A    intercepted ARP message data
     $T_H$  PMAC table for hosts (virtual or physical)
    R    installed PMAC routing rules
    E    set of edge (ToR) switches
     $\theta_S$  ARP stale time threshold

    // Function to handle intercepted ARP messages
1  function HANDLE_ARP(sw, A):
    // Generate and store PMAC
2    pmac  $\leftarrow$  GENERATE_PMAC(sw.id, sw.in_port, A.ip_src, A.eth_src);
3     $T_H[A.ip\_src] \leftarrow \{pmac, sw.id, sw.in\_port, A.eth\_src\}$ ;

    // Install PMAC routing rules
4    if  $\{sw.id, A.eth\_src, pmac\} \notin R$  then
5        | INSTALL_PMAC_RULES(sw.id, A.eth_src, pmac, sw.in_port);
6        |  $R \leftarrow \{sw.id, A.eth\_src, pmac\}$ ;
7    end

    // Update ARP data
8    A.eth_src  $\leftarrow pmac$ ;
9    UPDATE_TIME(A.dst_ip);
    // Check if ARP request
10   if A.oper = 1 then
        // Check if destination host is known
11       if A.dst_ip  $\notin T_H$  then
12           | actions  $\leftarrow [flood]$ ;
13           | for each e  $\in E$  do
14               | e.SEND_PACKET(A, actions);
15           | end
16       else
            // Check if stale time is shorter than threshold
17           if STALE_TIME(A.dst_ip) >  $\theta_S$  then
18               | CHECK_CONNECTION(A.dst_ip);
19           else
                // Send ARP reply with source MAC set to destination PMAC
20                 dst_pmac  $\leftarrow T_H[A.dst\_ip][pmac]$ ;
21                 reply  $\leftarrow$  ARP_REPLY(A);
22                 reply.eth_src  $\leftarrow dst\_pmac$ ;
23                 actions  $\leftarrow [A.in\_port]$ ;
24                 sw.SEND_PACKET(reply, actions);
25             end
26         end
27     else
        // Forward ARP reply
28         dst_swid, dst_port  $\leftarrow T_H[A.dst\_ip]$ ;
29         dst_sw  $\leftarrow$  GET_SWITCH(dst_swid);
30         actions  $\leftarrow [dst\_port]$ ;
31         dst_sw.SEND_PACKET(A, actions);
32     end
33 end
  
```

in the PMAC table record. In Figure 4.5, for example, the PMAC table associates the IP 10.0.0.1 with the PMAC 0a:01:01:01:00:00, which was generated using the ARP request

sent by host *A*, whose MAC is 03:45:21:a2:4b:61 and connects to the edge switch on port one.

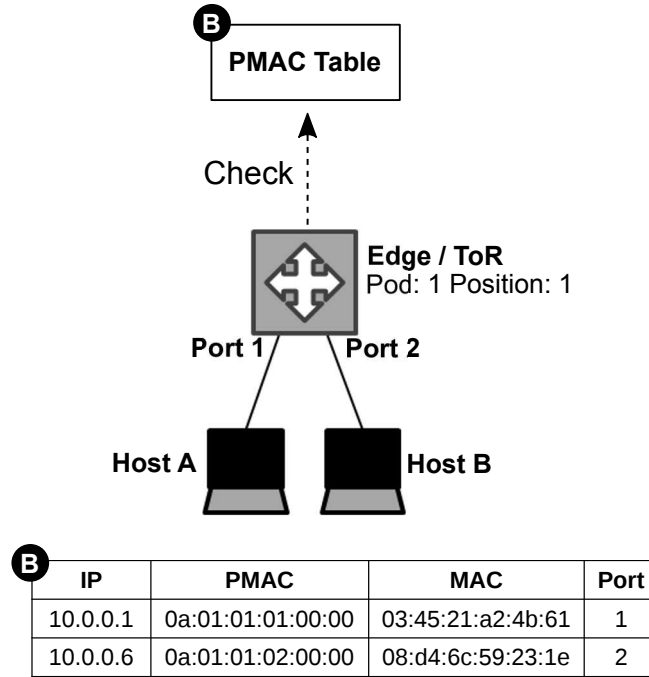


Figure 4.5: PMAC table in PM2

Continuing with Algorithm 4.1, PM2 defines and installs on the intercepting edge switch the routing rules associated with the generated PMAC, in case it has not been done yet. The PMAC routing rules consist of two routing rules per host on the edge switch the host connects to. The first rule updates the source MAC of any packet received from any of the hosts connected to the edge switch. This rule parses the actual MAC of the host to the PMAC generated for that host. For example, in Figure 4.5, this rule parses the source MAC from 03:45:21:a2:4b:61 to 0a:01:01:01:00:00 for any packet host *A* sends to the edge switch. The second rule updates the destination MAC of any packet targeting any host connected to the edge switch and forwards the packet through the corresponding port. This rule parses the PMAC generated for the host to the actual MAC of that host. For example, in Figure 4.5, this rule, first, parses the destination MAC from 0a:01:01:01:00:00 to 03:45:21:a2:4b:61 for any packet targeting host *A*, and then forwards the packet through port one.

Next, Algorithm 4.1 depicts that PM2 updates the source MAC (*A.eth_src*) of the ARP message by setting the generated PMAC. This enables each host to associate the IPs to the PMACs generated for the other hosts instead of using their actual MACs. Therefore, when the hosts send packets targeting any IP, they will set the source MAC with their own actual MAC addresses and the destination MAC with the corresponding PMAC that the controller reported for such IP. In our example, PM2 would parse the source MAC of the ARP request from 03:45:21:a2:4b:61 to 0a:01:01:01:00:00. Following, PM2 checks the operation code of the intercepted ARP message (*A.oper*) to discriminate ARP requests and ARP replies. For intercepted ARP requests (*A.oper* = 1), if the PMAC table ignores the IP destination address (*A.dst_ip*), PM2 instructs all the edge switches

to flood the updated ARP request (i.e., send the packet through all ports) to find the host with the destination IP. The destination host then receives the updated ARP request and responds with an ARP reply to inform that the destination IP is at the corresponding MAC. For example, in Figure 4.5, host *B* would respond that IP 10.0.0.6 is at MAC 08:d4:6c:59:23:1e. Note the ARP reply matches the intercepting routing rule at the edge switch, which forwards it to the controller.

As show in Algorithm 4.1, PM2 also leverages ARP replies for generating host PMACs, which are further stored in the PMAC table and used for defining and installing the PMAC routing rules on the intercepting edge switch. In Figure 4.5, for example, the PMAC table associates the IP 10.0.0.6 with the PMAC 0a:01:01:02:00:00, which was generated using the ARP reply sent by host *B*, whose MAC is 08:d4:6c:59:23:1e and connects to the edge switch on port two. Subsequently, PM2 updates the source MAC (*A.eth_src*) by setting the generated PMAC and recognizes the ARP message is an ARP reply (*A.oper* = 2), so it forwards the updated ARP reply to the destination host via the corresponding edge switch and port. In our example, PM2 would parse the source MAC of the ARP reply from 08:d4:6c:59:23:1e to 0a:01:01:02:00:00 and forward it through port one towards host *A*.

When validating if the PMAC table knows the IP destination from the ARP request (*A.dst_ip*), in case it does but the controller has not intercepted an ARP message (request or reply) from that IP for longer than a stale time threshold (θ_S), PM2 checks the connection by sending the updated ARP request directly to the known destination host. When the destination host replies, PM2 forwards the updated ARP reply to the source host, reporting that the destination IP is at the generated PMAC. If the controller does not intercepts an ARP reply from the destination host for longer than a removal time threshold (θ_R), PM2 removes the record associated with the destination IP from the PMAC table. On the other hand, in case the PMAC table knows the destination IP from the ARP request (*A.dst_ip*) and the time since the controller intercepted an ARP message from that IP is shorter than θ_S , PM2 generates and sends an ARP reply to the source host, reporting that the destination IP is at the generated PMAC. Note that θ_S and θ_R enable recognizing hosts down to avoid filling the network with traffic (e.g., UDP) that has no destination.

Finally, PM2 supports transparent migration of hosts across the whole SDDCN. Transparent migration refers to moving a host (virtual or physical) while keeping its IP and being able to communicate with that host without performing manual configurations. In PM2, when changing the position of a host, any source host communicating with it stops receiving replies for the direct ARP requests. Source hosts will receive an ARP reply either when one of them sends a broadcast ARP request or when the migrated host sends an ARP request for communicating with any other host. These ARP messages allow discovering the new position of the migrated host, for which PM2 updates the PMAC address and the corresponding routing rules.

4.1.3 Implementation

Figure 4.6 depicts the proof of concept we implemented for evaluating the feasibility of PM2. We deployed this implementation on a server at the Computer Networks Laboratory (LRC, Laboratório de Redes de Computadores) of the University of Campinas. This server runs Debian 8.11 server in a 2.66 GHz Intel Xeon Processor X5650 machine, with 12 physical cores, 24 logical cores, and 40 GB RAM. On the server, we deployed a VM for running Mininet [275], a well-known tool for emulating OpenFlow networks [48]. We installed Mininet 2.3.0 and Open vSwitch 2.5.9 [277], which Mininet uses for deploying the OpenFlow switches in the emulated network. Open vSwitch 2.5.9 fully supports up to OpenFlow 1.3 and OpenFlow 1.5 with missing features. The Mininet VM runs an Ubuntu 16.04 server with 6 logical cores and 20 GB RAM. Using the Mininet Python API, we developed a custom fat-tree topology that accepts as input a parameter k for setting the size of the fat-tree network (see Table 4.1). Our custom fat-tree topology implementation is available in [282]. We executed our custom fat-tree topology on the Mininet VM, varying the size of the network.

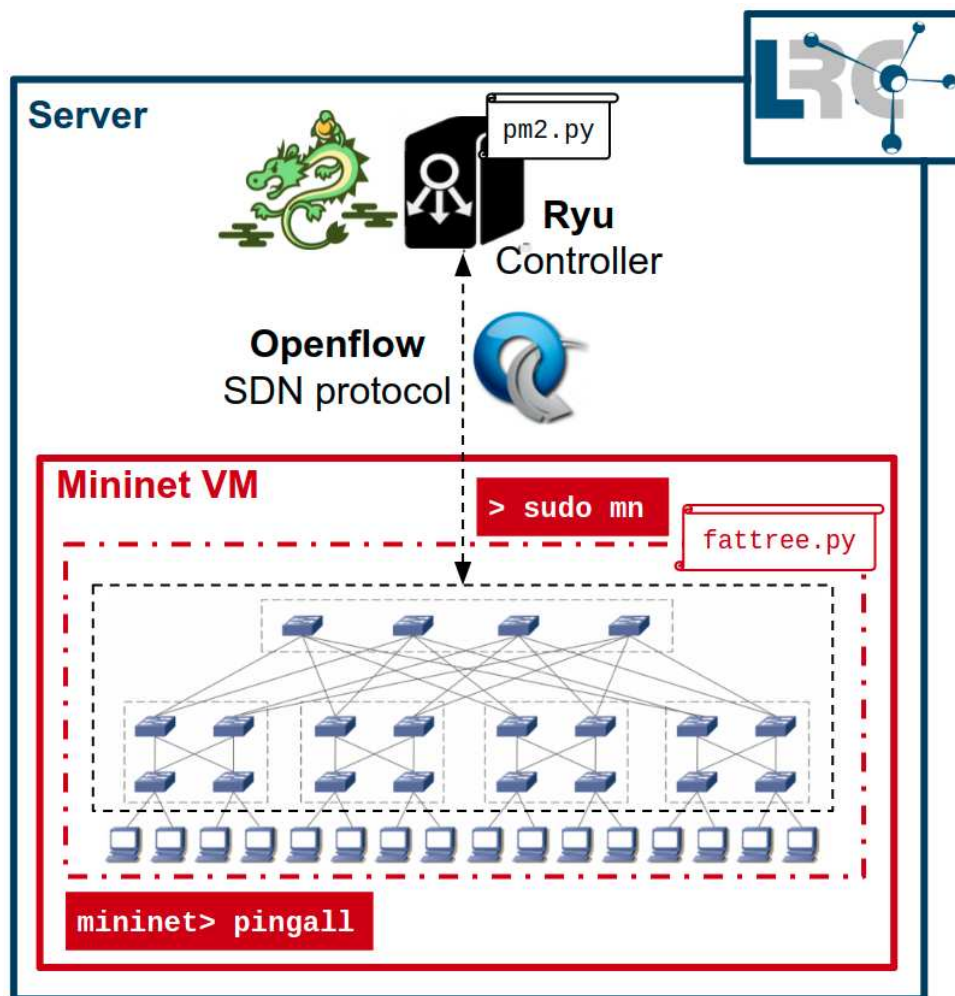


Figure 4.6: Implementation of PM2

On the server, we also deployed the Ryu SDN framework [66] as the controller for managing the OpenFlow switches in Mininet. We installed Ryu 4.34, which fully supports

up to OpenFlow 1.5 [55]. Using the Python libraries from the Ryu SDN framework, we developed a network application that implements PM2, including the topology discovery, the ARP management, and the process for generating PMAC addresses, defining the routing rules, and installing them on the OpenFlow switches. Note we assumed a fully connected network, so our PM2 implementation identifies as edge switches those ones that failed to receive LLDP packets from half of their ports. Our Ryu network application that implements PM2 is available in [282]. We executed our PM2 network application on the Ryu SDN framework.

To test that PM2 installs the appropriate routing rules, we performed a ping test between all the hosts in the emulated fat-tree network, validating that all of them were able to communicate with each other. Due to resource limitations, we were able to test the communication for a fat-tree topology of size 16, which deploys 1024 hosts, 320 switches (128 at edge layer, 128 at aggregation layer, and 64 at core layer), and 3072 links (see Table 4.1).

4.1.4 Analytic evaluation

PM2 enforces a PMAC-based routing for reducing the number of routing rules installed on the switches. PM2 is similar to PortLand [280] in that both assign PMAC addresses to the hosts according to their position in the fat-tree topology. However, for topology discovery, PortLand depends on a custom Location Discovery Protocol (LDP), whereas PM2 relies on the widely-used LLDP. Moreover, unlike PortLand, PM2 defines and installs the routing rules for supporting multipath routing (i.e., group routing rules). PortLand simply assumes a standard technique such as flow hashing in ECMP. Finally, PM2 validates the time that hosts have been in communication with others (i.e., θ_S and θ_R) to identify hosts down, avoiding filling the network with traffic (e.g., UDP) that has no destination. PortLand does not account for hosts down.

We further evaluated PM2 by analyzing the number of rules installed per switch for an all-to-all communication in an OpenFlow fat-tree topology, in comparison with other three feasible approaches: (i) Layer 2 (L2), which uses MAC addresses for communication, (ii) Hierarchical Layer 3 (HL3), which divides the network using IP prefixes; and (iii) a hybrid approach between L2 and HL3 (L2-HL3), which uses MAC addresses from the aggregation layer down and divides the pods using IP prefixes. Hereinafter, k denotes the order (i.e., size) of the fat-tree topology (see Table 4.1).

First, we compute the number of rules that PM2 installs on the switches of an OpenFlow fat-tree topology. Equation 4.1 describes the number of rules that PM2 installs per edge switch. The four (4) rules are the ARP intercepting rule, the group rule, a miss destination MAC rule that redirects to the group rule, and a miss source MAC rule that handles packets coming from the aggregation layer. Moreover, PM2 installs two PMAC routing rules per each host h in each server machine $s \in 1.. \frac{k}{2}$ connected to the edge switch. The first PMAC rule parses the source MAC, whereas the second rule parses the destination MAC and forwards the packet through the corresponding port. Note that h includes the physical host and the virtual hosts using a bridge adapter to connect to the network. Virtual hosts using NAT adapters communicate to the network using the IP

and MAC addresses of the physical host.

$$4 + 2 \cdot \sum_{s=1}^{k/2} h_s \quad (4.1)$$

Equation 4.2 depicts the number of rules that PM2 installs per aggregation switch. The three (3) rules are the ARP dropping rule, the group rule, and a miss destination MAC rule that redirects to the group rule. In addition, PM2 installs a wildcard routing rule per each of the $\frac{k}{2}$ edge switches connected to the aggregation switch. Note these wildcard rules use the PMAC prefixes generated for the edge switches (i.e., *lia:pod:pos:**).

$$3 + \frac{k}{2} \quad (4.2)$$

Equation 4.3 presents the number of rules that PM2 installs per core switch. One (1) ARP dropping rule and a wildcard routing rule per each of the k pods connected to the core switch. These wildcard rules use the PMAC prefixes generated for the pods (*lia:pod:**).

$$1 + k \quad (4.3)$$

Next, we calculate the number of rules installed on the switches by the other three OpenFlow-feasible approaches, namely, L2, HL3, and L2-HL3. Equation 4.4 denotes the number of rules that these three routing approaches install per edge switch. The three (3) rules are an ARP management rule, the group rule, and the miss destination MAC rule that redirects to the group rule. Furthermore, these approaches install a routing rule per each host h in each server machine $s \in 1..\frac{k}{2}$ connected to the edge switch.

$$3 + \sum_{s=1}^{k/2} h_s \quad (4.4)$$

Equation 4.5 describes the number of rules that L2 and L2-HL3 install per aggregation switch. The three (3) rules are an ARP management rule, the group rule, and the miss destination MAC rule that redirects to the group rule. Moreover, L2 and L2-HL3 install a routing rule per each host h in each server machine $s \in 1..\frac{k}{2}$ connected to each edge switch $e \in 1..\frac{k}{2}$ connected to the aggregation switch. In contrast, the number of rules that HL3 installs per aggregation switch is given by Equation 4.2. Similar to PM2, HL3 leverages address prefixes for reducing the number of rules at the aggregation layer.

$$3 + \sum_{e=1}^{k/2} \sum_{s_e=1}^{k/2} h_{s_e} \quad (4.5)$$

Lastly, Equation 4.6 depicts the number of rules that L2 installs per core switch. One (1) ARP management rule and a routing rule per each host h in each server machine $s \in 1..\frac{k}{2}$ connected to each edge switch $e \in 1..\frac{k}{2}$ belonging to each pod $p \in 1..k$. In contrast, the number of rules that HL3 and L2-HL3 install per core switch is shown in Equation 4.3. Similar to PM2, HL3 and L2-HL3 leverage address prefixes for reducing

the number of rules at the core layer.

$$1 + \sum_{p=1}^k \sum_{e_p=1}^{k/2} \sum_{s_{ep}=1}^{k/2} h_{s_{ep}} \quad (4.6)$$

For facilitating the analytic comparison, we make some assumptions to compute the number of rules all these approaches install per switch for each layer in a fat-tree topology. Table 4.2 simplifies Equations 4.1-4.6 by assuming only one host (i.e., the physical host) in each server machine (i.e., $h_s = 1$). Furthermore, Figure 4.7 shows the number of rules each approach (PM2, L2, HL3, L2-HL3) install per switch for each layer in a fat-tree topology of size $k = 48$ and only one host in each server machine (i.e., $h_s = 1$). Note that a fat-tree topology of size $k = 48$ consists of 27648 hosts, 2880 switches (1152 at edge layer, 1152 at aggregation layer, and 576 at core layer), and 82944 links (see Table 4.1). Moreover, the Y axis of Figure 4.7 is in logarithmic scale.

Table 4.2: Comparison of PM2 routing with related approaches

FEATURE	L2	HL3	L2-HL3	PM2
Rules per edge (ToR) switch*	$3 + \frac{k}{2}$	$3 + \frac{k}{2}$	$3 + \frac{k}{2}$	$4 + k$
Rules per aggregation switch*	$3 + \frac{k^2}{4}$	$3 + \frac{k}{2}$	$3 + \frac{k^2}{4}$	$3 + \frac{k}{2}$
Rules per core switch*	$1 + \frac{k^3}{4}$	$1 + k$	$1 + k$	$1 + k$
Transparent host migration	Network	Edge switch	Pod	Network

*Equations obtained assuming only one host in each server ($h_s = 1$)

* k denotes the order (i.e., size) of the fat-tree topology

The results show that PM2, HL3, and L2-HL3 installs much less rules per core switch than L2. Note the former three leverage address prefixes (either PMAC or IP) for reducing the number of rules at the core layer. At the aggregate layer, PM2 and HL3 install much fewer rules than L2 and L2-HL3. In this case, only PM2 and HL3 leverage address prefixes (PMAC and IP, respectively) for reducing the number of rules per aggregate switch. Lastly, PM2 installs a little more rules per edge switch than the other approaches. This is because PM2 requires the MAC-PMAC address parsing. Overall, PM2 installs much fewer rules than L2 and L2-HL3. In contrast, PM2 installs a little more rules than HL3.

However, as described in Table 4.2, HL3 limits the transparent host migration to the edge switch. Note that transparent host migration requires migrated hosts to keep their IPs since services running on other hosts might depend on it. As HL3 divides the whole network using IP prefixes, each edge switch supports only a specific range of IPs. Therefore, using HL3, a host cannot be migrated to another edge switch without changing its IP. On the other hand, L2-HL3 extends the transparent host migration scope to the pod since it uses IP prefixes only for dividing the pods. Whereas, PM2 and L2 enable transparent host migration across the whole fat-tree network.

To sum up, PM2 installs much fewer rules than the other OpenFlow-feasible approaches that support transparent host migration across a fat-tree topology area larger than the same edge switch. Such reduction in the number of rules installed on switches

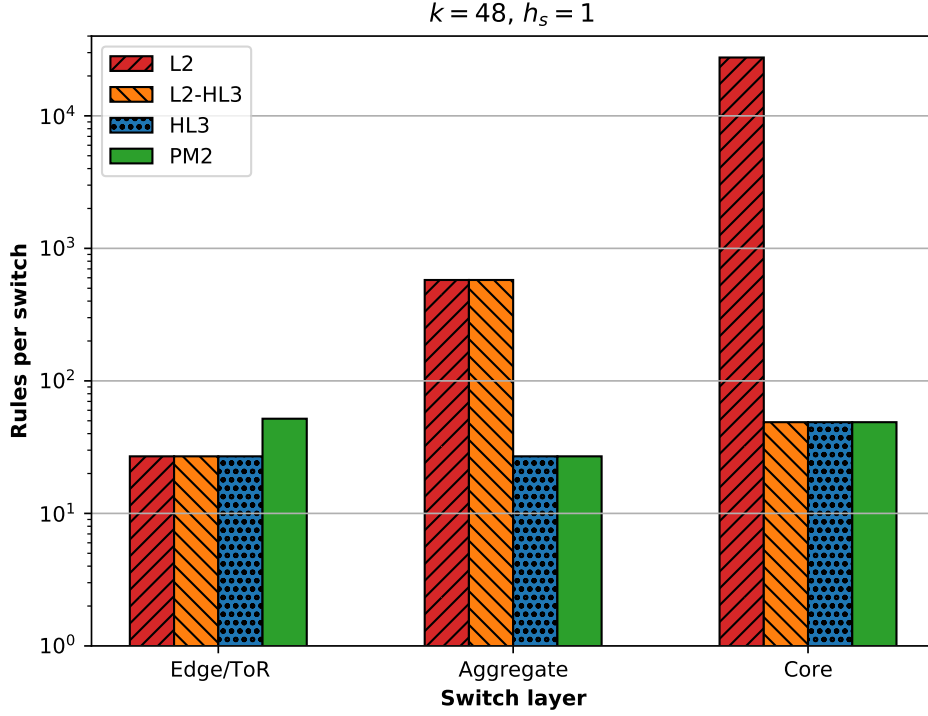


Figure 4.7: Number of rules installed per switch for each layer in a fat-tree topology with size $k = 48$ and only one host in each server ($h_s = 1$)

allows decreasing the delay introduced to flows (mainly mice) traversing the fat-tree topology network.

4.2 Rescheduling of elephants in software-defined data center networks using deep incremental learning

Recall that SDDCN multipath routing mechanisms (see Section 2.3.3) deploy a controller that dynamically reschedules paths for identified elephants while relying on a default multipath routing algorithm, like PM2 (see Section 4.1 and ECMP [7], for handling the rest of the flows (potentially, mice). These SDDCN multipath routing mechanisms depend on flow detection methods that discriminate elephants from mice, either reactively by using thresholds or proactively by incorporating ML (see NELLY, introduced in Chapter 3). However, these flow detection methods merely indicate which flows are elephants (i.e., binary classification) but do not provide specific traffic characteristics (e.g., size, rate, duration) of such flows.

Therefore, most SDDCN multipath routing mechanisms handle all the elephants flows in the same way, that is, all with the same traffic characteristics. This is not suitable for covering the broad distribution of traffic characteristics of elephant flows in DCNs [8–11]. Only FlowSeer [33] perform a multiclassification for dividing the rate of the elephants into five classes. Although a five-class classification is much better than a binary classification, finer granularity traffic characteristics are desirable for improving the decisions of the multipath routing algorithm. Moreover, FlowSeer relies on sampling-based data

collection, which increases the traffic overhead and the detection time while reducing the accuracy. As described in Section 3.3, FlowSeer reports to the controller 29% of mice as potential elephants, which can negatively affect the performance of the multipath routing algorithm.

In this section, we introduce a flow rescheduling method denominated intelligent Rescheduler of IDentified Elephants (iRIDE), which applies incremental learning at the controller-side of SDDCNs for predicting traffic characteristics of flows identified as elephants to compute and install the best path per flow across the network. Incremental learning allows iRIDE adapting to the variations in traffic characteristics and performing endless learning with limited memory resources. Quantitative evaluation demonstrates that iRIDE achieves low prediction error of flow rate and flow duration when using DNNs with regularization and dropout layers. Moreover, iRIDE enables intelligent elephant rescheduling algorithms that efficiently use the available bandwidth, generating higher throughput and shorter traffic completion time than conventional ECMP.

The remainder of this section is as follows. Section 4.2.1 introduces the architecture of iRIDE. Section 4.2.2 presents a quantitative evaluation of the prediction components in iRIDE using real data and incremental learning algorithms in both batch and incremental settings. Section 4.2.3 describes practical heuristics for implementing two rescheduling algorithms for iRIDE. Section 4.2.4 exposes the implementation of iRIDE, including the prediction and rescheduling components, for evaluating the networking performance. Section 4.2.5 discusses the networking performance results.

4.2.1 Architecture of iRIDE

Figure 4.8 introduces iRIDE, a flow rescheduling method at the controller-side of SDDCNs that applies incremental learning for predicting traffic characteristics of elephant flows to compute and install the best path per elephant flow across the network. iRIDE relays on flow detection methods, such as NELLY, for sending marked packets that identify elephant flows traversing the network.

As illustrated in Figure 4.8, iRIDE consists of five modules: *Catcher*, *Predictor*, *Rescheduler*, *Monitor*, and *Trainer*. For the sake of readability, Table 4.3 lists and describes the symbols defined in the architecture of iRIDE. The Catcher installs rules on edge switches (a.k.a. ToR) that forward the marked packets to the controller (i.e., catching rules). This installation can be conducted once the controller knows a switch belongs to the edge layer. The Catcher can receive this information from a *topology discovery* application (see Section 4.1). The catching rule can look for a predefined value in a code point header field supported by SDN switches. For example, OpenFlow switches support matching in two code point header fields. The first of these is the 6-bit DSCP field of the IPv4 header. This DSCP reserves a code point space for experimental and local usage (i.e., $****11$, where $*$ is 0 or 1). The second is the 3-bit 802.1Q PCP field of the Ethernet header. In practice, iRIDE can rely on either one of these fields, since it is improbable that a data center use both DSCP and PCP simultaneously [29].

As soon as the catching rule matches a marked packet in an edge switch, the marked packet is sent to the controller. The catcher receives the marked packet and extracts the

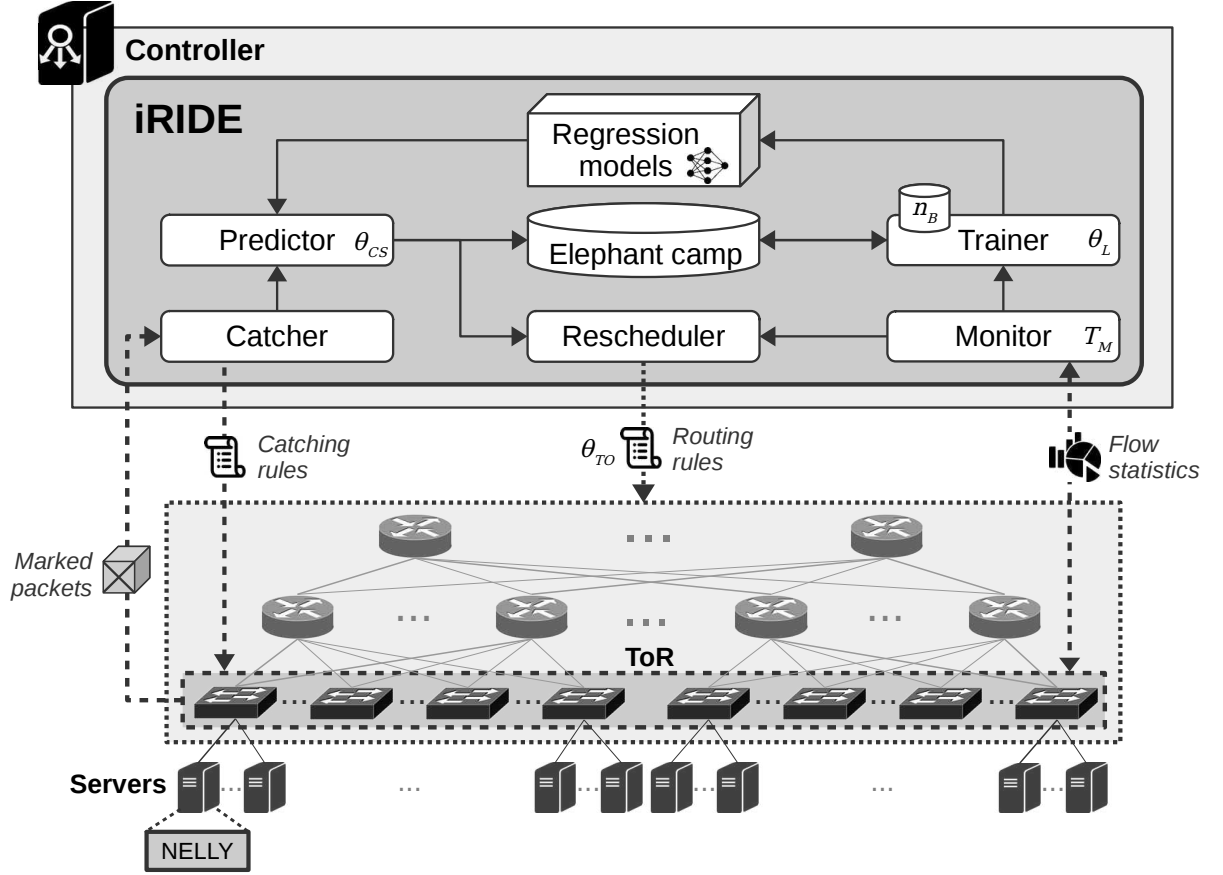


Figure 4.8: Architecture of iRIDE

Table 4.3: Symbols in the architecture of iRIDE

Symbol	Name	Description
θ_{CS}	Cold-start threshold	Value (e.g., number of training instances) above which the Predictor uses the regression models for predicting the traffic characteristics
θ_{TO}	Timeout threshold	Time limit above which switches acknowledge inactive flows as terminated for removing the corresponding routing rule
θ_L	Labeling threshold	Flow size limit below which the Trainer discards flows incorrectly classified as elephants
n_B	Mini-batch size	Number of elephant instances for training the regression models
T_M	Monitoring rate	Time interval at which the Monitor requests flow characteristics from the network

flow information from it, commonly, the header (e.g., 5-tuple) to identify the elephant flow. This marked packet might also include the size and IAT of the first N flow packets that flow detection methods, such as NELLY, usually collect for classifying flows as mice and elephants (see Chapter 3). Note that NELLY requires a small modification in the Marker

module of the Analyzer subsystem (see Section 3.1.1) to be able to communicate this extra information to iRIDE. Instead of marking the packets of flows classified as elephants, the Marker forwards the packets without changes and generates an extra marked packet for each elephant flow. Such an extra marked packet uses the same flow header (e.g., 5-tuple) and includes the size and IAT of the first N packets in its payload. In this case, the parameter M of the Marker would represent the number of extra marked packets generated for an elephant flow, thus enabling a trade-off between reliability and overhead. As M increases, the lesser the probability that the controller will miss elephant flows due to losses of extra marked packets in the SDDCN, but the more extra packets are sent to edge switches and to the controller. Once the controller has installed a higher priority routing rule for handling a specific elephant flow across the SDDCN, the subsequent extra marked packets of this flow are not required but they are still forwarded to the controller, which increases traffic overhead.

After the Catcher extracts the flow information from marked packets, it passes that data to the Predictor, which uses the regression models to predict the rate and duration of the flows identified by the marked packets. If the flow information includes the size and IAT of the first N packets, the Predictor's configuration can be extended by including a cold start threshold (θ_{CS}) that defines if estimating or predicting the flow traffic characteristics (i.e., rate and duration). For example, if the number of instances (i.e., flows) used to train the model is less than θ_{CS} , the Predictor computes the rate using the first N packets data and assigns a default value to the duration. When the number of trains reaches θ_{CS} , the Predictor starts using the regression models to predict the traffic characteristics. θ_{CS} allows warming up the regression models to avoid predictions about which they have not yet gathered sufficient information.

The Predictor stores all the flow information in the *elephant camp* (a temporal repository for elephant flows) and communicates the predicted traffic characteristics to the Rescheduler. The Rescheduler then can install specific routing rules per elephant flow across the network based on a path computed by a rescheduling algorithm that uses the predicted traffic characteristics. Section 4.2.3 describes two rescheduling algorithms that use either the predicted flow rate or flow duration to compute the path and install the routing rules for rerouting elephants in a fat-tree DCN topology.

Each routing rule includes a threshold timeout (θ_{TO}) that instructs SDN-enabled switches to remove the routing rule as soon as the corresponding flow has been inactive for θ_{TO} (i.e., the switch has not received a packet that matches the flow rule). Note that θ_{TO} is related to the flow definition in NELLY (see Section 3.1.1). Edge switches additionally include a mechanism that reports to the controller *flow statistics* of the removed routing rules due to time-out. Such a report must include the flow header, flow size, and flow duration of the routing rule. For example, OpenFlow allows inserting the flag `OFPPF_SEND_FLOW_REM` into the installed flow rules so when the OpenFlow switch removes one of them, it reports the removed flow rule to the controller, including the match header, removal cause (e.g., idle time-out), byte count, and duration in seconds. P4 enables a flexible data plane programming for easily implementing this reporting mechanism into P4 switches.

The Monitor receives the report of the removed routing rule from the edge switch

and extracts the flow statistics: flow header, flow size, and flow duration. The Monitor then communicates these flow statistics to the Trainer, which discards those flows that were incorrectly marked as elephants, that is, flows whose size (e.g., byte count) is less than an elephant size threshold (θ_L). In NELLY, θ_L represents the labeling threshold for tagging flows as either mice or elephants (see Section 3.1.2). The Trainer computes the rate and duration (*ground truth*) of the non-discarded flows (i.e., true elephants) and use them to train the regression models. Each regression model maps online features (i.e., packet header, size, and IAT of the first N packets) onto the corresponding value (i.e., rate and duration). Recall that the Predictor relies on the regression models to predict flow traffic characteristics. The Trainer avoids increasing memory consumption in iRIDE by removing all timed-out flows (both discarded and used for training) from the elephant camp. Instead of using only one elephant instance, the Trainer might hold a mini-batch of elephant instances of size n_B for training the regression models.

Since flows represent continuous and dynamic data streams, the Trainer uses an incremental learning algorithm for building the regression models. Incremental learning enables updating the regression models as the Trainer receives timed-out flows over time, rather than retraining from the beginning (see Section 2.2.1).

Note that the Rescheduler operates depending on the predicted traffic characteristics that the Predictor communicates. However, the Monitor might also support the Rescheduler by implementing a mechanism that frequently requests flow characteristics (e.g., rate) from the network. This mechanism should keep a low traffic overhead. Therefore, every T_M , the Monitor requests traffic characteristics from elephant flows from edge switches and passes that information to the Rescheduler. For example, the Monitor can compute the rate of each requested elephant flow and pass it to the Rescheduler to have an updated view of the network traffic for taking routing decisions.

4.2.2 Prediction

This section presents the evaluation of the ML modules in iRIDE (i.e., the Predictor and the Trainer) in relation to prediction accuracy and time by using real packet traces and incremental learning algorithms. We used both batch and incremental settings for evaluating these learning algorithms. Furthermore, we used the generic approach for designing ML-based solutions in networking (see Figure 2.4) to describe and conduct this evaluation: data collection, feature engineering, establishing the ground truth, model validation, and model learning.

Datasets

We reused the two datasets [262], UNI1 and UNI2, generated for evaluating NELLY (see Section 3.2.1). We selected the datasets with BinNom-header as they enabled the best performance of the majority of the algorithms (see Section 3.2.4). BinNom-header provides a total of 117 online features. Moreover, since iRIDE only reschedules flows marked as elephants, we removed those flows smaller than $\theta_L = 100$ kB from both datasets. Therefore, the newly generated UNI1 and UNI2 datasets consisted of approximately 8,500 and 20,000 flows, respectively.

The new datasets included the following flow information: start time, end time, 5-tuple header, size and IAT of the first 7 packets, as well as flow size. The start and end times allowed computing the duration of each flow. The 5-tuple header and the size and IAT of the first 7 packets represented the online features for the two regression models: flow rate and flow duration. The flow size was divided by the duration of the flow to compute its rate. The rate and duration of each flow represent the target value to predict and provide the ground truth for learning and validating the corresponding regression model.

To complement feature engineering, we converted to negative one (-1) the binary zero (0) values from the online features corresponding to the 5-tuple header since some ML techniques (particularly, NNs) perform better when the input values are centered around zero rather than ranging between 0 and 1 [283, 284]. We also transformed the numeric values in the online features (i.e., size and IAT of first 7 packets) using different feature scaling methods, as discussed later in the experiment setup.

Accuracy metrics

We used two accuracy metrics commonly used in the literature to report the performance of regression models: the coefficient of determination (R^2) and the Root Mean Square Error (RMSE). R^2 provides a goodness-of-fit score that measures how well the regression models fit the observed data (i.e., ground truth) [285]. We rely on the common R^2 definition that uses the first equation of Kvålseth (see Equation 4.7), which provides scores usually between 0 and 1. As the R^2 score gets closer to 1, the better the regression predictions ($\hat{y}$) approximate to the observed values (y). Note that R^2 scores below 0 might occur, which represent that the regression fit performs worse than a horizontal line [286]. R^2 enables comparisons across different types of data as it does not depend on the scale of the values. However, R^2 by itself cannot indicate if a regression model is adequate. Therefore, we used RMSE to complement the R^2 scores.

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}, \text{ where } \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i \quad (4.7)$$

As shown in Equation 4.8, RMSE measures the differences between the values predicted by the regression models ($\hat{y}$) and the observed values (y). RMSE provides a non-negative value that expresses a higher accuracy as the error gets smaller. We preferred RMSE over other similar error metrics often used to gauge the accuracy of regression models, such as MAE and MSE [286]. The three error metrics disregard the direction of under- and over-estimations in the predictions. Moreover, unlike R^2 , these metrics depend on the scale of the data so they cannot be used to compare the accuracy of regression models working with different types of data. However, MSE and RMSE are more useful than MAE for heavily penalizing large errors and outliers. Additionally, in contrast to MSE, RMSE expresses the standard deviation of the error, which is in the same units as the quantity being predicted. Our RMSE results display Megabits per second (Mbps) for

flow rate and seconds (s) for flow duration.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}} \quad (4.8)$$

At the end of the evaluation process, we also analyzed the residual plots of the selected regression models to check no bias nor problematic patterns exist in the residuals.

Experiment setup

Incremental learning algorithms are commonly evaluated using the interleaved test-then-train approach [266]. However, for selecting the data preprocessing techniques (i.e., feature scaling, target transformation, imbalance correction) and the regression model (i.e., learning algorithm and hyperparameter tuning), we followed the common 60/20/20% batch decomposition for dividing the datasets into training, validation, and test sets, respectively (see Section 2.2.1). This batch decomposition allows using the validation set for selecting the best model and evaluate it on the test set to get an unbiased estimate of the model’s performance [287]. Since our datasets are in the order of the tens of thousands, we applied the batch holdout method for validation and testing. Note the result from the test set represents the optimal performance to which the model would tend when using an incremental evaluation method (either holdout or interleaved test-then-train) [288].

Feature scaling is an essential data preprocessing step for ML algorithms that compute a distance function between input features, such as kNN [283]. Distance functions (e.g., Euclidean distance) depend heavily on the variability of the features and are biased towards numerically larger values. For example, the online features in our datasets contain binary values $[-1, 1]$ for the 5-tuple packet header and numeric values for the size and IAT of the first 7 packets. The packet size ranges from 60 bytes up to 1522 bytes, whereas the packet IAT can go from a microsecond up to five million of microseconds (i.e., $\theta_{TO} = 5\text{s}$). Therefore, our non-scaled numeric values (particularly, large IATs) could bias ML algorithms based on distance functions. Moreover, feature scaling for ML algorithms using gradient descent, such as NNs, might not be strictly required but can make their training to converge faster and with less chances of sticking in a local optima [283]. Conversely, DT algorithms (e.g., Hoeffding trees) are insensitive to feature scaling.

We analyzed different feature scaling methods on the numeric values of the online features [289]. The *min-max zero-center scaler* (see Equation 4.9) is a simple method that rescales the features to the range $[-1, 1]$ by using the minimum and maximum values of each feature. In our datasets, we defined 60 and 1522 (bytes) as the minimum and maximum values for the packet size, as well as one and five million (microseconds) for the packet IAT. The *standard scaler*, also called Z-score normalization (see Equation 4.10), removes the mean and scales the values to unit variance. The *robust scaler* removes the median and scales the data according to the interquartile range (i.e., range between the first and third quartiles). The *quantile transformer* provides a non-linear transformation that maps the features to follow either a *uniform* or a *normal* distribution. The *power transformer* applies either the *Box-Cox* transform (see Equation 4.11) or the *Yeo-Johnson* transform (see Equation 4.12). Both non-linear transformations use the maximum like-

lihood to estimate the optimal parameter λ that maps data to follow a Gaussian-like distribution. The *normalizer* performs the Euclidean norm (a.k.a. L2 norm) to scale features individually to unit form.

$$X'_i = \frac{2 \cdot (X_i - \min(X))}{\max(X) - \min(X)} - 1 \quad (4.9)$$

$$X'_i = \frac{X_i - \bar{X}}{\sigma}, \text{ where } \bar{X} \text{ is the mean and } \sigma \text{ is the standard deviation} \quad (4.10)$$

$$X_i^{(\lambda)} = \begin{cases} \frac{X_i^\lambda - 1}{\lambda} & \lambda \neq 0 \\ \ln(X_i) & \lambda = 0 \end{cases} \quad (4.11)$$

$$X_i^{(\lambda)} = \begin{cases} \frac{(X_i+1)^\lambda - 1}{\lambda} & \lambda \neq 0, X_i \geq 0 \\ \ln(X_i + 1) & \lambda = 0, X_i \geq 0 \\ -\frac{(-X_i+1)^{(2-\lambda)} - 1}{2-\lambda} & \lambda \neq 2, X_i < 0 \\ -\ln(-X_i + 1) & \lambda = 2, X_i < 0 \end{cases} \quad (4.12)$$

We also analyzed different target variable transformations by applying some of these feature scaling methods: *quantile-uniform*, *quantile-normal*, *Box-Cox*, and *Yeo-Jhonson*. Figures 4.9 and 4.10 depict these transformations on the two target variables, flow rate and flow duration, respectively, in both datasets, UNI1 and UNI2. Note both target variables present an imbalanced domain when no transformation has been applied to the data. Therefore, we further included the Synthetic Minority Over-sampling technique for regression with Gaussian Noise (SMOGN) [290] into our experiments for analyzing imbalance correction (i.e., under-sampling and over-sampling) in our datasets. SMOGN combines three methods proposed for addressing regression imbalance: random under-sampling [291], Synthetic Minority Over-sampling TEchnique for Regression (SMOTER) [292], and introduction of Gaussian Noise (GN) [293]. Random under-sampling enables removing less interesting instances, whereas SMOTER and GN introduction generate new synthetic data from close and distant instances, respectively.

Finally, we used the interleaved test-then-train approach [266] for evaluating the selected regression model in an incremental setting. Similar to NELLY (see Section 3.2.3, the prediction of flow characteristics (i.e., flow rate and flow duration) takes place at the flow start time, while retraining the regression models at the flow end time. Moreover, we analyzed different mini-batch sizes (n_B) for training the model incrementally.

Performance analysis

To determine the regression model with the best accuracy performance, first, we used the batch decomposition to evaluate different learning algorithms that can operate in an incremental approach. Three regression algorithms from scikit-learn [294]: Stochastic Gradient Descent (SGD) for Ordinary Least Squares (OLS), Passive-Aggressive (PA), and Multi-Layer Perceptron (MLP) (i.e., NN); plus four regression algorithms from scikit-

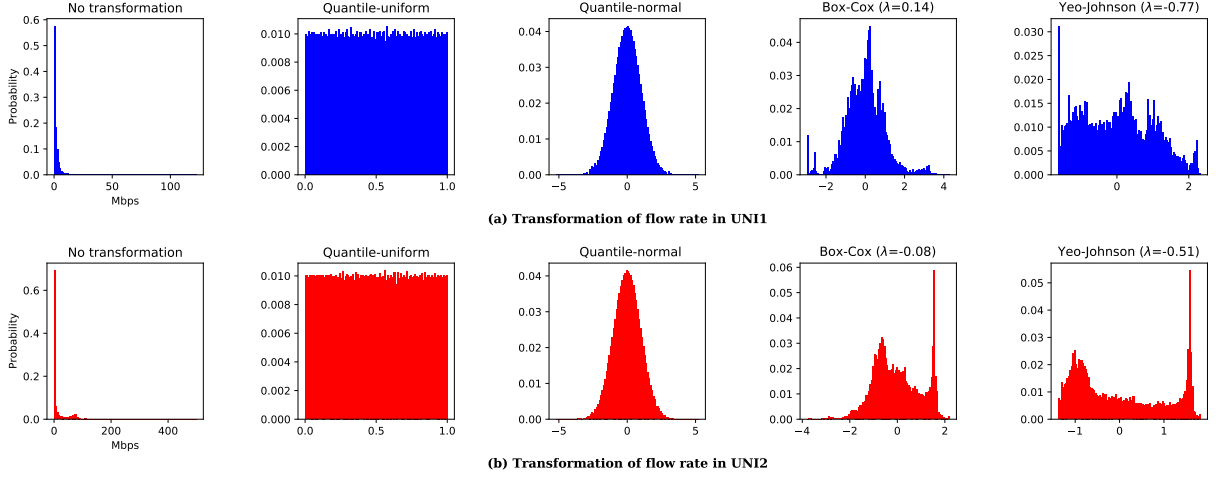


Figure 4.9: Transformations of flow rate in UNI1 and UNI2

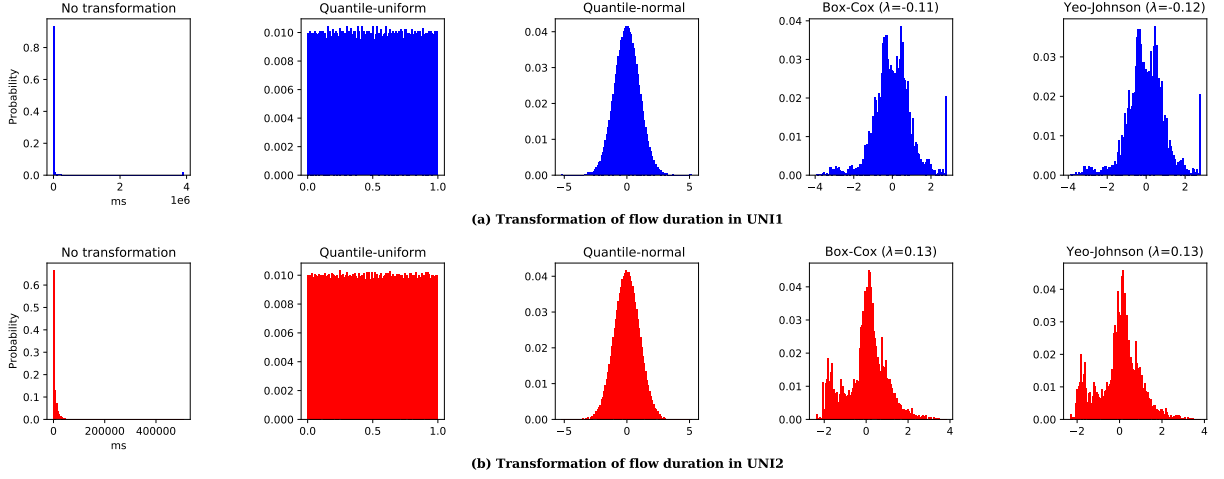


Figure 4.10: Transformations of flow duration in UNI1 and UNI2

multiflow [295]: k -Nearest Neighbors (kNN), Hoeffding tree, Hoeffding Adaptive Tree (HAT), and Adaptive Random Forest (ARF). The algorithms were executed with their default settings and without previous model initialization. We only modified the MLP structure to two hidden layers: the first one with 175 units and the second one with 117 units (i.e., respectively, 3/2 and 1 times the number of input units, which is the number of online features). Moreover, we evaluated all the feature scaling methods and target transformations (both explained in the experiment setup) for each learning algorithm.

For the sake of brevity, Table 4.4 presents the learning algorithm and data preprocessing techniques that achieved the best accuracy performance (i.e., the highest R^2 and the lowest RMSE) for predicting each target variable (i.e., flow rate and flow duration) in the training and validation sets of UNI1 and UNI2. The results show that MLP achieved the best accuracy for predicting the two target variables in both evaluation sets of UNI1 and UNI2, though the associated feature scaling methods and target transformations varied. Note Table 4.4 presents the second best accuracy performance for predicting both target variables in the validation set of UNI1. Although kNN achieved the best accuracy in the validation set, its accuracy results performed really bad in the training set. In fact,

kNN presented negative R^2 values in the training set, representing that the regression fit performed worse than a horizontal line. Therefore, the kNN results in the validation set of UNI1 were not reliable.

The results in Table 4.4 also depict that MLP fits better the rate (i.e., higher R^2) in UNI1 than in UNI2, achieving a lower error (i.e., RMSE) for the first one. In contrast, although MLP fits better the duration in UNI1 than in UNI2, the error in the first one is higher than in the other. This is because UNI1 presents some flow duration values that greatly differ from the others, which are highly penalized by the RMSE metric. Therefore, hereinafter, we make decisions based on RMSE results.

Table 4.4: Learning algorithm and data preprocessing with the best accuracy performance in a batch learning setting

Dataset	Target variable	Evaluation set	Learning algorithm	Feature scaling	Target transform	R^2	RMSE*
UNI1	Rate	Train	MLP	Box-Cox	None	0.9417	1.42
		Validation [†]	MLP	Robust	Yeo-Johnson	0.3123	5.61
	Duration	Train	MLP	Quantile-normal	Quantile-normal	0.9678	85.64
		Validation [†]	MLP	Min-max	None	0.9119	197.92
UNI2	Rate	Train	MLP	Quantile-normal	Quantile-normal	0.5424	23.07
		Validation	MLP	Robust	None	0.4367	24.81
	Duration	Train	MLP	Quantile-normal	Box-Cox	0.7378	14.28
		Validation	MLP	Min-max	Quantile-normal	0.5215	19.34

*RMSE units depend on target variable: Megabits per second (Mbps) for rate and seconds (s) for duration

[†]Showing the second best result, disregarding kNN

We further applied SMOGN to our datasets for evaluating the effect of imbalance correction (i.e., under-sampling and over-sampling) on the accuracy performance. In this experiment, we only used MLP as the learning algorithm as it performed the best in the results depicted in Table 4.4. However, we kept analyzing all the feature scaling methods and target transformations. MLP preserved the same NN structure (i.e., two hidden layers of 175 and 117 units), as well as the rest of default settings and no previous model initialization.

Table 4.5 shows the data preprocessing techniques (imbalance correction, feature scaling, and target transformation) with which MLP achieved the best accuracy performance in the validation set for predicting both target variables in UNI1 and UNI2. To summarize, for predicting the rate in UNI1, MLP achieved the best accuracy by using the under-sampling imbalance corrective, the robust feature scaler, and the Yeo-Johnson target transformation. Whereas, for the duration in UNI1, MLP performed the best by using over-sampling, the min-max scaler, and no target transformation. On the other hand, for predicting both target variables in UNI2, MLP required no imbalance corrective nor target transformation but used the feature normalizer to achieve the best performance. However, many data preprocessing techniques that enabled the best accuracy have been specially designed for batch learning (i.e., operate with the whole training set), including under-sampling and over-sampling, robust scaler, normalizer, and Yeo-Johnson transformer. Implementing these techniques in an incremental setting might require huge memory resources and it is still an open research challenge [296].

Table 4.5: Data preprocessing with the best accuracy performance for MLP in a batch learning setting

Dataset	Target variable	Imbalance corrective	Feature scaling	Target transform	Training		Validation	
					R^2	RMSE*	R^2	RMSE*
UNI1	Rate	None [†]	Robust	None	0.9722	1.12	0.4849	4.27
		Under-sampling	Robust	Yeo-Johnson	0.9745	1.62	0.4944	4.23
		Over-sampling	Yeo-Johnson	None	0.9801	1.75	0.4649	4.35
	Duration	None [†]	Standard	None	0.9979	24.82	0.8899	192.24
		Under-sampling	Min-max	None	0.9976	44.89	0.8885	193.46
		Over-sampling	Min-max	None	0.9978	49.58	0.9023	181.14
UNI2	Rate	None [†]	Normalizer	None	0.5167	23.45	0.4308	25.28
		Under-sampling	Normalizer	Yeo-Johnson	0.7081	24.72	0.3971	26.01
		Over-sampling	Robust	None	0.7532	23.09	0.4139	25.65
	Duration	None [†]	Normalizer	None	0.6335	17.01	0.4936	21.77
		Under-sampling	Min-max	None	0.6476	22.05	0.4647	22.38
		Over-sampling	Min-max	Yeo-Johnson	0.6786	22.15	0.4522	22.64

*RMSE units depend on target variable: Megabits per second (Mbps) for rate and seconds (s) for duration

[†]Original dataset, with no under-sampling nor over-sampling

Therefore, we evaluated a *feasible incremental* approach by combining MLP with those data preprocessing techniques that can operate in an incremental setting, namely, min-max feature scaler, no target transformation, and no imbalance correction. Note the min-max scaler only requires defining the minimum and maximum values of the features. In our datasets, 60 and 1522 (bytes) as the minimum and maximum values for the packet size, as well as one and five million (microseconds) for the packet IAT. Table 4.6 presents the accuracy performance of MLP using the feasible incremental data preprocessing techniques. As expected, MLP achieved worse validation errors in comparison with using the best data preprocessing techniques (see Table 4.5). RMSE in the validation set increased by 0.6 (Mbps) and 0.3 (seconds) when predicting the flow rate in both datasets and the flow duration in UNI2, respectively. Moreover, the flow duration prediction in UNI1 represents the worst case, incrementing the validation RMSE by 21 (seconds). Nevertheless, MLP achieved similar RMSE values in the training set, opening an opportunity to improve the validation error.

Aiming at improving the accuracy performance on the validation set, first, we focused on building NN structures that reduce the training errors by using Tensorflow [297] and Keras [298]. We set typical hyperparameters for DNNs [299–301], including the Rectified Linear Unit (ReLU) [302] as the activation function in the units of the hidden layers, He normal [303] to initialize the NN weights, Adam optimization [304] to improve SGD, and the default mini-batch size of 32 instances per gradient update [305]. For tuning the NN structure (see Figure 4.11), we varied the number of hidden layers (L_h) from one up to ten in steps of one (i.e., $L_h \in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$), and the number of units per each hidden layer ($n^{[l_h]} \forall l_h \in L_H$) from 15 up to 600 in steps of specific multiples of 15 (i.e., $n^{[l_h]} \in [15, 30, 60, 120, 240, 360, 480, 600]$). Note the data preprocessing techniques remained as the feasible incremental approach (i.e., min-max feature scaler, no target transformation, and no imbalance correction).

Table 4.6: Accuracy performance of MLP using feasible incremental data preprocessing techniques in a batch learning setting

Dataset	Target variable	Training		Validation	
		R^2	RMSE*	R^2	RMSE*
UNI1	Rate	0.9532	1.45	0.3259	4.88
	Duration	0.9969	29.77	0.8778	202.57
UNI2	Rate	0.4990	23.88	0.4032	25.88
	Duration	0.5832	18.14	0.4781	22.09

*RMSE units depend on target variable: Megabits per second (Mbps) for rate and seconds (s) for duration

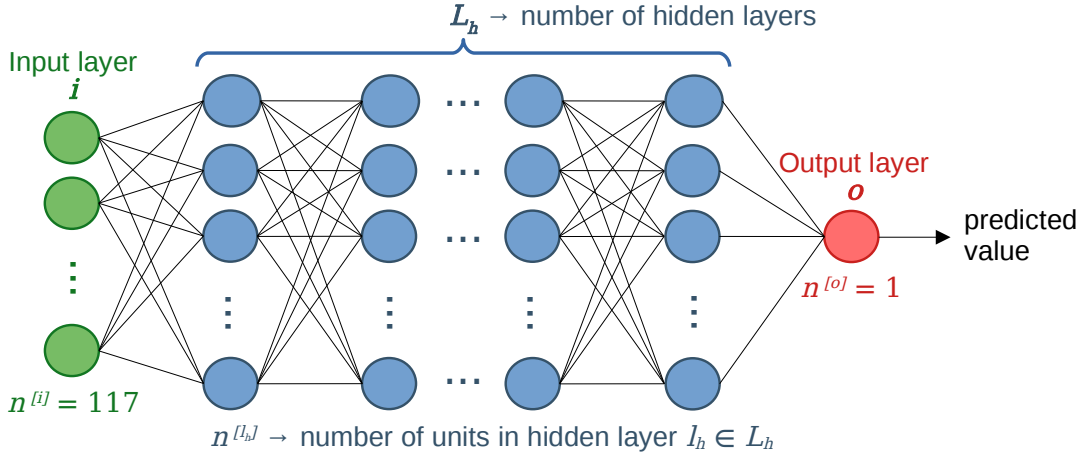


Figure 4.11: NN structure

Table 4.7 describes the NN structures that achieved the lowest RMSE values in the training sets of both datasets, UNI1 and UNI2, for predicting the two target variables. In general, the results show that for reducing the training errors, NN requires deep structures (i.e., DNN) from five up to nine hidden layers (L_h) and from 360 up to 600 units per each hidden layer ($n^{[l_h]}$). In comparison to the results of MLP using the feasible incremental data preprocessing techniques (see Table 4.6, the reduction of the training RMSE was good for predicting the rate and duration in UNI1 ($\sim 33\%$ and $\sim 24\%$, respectively), moderate for the duration in UNI2 ($\sim 11\%$), and minimal for the rate in UNI2 ($\sim 4.6\%$).

The problem about focusing on reducing the training error is that it causes overfitting [306], which means that the model would not be able to generalize in unseen data (i.e., poor accuracy performance on the validation and test sets). To tackle this problem, we incorporated a combination of two regularization methods to reduce the error on the validation sets. First, L2 regularization [307] (see Equation 4.13), for which we varied the regularization parameter (λ) from 0 up to 0.1 in steps of 1 thousand (i.e., $\lambda \in [0, 10^{-5}, 10^{-4}, 10^{-3}, 0.01, 0.1]$). Second, a dropout regularization layer [308] for each hidden layer, for which we varied the dropout rate from 0% up to 75% in steps of 25%

Table 4.7: DNN structures with the lowest training errors in a batch learning setting

Dataset	Target variable	Hidden layers (L_h)	Hidden units ($n^{[l_h]}$)	Training RMSE*
UNI1	Rate	5	480	0.97
	Duration	8	360	22.53
UNI2	Rate	8	600	22.78
	Duration	9	480	16.22

*RMSE units depend on target variable: Megabits per second (Mbps) for rate and seconds (s) for duration

(i.e., $[0, 25, 50, 75]\%$).

$$L2 = \frac{\lambda}{2m} \|w^{[l]}\|_F^2, \text{ where } \lambda \text{ is the regularization parameter, } m \text{ is the number of training instances, and } \|w^{[l]}\|_F^2 \text{ is the squared Frobenius norm of the weights in layer } l \quad (4.13)$$

Table 4.8 depicts the DNN structures and regularization methods (i.e., L2 and dropout layers) that achieved the lowest RMSE values in the validation sets of both datasets, UNI1 and UNI2, for predicting the two target variables. For predicting the rate, the results show the DNN structures achieved the lowest validation errors by using a combination of the two regularization methods, with $\lambda = 0.1$ and a dropout rate of 50% for UNI1, and $\lambda = 10^{-4}$ and a dropout rate of 25% for UNI2. Whereas, for predicting the duration, using only L2 regularization with $\lambda = 10^{-5}$ provided the best validation results. In comparison to the results of MLP with the feasible incremental data preprocessing techniques (see Table 4.6), the trained DNNs reduced the validation RMSE values, except for the rate in UNI2, where no improvement nor degradation was observed. In fact, in comparison to the results of MLP with the best data preprocessing techniques (see Table 4.5), the DNNs reduced the validation errors for predicting the duration in UNI1 and UNI2. Although the validation RMSE for predicting the rate in UNI1 is higher for the trained DNN than for the best MLP, this error is still lower than for MLP with the feasible incremental data preprocessing techniques.

To conclude the evaluation in a batch learning setting, we evaluated the obtained DNN regression models in the test set of both datasets, UNI1 and UNI2, for reporting an unbiased estimate of the model's accuracy performance when predicting the flow rate and the flow duration (see Table 4.8).

Continuing our prediction evaluation, we used the interleaved test-then-train approach for evaluating the tuned DNN regression models in an incremental learning setting. Figure 4.12 shows the mean RMSE over a sliding window of approximately 10% of the instances (1000 in UNI1 and 2000 in UNI2). Note that we also analyzed mini-batch sizes (n_B) other than one, varying the number of training instances from 8 up to 256 in dou-

Table 4.8: DNN structures and regularization methods with the lowest validation errors in a batch learning setting

Dataset	Target variable	Structure*	Regularization parameter (λ)	Dropout rate	Training RMSE [†]	Validation RMSE [†]	Testing RMSE [†]
UNI1	Rate	$L_h = 5, n^{[l_h]} = 480$	0.1	50%	4.61	4.34	5.42
	Duration	$L_h = 8, n^{[l_h]} = 360$	10^{-5}	0%	30.14	173.97	126.89
UNI2	Rate	$L_h = 8, n^{[l_h]} = 600$	10^{-4}	25%	25.51	25.88	27.08
	Duration	$L_h = 8, n^{[l_h]} = 360$	10^{-5}	0%	17.51	21.52	18.55

* Defines the number of hidden layers (L_h) and the number of units per each hidden layer ($n^{[l_h]}$)

[†] RMSE units depend on target variable: Megabits per second (Mbps) for rate and seconds (s) for duration

bling steps (i.e., $n_B \in [1, 8, 16, 32, 64, 128, 256]$). Moreover, the reference dotted-red line (Ref.) in the figure represents the RMSE values reported for the test sets in the batch learning evaluation (see Table 4.8). The results show, first, the DNN regression models incrementally tend to the test errors from the batch learning evaluation, except for the duration in UNI2. Second, the models suffer from higher error as new traffic characteristics appear in the data. However, the DNN regression models incrementally adapt to new traffic characteristics, lowering the RMSE values back. Third, using a large n_B generally provides no significant improvement over time. In fact, using $n_B = 256$ for predicting the duration in UNI2 might cause overfitting, producing a higher error than using a smaller n_B . This is not the case for the duration in UNI1, which greatly benefits from using $n_B > 1$ to reduce the negative impact of the flow duration outliers on the prediction error.

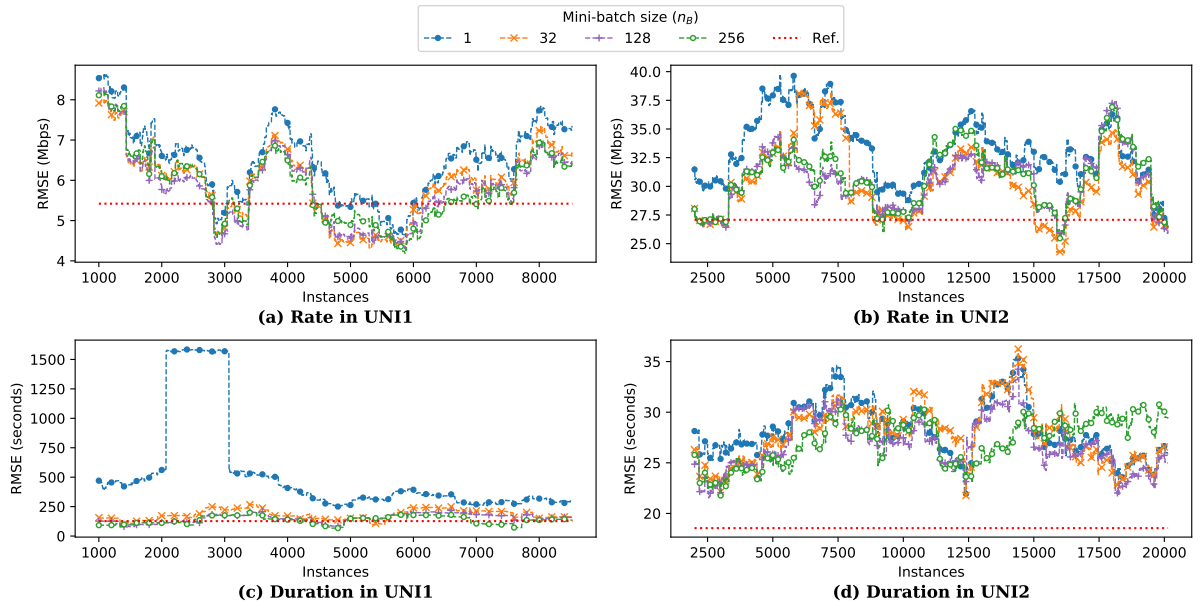


Figure 4.12: Mean RMSE over a sliding window of instances for DNN regression models with different mini-batch sizes (n_B) in an incremental learning setting

Figure 4.13 corroborates the impact of n_B on the prediction errors by presenting the mean RMSE over different values of n_B for the tuned DNN regression models. Similarly, the dotted-striped-red bar represents the RMSE values reported for the test sets in the

batch learning evaluation (see Table 4.8). The results show that using $n_B > 1$ for predicting the rate in both UNI1 and UNI2 reduces RMSE by a small amount (maximum 9.5%), achieving a minimum value that is around 11% over the test errors from the batch learning evaluation. Note the RMSE reduction is less significant for $n_B > 32$. Regarding the duration in UNI1, using $n_B > 1$ greatly reduces the prediction error by a maximum of 79%, achieving the lowest RMSE when using $n_B = 256$, which is only 7% over the test error from the batch learning evaluation. Similarly, the error reduction is minor for $n_B > 32$. In contrast, for predicting the duration in UNI2, the results show that using $n_B > 1$ provides no perceptible RMSE reduction. In fact, when $n_B = 8$, the RMSE grows up to 27% over the rest of n_B values. Note the lowest RMSE is 44% over the test error from the batch learning evaluation.

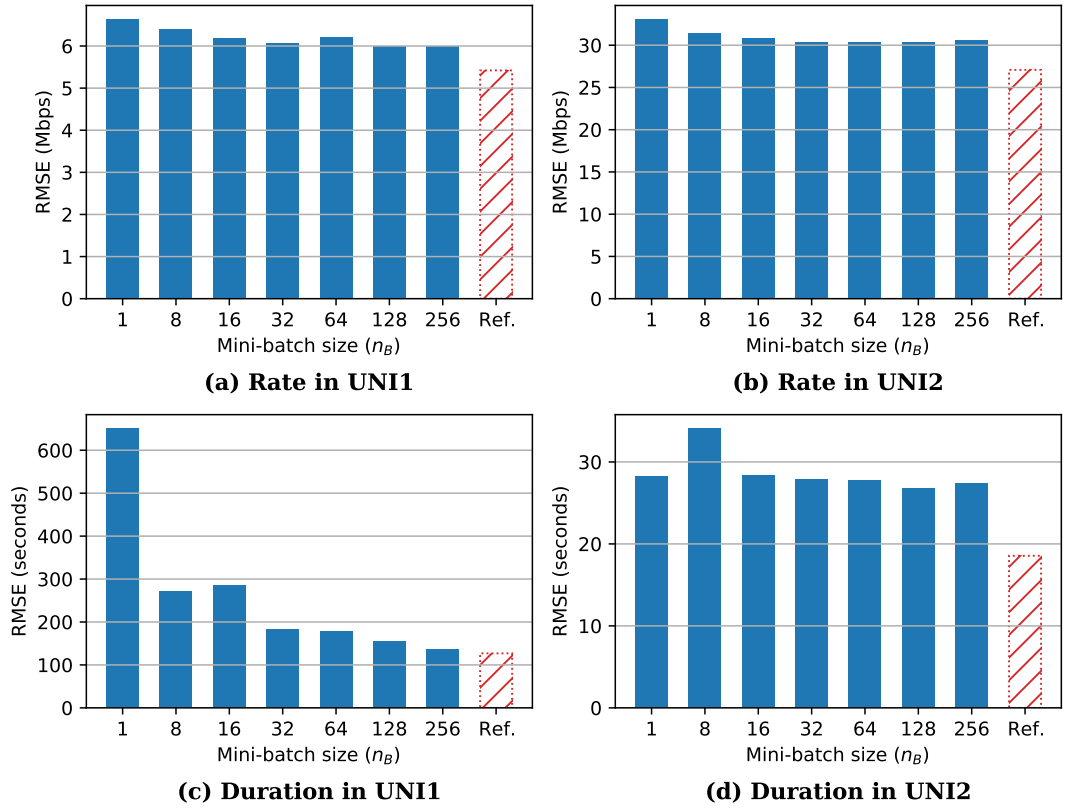


Figure 4.13: Mean RMSE over mini-batch sizes (n_B) for DNN regression models in an incremental learning setting

Lastly, Figure 4.14 depicts the mean prediction time per flow over different values of n_B for the tuned DNN regression models. Note that the time for predicting the two target variables, rate and duration, in both datasets UNI1 and UNI2, achieves the minimum value when $n_B = 1$. This prediction time per flow grows as n_B increases, achieving a maximum value when $n_B = 32$, which is between 1.8 and 3.4 milliseconds (ms) over the lowest prediction time. Based on the results of both the prediction time per flow and the mean RMSE (see Figure 4.13), we recommend using only one training instance at a time (i.e., $n_B = 1$) when predicting the rate in UNI1 and UNI2 as the error reduction is small (up to 9.5%) in comparison to the increment of the prediction time (> 2 ms) when using $n_B > 1$. Similarly, stick to $n_B = 1$ when predicting the duration in UNI2 since no error

reduction is perceptible when using $n_B > 1$ while the prediction time does increment by at least 2 ms. In contrast, for predicting the duration in UNI1, we recommend using $n_B = 256$ because the error is greatly reduced (up to 79%) while generating a prediction time per flow that is only 0.7 ms above the prediction time for $n_B = 8$.

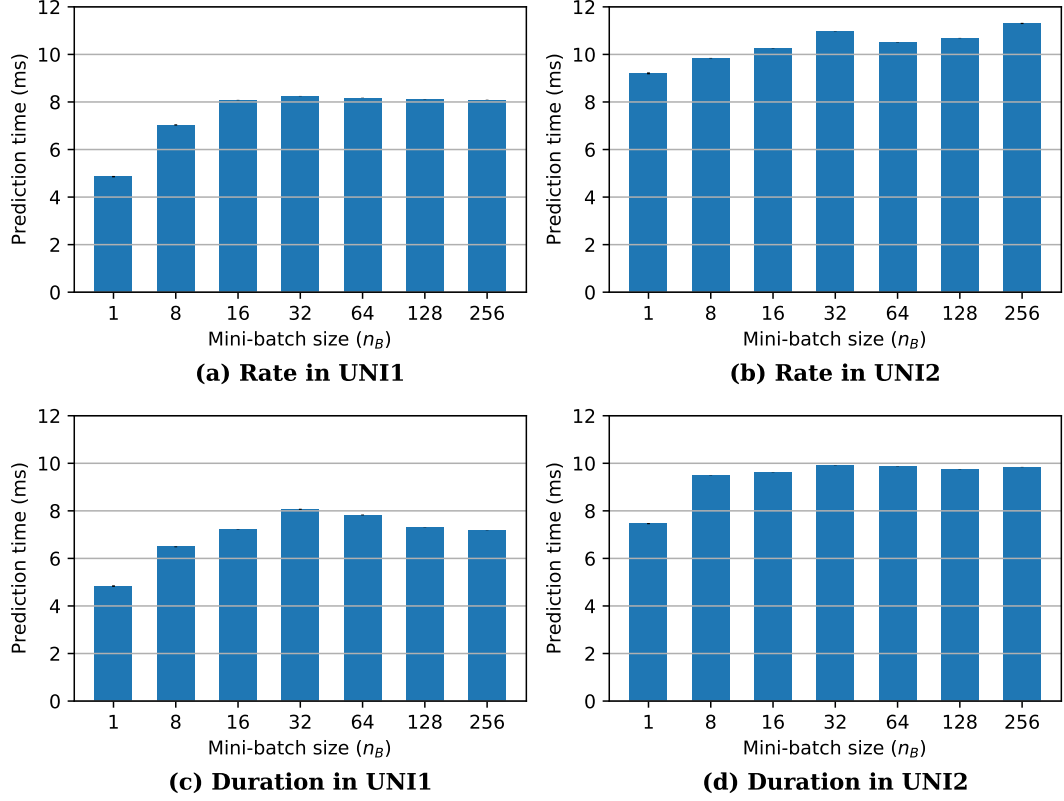


Figure 4.14: Mean prediction time per flow over mini-batch sizes (n_B) for DNN regression models in an incremental learning setting

4.2.3 Rescheduling

As described in Section 4.8, the Rescheduler module of iRIDE implements a rescheduling algorithm that uses the predicted traffic characteristics, reported by the Predictor, to compute a path and install specific routing rules per elephant flow across the network. The problem about finding the path for different flows while not exceeding the bandwidth capacity of any link is known as Multi-Commodity Flow, which is NP-complete [26]. To the best of our knowledge, no polynomial time algorithm exists for simultaneous flow routing in realistic DCN topologies, such as the 3-tier fat-tree topology (i.e., 5-stage Clos network) [26]. Therefore, this section describes practical heuristics for implementing two rescheduling algorithms that consider the fat-tree DCN topology, as seminal related works have done (e.g., simulated annealing [26] and increasing first-fit [29]).

Least Congested (LC) path

In a fat-tree topology, multiple equal-cost paths exist between any pair of hosts not connected to the same edge switch. When the Rescheduler receives a flow identified as

elephant, the Least Congested (LC) path algorithm linearly searches all the paths between the source and destination hosts to select the one whose link components carry the less traffic load. Note that only one path exists if the pair of hosts are connected to the same edge switch. The Rescheduler then places the flow on the selected path. First, the Rescheduler uses the predicted flow rate, reported by the Predictor, to reserve the bandwidth capacity for the flow on the links corresponding to the selected path. Second, the Rescheduler installs routing rules for bottom-up communication (i.e., from edge to aggregation and from aggregation to core) in the corresponding edge and aggregation switches. The top-down communication (i.e., from core to aggregation, from aggregation to edge, and from edge to destination host) relies on the default routing algorithm, such as PM2 (see Section 4.1). The Rescheduler then maintains the reserved bandwidth capacity for every link in the network to determine which paths carry the less traffic load for placing new flows identified as elephants. When receiving the notification from the Monitor that a flow has expired (i.e., a flow has been inactive for θ_{TO}), the Rescheduler clears the corresponding reservations of bandwidth capacity.

Worst-Fit Badwidth-Time-Fit (WF+BTF)

In contrast to LC, the Worst-Fit Badwidth-Time-Fit (WF+BTF) algorithm uses the two predicted traffic characteristics reported by the Predictor, namely, flow rate and flow duration. To do so, WF+BTF operates in two steps. First, when the Rescheduler receives a flow identified as elephant, WF+BTF linearly searches all the paths to find the one with the less traffic load whose link components can all accommodate the predicted rate of that flow. Note that this corresponds to the Worst-Fit (WF) algorithm from the bin packing problem [309]; WF places an item (flow) in a feasible bin (path) with most free space (bandwidth). When the network traffic load is light, finding such a path that can accommodate the predicted flow rate is likely to be easy. However, as the network traffic load grows and links become congested, WF does not guarantee that all flows identified as elephants will be placed in a path. When no path among the many existing ones between the pair of hosts can accommodate the predicted flow rate, WF+BTF moves to the second step, the Badwidth-Time-Fit (BTF) algorithm. BTF searches for the path with the maximum harmonic mean (see Equation 4.14) between the relative scores of the available bandwidth in the path (b_p) and the time the path would take to fit the predicted rate (t_p). BTF computes the relative scores b_p and t_p for each path by comparing the corresponding values of the path to the best values (i.e., the maximum free bandwidth and the minimum time to fit, respectively) among the many possible paths between the pair of hosts. Note that the time to fit the predicted flow rate is computed using the predicted duration of the flows.

$$H = \frac{2 \times b_p \times t_p}{b_p + t_p}, \text{ where } b_p = \frac{\text{available bandwidth in path}}{\text{maximum available bandwidth among paths}} \quad (4.14)$$

$$t_p = \frac{\text{minimum time to fit among paths}}{\text{time to fit in path}}$$

4.2.4 Implementation

As depicted in Figure 4.15, we reused the Mininet VM deployed for PM2 (see Section 4.1.3) for running our custom fat-tree topology to evaluate iRIDE. However, instead of executing a simple ping test between the hosts as in PM2, we installed the traffic generator Tcpreplay 4.3.4 [310] on the Mininet VM for the hosts to inject real traffic into the network. Tcpreplay supports replaying network traffic captured in the format of PCAP files. Moreover, we installed Ifstat 1.1 [311] on the Mininet VM for capturing the activity of the switch interfaces in our fat-tree emulated network. Ifstat reports interface statistics such as the incoming and outgoing bandwidth traffic. Note that all the hosts in our fat-tree emulated network concurrently execute a Tcpreplay process for generating traffic, which requires more resources (around a logical processor per process) than a simple ping test. Therefore, we redeployed our Mininet VM on to a more powerful server at LRC than the one used for the proof of concept of PM2. The new server also runs Debian 8.11 server but in a 2.00 GHz Intel Xeon Processor ES-2660 machine, with 28 physical cores, 56 logical cores, and 226 GB RAM. Then, we increased the resources of the Mininet VM to 18 logical cores and 100 GB RAM.

Similar to PM2, we installed Ryu 4.34 [66] on the new server for deploying the OpenFlow controller that manages the OpenFlow switches. Furthermore, we installed Tensorflow 2.3.0 [297] and Keras 1.1.2 [298] for training (incrementally) and inference of DNNs. Using the Python libraries from Ryu, Tensorflow, and Keras, we developed a network application that implements the five modules of iRIDE: Catcher, Predictor, Rescheduler, Monitor, and Trainer. The Catcher uses the value 001111 in the 6-bit DSCP field of the IPv4 header for installing the catching rules that match and forward to the controller the marked packets reporting the identified elephant flows. The Predictor implements no cold start threshold ($\theta_{CS} = 0$) for predicting the flow traffic characteristics. The Rescheduler sets the timeout threshold $\theta_{TO} = 5$ seconds for instructing OpenFlow switches to remove idle rules installed for routing identified elephant flows. Moreover, the Rescheduler inserts the OpenFlow flag OFPFF_SEND_FLOW_REM into the routing rules installed on the edge switches that report the elephant flows (i.e., forward the marked packets). That way, when a rule times out, the OpenFlow switch removes it and reports the flow rule statistics to the controller. The Monitor performs no frequent requesting of flow characteristics from the network ($T_M = \infty$). The Trainer sets the labeling threshold $\theta_L = 100$ kB for filtering out flows incorrectly marked as elephants. In addition, the Trainer implements a mini-batch size $n_B = 1$. Finally, we used the DNN structures and regularization methods from Table 4.8 for implementing the incremental regression models that the Trainer builds and the Predictor applies for inference. Our Ryu network application that implements iRIDE is available in [282]. We executed our iRIDE network application on the Ryu SDN framework.

Each host in our fat-tree emulated network runs a Tcpreplay process for replaying the traffic from a PCAP file. Figure 4.16 describes the process we followed to build a PCAP file for each host from public real packet traces. We used the Scapy 2.4.5 Python library to develop a PCAP parser that hashes the MAC and IP addresses, adds payload, and generates marked packets. Note the two real packet traces [255], UNI1 and UNI2, consist

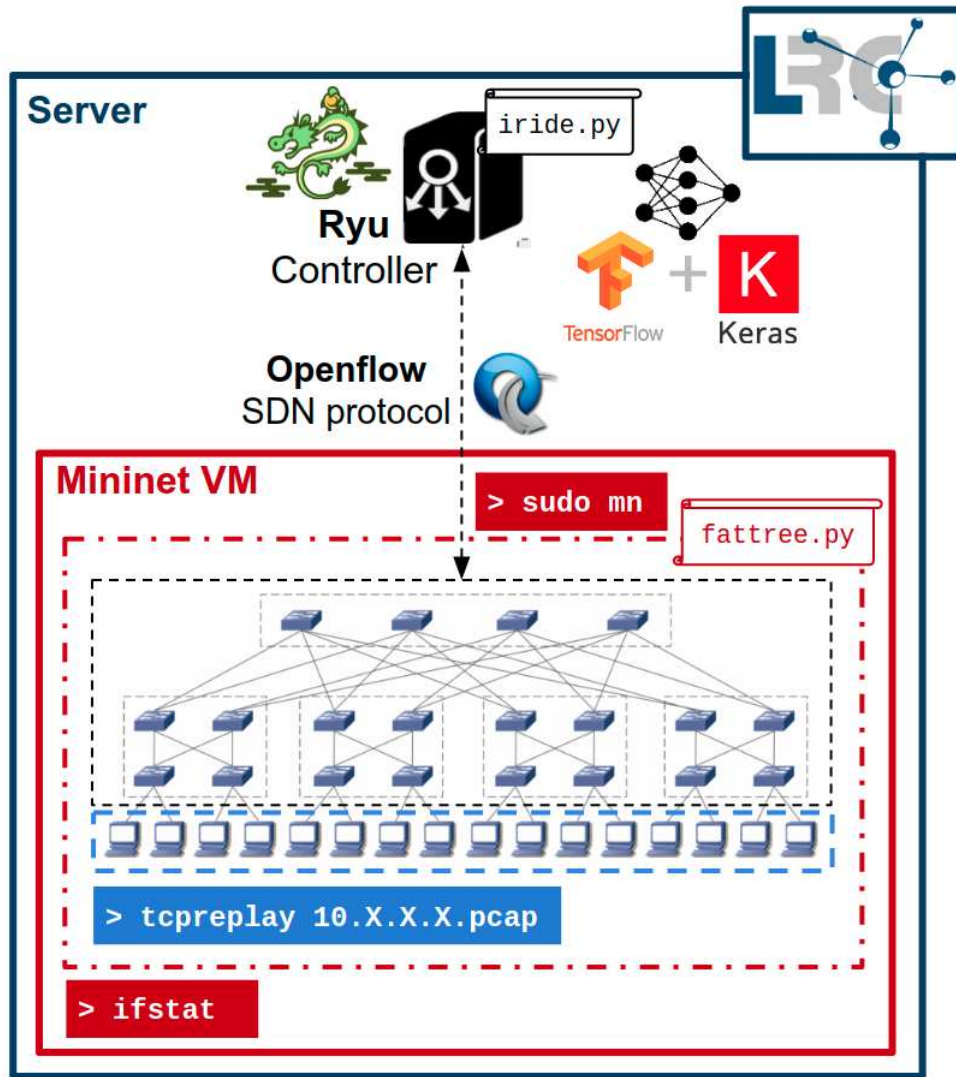


Figure 4.15: Implementation of iRIDE

of different PCAP files. For security reasons, these PCAP files come anonymized, which changes the source and destination MAC and IP addresses, remaps the transport layer ports, and truncates the payload of the captured packets [312]. Our PCAP parser then hashes the source and destination MAC and IP addresses for distributing the packets among the hosts in our fat-tree network. We use the source IP for assigning each packet to the corresponding PCAP file (e.g., 10.0.0.1.pcap). The PCAP parser also reads the packet length for padding the payload of each packet with empty data. Lastly, recall that iRIDE relies on a flow detection method, such as NELLY (see Chapter 3), that reports the elephant flows using marked packets. Therefore, the PCAP parser reads the classification results from NELLY for generating the marked packets that report those flows classified as elephants. The PCAP parser inserts the value 001111 into the DSCP field of the marked packets, so the catching rules will match and forward them to the controller. Moreover, the marked packets include the size and IAT of the first 7 packets of the flow into their payload.

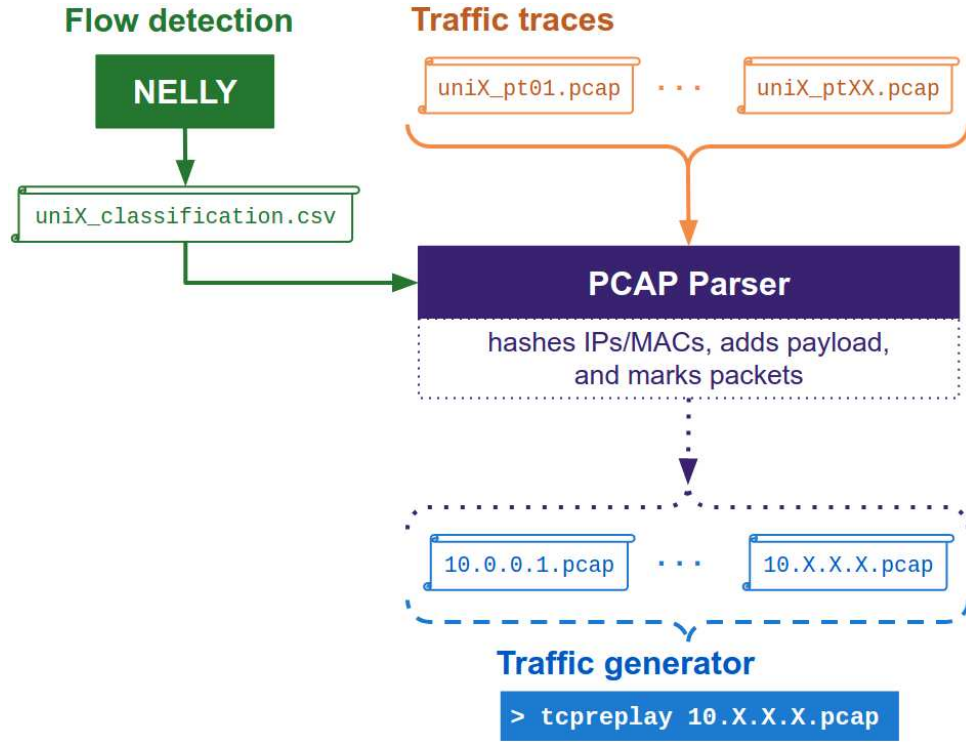


Figure 4.16: PCAP parsing for the traffic generator in iRIDE

4.2.5 Evaluation

Due to resource limitations, we executed our evaluation using a fat-tree topology of size $k = 4$, which deploys 16 hosts, 20 switches (8 at edge layer, 8 at aggregation layer, and 4 at core layer), and 48 links (see Table 4.1). Note that the number of logical cores in the server is insufficient for running a fat-tree topology of size $k = 8$ with all the 128 hosts concurrently executing a Tcpreplay process. Since the total bandwidth in our Mininet VM (obtained using the *iperf* test) was restricted to a maximum of 20 Gbps, we limited each link to a bandwidth of 100 Mbps, for a total of 4.8 Gbps. Moreover, we replayed only the traffic from the UNI1 packet trace, for which the PCAP parser generated the PCAP files with a total size of 12 GB. We were not able to generate the PCAP files for the UNI2 packet trace due to disk space limitations in the server. The hosts in our fat-tree emulated network replayed the traffic from UNI1 at top-speed.

For the evaluation, we measured the throughput over time in the links of the fat-tree emulated network when using iRIDE, with both rescheduling algorithms LC and WF+BTF, and when using PM2, which provides an ECMP implementation for OpenFlow networks. Figure 4.17 depicts the throughput over time in the bisection links of the fat-tree topology from the first run of the evaluation. The bisection links of a network is represented by “the minimum number of links to be removed to disconnect the network into two halves of equal size” [313]. It is noteworthy that we also measured the throughput on links from other areas of the fat-tree topology, including the links from the edge layer to the aggregation layer and from the aggregation layer to the core layer (the links from the edge layer to the hosts are not of our interest as they do not provide multiple paths). However, we observed the same pattern as in the bisection links, so we decided to present

only the throughput in the bisection links for simplicity. The results show that iRIDE is able to use more efficiently the bisection bandwidth than PM2. Moreover, iRIDE using the rescheduling algorithm WF+BTF is able to use more efficiently the bisection bandwidth than using LC. Note that as better the bandwidth efficiency, the faster to complete the traffic.

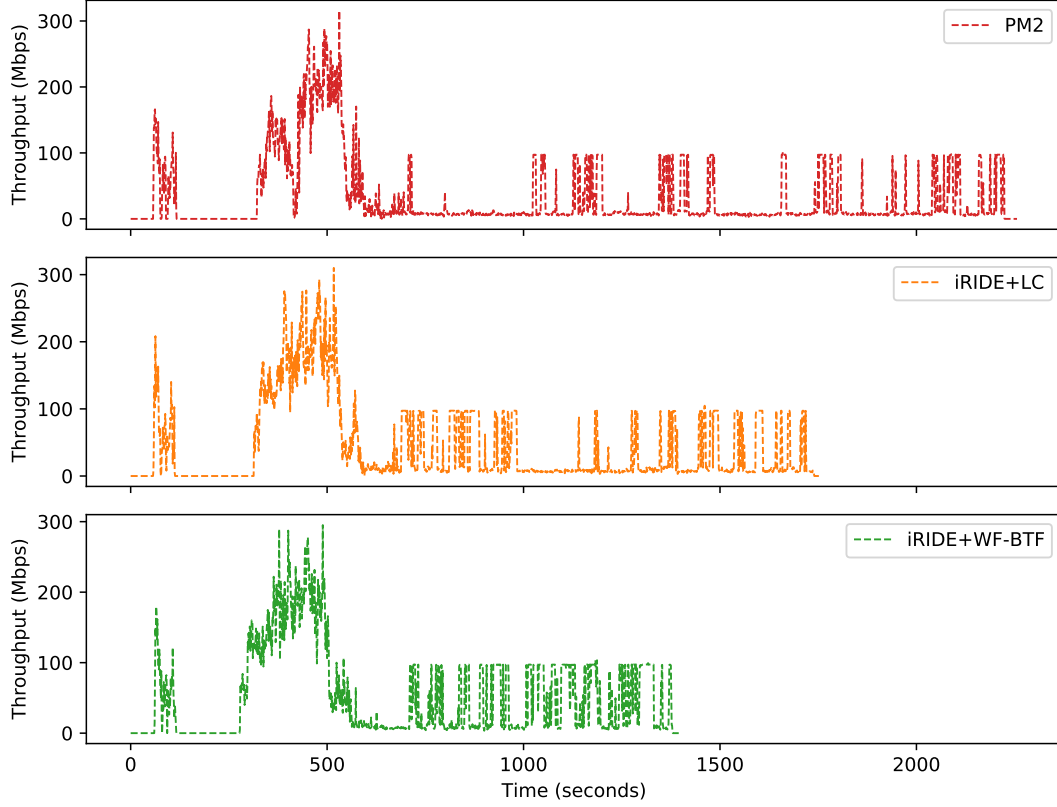


Figure 4.17: Throughput over time in the bisection links of a fat-tree topology of size $k = 4$ when using PM2 and iRIDE, with both LC and WF+BTF, for routing UNI1 traffic

Finally, Figure 4.18(a) presents the mean throughput in the bisection links of the fat-tree topology, whereas Figure 4.18(b) depicts the traffic completion time. Both figures display the average values and the corresponding standard deviation from ten runs for each routing algorithm. Figure 4.18(a) shows that PM2 achieves a mean throughput of 33 Mbps in the bisection links of the fat-tree topology, whereas iRIDE increments that throughput by 5 and 11 Mbps when using LC and WF+BTF, respectively. Conversely, Figure 4.18(b) depicts that all the hosts in the fat-tree topology completed the communication in about 38 minutes when using PM2 only, whereas iRIDE reduced such a traffic completion time by 5 and 9 minutes when using LC and WF+BTF, respectively. These results confirm that iRIDE is able to generate more throughput and to complete the traffic faster than PM2, particularly, when using the rescheduling algorithm WF+BTF, which uses the flow rates and flow durations predicted using the incremental DNNs.

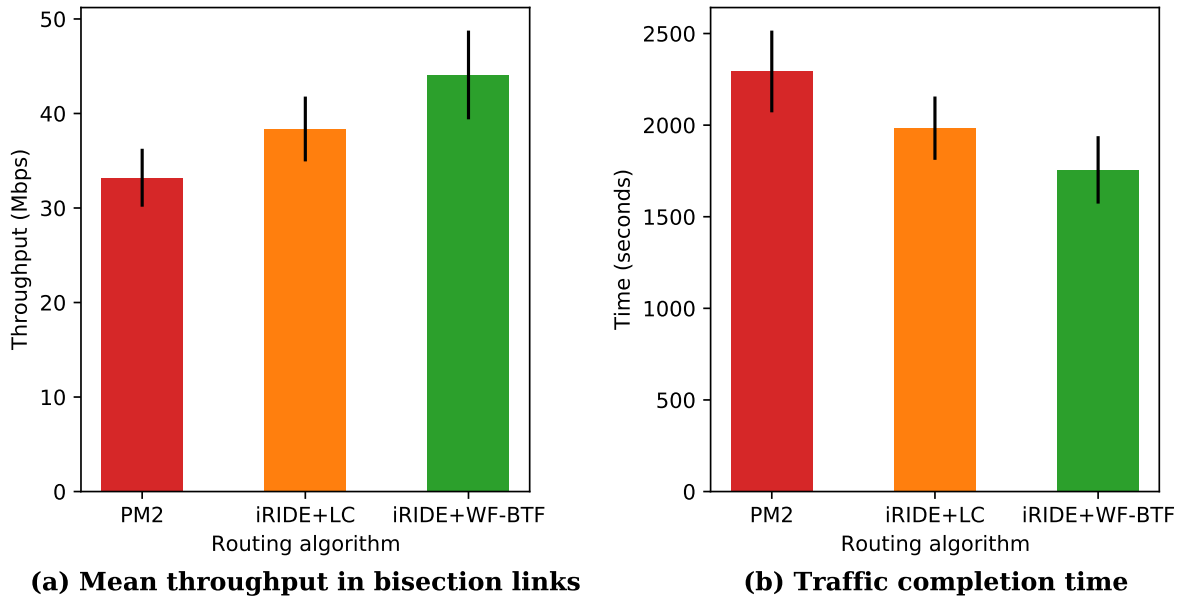


Figure 4.18: Mean throughput in the bisection links of a fat-tree topology of size $k = 4$ and traffic completion time when using PM2 and iRIDE, with both LC and WF+BTF, for routing UNI1 traffic

4.3 Final remarks

A relevant problem affecting the overall performance of multipath routing in SDDCNs is the coexistence of mice and elephant flows. Aiming at overcoming this problem, this chapter introduced PM2, a multipath routing algorithm for steering flows (mainly mice) in a fat-tree DCN topology. PM2 supports transparent host migration across the whole network while reducing the number of rules installed on SDN switches, decreasing the delay introduced to flows (mainly mice) traversing the network. An analytical comparison corroborated that PM2 installs much fewer rules than other OpenFlow-feasible multipath routing algorithms that support transparent host migration across a topology area greater than the same edge switch. Furthermore, this chapter proposed iRIDE, a flow rescheduling method that applies incremental learning at the controller-side of SDDCNs for predicting traffic characteristics of flows identified as elephants to compute and install the best path per flow across the network. An extensive evaluation based on real packet traces and various incremental learning algorithms demonstrated the low error for predicting the flow rate and duration of iRIDE when using DNNs with regularization and dropout layers. Furthermore, the evaluation results show the high throughput and short traffic completion time of iRIDE when implementing a rescheduling algorithm that uses the two predicted traffic characteristics.

Chapter 5

Conclusions

This chapter starts summarizing the research work carried out in this thesis. Then, it provides the answers for the fundamental questions that guided the verification of the hypothesis defended in this thesis. The last section outlines directions for future work.

This thesis presented the investigation carried out to verify the hypothesis: **using ML for fine-granularity prediction of flow characteristics and SDN for dynamic control of flow scheduling would allow building a multipath routing mechanism for DCNs that improves¹ the routing function.** Based on the hypothesis, this work proposed a multipath routing mechanism that leverages both SDN and ML to improve the routing function in DCNs. Three major components form the proposed multipath routing mechanism: NELLY, PM2, and iRIDE.

NELLY introduced a flow detection method that incorporates incremental learning at the server-side of SDDCNs to accurately and timely identify elephant flows at low traffic overhead while enabling continuous model adaptation under limited memory resources. An extensive evaluation based on real packet traces and various incremental learning algorithms demonstrated the high accuracy and speed of NELLY when used with the ARF and AHOT algorithms. Moreover, an analytical comparison to seminal related works corroborated the scalability of NELLY as well as its generation of low traffic overhead and the fact that no modifications in SDN standards are required.

PM2 provided a multipath routing algorithm that supports transparent host migration across the whole network while reducing the number of rules installed on SDN switches, decreasing the delay introduced to flows (mainly mice) traversing the SDDCN. A prototype implementation serves as a proof of concept for demonstrating the feasibility of PM2 in a fat-tree DCN topology. Moreover, an analytical comparison corroborated that PM2 installs much fewer rules than other OpenFlow-feasible multipath routing algorithms depending on either MAC or IP addresses and supporting transparent host migration across a topology area greater than the same edge switch.

iRIDE proposed a flow rescheduling method at the controller-side of SDDCNs that improves network throughput and traffic completion time by using incremental learning to predict the rate and duration of elephants for computing and installing the best path across the network. An extensive evaluation based on real packet traces and various

¹In terms of high throughput and low delay while efficient use of resources

incremental learning algorithms demonstrated the low error of iRIDE for predicting the flow rate and flow duration when using DNNs with regularization and dropout layers. Furthermore, the evaluation results show the high throughput and short traffic completion time of iRIDE when implementing the rescheduling algorithm WF+BTF, which uses both predicted traffic characteristics, flow rate and flow duration.

5.1 Answers for the fundamental questions

Two fundamental questions guided the investigation about using ML for fine-granularity prediction of flow characteristics and SDN for dynamic control of flow scheduling aiming at building a multipath routing mechanism for DCNs that improves the routing function. This section reviews and answers such questions.

Fundamental question I: *What is the accuracy and efficiency, in terms of time and memory, of ML techniques for predicting flow characteristics of network traffic from DCNs?*

In this work, NELLY focused on predicting the size of flows by classifying them as mice or elephants. The evaluation results demonstrated that using incremental learning algorithms for performing such a classification achieves high elephant detection with short classification time. In particular, NELLY achieved the best classification performance, in terms of accuracy and time, by using the BinNom headers along with the following adaptive decision trees algorithms. ARF provides the best classification accuracy for UNI1 and UNI2 traffic with a classification time less than $17 \mu s$ (i.e., less than 7.5% of the RTT in DCNs). AHOT is also good for UNI1 and UNI2 traffic, with a minor classification accuracy than ARF but reducing the classification time to less than $10 \mu s$. Finally, the Hoeffding tree is only good for traffic similar to that of UNI1 but achieves a classification accuracy similar to that of AHOT with a classification time less than $3 \mu s$.

Similarly, iRIDE focused on predicting the rate and duration of flows by using regression models. The evaluation results revealed that iRIDE achieved the lowest prediction errors of flow rate and flow duration when using DNNs with L2 regularization and dropout layers. In particular, the most accurate DNNs required deep structures from five up to nine hidden layers and from 360 up to 600 units per each hidden layer. Moreover, for predicting the flow rate, the DNN structures achieved the lowest errors by using a combination of L2 regularization and dropout layers, whereas, for predicting the duration, the most accurate DNN structures only required L2 regularization.

Finally, note that incremental learning reduces memory consumption by continuously updating the models from constantly generated data that is temporarily persisted. This enabled both NELLY and iRIDE, which rely on incremental learning algorithms, to adapt to the variations in traffic characteristics and perform endless learning with limited memory resources.

Fundamental question II: *Does incorporating ML techniques to an SDN-based multipath routing mechanism improve network traffic routing, in terms of throughput and delay, in DCNs?*

Recall that NELLY incorporates incremental learning at the server-side of SDDCNs for

proactively identifying elephant flows at low traffic overhead. An analytical comparison to seminal related works corroborated that NELLY reduces the switch table occupancy, the traffic overhead, and the flow detection time. In particular, NELLY requires no flow table entries on the SDN switches for collecting data since NELLY operates at the server-side of SDDCNs, having local access to the data. Regarding the traffic overhead, NELLY merely requires that edge (ToR) switches send control packets of flows marked as elephants, greatly reducing the control traffic overhead (to 4.4 kbps if assuming a control packet of 64 bytes). Lastly, NELLY detects elephant flows in a very short time as it relies on the first N packets (0.8 seconds when using the first 7 packets).

On the other hand, iRIDE incorporated incremental learning at the controller-side of SDDCNs to predict the rate and duration of flows. These predicted flow traffic characteristics enabled constructing intelligent elephant rescheduling algorithms, such as LC and WF+BTF. The results from a quantitative evaluation demonstrated that iRIDE efficiently uses the available bandwidth, generating higher throughput and shorter traffic completion time than conventional ECMP. In particular, when replaying the traffic from the UNI1 packet trace in a fat-tree emulated network, iRIDE with WF+BTF incremented the bisection throughput by 11 Mbps and reduced the traffic completion time by 9 minutes in comparison with PM2. Note that WF+BTF uses the flow rates and flow durations predicted using the incremental DNNs.

5.2 Future work

During the development of this thesis, we observed interesting opportunities for further research. These opportunities are outlined as follows.

- Implement NELLY as an in-kernel software component for evaluating its impact cost to server resources, including processing and memory consumption. This implementation would enable to evaluate NELLY in an emulated SDDCN by installing the software component into micro virtual machines connected to Open vSwitch instances.
- Although this paper has proven that incremental learning algorithms are efficient to detect elephant flows in DCNs, there is still no consistent and accepted method for defining the threshold value that discriminates between mice and elephants in DCNs. In this thesis, we evaluated different thresholds but did not specify how to select the appropriate threshold value for the traffic and routing requirements. RL algorithms can be useful for selecting a threshold that maximizes DCN routing performance (e.g., throughput and delay) for specific traffic conditions.
- A future analysis of network traffic using IPv6 or at other layers of IIoT systems would help to analyze if such traffic characteristics can also benefit from incremental learning for either classifying flows (NELLY) or predicting flow features (iRIDE). For example, fog layers are formed by micro data centers that analyze data that require a rapid return (low latency). Moreover, it is expected that IIoT systems introduce more diversity of network traffic (from elephant flows to mouse flows).

Therefore, there is a need to create publicly available datasets of network traffic with different characteristics (e.g., IPv6, fog layers) to evaluate the performance improvement of ML-based methods on such datasets.

- Extending iRIDE to achieve a full cognitive networking, such as the C-MAPE loop that we proposed in collaboration with other researchers (see Section 2.2.3. Note that the cognitive operation in iRIDE (i.e., predicting flow traffic characteristics) belongs to the C-Analyze function. However, the statistics collection, the path selection, and the rescheduling algorithm can be extended using an ML-based approach, providing cognition in the Monitor, Plan, and Execute functions, respectively. Different works [40, 314–316] in our research group have already explored some of these cognitive approaches but independently.

Bibliography

- [1] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, “Data center network virtualization: A survey,” *IEEE Communications Surveys Tutorials*, vol. 15, pp. 909–928, Second 2013.
- [2] R. K. Singh, N. S. Chaudhari, and K. Saxena, “Load balancing in IP/MPLS networks: A survey,” *Communications and Network*, vol. 4, pp. 151–156, May 2012.
- [3] N. Wang, K. H. Ho, G. Pavlou, and M. Howarth, “An overview of routing optimization for internet traffic engineering,” *IEEE Communications Surveys Tutorials*, vol. 10, pp. 36–56, First 2008.
- [4] S. K. Singh, T. Das, and A. Jukan, “A survey on internet multipath routing and provisioning,” *IEEE Communications Surveys Tutorials*, vol. 17, pp. 2157–2175, Fourthquarter 2015.
- [5] M. Chiesa, G. Kindler, and M. Schapira, “Traffic engineering with equal-cost-multipath: An algorithmic perspective,” *IEEE/ACM Transactions on Networking*, vol. 25, pp. 779–792, April 2017.
- [6] G. Detal, C. Paasch, S. van der Linden, P. Mérindol, G. Avoine, and O. Bonaventure, “Revisiting flow-based load balancing: Stateless path selection in data center networks,” *Computer Networks*, vol. 57, no. 5, pp. 1204 – 1216, 2013.
- [7] C. Hopps, “Analysis of an Equal-Cost Multi-Path Algorithm,” RFC 2992, Internet Engineering Task Force, Nov. 2000.
- [8] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, “Inside the social network’s (datacenter) network,” in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM ’15*, (New York, NY, USA), pp. 123–137, ACM, 2015.
- [9] T. Benson, A. Akella, and D. A. Maltz, “Network traffic characteristics of data centers in the wild,” in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, IMC ’10*, (New York, NY, USA), pp. 267–280, ACM, 2010.
- [10] T. Benson, A. Anand, A. Akella, and M. Zhang, “Understanding data center traffic characteristics,” *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 92–99, Jan. 2010.

- [11] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, “The nature of data center traffic: Measurements & analysis,” in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC ’09, (New York, NY, USA), pp. 202–208, ACM, 2009.
- [12] A. Andreyev, “Introducing data center fabric, the next-generation Facebook data center network.” Facebook Engineering, Nov. 2014. <https://code.facebook.com/posts/360346274145943/> (accessed Oct. 19, 2021).
- [13] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, “On the impact of packet spraying in data center networks,” in *2013 Proceedings IEEE INFOCOM*, pp. 2130–2138, April 2013.
- [14] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. A. Maltz, “Per-packet load-balanced, low-latency routing for clos-based data center networks,” in *CoNEXT*, 2013.
- [15] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, “Presto: Edge-based load balancing for fast datacenter networks,” in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM ’15, (New York, NY, USA), pp. 465–478, ACM, 2015.
- [16] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, “Wcmp: Weighted cost multipathing for improved fairness in data centers,” in *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys ’14, (New York, NY, USA), pp. 5:1–5:14, ACM, 2014.
- [17] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, “Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks,” in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT ’14, (New York, NY, USA), pp. 149–160, ACM, 2014.
- [18] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, “Scalable, optimal flow routing in datacenters via local link balancing,” in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT ’13, (New York, NY, USA), pp. 151–162, ACM, 2013.
- [19] S. Kandula, D. Katabi, S. Sinha, and A. Berger, “Dynamic load balancing without packet reordering,” *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 51–62, Mar. 2007.
- [20] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, “Let it flow: Resilient asymmetric load balancing with flowlet switching,” in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, (Boston, MA), pp. 407–420, USENIX Association, 2017.

- [21] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, “Hula: Scalable load balancing using programmable data planes,” in *Proceedings of the Symposium on SDN Research*, SOSR ’16, (New York, NY, USA), pp. 10:1–10:12, ACM, 2016.
- [22] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, “Conga: Distributed congestion-aware load balancing for datacenters,” in *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM ’14, (New York, NY, USA), pp. 503–514, ACM, 2014.
- [23] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, pp. 14–76, Jan 2015.
- [24] H. Kim and N. Feamster, “Improving network management with software defined networking,” *IEEE Communications Magazine*, vol. 51, pp. 114–119, February 2013.
- [25] H. Xu and B. Li, “Tinyflow: Breaking elephants down into mice in data center networks,” in *2014 IEEE 20th International Workshop on Local Metropolitan Area Networks (LANMAN)*, pp. 1–6, May 2014.
- [26] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic flow scheduling for data center networks,” in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI’10, (Berkeley, CA, USA), pp. 19–19, USENIX Association, 2010.
- [27] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “Devoflow: Scaling flow management for high-performance networks,” in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM ’11, (New York, NY, USA), pp. 254–265, ACM, 2011.
- [28] R. Trestian, G. M. Muntean, and K. Katrinis, “Micetrap: Scalable traffic engineering of datacenter mice flows using openflow,” in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pp. 904–907, May 2013.
- [29] A. R. Curtis, W. Kim, and P. Yalagandula, “Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection,” in *2011 Proceedings IEEE INFOCOM*, pp. 1629–1637, April 2011.
- [30] J. Lu, W. Liu, Y. Zhu, S. Ling, Z. Chen, and J. Zeng, “Scheduling mix-flow in sd-dcn based on deep reinforcement learning with private link,” in *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, pp. 395–401, 2020.
- [31] W.-X. Liu, J. Cai, Y. Wang, Q. C. Chen, and J.-Q. Zeng, “Fine-grained flow classification using deep learning for software defined data center networks,” *Journal of Network and Computer Applications*, vol. 168, p. 102766, 2020.

- [32] P. Poupart, Z. Chen, P. Jaini, F. Fung, H. Susanto, Y. Geng, L. Chen, K. Chen, and H. Jin, "Online flow size prediction for improved network routing," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pp. 1–6, Nov 2016.
- [33] S.-C. Chao, K. C.-J. Lin, and M.-S. Chen, "Flow classification for software-defined data centers using stream mining," *IEEE Transactions on Services Computing*, vol. 12, no. 1, pp. 105–116, 2019.
- [34] H. Zhang, F. Tang, and L. Barolli, "Efficient flow detection and scheduling for sdn-based big data centers," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, pp. 1915–1926, 2019.
- [35] F. Tang, L. Li, L. Barolli, and C. Tang, "An efficient sampling and classification approach for flow detection in sdn-based big data centers," in *IEEE AINA*, (Taipei, Taiwan), pp. 1106–1115, Mar. 2017.
- [36] C. F. Estrada Solano, "Software-defined networking management based on web 2.0 and web 3.0 technologies," Master's thesis, Telematics Dept., Univ. of Cauca, Popayan, Colombia, Jan. 2016. Available: <http://repositorio.unicauca.edu.co:8080/xmlui/handle/123456789/1358>.
- [37] F. Estrada-Solano, O. M. Caicedo, and N. L. S. Da Fonseca, "Nelly: Flow detection using incremental learning at the server side of sdn-based data centers," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 2, pp. 1362–1372, 2020.
- [38] F. Amezcuita-Suarez, F. Estrada-Solano, N. L. S. da Fonseca, and O. M. C. Rendon, "An efficient mice flow routing algorithm for data centers based on software-defined networking," in *IEEE ICC*, (Shanghai, China), pp. 1–6, May 2019.
- [39] F. Estrada-Solano, A. Ordonez, L. Z. Granville, and O. M. Caicedo Rendon, "A framework for sdn integrated management based on a cim model and a vertical management plane," *Computer Communications*, vol. 102, pp. 150 – 164, Apr. 2017.
- [40] M. A. Gironza-Ceron, W. F. Villota-Jacome, A. Ordonez, F. Estrada-Solano, and O. M. Caicedo Rendon, "Sdn management based on hierarchical task network and network functions virtualization," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1360–1365, 2017.
- [41] A. I. Montoya-Munoz, D. M. Casas-Velasco, F. Estrada-Solano, A. Ordonez, and O. M. Caicedo Rendon, "A yang model for a vertical sdn management plane," in *2017 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pp. 1–6, 2017.
- [42] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo Rendon, "Machine learning for cognitive network management," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 158–165, 2018.

- [43] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, “A comprehensive survey on machine learning for networking: evolution, applications and research opportunities,” *Journal of Internet Services and Applications*, vol. 9, p. 16, June 2018.
- [44] A. I. Montoya-Munoz, D. Casas-Velasco, F. Estrada-Solano, O. M. Caicedo Rendon, and N. L. Saldanha da Fonseca, “An approach based on yet another next generation for software-defined networking management,” *International Journal of Communication Systems*, vol. 34, no. 11, p. e4855, 2021.
- [45] S. Crawford and L. Stucki, “Peer review and the changing research record,” *J. Am. Soc. Inf. Sci.*, vol. 41, pp. 223–228, Mar. 1990.
- [46] N. Feamster, J. Rexford, and E. Zegura, “The road to sdn: An intellectual history of programmable networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–98, Apr. 2014.
- [47] P. Lin, J. Bi, H. Hu, T. Feng, and X. Jiang, “A quick survey on selected approaches for preparing programmable networks,” in *Proceedings of the 7th Asian Internet Engineering Conference*, AINTEC ’11, (New York, NY, USA), pp. 160–163, ACM, 2011.
- [48] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: Enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.
- [49] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, “A roadmap for traffic engineering in sdn-openflow networks,” *Comput. Netw.*, vol. 71, pp. 1–30, Oct. 2014.
- [50] L. Efremova and D. Andrushko, “What’s in opendaylight?,” Mirantis, Apr. 2015. <https://www.mirantis.com/blog/whats-opendaylight/> (accessed Oct. 19, 2021).
- [51] ONF, “Software-defined networking: The new norm for networks,” white paper, Open Network Foundation, Apr. 2012.
- [52] S. Kiran and G. Kinghorn, “Cisco open network environment: Bring the network closer to applications,” White Paper C11-728045-03, Cisco, Sept. 2015.
- [53] A. Singla and B. Rijsman, *Day One: Understanding OpenContrail Architecture*. Juniper Networks Books, Nov. 2013.
- [54] M. Casado, N. Foster, and A. Guha, “Abstractions for software-defined networks,” *Commun. ACM*, vol. 57, p. 86–95, Sept. 2014.
- [55] ONF, “Openflow switch specification version 1.5.1,” Technical Specification TS-025, Open Networking Foundation, Mar. 2015.
- [56] O. Michel, R. Bifulco, G. Rétvári, and S. Schmid, “The programmable data plane: Abstractions, architectures, algorithms, and applications,” *ACM Comput. Surv.*, vol. 54, May 2021.

- [57] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, “Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM ’17, (New York, NY, USA), p. 15–28, Association for Computing Machinery, 2017.
- [58] C. Kim, A. Sivaraman, N. P. K. Katta, A. Bas, A. A. Dixit, and L. J. Wobker, “In-band network telemetry via programmable dataplanes,” in *ACM SIGCOMM Symposium on SDN Research (SOSR) ’15 Demos*, June 2015.
- [59] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, “P4: Programming protocol-independent packet processors,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, p. 87–95, July 2014.
- [60] L. Yang, R. Dantu, T. Anderson, and R. Gopal, “Forwarding and control element separation (ForCES) framework,” RFC 3746, Internet Engineering Task Force, Apr. 2004.
- [61] H. Song, “Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane,” in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN ’13, (New York, NY, USA), p. 127–132, Association for Computing Machinery, 2013.
- [62] P4.org API Working Group, “P4runtime specification.” P4.org, July 2021. <https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html> (accessed Oct. 19, 2021).
- [63] R. Sherwood, G. Gibb, K. kiong Yap, M. Casado, N. McKeown, and G. Parulkar, “Flowvisor: A network virtualization layer,” Technical Report OpenFlow-tr-2009-1, OpenFlow, Oct. 2009.
- [64] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “Nox: Towards an operating system for networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, p. 105–110, July 2008.
- [65] NOX Repo, “The POX network software platform.” GitHub. <https://github.com/noxrepo/pox> (accessed Oct. 19, 2021).
- [66] R. Kubo, T. Fujita, Y. Agawa, and H. Suzuki, “Ryu sdn framework-open-source sdn platform software,” *NTT Technical Review*, vol. 12, 08 2014.
- [67] Project Floodlight, “Floodlight controller.” Project Floodlight. <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller> (accessed Oct. 19, 2021).
- [68] OpenDaylight Project, “Welcome to.opendaylight documentation.” OpenDaylight Documentation. <https://docs.opendaylight.org/> (accessed Oct. 19, 2021).
- [69] Trema, “Trema: Full-stack openflow framework in ruby.” GitHub. <https://github.com/trema/trema> (accessed Oct. 19, 2021).

- [70] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi, “SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains.” Internet Draft, June 2012.
- [71] R. Izard and Q. Wang, “Floodlight REST API.” Project Floodlight, Mar. 2018. <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343539/Floodlight+REST+API> (accessed Oct. 19, 2021).
- [72] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, “Modular sdn programming with pyretic,” *USENIX Login*, vol. 38, pp. 128–134, 10 2013.
- [73] A. Voellmy, H. Kim, and N. Feamster, “Procera: A language for high-level reactive network control,” in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN ’12, (New York, NY, USA), p. 43–48, Association for Computing Machinery, 2012.
- [74] M. Monaco, O. Michel, and E. Keller, “Applying operating system principles to sdn controller design,” in *In Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, HotNets-XII, (New York, NY, USA), p. 2:1–2:7, ACM, 2013.
- [75] J. A. Wickboldt, W. P. De Jesus, P. H. Isolani, C. B. Both, J. Rochol, and L. Z. Granville, “Software-defined networking: management requirements and challenges,” *IEEE Communications Magazine*, vol. 53, no. 1, pp. 278–285, 2015.
- [76] E. Haleplidis, J. Hadi Salim, S. Denazis, and O. Koufopavlou, “Towards a network abstraction model for sdn,” *Journal of Network and Systems Management*, vol. 23, no. 2, pp. 309–327, 2015.
- [77] ONF, “Sdn architecture,” Technical Reference TR-502, Open Network Foundation, June 2014. Issue 1.
- [78] ITU, “Framework of software-defined networking,” Recommendation Y.3300, International Telecommunication Union, June 2014.
- [79] ISO, “Information technology – Open Systems Interconnection – Systems management overview,” ISO/IEC 10040:1998, International Organization for Standardization, Nov. 1998.
- [80] T. Bray, “The JavaScript Object Notation (JSON) data interchange format,” Standards Track RFC 7159, Internet Engineering Task Force, Mar. 2014.
- [81] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1,” Standards Track RFC 2616, Internet Engineering Task Force, June 1999.
- [82] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, “Extensible Markup Language (XML) 1.0 (fifth edition),” recommendation, World Wide Web Consortium, Nov. 2008.

- [83] B. Pfaff and B. Davie, “The open vswitch database management protocol,” Informational RFC 7047, Internet Engineering Task Force, Dec. 2013.
- [84] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, “Network Configuration Protocol (NETCONF),” Standards Track RFC 6241, Internet Engineering Task Force, June 2011.
- [85] D. Harrington, R. Presuhn, and B. Wijnen, “An architecture for describing Simple Network Management Protocol (SNMP) management frameworks,” Standards Track RFC 3411, Internet Engineering Task Force, Dec. 2002.
- [86] ITU, “TMN management functions,” Recommendation M.3400, International Telecommunication Union, Feb. 2000.
- [87] DMTF, “Common Information Model (CIM) Infrastructure v2.7.0,” Specification DSP0004, Distributed Management Task Force, Apr. 2012.
- [88] A. Pras and J. Schoenwaelder, “On the difference between information models and data models,” Informational RFC 3444, Internet Engineering Task Force, Jan. 2003.
- [89] A. Afanasyev and S. K. Ramani, “Ndnconf: Network management framework for named data networking,” in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–6, 2020.
- [90] S. Popić, B. Majstorović, M. Vuleta, Đ. Sarić, and B. M. Todorović, “Efficient usage of resources in sdn by modifying yang modules in linux-based embedded systems,” in *2019 27th Telecommunications Forum (TELFOR)*, pp. 1–4, 2019.
- [91] G. Parladori, G. Gasparini, F. Ruggi, A. D. Broi, V. Simone, and F. Nicassio, “Yang modelling of optical nodes,” in *20th Italian National Conference on Photonic Technologies (Fotonica 2018)*, pp. 1–4, 2018.
- [92] M. Bjorklund, “The YANG 1.1 data modeling language,” Standards Track RFC 7950, Internet Engineering Task Force, Aug. 2016.
- [93] J. Schönwälder, M. Björklund, and P. Shafer, “Network configuration management using netconf and yang,” *IEEE Communications Magazine*, vol. 48, pp. 166–173, 2010.
- [94] S. Wallin, “Uml visualization of yang models,” in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pp. 1129–1134, 2011.
- [95] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, “Network function virtualization: State-of-the-art and research challenges,” *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.
- [96] C. Ullrich and E. Melis, “Pedagogically founded courseware generation based on htn-planning,” *Expert Systems with Applications*, vol. 36, pp. 9319–9332, 07 2009.

- [97] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd ed., 2009.
- [98] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [99] E. Alpaydin, *Introduction to Machine Learning*. The MIT Press, 2nd ed., 2010.
- [100] G. Tesauro, “Reinforcement learning in autonomic computing: A manifesto and case studies,” *IEEE Internet Computing*, vol. 11, pp. 22–30, Jan 2007.
- [101] M. Bramer, *Principles of Data Mining*. Springer Publishing Company, Incorporated, 2nd ed., 2013.
- [102] A. Ng, *Machine Learning Yearning*. Online Draft, 2017.
- [103] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, *Data Stream Mining A Practical Approach*. The University of Waikato.
- [104] V. Losing, B. Hammer, and H. Wersing, “Incremental on-line learning: A review and comparison of state of the art algorithms,” *Neurocomputing*, vol. 275, pp. 1261–1274, 2018.
- [105] A. R. T. Gepperth and B. Hammer, “Incremental learning algorithms and applications,” in *ESANN*, 2016.
- [106] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2nd ed., 2016.
- [107] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, Dec 1943.
- [108] D. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. A Wiley book in clinical psychology, Wiley, 1949.
- [109] A. M. Turing, “Computing machinery and intelligence,” *Mind*, vol. 59, no. 236, pp. 433–460, 1950.
- [110] A. Legendre, *Nouvelles méthodes pour la détermination des orbites des comètes*. Nineteenth Century Collections Online (NCCO): Science, Technology, and Medicine: 1780-1925, F. Didot, 1805.
- [111] S. M. Stigler, “Gauss and the invention of least squares,” *Ann. Statist.*, vol. 9, pp. 465–474, 05 1981.
- [112] A. S. Goldberger, “Econometric computing by hand,” *Journal of Economic and Social Measurement*, vol. 29, no. 1-3, pp. 115–117, 2004.
- [113] E. T. Jaynes, “Information theory and statistical mechanics,” *Phys. Rev.*, vol. 106, pp. 620–630, May 1957.

- [114] E. T. Jaynes, “Information theory and statistical mechanics. ii,” *Phys. Rev.*, vol. 108, pp. 171–190, Oct 1957.
- [115] D. R. Cox, “The regression analysis of binary sequences,” *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 20, no. 2, pp. 215–242, 1958.
- [116] E. Fix and J. L. Hodges, “Discriminatory analysis-nonparametric discrimination: consistency properties,” Report No. 4, Project 21-49-004, USAF School of Aviation Medicine, Feb. 1951.
- [117] C. Stanfill and D. Waltz, “Toward memory-based reasoning,” *Commun. ACM*, vol. 29, pp. 1213–1228, Dec. 1986.
- [118] M. Rosenblatt, “Remarks on some nonparametric estimates of a density function,” *Ann. Math. Statist.*, vol. 27, pp. 832–837, 09 1956.
- [119] E. Parzen, “On estimation of a probability density function and mode,” *Ann. Math. Statist.*, vol. 33, pp. 1065–1076, 09 1962.
- [120] H. Robbins and S. Monro, “A Stochastic Approximation Method,” *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400 – 407, 1951.
- [121] J. Kiefer and J. Wolfowitz, “Stochastic Estimation of the Maximum of a Regression Function,” *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462 – 466, 1952.
- [122] C. Darken and J. Moody, “Note on learning rate schedules for stochastic optimization,” in *Advances in Neural Information Processing Systems* (R. P. Lippmann, J. Moody, and D. Touretzky, eds.), vol. 3, Morgan-Kaufmann, 1991.
- [123] M. Moller, “Supervised learning on large redundant training sets,” in *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*, pp. 79–89, 1992.
- [124] D. Newton, R. Pasupathy, and F. Yousefian, “Recent trends in stochastic gradient descent for machine learning and big data,” in *2018 Winter Simulation Conference (WSC)*, pp. 366–380, 2018.
- [125] C. K. Chow, “An optimum character recognition system using decision functions,” *IRE Transactions on Electronic Computers*, vol. EC-6, pp. 247–254, Dec 1957.
- [126] M. E. Maron, “Automatic indexing: An experimental inquiry,” *J. ACM*, vol. 8, pp. 404–417, July 1961.
- [127] M. Bayes and M. Price, “An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, f. r. s. communicated by mr. price, in a letter to john canton, a. m. f. r. s.,” *Philosophical Transactions (1683-1775)*, vol. 53, pp. 370–418, 1763.
- [128] P. S. Laplace, *Théorie analytique des probabilités*. Paris, France: Courcier, 1812.

- [129] H. Steinhaus, "Sur la division des corp materiels en parties," *Bull. Acad. Polon. Sci*, vol. 1, pp. 801–804, 1956.
- [130] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, vol. 1, (Berkeley, CA, USA), pp. 281–297, University of California Press, 1967.
- [131] A. A. Markov, "An example of statistical investigation in the text of eugene onegin illustrating coupling of tests in chains," in *Proceedings of the Royal Academy of Sciences of St. Petersburg*, vol. 1, (St. Petersburg, Rusia), p. 153, 1913.
- [132] P. Gagniuc, *Markov Chains: From Theory to Implementation and Experimentation*. Wiley, 2017.
- [133] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1 ed., 1957.
- [134] F. Rosenblatt, "The perceptron, a perceiving and recognizing automaton," Report No. 85-460-1, Project PARA, Cornell Aeronautical Laboratory, Jan. 1957.
- [135] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Dev.*, vol. 3, pp. 210–229, July 1959.
- [136] R. L. Stratonovich, "Conditional markov processes," *Theory of Probability & Its Applications*, vol. 5, no. 2, pp. 156–178, 1960.
- [137] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *Ann. Math. Statist.*, vol. 37, pp. 1554–1563, 12 1966.
- [138] K. Astrom, "Optimal control of markov processes with incomplete state information," *Journal of Mathematical Analysis and Applications*, vol. 10, no. 1, pp. 174 – 205, 1965.
- [139] E. J. Sondik, *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Stanford University, California, 1971.
- [140] M. Tsetlin, *Automaton Theory and Modeling of Biological Systems*. Automaton Theory and Modeling of Biological Systems, Academic Press, 1973.
- [141] K. S. Narendra and M. A. L. Thathachar, "Learning automata - a survey," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-4, pp. 323–334, July 1974.
- [142] J. N. Morgan and J. A. Sonquist, "Problems in the analysis of survey data, and a proposal," *Journal of the American Statistical Association*, vol. 58, no. 302, pp. 415–434, 1963.

- [143] R. Messenger and L. Mandell, "A modal search technique for predictive nominal scale multivariate analysis," *Journal of the American Statistical Association*, vol. 67, no. 340, pp. 768–772, 1972.
- [144] A. G. Ivakhnenko, "Polynomial theory of complex systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-1, pp. 364–378, Oct 1971.
- [145] A. Ivakhnenko, V. Lapa, and P. U. L. I. S. O. E. ENGINEERING., *Cybernetic Predicting Devices*. Purdue University School of Electrical Engineering, 1965.
- [146] I. N. Aizenberg, N. N. Aizenberg, and J. P. Vandewalle, *Multi-Valued and Universal Binary Neurons: Theory, Learning and Applications*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.
- [147] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, 1972.
- [148] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1975.
- [149] J. S. Albus, "A new approach to manipulator control: the cerebellar model articulation controller (cmac)," *Journal of Dynamic Systems, Measurement, and Control*, vol. 97, pp. 220–227, 1975.
- [150] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.
- [151] I. H. Witten, "An adaptive optimal controller for discrete-time markov environments," *Information and Control*, vol. 34, no. 4, pp. 286 – 295, 1977.
- [152] J. Quinlan, "Discovering rules form large collections of examples: A case study," *Expert Systems in the Micro Electronic Age. Edinburgh Press: Edinburgh*, 1979.
- [153] R. Michalski and J. Larson, "Selection of most representative training examples and incremental generation of vl1 hypotheses: The underlying methodology and the description of programs esel and aq11," Technical Report 867, University of Illinois, Urbana, May 1978.
- [154] H. J. Kelley, "Gradient theory of optimal flight paths," *ARS Journal*, vol. 30, no. 10, pp. 947–954, 1960.
- [155] A. E. Bryson, "A gradient method for optimizing multi-stage allocation processes," in *Proc. Harvard Univ. Symposium on digital computers and their applications*, p. 72, April 1961.
- [156] A. Bryson and Y. Ho, *Applied optimal control: optimization, estimation, and control*. Blaisdell book in the pure and applied sciences, Blaisdell Pub. Co., 1969.

- [157] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient backprop,” in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, (Berlin, Heidelberg), p. 9–50, Springer-Verlag, 1998.
- [158] Y. Zhu, G. Zhang, and J. Qiu, “Network traffic prediction based on particle swarm bp neural network.,” *JNW*, vol. 8, no. 11, pp. 2685–2691, 2013.
- [159] R. S. Sutton and A. G. Barto, “A temporal-difference model of classical conditioning,” in *Proceedings of the ninth annual conference of the cognitive science society*, pp. 355–378, Seattle, WA, 1987.
- [160] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.
- [161] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological Cybernetics*, vol. 36, pp. 193–202, Apr 1980.
- [162] T. Kohonen, “Self-organized formation of topologically correct feature maps,” *Biological Cybernetics*, vol. 43, pp. 59–69, Jan 1982.
- [163] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.
- [164] D. H. Hubel and T. N. Wiesel, “Receptive fields of single neurones in the cat’s striate cortex,” *The Journal of Physiology*, vol. 148, no. 3, pp. 574–591, 1959.
- [165] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex,” *The Journal of Physiology*, vol. 160, no. 1, pp. 106–154, 1962.
- [166] M. I. Jordan, “Serial order: A parallel distributed processing approach,” Tech. Rep. ICS Report 8604, University of California, San Diego, 1986.
- [167] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart, “Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1,” in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1* (D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, eds.), ch. Distributed Representations, pp. 77–109, Cambridge, MA, USA: MIT Press, 1986.
- [168] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1,” in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1* (D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, eds.), ch. Learning Internal Representations by Error Propagation, pp. 318–362, Cambridge, MA, USA: MIT Press, 1986.

- [169] Y. Le Cun, *Learning Process in an Asymmetric Threshold Network*, pp. 233–240. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986.
- [170] P. Smolensky, “Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1,” in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1* (D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, eds.), ch. Information Processing in Dynamical Systems: Foundations of Harmony Theory, pp. 194–281, Cambridge, MA, USA: MIT Press, 1986.
- [171] A. S. Lapedes and R. M. Farber, “How neural nets work,” in *NIPS*, 1987.
- [172] D. S. Broomhead and D. Lowe, “Radial basis functions, multi-variable functional interpolation and adaptive networks,” Memorandum No. 4148, Royal Signals and Radar Establishment Malvern (United Kingdom), 1988.
- [173] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, “A learning algorithm for boltzmann machines,” *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.
- [174] G. E. Hinton, “Training products of experts by minimizing contrastive divergence,” *Training*, vol. 14, no. 8, 2006.
- [175] R. Salakhutdinov and G. Hinton, “Deep boltzmann machines,” in *Artificial Intelligence and Statistics*, pp. 448–455, 2009.
- [176] R. Dechter, “Learning while searching in constraint-satisfaction-problems,” in *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI’86, pp. 178–183, AAAI Press, 1986.
- [177] J. Pearl, “Bayesian networks: A model of self-activated memory for evidential reasoning,” in *Proceedings of the 7th Conference of the Cognitive Science Society*, (University of California, Irvine), pp. 329–334, Aug. 1985.
- [178] T. Dean and K. Kanazawa, “A model for reasoning about persistence and causation,” *Computational Intelligence*, vol. 5, no. 2, pp. 142–150, 1989.
- [179] P. Dagum, A. Galper, and E. J. Horvitz, *Temporal Probabilistic Reasoning: Dynamic Network Models for Forecasting*. Knowledge Systems Laboratory, Medical Computer Science, Stanford University, 1991.
- [180] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series, Taylor & Francis, 1984.
- [181] J. R. Quinlan, “Simplifying decision trees,” *Int. J. Man-Mach. Stud.*, vol. 27, pp. 221–234, Sept. 1987.
- [182] J. C. Schlimmer and D. Fisher, “A case study of incremental concept induction,” in *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI’86, p. 496–501, AAAI Press, 1986.

- [183] P. E. UTGOFF, “Id5: An incremental id3,” in *Machine Learning Proceedings 1988* (J. Laird, ed.), pp. 107–120, San Francisco (CA): Morgan Kaufmann, 1988.
- [184] P. E. Utgoff, “Incremental induction of decision trees,” *Mach. Learn.*, vol. 4, p. 161–186, Nov. 1989.
- [185] C. J. Watkins, *Models of Delayed Reinforcement Learning*. PhD thesis, Psychology Department, Cambridge University, 1989.
- [186] A. Jennings, “A learning system for communications network configuration,” *Engineering Applications of Artificial Intelligence*, vol. 1, no. 3, pp. 151 – 160, 1988.
- [187] K. J. Macleish, “Mapping the integration of artificial intelligence into telecommunications,” *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 892–898, Jun 1988.
- [188] L. Bernstein and C. M. Yuhas, “How technology shapes network management,” *IEEE Network*, vol. 3, pp. 16–19, July 1989.
- [189] D. J. Montana and L. Davis, “Training feedforward neural networks using genetic algorithms,” in *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI’89*, (San Francisco, CA, USA), pp. 762–767, Morgan Kaufmann Publishers Inc., 1989.
- [190] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, pp. 1942–1948 vol.4, Nov 1995.
- [191] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.
- [192] J. R. Quinlan, “Learning with continuous classes,” in *Proceedings of Australian Joint Conference on Artificial Intelligence*, pp. 343–348, World Scientific, November 1992.
- [193] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.
- [194] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.
- [195] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT ’92*, (New York, NY, USA), pp. 144–152, ACM, 1992.
- [196] N. A. Syed, H. Liu, and K. K. Sung, “Handling concept drifts in incremental learning with support vector machines,” in *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’99*, (New York, NY, USA), p. 317–321, Association for Computing Machinery, 1999.

- [197] G. A. Rummery and M. Niranjan, “On-line Q-learning using connectionist systems,” CUED/F-INFENG/TR 166, Cambridge University Engineering Department, Sept. 1994.
- [198] N. Littlestone and M. K. Warmuth, “The weighted majority algorithm,” in *30th Annual Symposium on Foundations of Computer Science*, pp. 256–261, Oct 1989.
- [199] R. E. Schapire, “The strength of weak learnability,” *Mach. Learn.*, vol. 5, pp. 197–227, July 1990.
- [200] L. Breiman, “Bagging predictors,” *Mach. Learn.*, vol. 24, pp. 123–140, Aug. 1996.
- [201] T. K. Ho, “Random decision forests,” in *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR ’95, (Washington, DC, USA), pp. 278–, IEEE Computer Society, 1995.
- [202] Y. Freund and R. E. Schapire, “Experiments with a new boosting algorithm,” in *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML’96, (San Francisco, CA, USA), pp. 148–156, Morgan Kaufmann Publishers Inc., 1996.
- [203] J. H. Friedman, “Stochastic gradient boosting,” *Comput. Stat. Data Anal.*, vol. 38, pp. 367–378, Feb. 2002.
- [204] S. N. Srihari and E. J. Kuebert, “Integration of hand-written address interpretation technology into the united states postal service remote computer reader system,” in *Proceedings of the 4th International Conference on Document Analysis and Recognition*, ICDAR ’97, (Washington, DC, USA), pp. 892–896, IEEE Computer Society, 1997.
- [205] ACM Special Interest Group on KDD, “KDD cup archives.” KDD. <http://www.kdd.org/kdd-cup> (accessed Oct. 19, 2021).
- [206] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., 1 ed., 1997.
- [207] J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” *Ann. Statist.*, vol. 29, pp. 1189–1232, 10 2001.
- [208] P. Geurts, D. Ernst, and L. Wehenkel, “Extremely randomized trees,” *Machine Learning*, vol. 63, pp. 3–42, Apr 2006.
- [209] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” *CoRR*, vol. abs/1603.02754, 2016.
- [210] G. E. Hinton, S. Osindero, and Y. W. Teh, “A fast learning algorithm for deep belief nets,” *Neural Computation*, vol. 18, pp. 1527–1554, July 2006.

- [211] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, “Greedy layer-wise training of deep networks,” in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS’06, (Cambridge, MA, USA), pp. 153–160, MIT Press, 2006.
- [212] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun, “Efficient learning of sparse representations with an energy-based model,” in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS’06, (Cambridge, MA, USA), pp. 1137–1144, MIT Press, 2006.
- [213] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, pp. 529–533, Feb. 2015.
- [214] G. Tesauro, “Practical issues in temporal difference learning,” *Machine Learning*, vol. 8, pp. 257–277, May 1992.
- [215] L.-J. Lin, *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1992. UMI Order No. GAX93-22750.
- [216] P. Domingos and G. Hulten, “Mining high-speed data streams,” in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’00, (New York, NY, USA), p. 71–80, Association for Computing Machinery, 2000.
- [217] G. Hulten, L. Spencer, and P. Domingos, “Mining time-changing data streams,” in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’01, (New York, NY, USA), p. 97–106, Association for Computing Machinery, 2001.
- [218] S. Nishimura, M. Terabe, K. Hashimoto, and K. Mihara, “Learning higher accuracy decision trees from concept drifting data streams,” in *Proceedings of the 21st International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems: New Frontiers in Applied Artificial Intelligence*, IEA/AIE ’08, (Berlin, Heidelberg), p. 179–188, Springer-Verlag, 2008.
- [219] A. Bifet and R. Gavaldà, “Adaptive learning from evolving data streams,” in *Advances in Intelligent Data Analysis VIII* (N. M. Adams, C. Robardet, A. Siebes, and J.-F. Boulicaut, eds.), (Berlin, Heidelberg), pp. 249–260, Springer Berlin Heidelberg, 2009.
- [220] B. Pfahringer, G. Holmes, and R. Kirkby, “New options for hoeffding trees,” in *Proceedings of the 20th Australian Joint Conference on Advances in Artificial Intelligence*, AI’07, (Berlin, Heidelberg), p. 90–99, Springer-Verlag, 2007.

- [221] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfahringer, G. Holmes, and T. Abdesslem, “Adaptive random forests for evolving data stream classification,” *Machine Learning*, vol. 106, pp. 1–27, 10 2017.
- [222] N. Oza, “Online bagging and boosting,” in *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 2340–2345 Vol. 3, 2005.
- [223] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, “Online passive-aggressive algorithms,” *J. Mach. Learn. Res.*, vol. 7, p. 551–585, Dec. 2006.
- [224] J. a. Gama and P. Kosina, “Learning decision rules from data streams,” in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI’11, p. 1255–1260, AAAI Press, 2011.
- [225] D. Brzeziński and J. Stefanowski, “Accuracy updated ensemble for data streams with concept drift,” in *Hybrid Artificial Intelligent Systems* (E. Corchado, M. Kurzyński, and M. Woźniak, eds.), (Berlin, Heidelberg), pp. 155–163, Springer Berlin Heidelberg, 2011.
- [226] D. Brzezinski and J. Stefanowski, “Combining block-based and online methods in learning ensembles from concept drifting data streams,” *Information Sciences*, vol. 265, pp. 50–67, 2014.
- [227] P. Domingos, *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Basic Books, Basic Books, 2015.
- [228] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett, *et al.*, “Knowledge-defined networking,” *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 2–10, 2017.
- [229] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, “A knowledge plane for the internet,” in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 3–10, 2003.
- [230] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart, “An architectural approach to autonomic computing,” in *International Conference on Autonomic Computing, 2004. Proceedings.*, pp. 2–9, May 2004.
- [231] T. Zimmerman and S. Kambhampati, “Learning-assisted automated planning: looking back, taking stock, going forward,” *AI Magazine*, vol. 24, no. 2, pp. 73–96, 2003.
- [232] W. Xia, P. Zhao, Y. Wen, and H. Xie, “A survey on data center networking (dcn): Infrastructure and operations,” *IEEE Communications Surveys Tutorials*, vol. 19, pp. 640–656, Firstquarter 2017.
- [233] K. Chen, C. Hu, X. Zhang, K. Zheng, Y. Chen, and A. V. Vasilakos, “Survey on routing in data centers: insights and future directions,” *IEEE Network*, vol. 25, pp. 6–10, July 2011.

- [234] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica, “A cost comparison of datacenter network architectures,” in *Proceedings of the 6th International Conference, Co-NEXT '10*, (New York, NY, USA), pp. 16:1–16:12, ACM, 2010.
- [235] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, “V12: A scalable and flexible data center network,” in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, (New York, NY, USA), pp. 51–62, ACM, 2009.
- [236] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, “Jellyfish: Networking data centers randomly,” in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, (Berkeley, CA, USA), pp. 17–17, USENIX Association, 2012.
- [237] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, “Bcube: A high performance, server-centric network architecture for modular data centers,” in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, (New York, NY, USA), pp. 63–74, ACM, 2009.
- [238] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, “Dcell: A scalable and fault-tolerant network structure for data centers,” in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, (New York, NY, USA), pp. 75–86, ACM, 2008.
- [239] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer, “Firefly: A reconfigurable wireless data center fabric using free-space optics,” in *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, (New York, NY, USA), pp. 319–330, ACM, 2014.
- [240] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, “Helios: A hybrid electrical/optical switch architecture for modular data centers,” in *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, (New York, NY, USA), pp. 339–350, ACM, 2010.
- [241] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Bov-ing, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, “Jupiter rising: A decade of clos topologies and centralized control in google’s datacenter network,” in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, (New York, NY, USA), pp. 183–197, ACM, 2015.
- [242] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, (New York, NY, USA), pp. 63–74, ACM, 2008.

- [243] D. O. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao, “Overview and Principles of Internet Traffic Engineering,” RFC 3272, Internet Engineering Task Force, May 2002.
- [244] R. Zhang-Shen and N. McKeown, “Designing a predictable internet backbone with valiant load-balancing,” in *Proceedings of the 13th International Conference on Quality of Service, IWQoS’05*, (Berlin, Heidelberg), pp. 178–192, Springer-Verlag, 2005.
- [245] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, “Improving datacenter performance and robustness with multipath tcp,” in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM ’11, (New York, NY, USA), pp. 266–277, ACM, 2011.
- [246] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, “Detail: Reducing the flow completion time tail in datacenter networks,” in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM ’12, (New York, NY, USA), pp. 139–150, ACM, 2012.
- [247] S. Sinha, S. Kandula, and D. Katabi, “Harnessing TCPs Burstiness using Flowlet Switching,” in *3rd ACM SIGCOMM Workshop on Hot Topics in Networks (Hot-Nets)*, (San Diego, CA), November 2004.
- [248] Z. Liu, D. Gao, Y. Liu, and H. Zhang, “An enhanced scheduling mechanism for elephant flows in sdn-based data center,” in *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*, pp. 1–5, Sept 2016.
- [249] C. Y. Lin, C. Chen, J. W. Chang, and Y. H. Chu, “Elephant flow detection in datacenters using openflow-based hierarchical statistics pulling,” in *2014 IEEE Global Communications Conference*, pp. 2264–2269, Dec 2014.
- [250] P. Phaál, S. Panchen, and N. McKee, “Inmon corporation’s sflow: A method for monitoring traffic in switched and routed networks,” RFC 3176, Internet Engineering Task Force, Sept. 2001.
- [251] L. A. Dias Knob, R. P. Esteves, L. Z. Granville, and L. M. R. Tarouco, “Sdefix — identifying elephant flows in sdn-based ixp networks,” in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 19–26, 2016.
- [252] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto, “Identifying elephant flows through periodically sampled packets,” in *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement, IMC ’04*, (New York, NY, USA), pp. 115–120, ACM, 2004.
- [253] P. Xiao, W. Qu, H. Qi, Y. Xu, and Z. Li, “An efficient elephant flow detection with cost-sensitive in sdn,” in *2015 1st International Conference on Industrial Networks and Intelligent Systems (INISCom)*, pp. 24–28, 2015.

- [254] MAWI Working Group, “Packet traces from WIDE backbone.” WIDE Project. <http://mawi.wide.ad.jp/mawi/> (accessed Oct. 19, 2021).
- [255] T. Benson, “Data set for IMC 2010 data center measurement.” University of Wisconsin-Madison. http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html (accessed Oct. 19, 2021).
- [256] D. Kotz, T. Henderson, I. Abyzov, and J. Yeo, “The dartmouth/campus dataset (v. 2009-09-09).” CRAWDAD, Sept. 2009. <http://crawdad.org/dartmouth/campus/20090909> (accessed Oct. 19, 2021).
- [257] Z. Liu, D. Gao, Y. Liu, H. Zhang, and C. H. Foh, “An adaptive approach for elephant flow detection with the rapidly changing traffic in data center network,” *International Journal of Network Management*, vol. 27, no. 6, pp. e1987–n/a, 2017. e1987 nem.1987.
- [258] T. Benson, A. Anand, A. Akella, and M. Zhang, “Microte: Fine grained traffic engineering for data centers,” in *Proceedings of the Seventh CONference on Emerging Networking EXperiments and Technologies*, CoNEXT ’11, (New York, NY, USA), pp. 8:1–8:12, ACM, 2011.
- [259] L. Nie, D. Jiang, L. Guo, S. Yu, and H. Song, “Traffic matrix prediction and estimation based on deep learning for data center networks,” in *2016 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6, Dec 2016.
- [260] N. L. S. da Fonseca and R. Boutaba, *Cloud Services, Networking, and Management*. Hoboken, NJ, USA: John Wiley & Sons, 1st ed., 2015.
- [261] A. Zarek, “Openflow timeouts demystified,” Master’s thesis, Department of Computer Science, University of Toronto, ON, Canada, 2012.
- [262] F. Estrada-Solano, “NELLY datasets.” GitHub. <https://github.com/festradasolano/nelly/tree/master/nelly-ml/analysis/datasets> (accessed Oct. 19, 2021).
- [263] D. Chicco and G. Jurman, “The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation,” *BMC Genomics*, vol. 21, pp. 6–, Jan. 2020.
- [264] S. Boughorbel, F. Jarray, and M. El-Anbari, “Optimal classifier for imbalanced data using matthews correlation coefficient metric,” *PLOS ONE*, vol. 12, pp. 1–17, June 2017.
- [265] D. Chicco, “Ten quick tips for machine learning in computational biology,” *BioData Mining*, vol. 10, pp. 1–17, Dec. 2017.
- [266] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, “MOA: massive online analysis,” *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, Aug. 2010.

- [267] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, “Data center tcp (dctcp),” *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 63–74, Aug. 2010.
- [268] M. Miglierina, “Application deployment and management in the cloud,” in *SYNASC*, (Timisoara, Romania), pp. 422–428, Sept. 2014.
- [269] F. F. Brasser, M. Bucicoiu, and A.-R. Sadeghi, “Swap and play: Live updating hypervisors and its application to xen,” in *ACM CCSW*, (Scottsdale, AZ, USA), pp. 33–44, Nov. 2014.
- [270] M. Afaq, S. U. Rehman, and W.-C. Song, “A Framework for Classification and Visualization of Elephant Flows in SDN-Based Networks,” *Procedia Computer Science*, vol. 65, pp. 672–681, 2015.
- [271] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, “Can the production network be the testbed?,” in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI’10, (USA), p. 365–378, USENIX Association, 2010.
- [272] B. Nunes Astuto, M. Mendonça, X. Nam Nguyen, K. Obraczka, and T. Turetti, “A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks,” *HAL Archive*, vol. 16, no. 3, pp. 1617–1634, 2014.
- [273] P. Song, Y. Liu, T. Liu, and D. Qian, “Controller-proxy: Scaling network management for large-scale SDN networks,” *Elsevier Computer Communications*, vol. 108, pp. 52–63, 2017.
- [274] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, “Efficient traffic splitting on commodity switches,” *CoNEXT*, pp. 1–13, 2015.
- [275] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: Rapid prototyping for software-defined networks,” in *ACM SIGCOMM*, Hotnets-IX, (New York, NY, USA), pp. 19:1–19:6, 2010.
- [276] M. Kerrisk, “ip-netns(8).” Linux manual page, Aug. 2021. <https://man7.org/linux/man-pages/man8/ip-netns.8.html> (accessed Oct. 19, 2021).
- [277] Open vSwitch, “What is open vswitch?.” Linux Foundation Collaborative Projects. <https://docs.openvswitch.org/en/latest/intro/what-is-ovs/> (accessed Oct. 19, 2021).
- [278] M. Kerrisk, “veth(4).” Linux manual page, Aug. 2021. <https://man7.org/linux/man-pages/man4/veth.4.html> (accessed Oct. 19, 2021).
- [279] Aruba, “Data sheet aruba 2920 switch series.” Aruba Networks. https://www.arubanetworks.com/assets/ds/DS_2920SwitchSeries.pdf (accessed Oct. 19, 2021).

- [280] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, “PortLand : A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric,” *ACM SIGCOMM*, pp. 39–50, 2009.
- [281] IEEE Standards Association, “Standard group mac addresses: A tutorial guide.” IEEE. <https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/tutorials/macgrp.pdf> (accessed Oct. 19, 2021).
- [282] F. Estrada-Solano, “iRide Ryu SDN applications.” GitHub. <https://github.com/festradasolano/iride/tree/master/ryu> (accessed Dec. 13, 2021).
- [283] W. Sarle, “Should i normalize/standardize/rescale the data?.” faqs.org, Mar. 2014. <http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-16.html> (accessed Dec. 19, 2021).
- [284] W. Sarle, “Why not code binary inputs as 0 and 1?.” faqs.org, Mar. 2014. <http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-8.html> (accessed Dec. 19, 2021).
- [285] T. O. Kvalseth, “Cautionary note about r^2 ,” *The American Statistician*, vol. 39, no. 4, pp. 279–285, 1985.
- [286] scikit-learn developers, “Metrics and scoring: quantifying the quality of prediction.” scikit-learn, 2021. https://scikit-learn.org/stable/modules/model_evaluation.html (accessed Dec. 19, 2021).
- [287] DeepLearning.AI, “Train/dev/test sets.” YouTube, Aug. 2017. <https://www.youtube.com/watch?v=1waHlpKiNyY> (accessed Dec. 19, 2021).
- [288] J. a. Gama, R. Sebastião, and P. P. Rodrigues, “Issues in evaluation of stream learning algorithms,” in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’09, (New York, NY, USA), p. 329–338, Association for Computing Machinery, 2009.
- [289] scikit-learn developers, “Preprocessing data.” scikit-learn, 2021. <https://scikit-learn.org/stable/modules/preprocessing.html> (accessed Dec. 19, 2021).
- [290] P. Branco, L. Torgo, and R. P. Ribeiro, “SMOGL: a pre-processing approach for imbalanced regression,” in *Proceedings of First International Workshop on Learning with Imbalanced Domains: Theory and Applications* (P. B. Luís Torgo and N. Moniz, eds.), vol. 74 of *Proceedings of Machine Learning Research*, pp. 36–50, PMLR, 22 Sep 2017.
- [291] L. Torgo, P. Branco, R. P. Ribeiro, and B. Pfahringer, “Resampling strategies for regression,” *Expert Systems*, vol. 32, no. 3, pp. 465–476, 2015.
- [292] L. Torgo, R. P. Ribeiro, B. Pfahringer, and P. Branco, “Smote for regression,” in *Progress in Artificial Intelligence* (L. Correia, L. P. Reis, and J. Cascalho, eds.), (Berlin, Heidelberg), pp. 378–389, Springer Berlin Heidelberg, 2013.

- [293] P. Branco, R. P. Ribeiro, and L. Torgo, “Ubl: an r package for utility-based learning,” 2016.
- [294] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [295] J. Montiel, J. Read, A. Bifet, and T. Abdesslem, “Scikit-multiflow: A multi-output streaming framework,” *Journal of Machine Learning Research*, vol. 19, no. 72, pp. 1–5, 2018.
- [296] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, and J. a. Gama, “Machine learning for streaming data: State of the art, challenges, and opportunities,” *SIGKDD Explor. Newsl.*, vol. 21, p. 6–22, nov 2019.
- [297] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Software available from tensorflow.org.
- [298] F. Chollet *et al.*, “Keras.” Keras, 2015. <https://keras.io> (accessed Dec. 19, 2021).
- [299] DeepLearning.AI, “Weight initialization in a deep network.” YouTube, Aug. 2017. <https://www.youtube.com/watch?v=s2coXdufOzE> (accessed Dec. 19, 2021).
- [300] DeepLearning.AI, “Adam optimization algorithm.” YouTube, Aug. 2017. https://www.youtube.com/watch?v=JXQT_vxqwIs (accessed Dec. 19, 2021).
- [301] DeepLearning.AI, “Understanding mini-batch gradient descent.” YouTube, Aug. 2017. https://www.youtube.com/watch?v=-_4Zi8fCZO4 (accessed Dec. 19, 2021).
- [302] A. F. Agarap, “Deep learning using rectified linear units (relu),” 2018.
- [303] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” 2015.
- [304] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014.
- [305] F. Chollet *et al.*, “Model training apis.” Keras, 2015. https://keras.io/api/models/model_training_apis/ (accessed Dec. 19, 2021).
- [306] DeepLearning.AI, “Bias/variance.” YouTube, Aug. 2017. <https://www.youtube.com/watch?v=SjQyLhQIXSM> (accessed Dec. 19, 2021).

- [307] DeepLearning.AI, “Regularization.” YouTube, Aug. 2017. <https://www.youtube.com/watch?v=6g0t3Phly2M> (accessed Dec. 19, 2021).
- [308] DeepLearning.AI, “Dropout regularization.” YouTube, Aug. 2017. <https://www.youtube.com/watch?v=D8PJAL-MZv8> (accessed Dec. 19, 2021).
- [309] J. Boyar, L. Epstein, L. M. Favrholt, J. S. Kohrt, K. S. Larsen, M. M. Pedersen, and S. Wøhlk, “The maximum resource bin packing problem,” *Theoretical Computer Science*, vol. 362, no. 1, pp. 127–139, 2006.
- [310] F. Klassen and AppNeta, “Tcpreplay - pcap editing and replaying utilities.” Tcpreplay. <https://tcpreplay.appneta.com/> (accessed Dec. 19, 2021).
- [311] G. Roualland, “ifstat(1) - linux man page.” die.net. <https://linux.die.net/man/1/ifstat> (accessed Dec. 19, 2021).
- [312] X. Mertens, “Anonymous packet capture.” /dev/random, May 2008. <https://blog.rootshell.be/2008/05/01/anonymous-packet-capture/> (accessed Dec. 19, 2021).
- [313] B. Schmidt, J. González-Domínguez, C. Hundt, and M. Schlarb, “Chapter 2 - theoretical background,” in *Parallel Programming* (B. Schmidt, J. González-Domínguez, C. Hundt, and M. Schlarb, eds.), pp. 21–45, Morgan Kaufmann, 2018.
- [314] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da Fonseca, “Drsir: A deep reinforcement learning approach for routing in software-defined networking,” *IEEE Transactions on Network and Service Management*, pp. 1–1, 2021.
- [315] E. F. Castillo, O. M. C. Rendon, A. Ordonez, and L. Zambenedetti Granville, “Ipro: An approach for intelligent sdn monitoring,” *Computer Networks*, vol. 170, p. 107108, 2020.
- [316] W. Villota, M. Gironza, A. Ordoñez, and O. M. Caicedo Rendon, “On the feasibility of using hierarchical task networks and network functions virtualization for managing software-defined networks,” *IEEE Access*, vol. 6, pp. 38026–38040, 2018.

Appendix A

Scientific Production

The research work presented in this thesis was reported to the scientific community through paper submissions to renowned conferences and journals. The process of doing research, submitting paper, gathering feedback, and improving the work helped to achieve the maturity hereby presented. The list of published papers to date are listed below in chronological order.

1. “A Framework for SDN Integrated Management based on a CIM Model and a Vertical Management Plane,” published in *Computer Communications*, 2017.
2. “SDN Management Based on Hierarchical Task Network and Network Functions Virtualization,” published in the proceedings of the 2017 IEEE Symposium on Computers and Communications (ISCC).
3. “A YANG Model for a vertical SDN Management Plane,” published in the proceedings of the 2017 IEEE Colombian Conference on Communications and Computing (COLCOM).
4. “Machine Learning for Cognitive Network Management,” published in *IEEE Communications Magazine*, 2018.
5. “A comprehensive survey on machine learning for networking: evolution, applications and research opportunities,” published in *Journal of Internet Services and Applications*, 2018.
6. “An Efficient Mice Flow Routing Algorithm for Data Centers based on Software-Defined Networking,” published in the proceedings of the 2019 IEEE International Conference on Communications (ICC).
7. “NELLY: Flow Detection Using Incremental Learning at the Server Side of SDN-based Data Centers,” published in *IEEE Transactions on Industrial Informatics*, 2020.
8. “An Approach based on YANG for SDN Management,” published in *International Journal of Communication Systems*, 2021.

There is other paper that is still under construction.

1. “iRIDE: Rescheduling of Elephant Flows in SDN-based Data Centers Using Incremental Deep Learning to Predict Traffic Characteristics,” in construction.

The published papers are available in the next pages.



Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A framework for SDN integrated management based on a CIM model and a vertical management plane



Felipe Estrada-Solano^{a,*}, Armando Ordonez^b, Lisandro Zambenedetti Granville^c,
Oscar Mauricio Caicedo Rendon^a

^a Telematics Engineering Group, Telematics Department, University of Cauca, Calle 5 No. 4-70, Popayan, CA, Colombia

^b Intelligent Management Systems Group, Foundation University of Popayan, Calle 5 No. 8-58, Popayan, CA, Colombia

^c Computer Networks Group, Institute of Informatics, Federal University of Rio Grande do Sul, Av. Bento Gonçalves, 9500 Porto Alegre, RS, Brazil

ARTICLE INFO

Article history:

Received 12 February 2016

Revised 12 August 2016

Accepted 13 August 2016

Available online 16 August 2016

Keywords:

Common information model

Software-defined networking

Heterogeneous environments

Resource characterization

Configuration management

ABSTRACT

The Software-Defined Networking (SDN) paradigm establishes a typical three-plane architecture (*i.e.*, Data, Control, and Application planes) that facilitates the deployment of network functions and simplifies traditional network management tasks. However, SDN lacks an integrated or standardized framework for managing its architecture. Some investigations have addressed such shortage by proposing different solutions that tackle specific management requirements for particular SDN technology instances. This isolated approach forces network administrators to use multiple frameworks to achieve a complete SDN management that is complex and time-consuming in heterogeneous environments. In this paper, we introduce an information model based on the common information model that establishes a technology-agnostic and consistent characterization of the SDN architecture. Such information model represents the core towards building a Management Plane aimed to facilitate the integrated SDN management in heterogeneous environments. To test our information model, we developed a prototype and conducted a performance evaluation in an SDN configuration scenario that deploys different managing technologies. The obtained results provide directions that corroborate the feasibility of our approach (in terms of time-response and network traffic) for configuring heterogeneous SDNs.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Over the past 20 years, programmable networks have evolved as a key driver to innovate and to cope with complexity and management in computer networks. Nowadays, Software-Defined Networking (SDN) paradigm is an attractive trend to program networks in both research and industry [1,2]. From a high-level point of view, SDN separates control and forwarding planes, allowing to operate networks in a simpler way from a logically centralized software program often referred to as controller [3].

SDN standardization bodies (*e.g.*, Open Network Foundation [ONF] and Linux Foundation) and networking vendors (*e.g.*, Cisco and Juniper) describe a typical SDN architecture as three horizontal planes [4–7]: (i) a lower Data Plane to forward packets, (ii) a middle Control Plane to compile decision policies and to enforce them on the Data Plane through Southbound Interfaces (SBI);

and (iii) an upper Application Plane to orchestrate business functions and high-level services that manage network behavior using Northbound Interfaces (NBI) provided by the Control Plane. Additionally, the SDN architecture includes East/Westbound Interfaces (EWBI) to enable deploying a distributed Control Plane.

At the top of the SDN architecture, a lot of research has proposed services and applications to simplify traditional network management tasks, such as load-balancing [8], efficient energy usage [9], and access control [10,11]. Furthermore, some studies have addressed the need to manage the SDN architecture, including, for example, frameworks to configure the Data Plane [12,13], to deploy [14,15] and monitor [16,17] the Control Plane, to virtualize SDNs [18,19], and to develop the Application Plane [20,21]. However, to the best of our knowledge, no integrated solution exists to manage SDN as a whole by employing well-defined interfaces and supporting different deployment technologies.

The lack of frameworks for integrated network management forces network administrators to handle several isolated solutions to manage resources from distinct planes of the SDN architecture as well as various technology instances. Thus, SDN management remains complex and time-consuming because of the diversity of

* Corresponding author.

E-mail addresses: festradasolano@unicauca.edu.co (F. Estrada-Solano), jaordonez@fup.edu.co (A. Ordonez), granville@inf.ufrgs.br (L.Z. Granville), omcaicedo@unicauca.edu.co (O.M. Caicedo Rendon).

solutions. In our previous work, we evaluated the feasibility of using mashups to control and monitor SDN in heterogeneous environments by composing management applications at the top of the SDN architecture [22–25]. Still, there is the need to characterize SDN as a whole from the management perspective to accomplish a joint understanding in heterogeneous and distributed environments. Some approaches have addressed the formal representation of SDN [26–29], nonetheless, they are focused exclusively on specific SDN technology instances or fall short in modeling the SDN architecture. In a later section, we define how these approaches provide an insufficient solution for representing the SDN architecture.

Recent proposals have considered a Management Plane in the SDN architecture for carrying out Operation, Administration, and Maintenance (OAM) functions [4–7]. These proposals expose a very high-level view of their management component. We argue that is needed to extend and detail such Management Plane aiming to facilitate integrated control and monitoring of heterogeneous SDNs. As a first critical step, this Management Plane requires an information model that homogenizes management data to achieve consistency among all OAM tasks. In this paper, we specify the SDN Management Plane using the four Open System Interconnection (OSI) submodels [30] (i.e., Information, Organizational, Communication, and Functional). Then, we focus on introducing a novel information model that represents the SDN architecture from a management perspective as a Common Information Model (CIM) conceptual framework [31]. We preferred CIM over other information definition languages (e.g., Structure of Management Information [SMI] [32] and Guidelines for the Definition of Managed Objects [GDMO] [33]) because its high expressiveness affords future robust semantic integration [34]. Leveraging CIM features, we provide a technology-independent and consistent model across distinct vendors and SDN instances. Furthermore, our information model establishes a shared abstraction of managed and managing SDN resources from Data, Control, and Application planes to achieve a complete SDN management. It is noteworthy that the proposed model provides concepts and artifacts that may complement or enhance the information model structured by ONF (i.e., ONF-CIM¹) [26].

The main contributions presented in this paper are:

- An information model based on CIM that describes managed and managing SDN resources regardless of deploying technologies;
- An SDN management system prototype that is based on the above information model;
- The demonstration that our proposal is effectively feasible (in terms of time-response and network traffic) when managing an SDN deployed with heterogeneous technologies.

The remainder of this paper is organized as follows. In Section 2, it is discussed both the background and related work. In Section 3, we overview the proposed Management Plane. In Section 4, we introduce our CIM-based information model. In Section 5, we present a case study used to evaluate the proposed approach. The paper concludes in Section 6.

2. Background

In this section, first, we present the SDN architecture. Second, we discuss the related work about the SDN management.

2.1. Software-Defined Networking architecture

Multiple standardization bodies, such as ONF and Linux Foundation, focus on encouraging and normalizing open SDN frameworks. Also, various private networking vendors, such as Cisco and Juniper, offer proprietary SDN deployments. In turn, several research efforts work on improving architectural aspects of SDN. These open, proprietary, and research SDN solutions establish a typical SDN architecture [4–7] composed of three horizontal planes (i.e., Data Plane, Control Plane, and Application Plane) and three interfaces (i.e., SBI, NBI, and EWBI).

The Data Plane deploys the network infrastructure composed of interconnected hardware and software-based Network Devices (NetDev) that perform forwarding operations. A NetDev ranges from dumb switches to custom switches. A dumb switch merely carries out simple forwarding functions, such as Longest Prefix Match (LPM). For example, OpenFlow-Only switches [35] just forward packets regarding their flow tables that are updated by the Control Plane. A custom switch relies on programmable platforms (e.g., OpenWrt and NetFPGA) to integrate more complex forwarding functions, such as Network Address Translation (NAT) and firewall. For example, Forwarding Elements (FE) in ForCES [36] include multiple associated Logical Functional Blocks (LFB) to carry out such forwarding functions. An LFB defines either a punctual action for handling packets or a configuration entity operated by the Control Plane.

The Control Plane enforces decision policies on the Data Plane through SBIs. Each SBI defines the set of instructions and the communication protocols to allow the interaction between components in the Control Plane and in the Data Plane. The OpenFlow protocol is the most well-known open standard SBI because its widespread usage by vendors and research [1]. Other SBI proposals are ForCES [36] and Protocol-Oblivious Forwarding (POF) [37].

The Control Plane comprises Network Slicers (NetSlicer) and Network Operating Systems (NOS). A NetSlicer divides the underlying network infrastructure into several isolated logical network instances (a.k.a. slices), assigning their control to multiple tenant NOSs. For example, FlowVisor [38] acts as an OpenFlow proxy between switches and controllers, redirecting messages according to specific slicing dimensions, such as bandwidth and forwarding tables. An NOS compiles the network logic for instructing the underlying Data Plane and provides generic services (e.g., topology discovering and host tracking) and NBIs to the Application Plane, facilitating to add custom Network Applications (NetApp). OpenFlow Controllers [35] and ForCES Control Elements (CE) [36] are NOS instances. Although NetSlicers can be considered as a specific NOS, it is important to describe them as separate components in order to demarcate their functionality: network virtualization versus decision making. In addition, some approaches may provide network virtualization as an NOS service for multiple tenant NetApps [39,40].

As aforementioned, the Control Plane provides NBIs to the Application Plane. An NBI encompasses common Application Programming Interfaces (API) based on NOS native bundles (e.g., Floodlight Java API [41] and Ryu application API [42]), programming languages (e.g., Pyretic [20] and Procera [21]), protocols (e.g., Floodlight REST API [43]), file systems (e.g., YANC [44]), among others. The Control Plane also defines EWBI to enable information exchange between NOSs distributed in different domains. For example, the SDN Inter-networking (SDNI) Negotiation Interface, the West-East Bridge (WE-Bridge) [45], and the ForCES CE-CE interface [36].

The Application Plane consists of NetApps that deploy and orchestrate business logic and high-level network functions, such as routing policies and access control. NetApps run either locally or remotely regarding NOSs. Local NetApps prefer NBIs based on

¹ ONF-CIM is not based on the CIM specification defined by the Distributed Management Task Force (DMTF).

programming languages. Remote NetApps usually employ protocol-based APIs.

2.2. Software-Defined Networking management

Most SDN proposals have tackled traditional network management tasks by carrying out managing *functions in NetApps* at the Application Plane. For example, wildcard-based algorithms [8] to better redistribute traffic in SDN networks, ElasticTree [9] to efficiently provide energy for SDN components, and Resonance [10] and OpenRoads [11] to control access to SDN resources. However, *functions in NetApps* lack of mechanisms to deal with several management requirements from distinct SDN architectural planes, such as: (i) in the Data Plane, configure certain NetDevs to communicate with a preferred NOS, (ii) in the Control Plane, set up a NetSlicer to link NOSs to their corresponding virtual network instances; and (iii) in the Application Plane, modify business parameters to customize NetApps logic.

Some investigations have tackled the above gap by providing *isolated tools* that address specific management requirements for particular SDN technology instances. For example: (i) OpenFlow Management and Configuration Protocol (OF-CONFIG) [12] and Open vSwitch Database (OVSDb) [13] that define protocols to configure NetDevs, (ii) Kandoo [14] and HyperFlow [15] to scale and distribute NOSs, (iii) OpenFlow Management Infrastructure (OMNI) [16] and ROVIZ [17] that provide graphic interfaces to monitor NOSs, (iv) VeRTIGO [18] and ADVisor [19] to configure NetSlicers; and (v) Pyretic [20] and Procera [21] that supply development tools to build NetApps. Considering that heterogeneous SDNs deploy a variety of resources from multiple vendors and distinct technologies, it is to highlight that a classic solution based on using *isolated tools* to accomplish a complete SDN management is complex and time-consuming.

In our previous work, we assessed the feasibility of a *mashup-based* approach to allow network administrators to compose NetApps that control and monitor heterogeneous SDNs [22–25]. In such *mashup-based* approach, management applications relied on services and middlewares developed and extended by builder actors. These builder actors realized their job according to deployed SDN technology and their own managing data models, constraining such management applications to operate only on particular SDN domains. Therefore, as an essential step, an SDN management solution requires an information model that establishes a shared characterization of the entire SDN to enable integrated management in heterogeneous environments.

Few approaches have defined *information models* to characterize the SDN architecture from a management perspective: (i) ONF-CIM [26] defines a Core Information Model (CoreModel) [27] that uses the Unified Modeling Language (UML) to structure the forwarding functions of the Data Plane, (ii) Network Abstraction Model (NAM) [28] employs a building block approach to represent all resources of NetDevs; and (iii) CIM-SDN [29] proposes a CIM extension schema to model SDN. It is worth noting that these *infor-*

mation models fall short in representing the SDN architecture or are tied to specific SDN technology instances. ONF CoreModel describes only the Data Plane and was designed for OpenFlow. NAM focuses on NetDevs and is extensible enough to represent different functionalities of the SDN architecture, however, it is deeply tied to the ForCES FE Model [46]. CIM-SDN merely includes the main elements from the Data and the Control Planes. Although CIM-SDN is based on a technology-neutral model (*i.e.*, CIM), the extended schema is highly attached to the OpenFlow architecture. Finally, ONF-CIM simply includes CoreModel so far but provides a flexible environment that allows to expand and refine its structure as new insights emerge, such as the approach described in this paper.

Table 1 summarizes the targets and gaps in SDN management of the above-reviewed proposals. Unlike these proposals, we consider an SDN management approach based on a complete, technology-agnostic reference model of the SDN architecture in order to achieve integrated management in heterogeneous environments.

3. A Management Plane for SDN

To define our approach, we extend the Management Plane concept considered by recent SDN proposals for covering OAM functions omitted and restricted in the traditional SDN architecture [4–7]. For example, assigning the Data Plane resources to the corresponding control components and configuring the policies and Service Level Agreements (SLA) of the Control and Application planes. Although NOSs may implement many of these OAM functions, flooding the Control Plane with a lot of management tasks may cause low network performance. Unlike the above proposals, our Management Plane aggregates components that facilitate the integrated management of heterogeneous SDN resources. To better explain our approach, we present a high-level overview of the Management Plane using the four OSI submodels [30]: Information, Organizational, Communication, and Functional.

3.1. Overview

Fig. 1 depicts our Management Plane. This plane is formed by the Data Repository, the Manager, Adapters, Management Interfaces, and Agents. The Data Repository holds the Resource Representation Model (RRM) and serves the Manager to store management instance data. RRM handles metadata to provide an abstract, technology-neutral characterization of SDN resources. The Manager orchestrates and deploys Management Services to carry out different SDN management functions. These Management Services expose user interfaces to allow interaction of Network Administrators. Adapters afford a protocol-agnostic communication between the Manager and Agents through well-defined Management Interfaces. Each Management Interface connects both corresponding Adapter and Agent. Agents situate on SDN resources to act on behalf of the Manager. The whole operation of the Management Plane is based on RRM to achieve an integrated and technology-independent SDN management.

Table 1
Proposals on SDN management.

References	Approach	Requirements for SDN management			
		Complete architecture	Integrated management	Heterogeneous environment	Information model
[8]–[11]	Functions in NetApps				
[12]–[21]	Isolated tools	✓			
[22]–[25]	Mashup based		✓	✓	
[26]–[29]	Information models		✓		✓
–	This approach	✓	✓	✓	✓

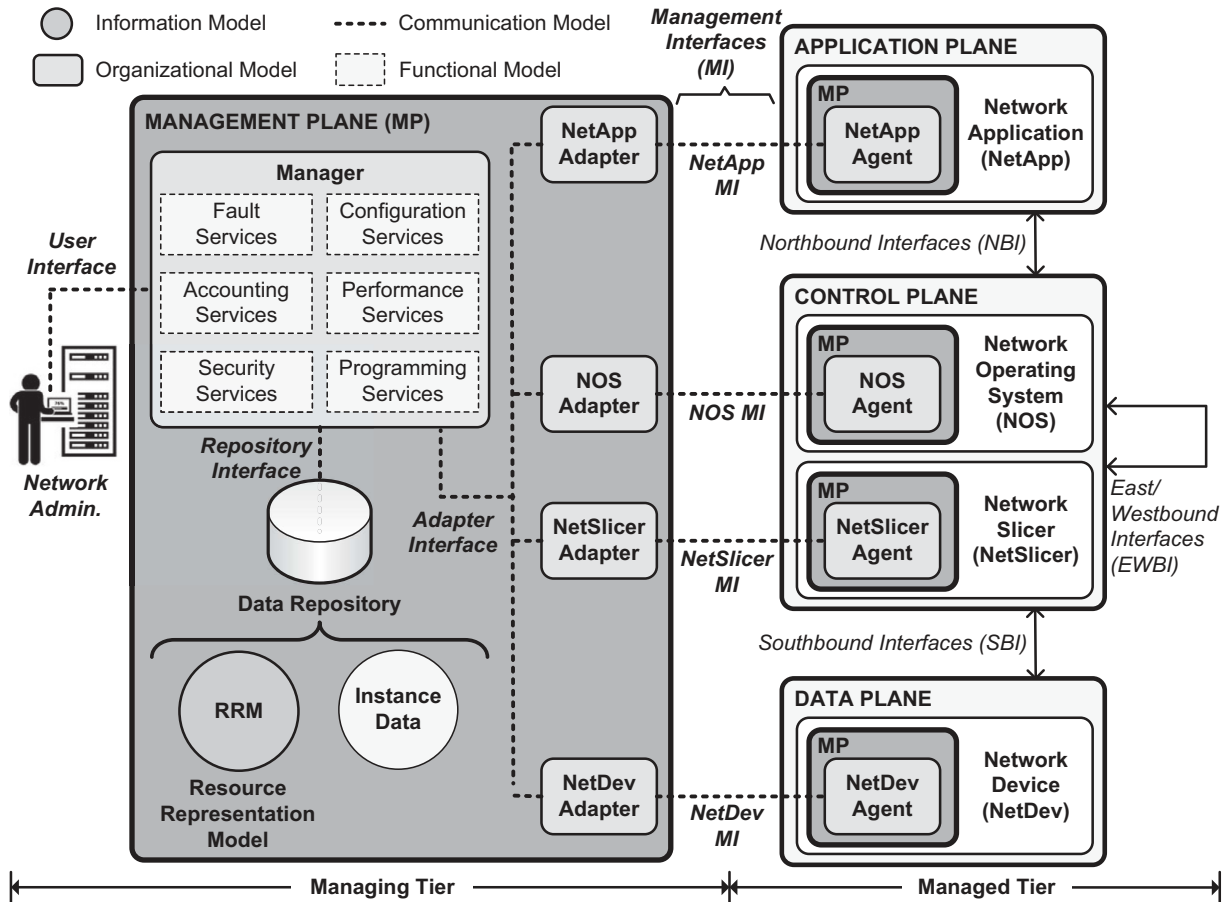


Fig. 1. High-level SDN architecture with Management Plane.

It is important to highlight that we define the Management Plane by referencing the four OSI network management submodels [30]: (i) an information model to establish a shared abstraction of SDN resources, (ii) an Organizational Model to specify roles and collaboration forms, (iii) a Communication Model to delineate the exchange of management data; and (iv) a Functional Model to structure management requirements.

3.2. Submodels

Following we overview the four submodels of the proposed Management Plane.

3.2.1. Information model

Our approach introduces a CIM model to describe the SDN architecture from a management perspective at a conceptual level regardless of deploying technologies. We use UML to graphically represent SDN resources and their relationships as CIM classes and associations, respectively. This object-oriented, well-understood abstract framework standardizes SDN management information across disparate vendors and SDN instances. Thus, enabling to carry out integrated management in heterogeneous SDNs. Further network designers may extend the proposed CIM model to include customized resource behavior. In our approach, the information model is realized by RRM in the Data Repository. We focus on the details of the proposed information model in Section 4.

3.2.2. Organizational model

We depict a two-tier like network management model that incorporates three kinds of entities (*a.k.a.* roles). A Managing Tier

that encloses *manager* and *adapter* entities, and a Managed Tier that contains *agent* entities. A *manager* entity is responsible for: (i) housing and coordinating logic of management functions, (ii) providing user interaction with deployed management functions through tailored user interfaces (*e.g.*, command-line, graphical, and Web-based); and (iii) sending requests to and receiving replies and events from *agents* by means of *adapters*. An *adapter* entity allows a *manager* to interact with any specific *agent* by parsing data formats and protocols handled by their communication interfaces (*i.e.*, Adapter Interface and Management Interfaces). An *agent* entity resides on managed resources to carry out management requests delegated by a *manager*, such as performing an operation or responding to a query. In addition, an *agent* entity may dispatch unsolicited events to a *manager*. Each organizational component in our Management Plane gets the same name as its corresponding role. The Manager acts as a *manager* entity. NetApp Adapter, NOS Adapter, NetSlicer Adapter, and NetDev Adapter play an *adapter* role. NetApp Agent, NOS Agent, NetSlicer Agent, and NetDev Agent perform *agent* tasks. We differentiate Adapters and Agents regarding of SDN managed resources (*i.e.*, NetApp, NOS, NetSlicer, and NetDev) to demarcate the communication between such kind of entities located at different architectural planes.

3.2.3. Communication model

Our Management Plane defines the User Interface, the Repository Interface, the Adapter Interface, and the set of Management Interfaces. The User Interface enables Network Administrators to interact with Management Services exposed by the Manager. The Repository Interface connects the Manager with the Data Repository. The Adapter Interface and Management Interfaces transport

request messages (i.e., operations and queries) from the Manager to a particular Agent, passing through the related Adapter. Both of these interfaces also transmit reply messages and unsolicited events sent by an Agent towards the Manager. In order to match each Agent with its respective Adapter, we establish a Management Interface (MI) per SDN managed resource: NetApp MI, NOS MI, NetSlicer MI, and NetDev MI. Regarding communication support, the User Interface and the Adapter Interface must employ a consistent data format (e.g., JavaScript Object Notation [JSON] [47] and eXtensible Markup Language [XML] [48]) and a standardized protocol (e.g., HyperText Transfer Protocol [HTTP] [49] and Simple Object Access Protocol [SOAP] [50]) to exchange management data. The Repository Interface relies on technologies deployed by the Data Repository (e.g., XML over HTTP). Finally, Management Interfaces handle data formats and protocols implemented by Agents (e.g., OVSDB [13], Network Configuration Protocol [NETCONF] [51], and Simple Network Management Protocol [SNMP] [52]).

3.2.4. Functional model

As considered by recent Management Plane approaches for SDN [5,7], this proposal also references the five OSI management functional areas [53] to classify Management Services: Fault Services, Configuration Services, Accounting Services, Performance Services, and Security Services. Fault Services detect, separate, and fix failures in physical and logical SDN resources. Configuration Services modify and update behavior of SDN resources. Accounting Services tracks and allocate usage of SDN resources. Performance Services monitor, collect, and report information about the operation of SDN resources. Security Services control and analyze access to SDN resources. In addition, we include Programming Services to coordinate programmable software of SDN resources, such as checking and deploying versions of a particular NetApp running on a specific NOS. By using or combining the aforementioned Management Services, our Management Plane allows network administrators to carry out different SDN management requirements, as those described by Wickboldt et al. [4].

4. Information model

As aforementioned, our Management Plane requires an information model that provides a technology-agnostic and consistent abstraction of the SDN architecture to enable integrated management. Few approaches provide models that attempt to characterize the SDN architecture from a management perspective [12,28,29], but they are tied to specific SDN instances and expose incomplete SDN representations.

In this paper, we introduce a CIM-based information model that provides a technology-independent and consistent abstraction of

SDN managed and managing resources. This information model represents every plane in the SDN architecture to encourage a complete SDN management regardless of deploying technologies. We adopted CIM because it offers higher expressiveness than other information definition languages (e.g., SMI [32] and GDMO [33]), affording future robust semantic integration [34]. CIM supplies several classes, associations, properties, and methods to describe network resources at a conceptual level, such as Ethernet ports, LAN endpoints, and VLANs [54,55]. However, CIM lacks elements that represent specific SDN features for management [29]. Thus, our information model introduces new elements that extend the actual CIM Schema to characterize the SDN architecture from a management perspective. We present this extended schema as a graphical visualization composed of UML classes and associations that represent SDN managed and managing resources and their relationships. Although CIM schemas can be considered as a Data Model, their graphical notation support (i.e., UML) provides a nearby approach to an information model [56]. In fact, the extended CIM schema described in this paper includes important aspects of information models: the independence of particular implementations or protocols and the relationships between managed objects.

In next paragraphs and figures (Figs. 2–5) we describe a simple version of the proposed information model. Specific properties and methods from each class are out of scope. We exclude the *CIM_* prefix from the current CIM elements and the *SDN_* prefix from the new elements. For example, *CIM_System* appears as *System* and *SDN_AgentService* as *AgentService*. To provide a better visualization, the proposed class schema displays gray background for the new classes, white background for the current CIM classes, bold characters for the new associations, and thin characters for the current CIM associations. It is noteworthy that this class schema represents the main contribution of this paper, particularly the new classes and the new associations. For the sake of simplicity, we omit inheritance associations between the new classes and the current CIM classes. Unless otherwise stated, general inheritance associations satisfy the following: (i) the new classes with suffix *Capabilities* represent subclasses from the *EnabledLogicalElementCapabilities* CIM class, (ii) with suffix *Service* from the *Service* CIM class; and (iii) with suffix *Settings* from the *SettingData* CIM class. In addition, we skip the *BindsTo* CIM associations for the CIM classes *ServiceAccessPoint* and *ProtocolEndpoint*. The *BindsTo* association connects the class itself to define a protocol layering dependency between an upper and a lower protocol. For example, the OpenFlow protocol binds the TCP protocol to set the port and address enabled for OpenFlow communication.

Fig. 2 illustrates the extended class schema for the proposed Management Plane. We introduced five new classes to characterize the novel components defined in this approach: in the

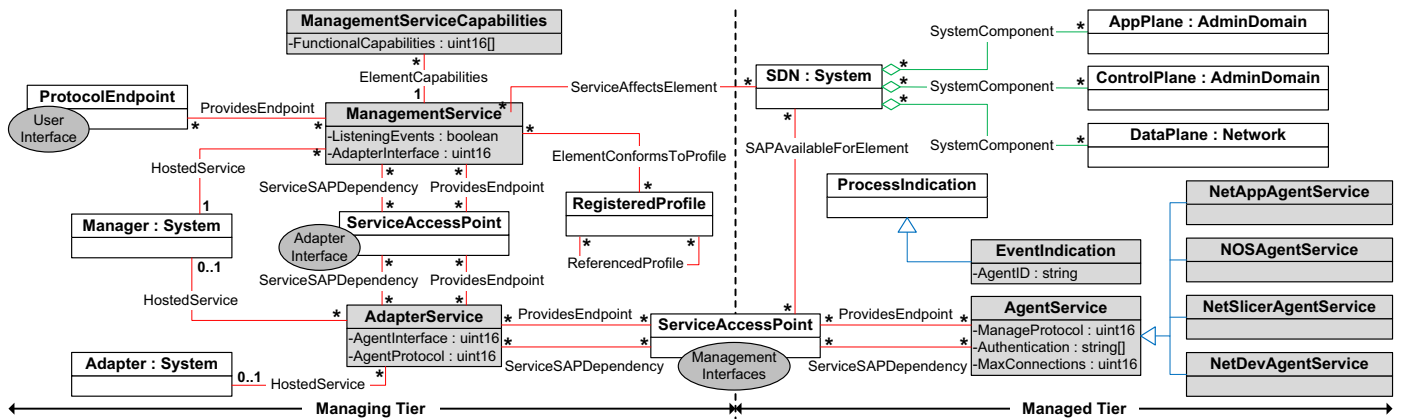


Fig. 2. Class schema to model SDN Management Plane.

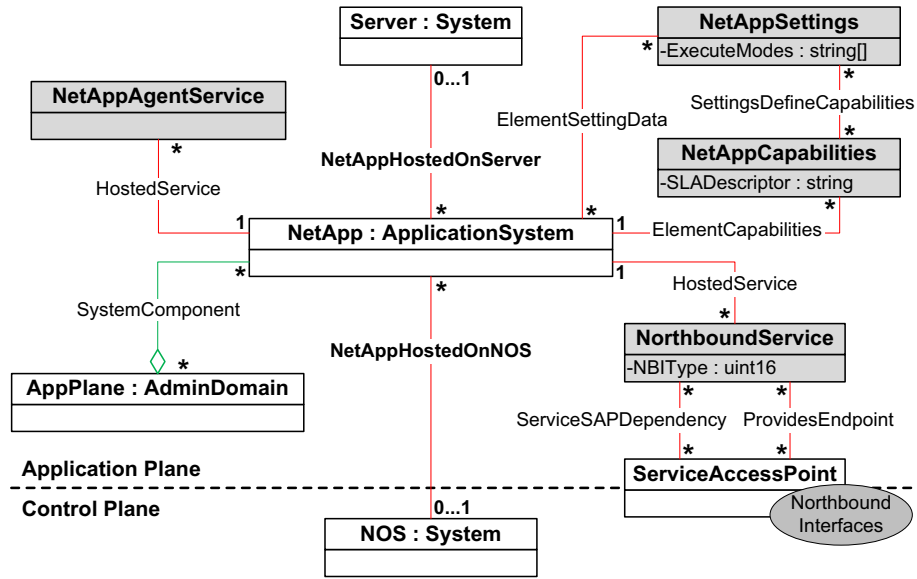


Fig. 3. Class schema to model SDN Application Plane.

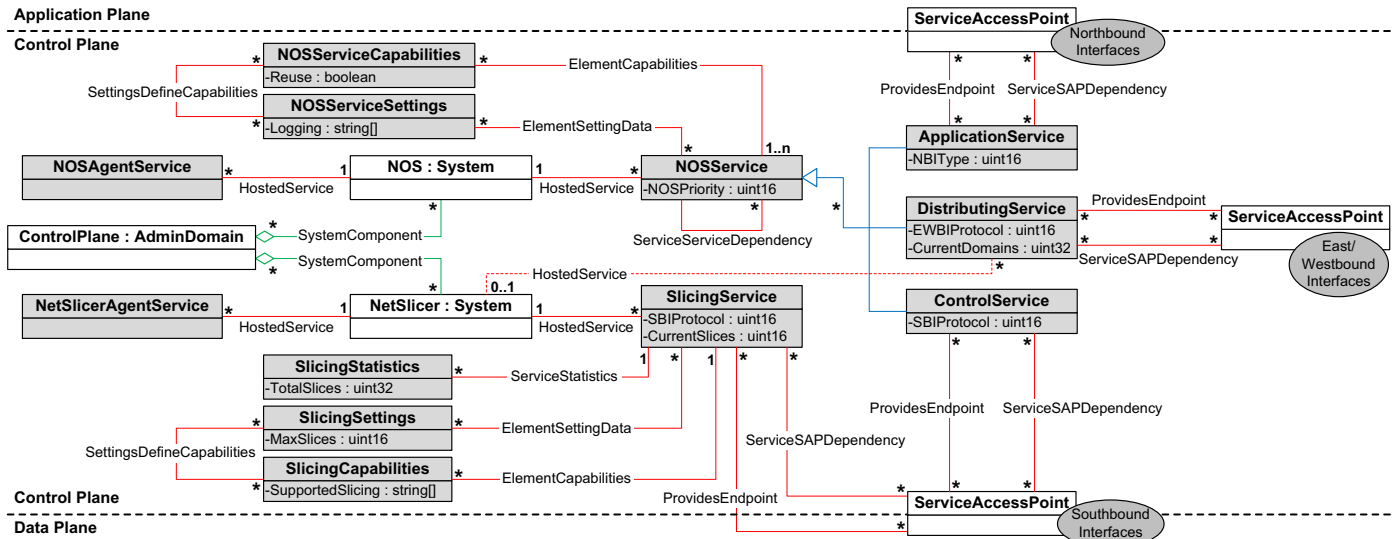


Fig. 4. Class schema to model SDN Control Plane.

Managing Tier, the *ManagementService*, the *ManagementServiceCapabilities*, and the *AdapterService*; and in the Managed Tier, the *AgentService* and the *EventIndication*.

The *ManagementService* class represents Management Services that allow carrying out different SDN management functions. Through the *ElementCapabilities* association, the *ManagementServiceCapabilities* class describes both supported and excluded abilities for Management Services. For example, the property *FunctionalCapabilities* establishes a numeric value map based on the Functional Model for classifying SDN Management Services as Fault, Configuration, Accounting, Performance, Security, or Programming services (see Section 3.2.4). Thus, a Management Service that modifies the SBI communication of NetDevs may declare capabilities of Configuration Services.

The *RegisteredProfile* class models a CIM profile specification defined by any standard organization for managing SDNs. Each profile specification includes a small subset of the proposed class schema and delineates corresponding behavior as management requirements. The *ReferencedProfile* association indicates that a profile specification may reference others. In addition, the *ElementConformsToProfile* association describes which CIM profile

specifications a Management Service apply. For example, a Configuration Service fulfills with a profile specification of DMTF that standardizes how to achieve seamless migration in NetDevs.

The *Manager* represents the system hosting the SDN Management Services. The *HostedService* association realizes this relationship between the *ManagementService* and the *Manager*. Although this model presents the *Manager* as an instance of the *System* class, it also may implement an instance of a subclass from *System*, such as *ComputerSystem*, *J2eeServer*, or a new class. For example, a Configuration Service may be carried out as a Web application running on either an Apache Tomcat Server or a GlassFish Server.

The *ProtocolEndpoint* class tagged as *User Interface* models the communication point that enables access of Network Administrators. The corresponding *ProvidesEndpoint* association implies that the *ManagementService* supplies such user *ProtocolEndpoint*. For example, a Configuration Service provides an HTTP interface to allow Network Administrators to set SBI parameters of NetDevs through a Web browser.

The *ServiceAccessPoint* class tagged as *Adapter Interface* represents the communication point between the *ManagementService* and the *AdapterService*. The *ProvidesEndpoint* associations con-

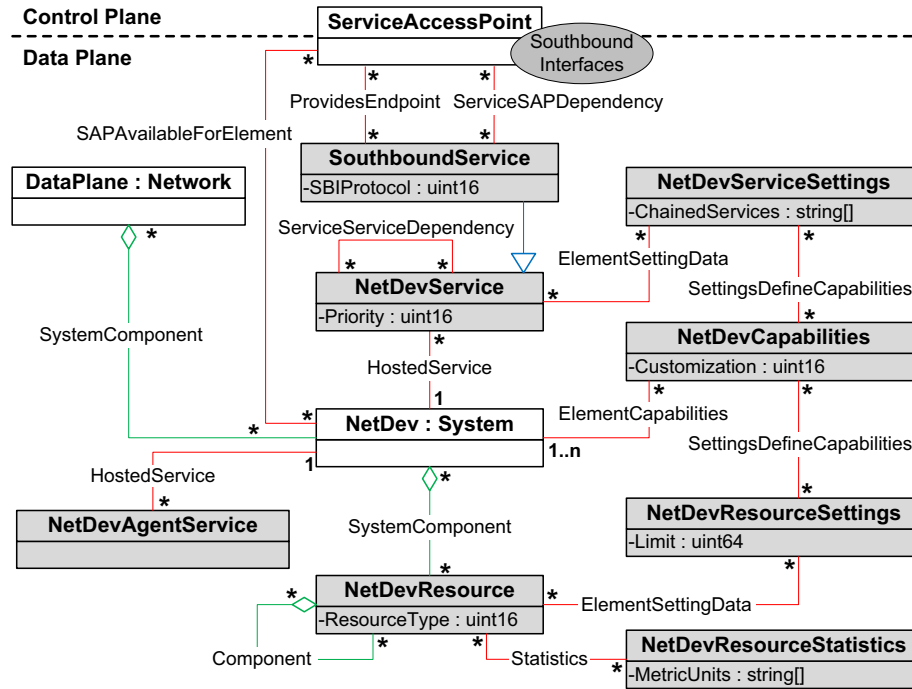


Fig. 5. Class schema to model SDN Data Plane.

nected to the adapter *ServiceAccessPoint* indicate that both the *ManagementService* and the *AdapterService* establish their own communication points to allow access from the other. The *ServiceSAPDependency* associations imply that both the *ManagementService* and the *AdapterService* utilize the adapter *ServiceAccessPoint* to access the other. The *ManagementService* and the *AdapterService* support properties and methods for sending and receiving requests, responses, and events through the adapter *ServiceAccessPoint*, such as the property *ListeningEvents* that specifies if the *ManagementService* handles event notifications and the properties *AdapterInterface* and *ManagerInterface* that identify the access APIs provided by the *ManagementService* and the *AdapterService*, respectively. For example, a Configuration Service and a NetDev Adapter establish a mutual communication using JSON over HTTP. Using this channel, the NetDev Adapter forwards to the Configuration Service an event from a NetDev Agent that notifies about failures with misconfiguration. Similarly, the Configuration Service uses the same channel to fix this failure by sending a configuration request to the NetDev Adapter. The NetDev Adapter forwards this request to the corresponding NetDev Agent.

The *AdapterService* class models an Adapter in charge of parsing and forwarding requests, responses, and events between the *ManagementService* and the *AgentService*. The *AdapterService* is a superclass that holds properties and methods for handling the communication through the adapter and management interfaces. Besides the aforementioned property *ManagerInterface*, the *AdapterService* also provides the property *AgentProtocol* that identifies the communication protocol used to interact with a specific *AgentService*. Four subclasses inherit from the *AdapterService*: the *NetDevAdapterService*, the *NetSlicerAdapterService*, the *NOSAdapterService*, and the *NetAppAdapterService*. For the sake of brevity and because the behavior of these subclasses is very similar, we decide to exclude them in Fig. 2. Each subclass from the *AdapterService* adds properties and methods to support functionality provided by the subclass from the *AgentService* that uses the same managed resource name (e.g., the *NetDevAdapterService* matches the *NetDevAgentService*). In addition, regarding this correlation, every subclass deriving from the *AdapterService* instruments specific aspects

from the proposed class schema. For example, a NetDev Adapter, which matches a NetDev Agent, only concerns about functionality for managing NetDevs (see Fig. 5).

The *AdapterService* may be hosted by either the *Manager* or the *Adapter*. Both *HostedService* associations linked to the *AdapterService* indicate this relationship. As well as the *Manager*, the *Adapter* may be an instance of either the *System* class or one of its subclasses. For example, a NetDev Adapter may be executed on either the same server running Management Services or a different server.

The *ServiceAccessPoint* class tagged as *Management Interfaces* represents the communication point between the *AdapterService* and the *AgentService*. The *ProvidesEndpoint* and the *ServiceSAPDependency* associations related to the management *ServiceAccessPoint* indicate that both the *AdapterService* and the *AgentService* provide and utilize management interfaces to perform their functionality. Subclasses from the *AdapterService* and from the *AgentService* inherit these associations. Each instance of the subclasses from the *AdapterService* handles the protocol used by the corresponding instance of the subclasses from the *AgentService*, affording a protocol-agnostic communication for the *ManagementService*. Both the property *AgentProtocol* from the *AgentService* and the property *ManagerProtocol* from the *AdapterService* define the communication protocol. For example, a NetDev Adapter uses the OF-CONFIG protocol to access a NetDev Agent for OpenFlow switches. A second NetDev Adapter utilizes the SNMP protocol to contact a second NetDev Agent for ForCES FEs. A Configuration Service communicates with both NetDev Adapters using a standardized data format and protocol (e.g., JSON over HTTP). The NetDev Adapters forward to the NetDev Agents the management requests received from the Configuration Service. Similarly, the NetDev Adapters forward to the Configuration Service responses and events received from the NetDev Agents. Thus, the Configuration Service carry out a protocol-agnostic management on different NetDev technology instances.

The *AgentService* class represents an Agent running on SDN managed resources, such as NetDev, NetSlicer, NOS, and NetApp. This is a superclass that defines properties and methods for sup-

porting the management *ServiceAccessPoint* and for handling the *EventIndication*. Besides the above described property *ManageProtocol*, the *AgentService* also provides the property *Authentication* that declares the access parameters (e.g., login and password) and the property *MaxConnections* that defines the maximum number of concurrent connections supported. The *EventIndication* is a subclass from the *ProcessIndication* class. The *EventIndication* maps an unsolicited event sent by the *AgentService* towards the *ManagementService* to notify about changes and alerts in SDN managed resources. The property *AgentID* of the *EventIndication* class identifies the instance of *AgentService* that generates the notification. For example, a NetDev Agent dispatches an event that notifies a detected misconfiguration on its hosting NetDev. The corresponding NetDev Adapter receives this unsolicited event and forwards it to a Configuration Service.

Four subclasses derive from the *AgentService*: the *NetDevAgentService*, the *NetSlicerAgentService*, the *NOSAgentService*, and the *NetAppAgentService*. Each subclass supports methods to carry out management tasks in its hosting SDN managed resource, such as retrieving statistical information, modifying configuration parameters, discovering capabilities, and changing communication attributes.

We use the *System* class to model the SDN as an entity composed of the *DataPlane*, the *ControlPlane*, and the *AppPlane*. The *Network* class represents the *DataPlane* as a logical, virtual, or physical network that groups interconnected NetDevs capable of exchanging information. The *AdminDomain* class indicates that the *ControlPlane* and the *AppPlane* gather similarly managed components, such as NetSlicers and NOSs for the Control Plane, and NetApps for the Application Plane.

The *ServiceAffectsElement* association between the SDN and the *ManagementService* reflects that Management Services have an effect in the SDN architecture, such as changing resource behavior, monitoring failures, and analyzing performance. Besides, the *SAPAvailableForElement* association between the SDN and the management *ServiceAccessPoint* implies that management interfaces provide managing access for the SDN architecture.

Fig. 3 shows the extended class schema for the Application Plane. We introduced three new classes and two novel associations to describe specific management features of NetApps. The new classes are the *NetAppCapabilities*, the *NetAppSettings*, and the *NorthboundService*. The new associations are the *NetAppHostedOnNOS* and the *NetAppHostedOnServer*.

The *AppPlane*, modeled with the *AdminDomain* class, uses the *SystemComponent* association to aggregate instances of the *NetApp*. Leveraging the *ApplicationSystem* class, the *NetApp* represents NetApps holding business logic on top of the SDN architecture. For example, NetApps that carries out load-balancing and access control tasks.

We use the *HostedService* association to indicate that the *NetApp* hosts the *NetAppAgentService* and the *NorthboundService*. The *NorthboundService* class models modules that communicate with services exposed by NOSs. The *ProvidesEndpoint* and the *ServiceSAPDependency* associations reflect that the *NorthboundService* uses and provides functions through the northbound *ServiceAccessPoint*. For example, a load-balancing and access-control NetApps retrieve and supply data from and to tracking and firewall services deployed by an NOS.

The *ServiceAccessPoint* tagged as *Northbound Interfaces* models the communication between the Application Plane and the Control Plane. This northbound *ServiceAccessPoint* encompasses different NBIs, such as APIs based on NOS native bundles (e.g., Floodlight and Ryu APIs), programming languages (e.g., Pyretic and ProCera), and protocols (e.g., REST). The property *NBType* from the *NorthboundService* identifies the API type used for the communication through the northbound *ServiceAccessPoint*.

The *NetAppHostedOnNOS* association between the *NetApp* and the NOS represents local NetApps running on NOSs. Usually, these local NetApps utilize NBIs based on NOS native bundles and programming languages to access and supply functionality from and to NOS services. The *NetAppHostedOnServer* between the *NetApp* and the *Server* system models NetApps running on remote servers. These remote NetApps prefer NBIs based on protocols for communicating with the Control Plane.

Using the *ElementCapabilities* association, the *NetAppCapabilities* class describes the supported and excluded abilities of NetApps. For instance, the property *SLADescriptor* allows to describe the service level policies of a specific *NetApp*. The *NetAppSettings* class establishes configuration parameters for the *NetApp*, such as the property *ExecuteModes* that defines the different execution modes supported by a particular *NetApp*. The *ElementSettingData* association depicts the relationship between the *NetAppSettings* and the *NetApp*. The *SettingsDefineCapabilities* association between the *NetAppSettings* and the *NetAppCapabilities* reflects that the setting data affect some NetApps capabilities. For example, configuring a different load-balancing algorithm modifies the behavior of the corresponding *NetApp*.

Fig. 4 describes the extended class schema for the Control Plane. Considering that CIM lacks elements that characterize the management information of NetSlicers and NOSs, we introduce seven new classes: the *SlicingService*, the *SlicingStatistics*, the *SlicingCapabilities*, the *SlicingSettings*, the *NOSService*, the *NOSServiceCapabilities*, and the *NOSServiceSettings*.

The *AdminDomain* class uses the *SystemComponent* association to describe the *ControlPlane* as an entity composed of NOSs and NetSlicers. The NOS models an NOS, such as OpenFlow controllers and ForCES CEs. The *NetSlicer* represents a NetSlicer system, such as FlowVisor for OpenFlow-based networks. The NOS is the hosting system for the *NOSAgentService* and the *NetSlicer* for the *NetSlicerAgentService*.

The *NOSService* is a superclass that models network services hosted in NOSs. The *HostedService* association between the *NOSService* and the NOS indicates this relationship. The property *NOSPRIORITY* from the *NOSService* designates the order at which the NOS processes the instances of the different hosted services. Subclasses must inherit from the *NOSService* in order to define specific NOS services, such as tracking, route calculation, and firewall. We present three subclasses for NOS services: the *ApplicationService*, the *DistributingService*, and the *ControlService*. The *ApplicationService* depicts services that expose functionality to the Application Plane through the northbound *ServiceAccessPoint*. Its property *NBType* specifies the provided API for the northbound communication. The *DistributingService* defines services that enable to deploy a distributed Control Plane across distinct domains through the east/westbound *ServiceAccessPoint*. This class includes the property *EWBIPProtocol* that declares the protocol for the east/westbound communication and the property *CurrentDomains* that indicates the number of different domains currently handled by the *DistributingService*. The *ControlService* describes services that handle the communication with NetDevs and NetSlicers through the southbound *ServiceAccessPoint*. Its property *SBIPProtocol* specifies the protocol for the southbound communication.

The *ServiceServiceDependency* association reflects that NOS services collaborate with or are necessary for other NOS services to perform their operation. For example, a topology service requires a tracking service to recognize hosts connected to specific switches.

We use the *ProvidesEndpoint* and the *ServiceSAPDependency* associations to correlate the *ApplicationService* with the northbound *ServiceAccessPoint*, the *DistributingService* with the east/westbound *ServiceAccessPoint*, and the *ControlService* with the southbound *ServiceAccessPoint*.

The *ServiceAccessPoint* tagged as *East/Westbound Interfaces* represents the communication point between distinct Control Plane domains. For example, the SDNI Negotiation Interface and the WE-Bridge mechanism. Although the exchange of information is usually carried out by NOSs, NetSlicers may also need to deploy a distributed architecture. This is reflected by the *HostedService* association between the *DistributingService* and the *NetSlicer*.

The *ServiceAccessPoint* tagged as *Southbound Interfaces* models the communication point between the Control Plane and the Data Plane. The *ServiceSAPDependency* and the *ProvidesEndpoint* associations reflect that the *ControlService* uses and supplies the southbound *ServiceAccessPoint* to send and receive messages to and from the Data Plane. This southbound *ServiceAccessPoint* encompasses different SBI protocols, such as OpenFlow, ForCES, and POF.

The *NOSServiceCapabilities* class declares the supported abilities of NOSs services, like the property *Reuse* that specifies the reusability of the *NOSService*. The *ElementCapabilities* association between the *NOSServiceCapabilities* and the *NOSService* reflects this relationship. The *NOSServiceSettings* class defines the configuration parameters for NOSs services, such as the property *Logging* that allows to set the method for storing data generated by the *NOSService*. The *ElementSettingData* association between the *NOSServiceCapabilities* and the *NOSService* indicates this relationship. The *SettingsDefineCapabilities* association between the *NOSServiceSettings* and the *NOSServiceCapabilities* implies that configuring NOS services establishes some capabilities. For example, the time interval for sending discovery messages updates the behavior of a tracking service.

The *SlicingService* class represents the functionality of a NetSlicer: divide the Data Plane into several isolated logical network instances (*a.k.a.* slices) and assign them to different NOSs. The *NetSlicer* hosts the *SlicingService* using the *HostedService* association. The *SlicingService* includes the property *SBIProtocol* that describes the protocol for the southbound communication and the property *CurrentSlices* that reports the number of slices currently operated by a particular NetSlicer. The *ProvidesEndpoint* and the *ServiceSAPDependency* associations between the *SlicingService* and the southbound *ServiceAccessPoint* indicate that NetSlicers provide and use SBIs to communicate with NetDevs and NOSs. For example, FlowVisor uses the OpenFlow protocol to communicate with both OpenFlow switches and OpenFlow controllers.

The *SlicingStatistics* class defines collections of metrics suitable to instances of the *SlicingService*. The *SlicingStatistics* is a subclass that derives from the *StatisticalData*. The *ServiceStatistics* association relates the *SlicingStatistics* with the *SlicingService*. For example, the property *TotalSlices* from the *SlicingStatistics* reports the total number of slices handled by a NetSlicer.

Through the *ElementCapabilities* association, the *SlicingCapabilities* class describes the supported and excluded capabilities of the *SlicingService*, like the property *SupportedSlicing* that indicates the different slicing methods defined by a NetSlicer. Some of these capabilities are specified in the *SlicingSettings* class by means of the *SettingsDefineCapabilities* association. The *SlicingSettings* delineates the configuration parameters for the *SlicingService*. The *ElementSettingData* association between the *SlicingSettings* and the *SlicingService* reflects this relationship. For example, the property *MaxSlices* declares the maximum number of concurrent slices supported by a NetSlicer.

Fig. 5 depicts the extended class schema for the Data Plane. In order to describe specific management features of NetApps, we introduce five new classes: the *NetDevCapabilities*, the *NetDevResource*, the *NetDevResourceSettings*, the *NetDevService*, and the *NetDevServiceSettings*.

As aforementioned, the *Network* class indicates that the *DataPlane* models a network composed of interconnected NetDevs. The *NetDev* represents a NetDev system within a network, such as

OpenFlow switches and custom forwarding hardware (e.g., OpenWrt and NetFPGA). The *DataPlane* aggregates the *NetDev* using the *SystemComponent* association. The *NetDev* hosts the *NetDevAgentService*.

The supported and excluded abilities of a NetDev are described by the *NetDevCapabilities*. The *ElementCapabilities* association between the *NetDev* and the *NetDevCapabilities* indicates this relationship. For example, the property *Customization* defines the degree of personalization for an instance of *NetDev*. An OpenFlow switch declares low customization capabilities because it provides simple forwarding functions based on match/action flow tables. A NetFPGA programmable hardware exposes higher customization capabilities because it offers complex plugging modules that enable to build specific functions. In addition, both kinds of NetDevs also reveal network capacity enabled by its components, such as speed of ports and size of queues.

The *NetDevResource* class inherits from the *EnabledLogicalElement* to model network elements composing a NetDev, like ports, queues, and tables. The *SystemComponent* association between the *NetDev* and the *NetDevResource* implies this aggregation. The latter includes the property *ResourceType* that allows to identify the type of network element composing the *NetDev*. Our schema presents the *NetDevResource* as a superclass from which individual subclasses inherit to represent NetDev components. For example, a subclass called *FlowTable* for characterizing flow tables that compose OpenFlow switches. In addition, the *Component* association connected to the *NetDevResource* models a network element composed of others, such as ports including various queues.

The *NetDevResourceStatistics* defines arbitrary collections of statistical information applicable to instances of the *NetDevResource*. The *NetDevResourceStatistics* is a subclass that derives from the *StatisticalData*. The *Statistics* association attaches the *NetDevResourceStatistics* with the *NetDevResource*. For example, ports in switches delineate transmission metrics, such as received and transmitted bytes, packets, and errors. The property *MetricUnits* from the *NetDevResourceStatistics* declares the units of measurement used by a network element for statistical data.

The *NetDevResourceSettings* class describes the configuration of network elements that compose NetDevs. The *ElementSettingData* association between the *NetDevResourceSettings* and the *NetDevResource* reflects this relationship. The *SettingsDefineCapabilities* association between the *NetDevResourceSettings* and the *NetDevCapabilities* indicates that the configuration parameters of NetDev components specify some capabilities of NetDevs. For example, the property *Limit* establishes boundaries for NetDev components, such as the highest speed operation of ports and the maximum buffer size of queues.

The *NetDevService* is a superclass that represents network services hosted in NetDevs. This hosting relationship is depicted with the *HostedService* association between the *NetDevService* and the *NetDev*. The property *Priority* from the *NetDevService* indicates the preference for processing a group of services hosted by a NetDev. Subclasses must derive from the *NetDevService* in order to model particular NetDev services, such as forwarding, route calculation, and firewall. This is the case of the *SouthboundService*, which defines services that query, receive, and execute instructions to and from the Control Plane. The *SouthboundService* inherits from the *NetDevService* and includes properties and methods to handle the communication through the SBIs. For example, the property *SBIProtocol* may specify that the southbound communication uses the OpenFlow protocol, which defines a secure channel in switches for communicating with external controllers and updating internal flow tables.

The *ServiceServiceDependency* association indicates that NetDev services cooperate with or are required for other NetDev services to perform their functions. For example, an inspection service

requires a forwarding service to redirect malicious packets to a specific destination.

The *ServiceSAPDependency* and the *ProvidesEndpoint* associations reflect that the *SouthboundService* uses and supplies the southbound *ServiceAccessPoint* to send and receive messages to and from the Control Plane. The *SAPAvailableForElement* association between this *ServiceAccessPoint* and the *NetDev* implies that the SBIs allow access from the Control Plane for managing NetDevs components and hosted services. For example, the OpenFlow protocol enables to manipulate flow tables within OpenFlow switches and the ForCES protocol facilitates to configure logical functions residing in ForCES FEs.

Through the *ElementSettingData* association, the *NetDevServiceSettings* class describes the configuration parameters of services hosted in NetDevs. For example, the property *ChainedServices* allows to delineate the sequence of services that forms a specific *NetDevService*. In particular, a subclass from the *NetDevServiceSettings* may include properties for establishing the routing algorithm of a route calculation service or the list of OpenFlow controllers of a southbound service. The *SettingsDefineCapabilities* association between the *NetDevServiceSettings* and the *NetDevCapabilities* implies that the configuration of NetDev services characterizes some capabilities of NetDevs.

The presented information model leverages the extensibility of CIM schemas to allow vendors and providers to represent specific implementation features of SDN resources. As above described, a specific implementation model must inherit from the classes defined in the proposed class schema. Furthermore, network designers may use or extend our information model to represent management requirements of new network paradigms seamlessly together with the functionality described in the proposed class schema. For example, network functions from Network Function Virtualization (NFV) can be successfully adapted to the conceptual model of NetDevs from SDN [28]. Finally, since the proposed class schema follow CIM conventions, our information model may include elements from the existing CIM Schema to represent other managed resources, such as legacy networks [54] and virtual networking [55].

5. Case study

To assess our approach, first, we establish a network management scenario that deploys different SDN management technologies. Second, we implement the system prototype that relies on the present approach. Third, we build up a test environment based on the described scenario. Fourth, we conduct a performance evaluation to determine the feasibility of using the proposed approach in terms of time-response and network traffic. These metrics are related to the behavior on runtime of solutions used to manage heterogeneous SDNs. Finally, we expose the qualitative features of this approach.

5.1. Scenario: configuring SDN-based networks by using heterogeneous management interfaces

Let us suppose that a Cloud Service Provider (CSP) enables access to its cloud resources by deploying a basic SDN data center network: three tiers of NetDevs (i.e., core, aggregation, and edge) handled by a *Current NOS* and arranged in a simple tree topology (i.e., each NetDev has a single parent). Usually, the CSP Network Administrator purchases SDN forwarding resources from *Vendor A*. However, at some point in time, the Network Administrator decided to buy NetDevs from *Vendor B* because it offered a better benefit-cost ratio than *Vendor A*. As the network became bigger and since the implementations of SDN resources are constantly updated, the CSP decided to install a *New NOS* that offers better per-

formance, reliability, and security features. Now, the Network Administrator faces the challenge of configuring heterogeneous NetDevs for being controlled by the *New NOS*.

Considering that *Vendor A* provides a different NetDev management interface than *Vendor B*, the Network Administrator typically would use an *Isolated Solution* (i.e., *Vendor A Tool* and *Vendor B Tool*) to execute specific configuration commands on NetDevs from distinct vendors. This solution hinders and retards managing tasks of Network Administrator. Instead, our approach hides network heterogeneity by establishing a common NetDev configuration model and by adapting to each vendor management interface. Thus, we afford an *Integrated Solution* that allows the Network Administrator to seamlessly configure every NetDev to be controlled by the new NOS, mitigating the complexity and time consumption of managing heterogeneous SDN resources.

Fig. 6 illustrates the above-described scenario. NetDevs provided by *Vendor A* are OpenFlow switches that enable the OVSDB management interface to accept configuration requests (i.e., OVSDB switches). NetDevs from *Vendor B* are OpenFlow switches that run the OF-CONFIG server to support configuration through the OF-CONFIG protocol (i.e., OF-CONFIG switches). The *Current NOS* that initially handles the OpenFlow switches is a Floodlight controller. The *New NOS* that has to be set in the OpenFlow switches for controlling them is an OpenDaylight controller. In addition, we defined a managing operation called *SetController*. This operation describes the process of configuring a number of NetDevs that provide distinct management interfaces (i.e., OVSDB switches and OF-CONFIG switches) in order to establish the *New NOS* (i.e., OpenDaylight) as their controller. *SetController* represents a common management task that Network Administrators must perform when conducting updating, maintenance, and recovery functions in heterogeneous SDN-based networks.

5.2. Implementation

To evaluate our approach, we developed two prototypes to conduct the *SetController* operation: *Integrated Solution* and *Isolated Solution*.

5.2.1. Integrated solution

We built this prototype upon the proposed approach for performing *SetController* regardless the different configuration interfaces. Fig. 7 depicts the implemented *Integrated Solution*. The Data Repository is a CIM Object Manager (CIMOM) that provides RRM as CIM schemas and stores instance data as CIM instances. CIMOM is the main component of a Web-Based Enterprise Management (WBEM) framework. CIM schemas characterize the SDN architecture from a management perspective while CIM instances represent the SDN managed resources. To build RRM, we compiled both the CIM Schema 2.18.1 and the SDN Extension Schema. The SDN Extension Schema implements the proposed information model.

Since this prototype focuses on the *SetController* operation, the compiled SDN Extension Schema is limited to the following new classes from the proposed information model: *AgentService*, *NetDevAgentService*, *AdapterService*, *NetDevAdapterService*, *NetDevService*, and *SouthboundService*. In addition, CIMOM stores CIM instances of the above-mentioned classes and of classes included in the CIM Schema 2.18.1: *ComputerSystem*, *HostedService*, *RemotePort*, *ServiceSAPDependency*, *TCPPProtocolEndpoint*, *ProvidesEndpoint*, *IPProtocolEndpoint*, and *BindsTo*. We used the Managed Object Format (MOF) to formally express the SDN Extension Schema and the CIM instances.

The Manager is carried out in a Java application. This Manager deploys the *SetController* operation as a Configuration Service. The User Interface of the Manager is a simple Command Line Interface

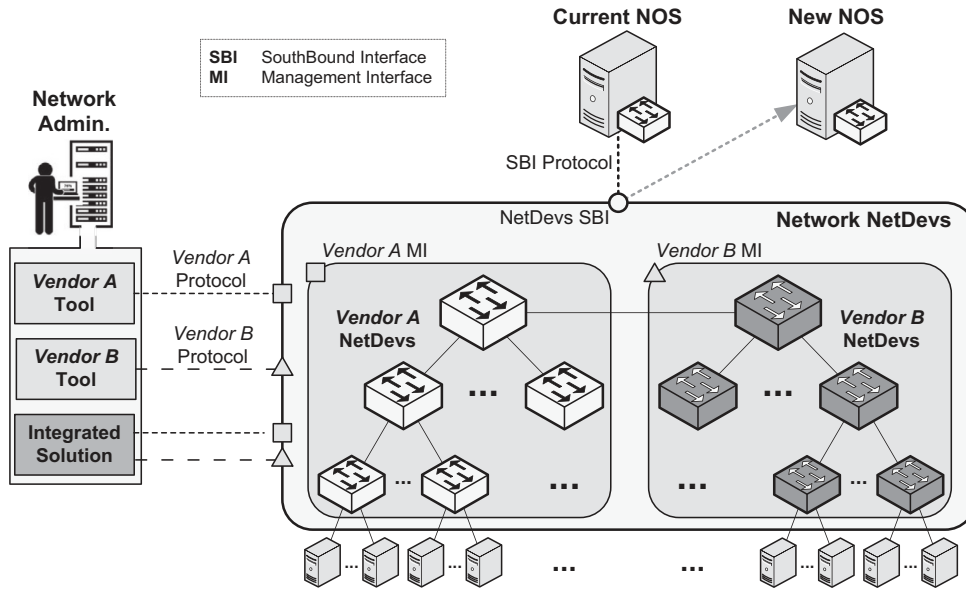


Fig. 6. Scenario: configuring NOS of heterogeneous NetDevs.

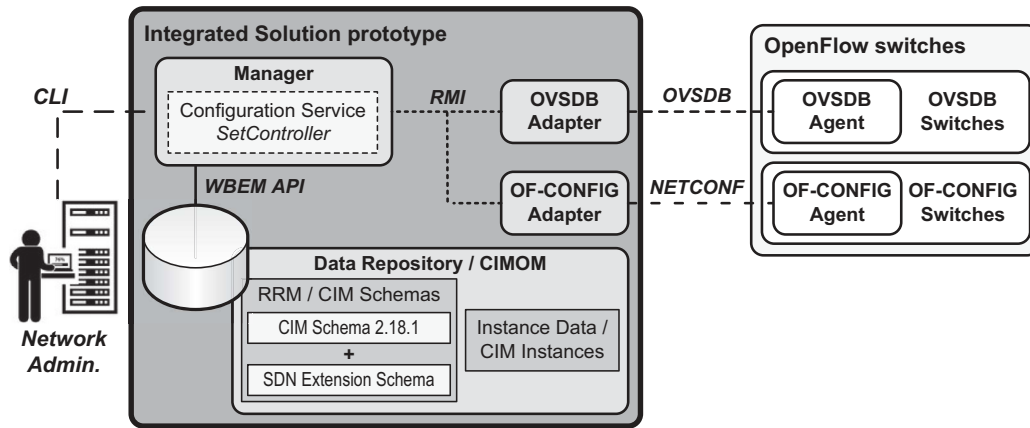


Fig. 7. Integrated solution prototype.

(CLI) that allows executing the configuration commands of *SetController*. The Repository Interface uses the WBEM API to read and write instances stored in CIMOM. The Adapter Interface relies on the Remote Method Invocation (RMI) to communicate the Manager and the Adapters.

The Java application also deploys the NetDev Adapters: the OVSDB Adapter and the OF-CONFIG Adapter. The OVSDB Adapter employs the Opendaylight OVSDB API to communicate with the OVSDB Agent that maintains the configuration database of OVSDB switches. The OF-CONFIG Adapter relies on the NETCONF4J API to connect with the OF-CONFIG Agent deployed by the OF-CONFIG switches for accepting configuration requests. OF-CONFIG utilizes NETCONF as the associated protocol.

Based on the information retrieved from CIMOM, the Manager invokes the appropriate Adapter. Once achieved the configuration in each requested OpenFlow switch, the Manager updates the instance data stored in CIMOM.

5.2.2. Isolated solution

This prototype describes a classic solution of using a configuration tool for each management technology (i.e., OVSDB and OF-CONFIG) to perform operations as *SetController*. We built both the OVSDB Tool and the OF-CONFIG Tool as bash scripts that automatize the usage of their underlying software. The OVSDB Tool uses

the *ovs-vsctl* program to configure OVSDB switches. The OF-CONFIG Tool employs the NETCONF client *netopeer-cli* to communicate with OF-CONFIG switches. Both tools provide a simple CLI to specify the configuration parameters.

5.3. Test environment

To evaluate the proposed approach, we conducted a case study in a test environment that allowed deploying the described scenario. Fig. 8 depicts this environment formed by two OpenFlow networks, two OpenFlow controllers, the Manager Client, and CIMOM. Each OpenFlow network ran on an Ubuntu Server 14.04 Virtual Machine (VM) with one virtual processor and 1.5 GB RAM assigned, both hosted by an Ubuntu 14.04 machine with 2.53 GHz Intel Core i5 processor and 4 GB RAM. Each VM executed Mininet 2.2.1, a software for emulating OpenFlow-based networks, to deploy a simple tree topology with 111 Open vSwitches 2.3.1. A tunnel over an IP network interconnected the root switches from each tree topology.

The Open vSwitches used the OpenFlow protocol over a second IP network to communicate with a specific OpenFlow controller: Floodlight v1.0.1 or Opendaylight Helium. Later in each evaluation, we will define the exact quantity of switches per controller. Each

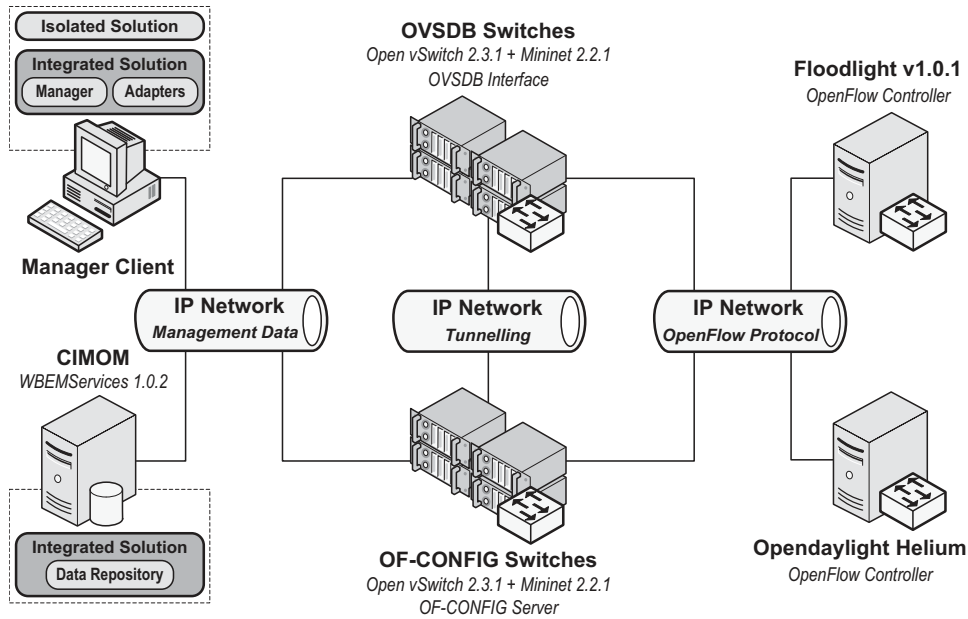


Fig. 8. Test environment.

OpenFlow controller operated on an Ubuntu 14.04 machine with 2.4 GHz Intel Core 2 duo processor and 2 GB RAM.

A third IP Network transmitted management data among the OpenFlow networks, the Manager Client, and CIMOM. The Manager Client was an Ubuntu 14.04 machine with 2.33 GHz Intel Core 2 duo processor and 2 GB RAM. The Manager Client hosted the Manager and Adapters components of the *Integrated Solution*, and the configuration tools of the *Isolated Solution*. We executed CIMOM from the WBE Services 1.0.2 on an Ubuntu 14.04 machine with 2.53 GHz Intel Core 2 duo processor and 4 GB RAM. CIMOM realizes the Data Repository of the *Integrated Solution* prototype.

5.4. Evaluation and analysis

To evaluate the proposed approach, it was proceeded to measure the time-response and the network traffic of the *Integrated Solution* and the *Isolated Solution* when used in the test environment (see Fig. 8) to conduct the operation *SetController*. Although the test environment allows to perform such operation in parallel for reducing the overall time-response, we carried out a sequential configuration for assuming the worst scenario. Furthermore, since many Open vSwitches run on the same VM, executing the operation in sequence avoided readings distorted by the overuse of shared resources. The number of configured switches for each evaluation was 2, 20, 50, 100, 150, and 200. Half were OVSDB Switches, and the other half were OF-CONFIG Switches. It is worth to mention that the values of 2 and 20 configured switches allowed us to demarcate a boundary for the analysis in terms of time-response and network traffic. In all evaluation cases, we took the average values for 30 measurements with a 95% confidence level.

Fig. 9 depicts the time-response results. Time-response is the time in seconds (s) measured since the Network Administrator executes the operation *SetController* on the Manager Client until receiving the reply of the last configured switch. Nevertheless, since a configuration reply is received for each switch, we provide a per switch basis analysis. Considering that the time-response (r in s) of Web systems can be ranked as optimal ($r \leq 0.1$), good ($0.1 < r \leq 1$), admissible ($1 < r \leq 10$), and deficient ($r > 10$) [57], the time-response results reveal: (i) *SetController* of both the *Isolated Solution* and the *Integrated Solution* has an admissible r that grows

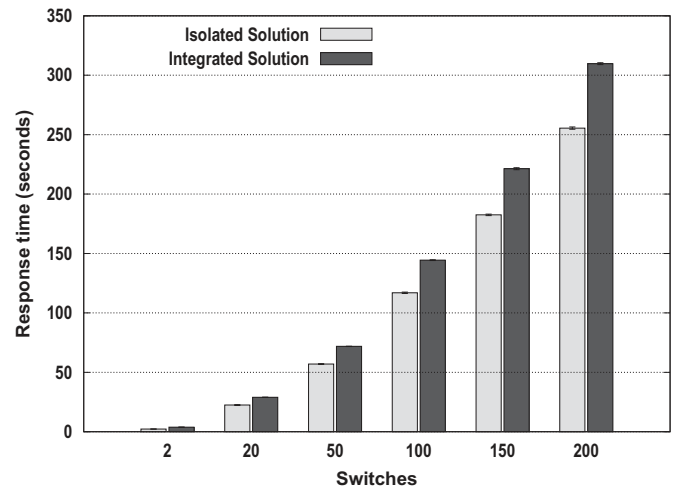


Fig. 9. Configuring evaluation: time-response.

moderately (less than 1.5s and 1.8s per switch, respectively) when the number of configured switches increases, (ii) the *Integrated Solution* takes longer than the *Isolated Solution*, as expected for using additional components (e.g., CIMOM and Adapters) to cope with the heterogeneity; and (iii) the time-response overhead per switch of the *Integrated Solution* is 0.8s for 2 switches and less than 0.35s for 20 switches or more. Based on the above results, the time-response overhead of the *Integrated Solution* ($Tr_{oh:integrated}$) depends on the number of configured switches (N_{sw}): (i) if $N_{sw} < 20$, $Tr_{oh:integrated} \leq 0.8N_{sw}$ and (ii) if $N_{sw} \geq 20$, $Tr_{oh:integrated} \leq 0.35N_{sw}$.

Let us compare the time-response results with the time that the Network Administrator takes to build the configuration command on the CLIs (i.e., time composing). In this case, the *Isolated Solution* aggregates an overhead to the time composing of the *Integrated Solution*. This is because the *Isolated Solution* forces the Network Administrator to decide which tool must use to configure each OpenFlow switch. Unlike this, the *Integrated Solution* abstracts the heterogeneity of the configuration technologies of the OpenFlow switches.

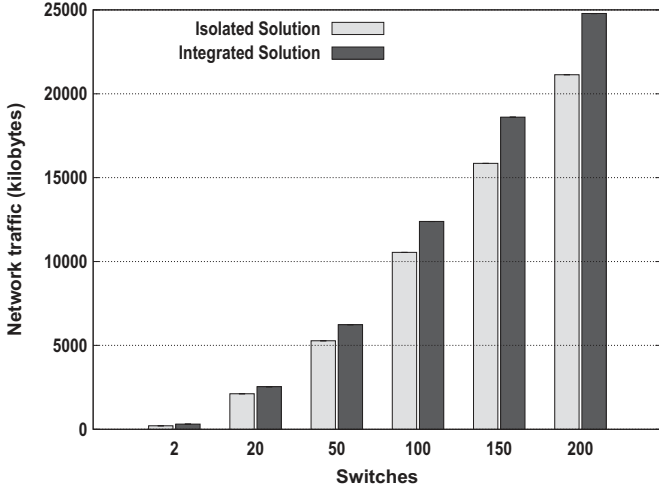


Fig. 10. Configuring evaluation: network traffic.

To compute the time composing overhead of the *Isolated Solution* ($T_{oh:isolated}$), we use the Keystroke-Level Model (KLM) [58] because it is useful to estimate the time that an expert user (i.e., the Network Administrator) spends to accomplish a routine task supported on computer keyboard and mouse (i.e., build the configuration command on the CLI). Previous researches demonstrate the feasibility of using KLM for conducting this kind of time evaluation [24,59]. In KLM, each task is modeled as a sequence of actions. We take two KLM operators: (i) press and release a key, $K = 0.2s$, and (ii) mentally preparing for decision making, $M = 1.35s$. In the best case, the Network Administrator carries out on the *Isolated Solution* the following additional actions: (i) change once from one tool to another by pressing ALT and TAB keys, and (ii) decide which tool must use for each switch. Based on these actions, $T_{oh:isolated}$ is also proportional to N_{sw} , therefore: $T_{oh:isolated} = K + MN_{sw} = 0.4 + 1.35N_{sw}$. The results obtained and estimated reveal that $T_{oh:integrated}$ is always less than $T_{oh:isolated}$.

Summing up, although the *Integrated Solution* includes additional modules (e.g., CIMOM and adapters) to cope with the complexity of managing heterogeneous SDN resources, it introduces a time-response overhead shorter than the time composing overhead of the *Isolated Solution*. Certainly, the difference between the time overheads increases as more switches and distinct technologies incorporate the managed SDN architecture. Additionally, considering that the time-consumption (Tt) is the sum of the time-response and the time composing, the difference between the time overheads demonstrates that the *Integrated Solution* reduces the time-consumption for carrying out the operation *SetController*, as can be seen in Eq. (1). Therefore, the time-response results corroborate that, in terms of such metric, it is feasible to use the proposed approach for executing management operations like the proved *SetController*.

$$\begin{aligned}
 Tt_{integrated} &= Tr_{integrated} + Tc_{integrated} \\
 Tt_{isolated} &= Tr_{isolated} + Tc_{isolated} \\
 Tr_{integrated} &= Tr_{oh:integrated} + Tr_{isolated} \\
 Tc_{isolated} &= Tc_{oh:isolated} + Tc_{integrated} \\
 Tt_{integrated} &= Tr_{isolated} + Tc_{isolated} + Tr_{oh:integrated} - Tc_{oh:isolated} \\
 Tt_{integrated} &= \begin{cases} Tt_{isolated} - 0.4 - 0.55N_{sw}, & N_{sw} < 20 \\ Tt_{isolated} - 0.4 - N_{sw}, & N_{sw} \geq 20 \end{cases} \quad (1)
 \end{aligned}$$

Fig. 10 presents the network traffic results. Network traffic is the amount of data in kilobytes (KB) transmitted and received by the network interface of the Manager Client. These results reveal:

(i) the traffic generated by *SetController* of both the *Isolated Solution* and the *Integrated Solution* grows moderately (approx 106 KB and 124 KB per switch, respectively) when the number of configured switches increases, (ii) the *Integrated Solution* generates more traffic than the *Isolated Solution*, as expected for handling management information of switches in CIMOM; and (iii) the additional traffic generated by the *Integrated Solution* is 32% for 2 switches and less than 17% for 20 switches or more. Considering that the *Integrated Solution*, unlike the *Isolated Solution*, copes with the heterogeneity of SDN resources by operating with standardized management data, the above facts corroborate that *SetController* of the *Integrated Solution* has a good behavior on network traffic.

Regarding the results obtained in the time-response and network traffic evaluation of the operation *SetController*, it is important to mention: (i) approx 94% of the time-response of *Isolated Solution* corresponds to the OF-CONFIG Tool, (ii) the OVSDb Tool generated approx 87% of the network traffic of *Isolated Solution*; and (iii) the time-response and network traffic overheads introduced by the *Integrated Solution* is smaller for many switches than for a few; this is because both the connection and the authentication with CIMOM were realized just once for any number of configured switches. Summarizing, the time-response and network traffic results demonstrated that, in terms of such metrics, it is feasible to use the proposed approach to perform SDN management operations in heterogeneous environments, as the executed *SetController*.

From a qualitative point of view, our approach provides mainly simplicity and formalization. The simplicity refers to that the proposed Management Plane facilitates integrating the SDN management operations of network administrators. They do not require to employ multiple frameworks to completely manage SDNs deployed with various technologies because the proposed plane addresses the management requirements of all the SDN architecture and hides the heterogeneity of the deployed resources. Regarding the formalization, the information model introduced in this paper can be considered as a step forward in unifying a conceptual understanding of the SDN architecture from the management perspective. It is possible because the information model relies on CIM to provide a technology-agnostic and consistent characterization of SDN. Although CIM schemas might be complex to understand, the working groups from DMTF continuously develop management profiles for clarifying how to use the classes and associations from CIM for specific areas of management. Thus, our proposed CIM schema provides a reference to DMTF for building a management profile that clearly explains how to characterize managing functions in the SDN architecture. Moreover, future SDN proposals may extend this approach for tackling arising challenges in SDN management.

6. Conclusions and future work

In this paper, we introduced a Management Plane aimed to facilitate the integrated management of the SDN architecture in heterogeneous environments. We provided a description of this Management Plane by referencing the four OSI network management submodels: Information, Organization, Communication, and Function. Furthermore, we relied on CIM to define a consistent information model that characterizes the entire SDN architecture from a management perspective regardless of the deploying technologies. This information model extends the CIM Schema to accomplish a generic abstraction of the SDN managed and managing resources and their relationships. It is noteworthy that our proposal looks at the complete SDN management aspect instead of only modeling a certain part, empowering a fully integrated solution for managing heterogeneous SDNs.

We carried out and evaluated the proposed approach with a prototype in a realistic scenario based on SDN. This scenario established a particular challenge: configuring a heterogeneous SDN composed of NetDevs that deploy distinct management technologies. Our approach corroborated to be feasible when effectively (in terms of time-response and network traffic) overcoming such challenge. Through a quantitative perspective, the evaluation results revealed: (i) regarding the performance analysis for Java Websites [57], it has an admissible behavior in terms of time-response, similar than using isolated tools, (ii) it introduces a small time-response overhead ($< 0.8s$ per switch) compared with the minimal time required by network administrators to handle dispersed solutions ($> 1.35s$ per switch), and (iii) it has a good behavior on network traffic when managing several devices ($< 17\%$ for 20 switches or more) in relation to employing distinct tools. From a qualitative point of view, the proposed approach reduces the complexity of SDN management by including modules (e.g., Data Repository and Adapters) that hide heterogeneity of SDN resources.

As future research, we plan to evaluate the proposed information model with other SDN technology instances (e.g., ForCES). We are also interested in focusing on assessing the remaining submodels from the Management Plane (e.g., Communication and Functions) in order to afford a complete SDN management architecture.

Acknowledgments

This research was supported in part by the University of Cauca (Colombia), the Foundation University of Popayan (Colombia), the Federal University of Rio Grande do Sul (Brazil), and by Colciencias (Colombia) under Ph.D. scholarship 647-2014 granted to M.Sc. Estrada-Solano.

References

- [1] N. Feamster, J. Rexford, E. Zegura, The road to SDN, *Queue* 11 (12) (2013) 20:20–20:40, doi:10.1145/2559899.2560327.
- [2] P. Lin, J. Bi, H. Hu, T. Feng, X. Jiang, A quick survey on selected approaches for preparing programmable networks, in: Proceedings of the 7th Asian Internet Engineering Conference, in: AINTEC '11, ACM, New York, NY, USA, 2011, pp. 160–163, doi:10.1145/2089016.2089044.
- [3] H. Kim, N. Feamster, Improving network management with software defined networking, *Commun. Mag. IEEE* 51 (2) (2013) 114–119, doi:10.1109/MCOM.2013.6461195.
- [4] J. Wickboldt, W. De Jesus, P. Isolani, C. Both, J. Rochol, L. Granville, Software-defined networking: management requirements and challenges, *Commun. Mag. IEEE* 53 (1) (2015) 278–285, doi:10.1109/MCOM.2015.7010546.
- [5] E. Haleplidis, K. Pentikousis, S. Denazis, J.H. Salim, D. Meyer, O. Koufopavlou, Software-Defined Networking (SDN): Layers and Architecture Terminology, Information, RFC 7426, IETF, 2015. URL <https://tools.ietf.org/html/rfc7426>.
- [6] ONF, SDN Architecture V1.0, Technical reference TR-502, Open Network Foundation, 2014. URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf.
- [7] ITU, Framework of Software-Defined Networking, Recommendation Y.3300, International Telecommunication Union, 2014. URL <https://www.itu.int/rec/T-REC-Y.3300-201406-I>.
- [8] R. Wang, D. Butnariu, J. Rexford, Openflow-based server load balancing gone wild, in: Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, in: Hot-ICE'11, USENIX Association, Berkeley, CA, USA, 2011. 12–12. URL <http://dl.acm.org/citation.cfm?id=1972422.1972438>.
- [9] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yakoumis, P. Sharma, S. Banerjee, N. McKeown, Elastictree: saving energy in data center networks, in: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, in: NSDI'10, USENIX Association, Berkeley, CA, USA, 2010. 17–17. URL <http://dl.acm.org/citation.cfm?id=1855711.1855728>.
- [10] A.K. Nayak, A. Reimers, N. Feamster, R. Clark, Resonance: dynamic access control for enterprise networks, in: Proceedings of the 1st ACM Workshop on Research on Enterprise Networking, in: WREN '09, ACM, New York, NY, USA, 2009, pp. 11–18, doi:10.1145/1592681.1592684.
- [11] K.-K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, N. McKeown, The stanford openroads deployment, in: Proceedings of the 4th ACM International Workshop on Experimental Evaluation and Characterization, in: WINTECH '09, ACM, New York, NY, USA, 2009, pp. 59–66, doi:10.1145/1614293.1614304.
- [12] ONF, OpenFlow Management and Configuration Protocol (OF-CONFIG) v1.2, Technical Specification TS-016, Open Network Foundation, 2014. URL <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf>.
- [13] B. Pfaff, B. Davie, The Open vSwitch Database Management Protocol, Information RFC 7047, IETF, 2013. URL <https://tools.ietf.org/html/rfc7047>.
- [14] S. Hassas Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, in: HotSDN '12, ACM, New York, NY, USA, 2012, pp. 19–24, doi:10.1145/2342441.2342446.
- [15] A. Tootoonchian, Y. Ganjali, Hyperflow: a distributed control plane for openflow, in: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, in: INM/WREN'10, USENIX Association, Berkeley, CA, USA, 2010. 3–3. URL <http://dl.acm.org/citation.cfm?id=1863133.1863136>.
- [16] D. Mattos, N. Fernandes, V. da Costa, L. Cardoso, M. Campista, L. Costa, O. Duarte, Omni: openflow management infrastructure, in: Network of the Future (NOF), 2011 International Conference on the, 2011, pp. 52–56, doi:10.1109/NOF.2011.6126682.
- [17] S. Natarajan, X. Huang, An interactive visualization framework for next generation networks, in: Proceedings of the ACM CoNEXT Student Workshop, in: CoNEXT '10 Student Workshop, ACM, New York, NY, USA, 2010, pp. 14:1–14:2, doi:10.1145/1921206.1921222.
- [18] R. Corin, M. Gerola, R. Riggio, F. De Pellegrini, E. Salvadori, Vertigo: network virtualization and beyond, in: Software Defined Networking (EWSN), 2012 European Workshop on, 2012, pp. 24–29, doi:10.1109/EWSN.2012.19.
- [19] E. Salvadori, R. Doriguzzi Corin, M. Gerola, A. Broglio, F. De Pellegrini, Demonstrating generalized virtual topologies in an openflow network, *SIGCOMM Comput. Commun. Rev.* 41 (4) (2011) 458–459, doi:10.1145/2043164.2018520.
- [20] J. Reich, C. Monsanto, N. Foster, J. Rexford, D. Walker, Modular SDN programming with pyretic, *USENIX* 38 (5) (2013) 40–47. URL https://www.usenix.org/system/files/login/articles/09_reich-online.pdf.
- [21] A. Voellmy, H. Kim, N. Feamster, Protera: a language for high-level reactive network control, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, in: HotSDN '12, ACM, New York, NY, USA, 2012, pp. 43–48, doi:10.1145/2342441.2342451.
- [22] O.M.C. Rendon, C.R.P. dos Santos, A.S. Jacobs, L.Z. Granville, Monitoring virtual nodes using mashups, *Comput. Netw.* 64 (0) (2014) 55–70. <http://dx.doi.org/10.1016/j.comnet.2014.02.007>. URL <http://www.sciencedirect.com/science/article/pii/S1389128614000498>.
- [23] O. Caicedo Rendon, F. Estrada Solano, L. Zambenedetti Granville, An approach to overcome the complexity of network management situations by mashups, in: Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on, 2014, pp. 875–883, doi:10.1109/AINA.2014.142.
- [24] O. Caicedo Rendon, F. Estrada-Solano, L. Zambenedetti Granville, A mashup ecosystem for network management situations, in: Global Communications Conference (GLOBECOM), 2013 IEEE, 2013, pp. 2249–2255, doi:10.1109/GLOCOM.2013.6831409.
- [25] O.M.C. Rendon, F. Estrada-Solano, L.Z. Granville, A mashup-based approach for virtual SDN management, in: Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual, 2013, pp. 143–152, doi:10.1109/COMPSAC.2013.222.
- [26] ONF, Common Information Model (CIM) Overview, Technical Reference TR-513, Open Network Foundation, 2015. URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Common_Information_Model_CIM_Overview_1.pdf.
- [27] ONF, Core Information Model (CoreModel) v1.1, Technical Reference TR-512, Open Network Foundation, 2015. URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/ONF-CIM_Core_Model_base_document_1.1.pdf.
- [28] E. Haleplidis, J. Hadi Salim, S. Denazis, O. Koufopavlou, Towards a network abstraction model for SDN, *J. Netw. Syst. Manage.* 23 (2) (2015) 309–327, doi:10.1007/s10922-014-9319-3.
- [29] B. Pinheiro, R. Chaves, E. Cerqueira, A. Abelem, CIM-SDN: a common information model extension for software-defined networking, in: Globecom Workshops (GC Wkshps), 2013 IEEE, 2013, pp. 836–841, doi:10.1109/GLOCOMW.2013.6825093.
- [30] ISO, Information Technology – Open Systems Interconnection – Systems Management Overview, ISO/IEC 10040:1998, International Organization for Standardization, Geneva, Switzerland, 1998.
- [31] DMTF, Common Information Model (CIM) Infrastructure v2.7.0, Specification DSP0004, Distributed Management Task Force, 2012. URL http://www.dmtf.org/sites/default/files/standards/documents/DSP0004_2.7.0.pdf.
- [32] K. McCloghrie, D. Perkins, J. Schoenwaelder, J. Case, M. Rose, S. Waldbusser, Structure of Management Information version 2 (SMIv2), Standards Track RFC 2578, IETF, 1999. URL <https://tools.ietf.org/html/rfc2578>.
- [33] ITU, Information Technology Open Systems Interconnection Structure of Management Information: Guidelines for the Definition of Managed Objects, Recommendation X.722, International Telecommunication Union, 1992. URL <https://www.itu.int/rec/T-REC-X.722-199201-I>.
- [34] J. de Vergara, V. Villagra, J. Asensio, J. Berrocal, Ontologies: giving semantics to network management models, *Netw. IEEE* 17 (3) (2003) 15–21, doi:10.1109/MNET.2003.1201472.

- [35] ONF, OpenFlow Switch Specification v1.5.0, Technical Specification TS-020, Open Network Foundation, 2014. URL <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf>.
- [36] L. Yang, R. Dantu, T. Anderson, R. Gopal, Forwarding and Control Element Separation (ForCES) Framework, Informational RFC 3746, IETF, 2004. URL <http://datatracker.ietf.org/doc/rfc3746/>.
- [37] H. Song, Protocol-oblivious forwarding: unleash the power of SDN through a future-proof forwarding plane, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, in: HotSDN '13, ACM, New York, NY, USA, 2013, pp. 127–132, doi:10.1145/2491185.2491190.
- [38] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, G. Parulkar, FlowVisor: A Network Virtualization Layer, Technical Report OpenFlow-tr-2009-1, OpenFlow Team, Stanford University, 2009. URL <http://www.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf>.
- [39] X. Jin, J. Rexford, D. Walker, Incremental update for a compositional SDN hypervisor, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, in: HotSDN '14, ACM, Chicago, Illinois, USA, 2014, pp. 187–192, doi:10.1145/2620728.2620731.
- [40] D. Drutskey, E. Keller, J. Rexford, Scalable network virtualization in software-defined networks, IEEE Internet Comput. 17 (2) (2013) 20–27, doi:10.1109/MIC.2012.144. URL <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6362137>.
- [41] Project Floodlight, The floodlight controller, [Accessed: June 16, 2016]. URL <https://floodlight.atlassian.net/wiki/display/floodlightcontroller/The+Controller>.
- [42] Ryu Project Team, Ryu application API, [Accessed: June 16, 2016]. URL https://ryu.readthedocs.io/en/latest/ryu_app_api.html.
- [43] Project Floodlight, Floodlight REST API, [Accessed: June 16, 2016]. URL <https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Floodlight+REST+API>.
- [44] M. Monaco, O. Michel, E. Keller, Applying operating system principles to SDN controller design, in: Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks, in: HotNets-XII, ACM, New York, NY, USA, 2013, pp. 2:1–2:7, doi:10.1145/2535771.2535789.
- [45] P. Lin, J. Bi, S. Wolff, Y. Wang, A. Xu, Z. Chen, H. Hu, Y. Lin, A west-east bridge based SDN inter-domain testbed, IEEE Commun. Mag. 53 (2) (2015) 190–197, doi:10.1109/MCOM.2015.7045408.
- [46] J. Halpern, J.H. Salim, Forwarding and Control Element Separation (ForCES) Forwarding Element Model, Standards Track RFC 5812, IETF, 2010. URL <https://tools.ietf.org/html/rfc5812>.
- [47] T. Bray, The JavaScript Object Notation (JSON) Data Interchange Format, Standards Track RFC 7159, IETF, 2014. URL <https://tools.ietf.org/html/rfc7159>.
- [48] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan, Extensible Markup Language (XML) 1.1 (Second Edition), Recommendation, W3C, 2006. URL <https://www.w3.org/TR/xml11/>.
- [49] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, Standards Track RFC 2616, IETF, 1999. URL <https://tools.ietf.org/html/rfc2616>.
- [50] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H.F. Nielsen, A. Karmarkar, Y. Lafon, SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), Recommendation, W3C, 2007. URL <https://www.w3.org/TR/2007/REC-soap12-part1-20070427/>.
- [51] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, Network Configuration Protocol (NETCONF), Standards Track RFC 6241, IETF, 2011. URL <https://datatracker.ietf.org/doc/rfc6241/>.
- [52] D. Harrington, R. Presuhn, B. Wijnen, An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks, Standards Track RFC 3411, IETF, 2002. URL <https://tools.ietf.org/html/rfc3411>.
- [53] ITU, TMN Management Functions, Recommendation M.3400, International Telecommunication Union, 2000. URL <https://www.itu.int/rec/T-REC-M.3400-200002-I>.
- [54] DMTF, Network Management Profile v1.0.0b, Specification DSP1046, Distributed Management Task Force, 2015. URL http://www.dmtf.org/sites/default/files/standards/documents/DSP1046_1.0.0b.pdf.
- [55] DMTF, Virtual Networking Management, White Paper DSP2025, Distributed Management Task Force, 2012. URL http://www.dmtf.org/sites/default/files/standards/documents/DSP2025_1.0.0.pdf.
- [56] A. Pras, J. Schoenwaelder, On the Difference between Information Models and Data Models, Informational RFC 3444, IETF, 2003. URL <https://tools.ietf.org/html/rfc3444>.
- [57] S. Joines, R. Willenborg, K. Hygh, Performance Analysis for Java Websites, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
- [58] D. Kieras, Using the Keystroke-Level Model to Estimate Execution Times, Unpublished Report, University of Michigan, 2001. [Online]. Available: <http://www.ict.griffith.edu.au/marilyn/uidweek10/klm.pdf>.
- [59] C.R.P. dos Santos, L.Z. Granville, W. Cheng, D. Loewenstern, L. Schwartz, N. Anerousis, Performance management and quantitative modeling of it service processes using mashup patterns, in: Network and Service Management (CNSM), 2011 7th International Conference on, 2011, pp. 1–9. URL <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6104022>.

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/318092682>

SDN Management Based on Hierarchical Task Network and Network Functions Virtualization

Conference Paper · June 2017

DOI: 10.1109/ISCC.2017.8024713

CITATION

1

READS

141

5 authors, including:



Mario Gironza

Universidad del Cauca

3 PUBLICATIONS 4 CITATIONS

SEE PROFILE



William Villota

University of Campinas

3 PUBLICATIONS 4 CITATIONS

SEE PROFILE



Felipe Estrada-Solano

Universidad del Cauca

11 PUBLICATIONS 271 CITATIONS

SEE PROFILE



Armando Ordonez

Universidad del Cauca

62 PUBLICATIONS 156 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



µVIMS: A Finer-Scalable Architecture Based on Microservices [View project](#)



NFV Management [View project](#)

SDN Management Based on Hierarchical Task Network and Network Functions Virtualization

Mario Alejandro Gironza-Ceron*, William Fernando Villota-Jacome*,
Armando Ordóñez†, Felipe Estrada-Solano* and Oscar Mauricio Caicedo Rendon*

*Telematics Engineering Group, Telematics Department, University of Cauca,
e-mail: {magironza, wvillota, festradasolano, omcaicedo}@unicauca.edu.co

†University Foundation of Popayán, e-mail: jaordonez@unicauca.edu.co

Abstract—Ensuring proper operation of networks based on the Software-Defined Networking (SDN) is essential. Many proposals exist for managing SDN. However, they present low automation, long-time for Administrators to manage situations, and lack of flexibility and workability. In this paper, we introduce an architecture based on the Hierarchical Task Network (HTN) planning technique and the Network Functions Virtualization (NFV) for managing SDN. This architecture presents how to carry out the orchestrator of NFV by using HTN for SDN management. Furthermore, it presents how to perform a Virtualized Network Functions Manager and a Virtualized Infrastructure Manager of NFV by using a request-response model for communicating to and from managed networks. Furthermore, our architecture inherited flexibility and workability from NFV. We evaluate the architecture in a proof-of-concept regarding time-planning and network traffic. The evaluation results evidenced short time-planning and small traffic overhead.

I. INTRODUCTION

Management is an essential part of networks because it ensures operation, maintenance, administration, and provisioning of network systems [1]. In a traditional network environment, management is complex because forwarding and control planes are both in the same network device [2] [3] [4]. This means that Administrators have to manage each network device separately for controlling the whole network. The Software-Defined Networking (SDN) is a new network technology that makes management tasks easier than in traditional networks by centralizing the network control [2].

As any network technology, SDN needs to be managed properly. However, SDN presents a typical problem: when a new technology is deployed, the importance of its management is underestimated. This means that management becomes a patch because it does not evolve simultaneously with the network technology [2]. There are many proposals for managing SDN, which are based on different techniques, such as Set Cover Problem [5], middleboxes [6], interactive visualization [7], Software Defined Infrastructure [8], RESTful APIs [9] and selective and adaptive monitoring [10]. Nonetheless, these proposals share some shortcomings: (i) low automation of network tasks, (ii) long-time required for Administrators to manage situations (e.g, a failure in packets transmission or an unforeseen slowness in a link formed by virtual switches); and (iii) lack of flexibility and workability for network management. Also, in our previous work [11]–[14], we have proposed solutions based on mashups and situational management

for managing SDN. Nevertheless, we have only achieved a medium-automation level because of intrinsic limitations of mashups.

To overcome the above shortcomings, we introduce an architecture for monitoring, configuring, and controlling SDN-based networks. Our architecture is based on the Hierarchical Task Network (HTN) and the Network Functions Virtualization (NFV). HTN is a planning technique that decomposes tasks into subtasks and leads their execution to meet a definite goal [15]. We argue that HTN is a key tool for managing SDN because it makes the work of Administrators easier by introducing automation of network tasks. Such automation could reduce management complexity and eliminate the need for Administrators to manage network situations manually and one at a time. Thus, the time that Administrators need to face situations could be reduced.

NFV is a promising technology that aims to decouple Network Functions from the hardware on which they run [16]. Although some proposals have investigated SDN and NFV relationship [16]–[18], the use of NFV for managing SDN is a major open research topic yet. We consider that the Management and Orchestrator (MANO) is fundamental for managing SDN because it brings benefits like flexibility (this involves scalability and elasticity) and workability (*i.e.*, the ability to work in existing networks) from NFV.

To sum up, the key contributions presented in this paper are to: (i) propose an architecture based on NFV and HTN for managing SDN, (ii) present a prototype of this architecture; and (iii) demonstrate by a proof-of-concept that our architecture has short values for time-planning and network traffic.

The remainder of this paper is organized as follows. In Section II, we present HTN, NFV and the related work. In Section III, we introduce the architecture for managing SDN. In Section IV, we expose and analyze the proof-of-concept of our architecture. In Section V, we provide some conclusions and implications for future work.

II. BACKGROUND AND RELATED WORK

In this section, we present a background about HTN and NFV. Afterward, we present the related work of our proposal.

A. Hierarchical Task Network

HTN is an Artificial Intelligence planning technique that aims to create a plan (*i.e.*, a set of tasks) for achieving

a goal state by decomposing tasks into smaller tasks [15]. HTN includes a planning problem and a planning domain. The planning problem has an initial state and desired goals while the planning domain contains a set of tasks useful to achieve these goals [19]. Several tasks are organized in a hierarchy called task network. Each task represents an activity to be performed and may be primitive or compound [20]. A primitive task represents a low-level action that is indivisible and executed directly. A compound task is a high-level action that needs to be decomposed into subtasks. A method describes how to decompose a compound task [21]. Several methods may be associated with a compound task [22].

We consider that HTN represents a powerful tool for supporting SDN management. The work of Administrators should become easier as HTN introduces automation of network tasks. Such automation could lower management complexity because it would eliminate the need for Administrators to manage network situations individually. As a consequence, the time that Administrators spend to face complex situations (*i.e.*, situations composed by other ones) could be reduced.

B. Network Functions Virtualization

NFV is an initiative of the European Telecommunications Standards Institute (ETSI), aiming to enhance the delivery of network services by separating network functions from the hardware [18]. MANO of NFV is formed by [23]: an Orchestrator for composing software resources and virtualized hardware; a Virtualized Network Functions Manager (VNFM) for managing the lifecycle of the Virtualized Network Functions (VNFs); and a Virtualized Infrastructure Manager (VIM) for virtualizing and managing network resources [24].

We argue that MANO is an essential element for monitoring, configuring and controlling SDN because its use can bring from NFV benefits like scalability, elasticity and workability [23]. Such benefits aid to ensure the robustness of network management tools that follow our architecture.

C. Literature Review

In the literature, there are a lot of proposals for managing SDN. In [5], the authors proposed a low-cost monitoring scheme for recovering flow statistics across an SDN-based network in a timely way. In [7], the authors put forward an approach for managing SDN by configuring control traffic that allows Administrators to interact with the network. In [8], the authors built a system based on Software-Defined Infrastructure that provides integrated functions for monitoring SDN. In [10], the authors presented Flo-v, a framework to provide network monitoring in a virtual SDN. The above proposals share some shortcomings like low automation of network management tasks, long-time required for Administrators to handle situations, and lack of flexibility, elasticity, and workability. Furthermore, they do not use NFV and HTN for managing SDN.

There are also proposals that use SDN and NFV for different purposes. For instance, in [18], the authors described how NFV is applied to virtualize a particular SDN function. In [16], the

authors performed a study for flexible management and deployment of VNFs. In [17], the authors presented a taxonomy to depict the evolution of the NFV/SDN relationship. However, the use of NFV for managing SDN is still a relevant open research topic.

In our previous work, we have developed solutions for managing SDN. In [11]–[13], we proposed an approach that allows Administrators to customize and combine solutions for composing mashups, based on situation management, that aim to tackle unexpected, dynamic, and heterogeneous network situations. Nevertheless, we have only achieved a medium-automation level because of inner limitations of mashups. Also, we introduced a CIM-based Information Model for a vertical Management Plane to facilitate the integrated management in heterogeneous SDNs [14]. Such a model is more abstract than the one presented in this paper.

On the other hand, HTN has been used for diverse purposes. The work [25] proposed an automated system that allows creating mission plans for environmental disaster situations. The work [20] presented a framework based on automated planning for composing convergent telecommunications services centered in the final user. The work [26] put forward an automated solution to manage human-robot collaborative activities. It is remarkable that, to the best of our knowledge, we are pioneers in using HTN to carry out NFV MANO for managing SDN.

III. ARCHITECTURE BASED ON HTN AND NFV

Here, we present motivating scenarios and an overview of our architecture. Also, we introduce its layers and elements.

A. Motivating Scenario

Let us consider the next scenario: a situation in an SDN-based network formed by several slices (*i.e.*, portions of a network) that support Internet services to companies like banks, hospitals, and governments. Sometimes, unexpected situations, like faults or wrong configuration in an SDN-enabled switch, can break the connection and stop the normal functioning of these services. Usually, providers cope with these situations by redirecting, reconfiguring and, even, rebooting slices or network in a traditional way, that is, with a human intervention that tends to be slow and prone to mistakes.

We found some shortcomings when network situations are handled in a traditional way: (i) Administrators have to manage network devices one at a time, (ii) as a consequence, Administrators need a lot of time to deal with these situations; and (iii) Administrators typically have just one option to solve a situation. These shortcomings bring loss of money for both providers and customers and, mainly, highlight the need for automating the process of overcoming network situations. To address these shortcomings, our architecture offers: (a) automated planning (*i.e.*, composition) of solutions in a less complex, time-consuming and network traffic way, (b) visualization of planning process information in an understandable and friendly way to make decisions; and (c) taking into account several alternatives to generate the automatic solutions.

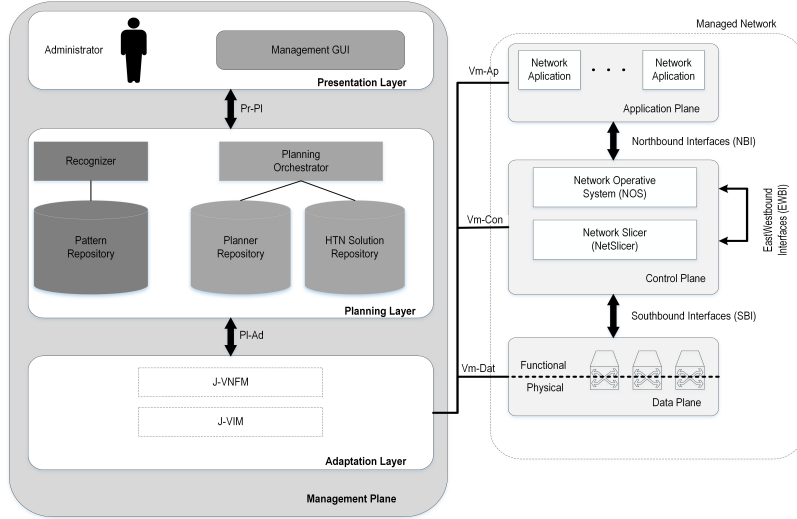


Fig. 1. Architecture based on HTN and NFV

B. Overview

Our architecture, based on HTN and NFV MANO, aims to automate the tasks of monitoring, configuring and controlling SDN-enabled networks. Here, we propose how to instantiate: (i) the orchestrator of MANO by using HTN that is a key enabler for facilitating the composition of network management tasks because it allows automated planning, (ii) VNFM and VIM of MANO for enabling the communication to and from the Managed Networks. By these instantiations, our architecture inherits from NFV important features like flexibility and workability.

The proposed architecture is formed by three layers (Figure 1) located in a vertical Management Plane [2] [14] that communicates with classical SDN planes. From a high level point of view, this architecture allows: (i) collecting data from the Managed Network to recognize situations (*i.e.*, a specific condition or problem in a SDN [13]), (ii) building automatically plans (*i.e.*, a solution formed by a set of management tasks) to manage recognized situations, (iii) storing, executing, and publishing built plans; and (iv) depicting information of such plans and their tasks.

C. Layers and Elements

The Adaptation Layer is in charge of getting information (*e.g.*, routers and switches condition) from the Managed Network to send it to the Planning Layer. Furthermore, the Adaptation Layer receives management operations from the Planning Layer to forward them to the Managed Network. This Layer is formed by J-VNFM and J-VIM that are based on VNFM and VIM, respectively. In particular, J-VIM is in charge of virtualizing and managing the network infrastructure including the Planes of Data, Control, and Application. In turn, J-VNFM is responsible for managing lifecycle of Virtual Machines or Dockers in which VNFs are deployed. Here,

example of VNFs are firewalls, load balancers and obviously the management network functions of our Management plane.

The communication between architectural layers is made by reference points that support the bidirectional information exchange. The Managed Network and J-VNFM and J-VIM of the Adaptation Layer are linked by $Vm-Dat$, $Vm-Con$, and $Vm-Ap$ reference points. $Vm-Dat$ links to the Data Plane, $Vm-Con$ connects to the Control Plane, and $Vm-Ap$ links to the Application Plane. The communication between the Planning and Adaptation layers is by $Pl-Ad$. The reference points might be instantiated by using the Representational State Transfer (REST) [27] or the Simple Object Access Protocol (SOAP) [28].

The Planning Layer is in charge of automatically generating plans for monitoring, configuring and controlling situations in the Managed Network by two steps: (i) recognizing a situation in an automated way; and (ii) automatically planning a solution to manage the identified situation. These steps are conducted by Recognizer and Planning Orchestrator.

Figure 2 introduces the Recognizer that automatically identifies network situations (*e.g.*, faults and configuration) by comparing network information (*i.e.*, samples) and patterns existing in the Pattern Repository that we explained in [13]. The Recognizer is formed by Sensing and Matching Mechanism. The Sensing is in charge of retrieving information from the Managed Network via the Adaptation Layer and delivering it as streaming to the Matching Mechanism. The Matching Mechanism recognizes a situation as follows: (1) it reads and loads situation patterns from the Pattern Repository, (2) it obtains samples by sensing, (3) it conducts matching operations (*i.e.*, samples vs patterns), (4) per detected situation (*i.e.*, there is a match), an HTN descriptor (*i.e.*, the representation of a goal to achieve) is called by using a planner identifier; and (5) the Planning Orchestrator is invoked to build a plan intended to face the recognized situation.

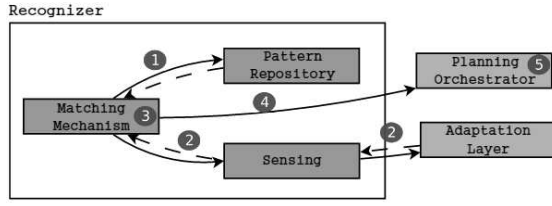


Fig. 2. Automatic Recognition

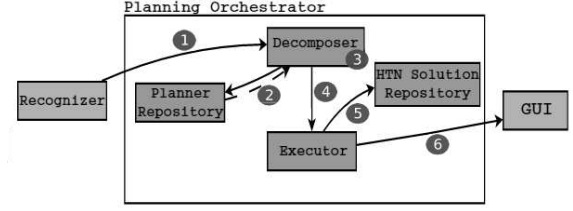


Fig. 3. Automatic Composition

The Planning Orchestrator is an HTN-based MANO. Our Orchestrator build plans automatically for managing network situations, and it is formed by the Decomposer, Executor, Planner Repository and HTN Solution Repository. The Planner Repository storages planners that include two descriptors called Planning Domain and Planning Problem. These descriptors must be written in an HTN notation by experts in automated planning and Networking. The Planning Domain contains the operators and methods used to compose plans. The Planning Problem contains an initial state and a final state to be reached.

Listings 1 and 2 present, using the SHOP2 [29] planner notation, the Planning Problem and Planning Domain for a planner of example. This example aims to change the state (*i.e.*, on and off) of two OpenFlow-enabled switches when one of them fails. The Planning Problem includes a description of the initial state where switch2 is on, and a description of the goal in which switches states must be exchanged. The Planning Domain consists of two operators and one method that allow decomposing tasks. The method allows to exchange the switches states by calling the operators. Once operators are called, they are executed to achieve the goal. It is important to emphasize that the Planning Domain is fundamental for our Orchestrator because it includes all possible actions for facing network situations.

Listing 1. HTN Planning Problem

```
(defproblem problem basic
  ((on link2))
  ((exchange link2 link1))
):defproblem
```

Listing 2. HTN Planning Domain

```
(defdomain basic(
  (:operator (!enable ?a) ()) ((on ?a)))
  (:operator (!disable ?a) ((on ?a) ()))
  (:method (exchange ?x ?y)
    ((on ?x) (not (on ?y)))
    ((!disable ?x) (!enable ?x)))
  ); basic
):defdomain
```

Figure 3 depicts the automated composition of a plan for a network situation. This composition is as follows. (1) Decomposer receives a planner identifier from the Recognizer. (2) Decomposer retrieves a planner from the Planner Repository by using such an identifier. (3) Decomposer creates a plan by using SHOP2 [20] that is an automated planning system for planning tasks in the same order that they will be executed

later. The algorithm begins loading the Planning Problem (goal to achieve) and the Planning Domain (tasks useful to reach a goal) included in the planner, then the algorithm chooses a task from the set of tasks, if such a task is a primitive, it is executed, if the task is compound, the algorithm looks for a method to decompose it into smaller ones. This decomposition process is continuously repeated and it finishes when the algorithm finds out only primitive tasks which form the plan. (4) Executor allows storing, publishing, and executing plans. (5) Executed plans are stored in the HTN Solution Repository for later reference. (6) Once plans are stored, they are ready for being displayed in the Presentation Layer.

The Presentation Layer can be accessed by the Management GUI in any PC or laptop connected to the Internet. This Layer receives management information about generated, executed and on running plans from the Planning Layer by *Pr – Pl* reference point. This point could also be instantiated using REST or SOAP. By the Management GUI, Administrators can mainly: (i) visualize plans executed and on running, (ii) select plans by clicking, (iii) visualize alternative plans; and (iv) customize plans.

IV. PROOF-OF-CONCEPT

This section introduces a prototype of our architecture. Also, it presents and discusses the evaluation of our prototype.

A. Prototype

Figure 4 depicts the prototype built to assess our proposal. We implemented the Presentation Layer as a typical Web application based on Java. The Planning Orchestrator was created with JSHOP2 [20] that is an implementation of SHOP2 developed in Java. The Matching Mechanism was created using an instantiation of the PHREAK algorithm provided by JBOSS Drools [30]. The J-VIM and J-VNFM were created using RESTful [31] that is a Java API useful to implement the REST architectural model. It is important to highlight that the Planning Orchestrator, Matching Mechanism, J-VIM, and J-VNFM were deployed in Virtual Machines running on OpenStack [32] as VNFs specialized in management tasks.

The Managed Network was a SDN formed by a controller, switches and hosts deployed in Mininet [33]. The controller was deployed by using Ryu [34], a SDN framework that provides software components with APIs to create network management and control applications. In turn, the switches were deployed with Open vSwitch [35].

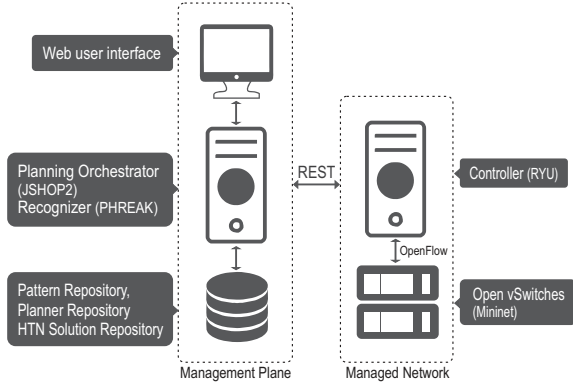


Fig. 4. Prototype

B. Evaluation and analysis

During the evaluation, first, we measured the time-planning (*i.e.*, time required by the Planning Orchestrator for building an automated plan) to analyze the time added by our approach for facing a situation. Second, we measured the traffic to analyze the impact in the network of using our prototype.

In the time-planning evaluation, we automatically generated a plan from the planner described in Listings 1 (Planning Problem) and 2 (Planning Domain). Afterwards, we generated the same plan increasing the number of primitives (tasks applied to each switch forming the Managed SDN) to 20, 40, 60, 80, and 100. Table I presents the corresponding results, revealing that time-planning increases linearly with the number of primitives.

TABLE I
TIME-PLANNING FOR A SINGLE PLAN

Primitive tasks	Time-Planning (ms)
20	68
40	105.2
60	127.2
80	149.9
100	171.2

During the time-planning evaluation, we also conducted a test by automatically generating 2, 4, 8, and 16 plans. Here, per plan, we varied the number of primitive tasks from 4 to 28. Figure 5 presents the corresponding results, corroborating that time-planning increases slightly and linearly with the number of primitives. It is important to highlight that our proposal has similar time-planning behavior than other works [13] [36]. According to the foregoing results, we can conclude that our proposal is a feasible alternative for Administrators manage situations in SDN by spending short time. This feasibility is because we use HTN and NFV to address the low automation problem of network management.

For the traffic evaluation, we managed a SDN formed by a controller and 255 Open vSwitches organized in a tree topology (depth 8, fan-out 2). During this evaluation, we measured the traffic in bytes when the controller receives

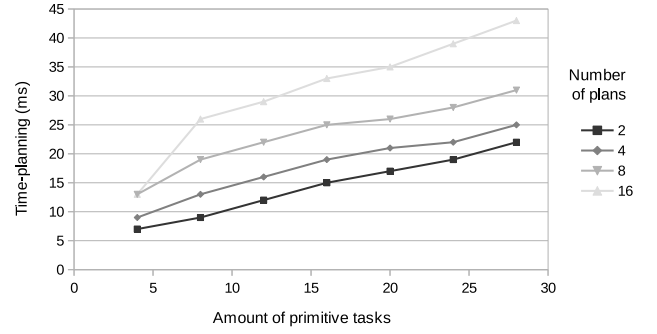


Fig. 5. Time-planning for several plans

from our prototype 10, 20, 40, 80, 160 and 320 plans with 2, 4, 8, and 16 primitives; in Figure 4, the controller is obviously located in the Managed Network and our prototype in the Management Plane. It has to be noted that in this evaluation, first, an OpenFlow message is generated by a primitive task (*i.e.*, a management operation) into a plan. Second, our prototype was deployed in a cloud environment to offer it on-demand.

Figure 6 depicts the network traffic evaluation results, revealing that the number of OpenFlow messages raised linearly with the number of plans. In our solution, it is possible to know in advance the traffic that a plan can generate because such a traffic is related to the number of primitives per plan; in this test we have a one to one relationship between primitives and OpenFlow messages. Furthermore, the evaluation results confirmed that our proposal does not affect the normal traffic between switches, but it increases the number of OpenFlow messages between the SDN controller and switches; this increasing may be a limiting in front of high traffic conditions. According to the above results, our proposal is a good option for managing SDN by using few resources and delivering a good performance.

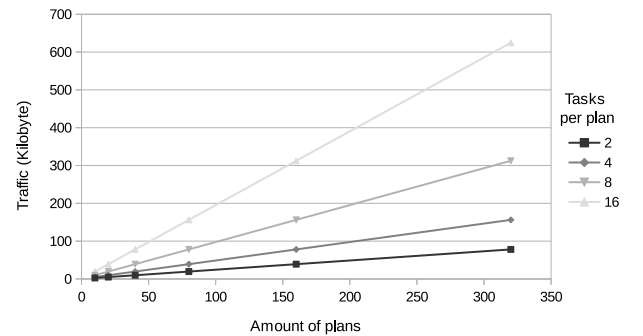


Fig. 6. Traffic versus plans launched

From a qualitative perspective, our proposal inherited the flexibility and workability from MANO and HTN. It is because we instantiate the Orchestrator, VNFM, and VIM as VNFs specialized in performing network management tasks.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced an approach, based on HTN and NFV, for monitoring, configuring and controlling SDN. The time-planning results corroborated that our architecture aids to overcome the low automation of tasks problem presented in related works. We solved this problem by providing an HTN-based proposal with minimal human intervention. This intervention is just necessary for getting information about plans executed and on running and to adapt plans to manage similar network situations, facilitating the work of Administrators. Also, the network traffic results confirmed that our proposal is a feasible solution for managing SDN by using few resources; it is because the plans automatically built do not affect the normal traffic between switches, these plans just increase the number of OpenFlow messages between the controller and switches. Furthermore, our proposal is flexible and workable because it works in a vertical management plane based on NFV MANO and deployed on OpenStack.

As future work, we are interested in supporting Natural Language Processing. We think that the work of Administrators will be even easier by using natural language that allows adding and improving interactions through any device connected to the Internet. Furthermore, we also want to add learning of network situations to build an autonomic network management system for SDN.

REFERENCES

- [1] Y. Zhang, X. Gong, Y. Hu, W. Wang, and X. Que, "Sdnmp: Enabling sdn management using traditional nms," in *IEEE ICCW*, 2015, pp. 357–362.
- [2] J. Wickboldt, W. De Jesus, P. Isolani, C. Both, J. Rochol, and L. Granville, "Software-defined networking: management requirements and challenges," *IEEE Comm Magazine*, vol. 53, no. 1, pp. 278–285, 2015.
- [3] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.
- [4] S. Sezer, S. Scott-Hayward, P.-K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Comm Magazine*, vol. 51, no. 7, pp. 36–43, 2013.
- [5] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "Flowcover: Low-cost flow monitoring scheme in software defined networks," in *IEEE GLOBECOM*, 2014, pp. 1956–1961.
- [6] T. Choi, S. Kang, S. Yoon, S. Yang, S. Song, and H. Park, "Suvmf: software-defined unified virtual monitoring function for sdn-based large-scale networks," in *Proceedings of CFI*. ACM, 2014, p. 4.
- [7] P. H. Isolani, J. A. Wickboldt, C. B. Both, J. Rochol, and L. Z. Granville, "Interactive monitoring, visualization, and configuration of openflow-based sdn," in *IFIP/IEEE IM*, 2015, pp. 207–215.
- [8] J. Lin, R. Ravichandiran, H. Bannazadeh, and A. Leon-Garcia, "Monitoring and measurement in software-defined infrastructure," in *IFIP/IEEE IM*, 2015, pp. 742–745.
- [9] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *IEEE NOMS*, 2014, pp. 1–9.
- [10] G. Yang, K. Lee, W. Jeong, and C. Yoo, "Flo-v: Low overhead network monitoring framework in virtualized software defined networks," in *Proceedings of CFI*. ACM, 2016, pp. 90–94.
- [11] O. M. C. Rendon, F. Estrada-Solano, and L. Z. Granville, "A mashup-based approach for virtual sdn management," in *IEEE COMPSAC*, 2013, pp. 143–152.
- [12] —, "A mashup ecosystem for network management situations," in *IEEE GLOBECOM*, 2013, pp. 2249–2255.
- [13] O. M. C. Rendon, F. Estrada-Solano, V. Guimarães, L. M. R. Tarouco, and L. Z. Granville, "Rich dynamic mashments: An approach for network management based on mashups and situation management," *Computer Networks*, 2015.
- [14] F. Estrada-Solano, A. Ordóñez, L. Z. Granville, and O. M. C. Rendon, "A framework for sdn integrated management based on a cim model and a vertical management plane," *Computer Communications*, 2016.
- [15] C. Ullrich and E. Melis, "Pedagogically founded courseware generation based on htn-planning," *Expert Systems with Applications*, vol. 36, no. 5, pp. 9319–9332, 2009.
- [16] K. Giotis, Y. Kryftis, and V. Maglaris, "Policy-based orchestration of nfv services in software-defined networks," in *IEEE NetSoft*, 2015, pp. 1–5.
- [17] J. Matias, J. Garay, N. Toledo, J. Unzilla, and E. Jacob, "Toward an sdn-enabled nfv architecture," *IEEE Comm Magazine*, vol. 53, no. 4, pp. 187–193, 2015.
- [18] G. Monteleone and P. Paglierani, "Session border controller virtualization towards" service-defined" networks based on nfv and sdn," in *IEEE SDN4FNS*, 2013, pp. 1–7.
- [19] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "Htn planning for web service composition using shop2," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 4, pp. 377–396, 2004.
- [20] A. Ordóñez, J. C. Corrales, and P. Falcarin, "Hauto: Automated composition of convergent services based in htn planning," *Ingeniería e Investigación*, vol. 34, no. 1, pp. 66–71, 2014.
- [21] K. Chen, J. Xu, and S. Reiff-Marganiec, "Markov-htn planning approach to enhance flexibility of automatic web service composition," in *IEEE ICWS*, 2009, pp. 9–16.
- [22] I. Georgievski and M. Aiello, "An overview of hierarchical task network planning," *arXiv preprint arXiv:1403.7426*, 2014.
- [23] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Comm Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.
- [24] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Comm Magazine*, vol. 53, no. 2, pp. 90–97, 2015.
- [25] S. Biundo and B. Schattberg, "From abstract crisis to concrete relief—a preliminary report on combining state abstraction and htn planning," in *ECP*, 2014.
- [26] G. Milliez, R. Lallement, M. Fiore, and R. Alami, "Using human knowledge awareness to adapt collaborative plan generation, explanation and monitoring," in *HRI*, 2016, pp. 43–50.
- [27] R. T. Fielding and R. N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Transactions on Internet Technology*, vol. 2, no. 2, pp. 115–150, May 2002.
- [28] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: an introduction to soap, wsdl, and uddi," *IEEE Internet Computing*, vol. 6, no. 2, pp. 86–93, March 2002.
- [29] D. Nau, T. C. Au, O. Ilghami, U. Kuter, D. Wu, F. Yaman, H. Munoz-Avila, and J. W. Murdock, "Applications of shop and shop2," *IEEE Intelligent Systems*, vol. 20, no. 2, pp. 34–41, March 2005.
- [30] P. Browne, *JBoss Drools business rules*. Packt Publishing Ltd, 2009.
- [31] S. Schreier, "Modeling restful applications," in *Proceedings of the Second International Workshop on RESTful Design*, ser. WS-REST '11. New York, NY, USA: ACM, 2011, pp. 15–21.
- [32] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, 2012.
- [33] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *ACM SIGCOMM Wksp on Hot Topics in Networks*. ACM, 2010, p. 19.
- [34] RYU. (2011) Welcome to ryu the network operating system (nos) — ryu 4.5 documentation. [Online]. Available: URLhttps://ryu.readthedocs.io/en/latest/
- [35] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar et al., "The design and implementation of open vswitch," in *NSDI*, 2015, pp. 117–130.
- [36] A. Ordóñez, V. Alcázar, J. C. Corrales, and P. Falcarin, "Automated context aware composition of advanced telecom services for environmental early warnings," *Expert Systems with Applications*, vol. 41, no. 13, pp. 5907–5916, 2014.

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/319321474>

A YANG model for a vertical SDN management plane

Conference Paper · August 2017

DOI: 10.1109/ColComCon.2017.8088217

CITATION

1

READS

347

5 authors, including:



Ana Isabel Montoya-Muñoz

Universidad del Cauca

1 PUBLICATION 1 CITATION

SEE PROFILE



Daniela Maria Casas-Velasco

University of Campinas

2 PUBLICATIONS 1 CITATION

SEE PROFILE



Felipe Estrada-Solano

Universidad del Cauca

11 PUBLICATIONS 270 CITATIONS

SEE PROFILE



Armando Ordonez

Universidad del Cauca

62 PUBLICATIONS 156 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



Heavy-Hitters - Elephant Flows [View project](#)



intelligent systems [View project](#)

A YANG Model for a vertical SDN Management Plane

Ana Isabel Montoya-Muñoz*, Daniela Maria Casas-Velasco*, Felipe Estrada-Solano*, Armando Ordonez† and Oscar Mauricio Caicedo Rendon*

*Grupo de Ingeniería Telemática, Departamento de Telemática, Universidad del Cauca

†University Foundation of Popayán

e-mail: {aimontoya,danikasas,festradasolano,jaordonez,omcaicedo}@unicauca.edu.co

Abstract—Specifying a Management Plane for the Software-Defined Networking (SDN) architecture is essential for monitoring, configuring and handling computer networks. Diverse solutions have proposed it but they share some shortcomings: incomplete representations of SDN elements, few human readable languages for managing networks, no communication management aimed at Internet Engineering and absence of supporting communication between Autonomous Systems. In this paper, we introduce a protocol-agnostic Data Model based on YANG that specifies a vertical SDN management Plane, handles technology heterogeneity and supports inter-domain communication. To test our Data Model, we evaluate a prototype in an SDN configuration scenario that includes devices supporting diverse technologies. The obtained results provide directions that corroborate the efficiency of our approach in terms of time-response and network traffic.

Keywords—Software-Defined Networking, Network Management, Data Model, YANG.

I. INTRODUCTION

Computer networks are formed by a lot of elements that involve complex management tasks [1]. The Software-Defined Networking (SDN) allows managing networks by a centralized logic, a programmatic configuration and open standards-based protocols [2]. The SDN architecture is composed of four Planes [3-6]: Data, Control, Application and Management. The last Plane is in charge of carrying out the functions of Operation, Administration, and Management (OAM). The management of SDN presents some challenges [5] [7-8] such as performance and scalability, monitoring and visualization, continually changing network state, low-level per-device network configuration, and lack of frameworks for integrated network management. To address these challenges, the specification of a Management Plane became an important requirement because it allows integrally monitoring, configuring, and handling an entire SDN and avoids the recurrent mistake of patching management solutions [5].

To define a Management Plane is priority to describe its Information Model (IM) [9]. An IM represents entities, relationships and operations that specify the semantic of data in a particular domain such as network management [10]. In this domain, for instance, an IM allows conceptually modeling

managed objects, regardless of specific protocols used to transport data. An IM is formed by a collection of Data Models (DMs) and relationships which are defined at a lower level of abstraction [11].

Diverse works have proposed management solutions considering DMs and, in general, modeling languages. For instance, the Common Information Model (CIM) has been used as an extension schema to model both SDN [12] and Core Information Model [1] that provides a Data Plane representation. The Yet Another Next Generation (YANG) language has been used jointly with the OpenFlow Protocol to carry out conversion of parameters from OFConfig to Open Virtual Switch Data Base (OVSDB) [13], in configuration state models for OpenConfig [14], and for defining data model routing [15]. Notwithstanding the contributions performed by the above works, they share shortcomings: *a)* incomplete representation of SDN elements, *b)* use of languages for network management that are not very understandable by humans, *c)* no communication management aimed at Internet Engineering; and *d)* no supporting on communication between multiple Autonomous Systems (ASs).

In previous work, we introduced a DM that characterizes an SDN vertical Management Plane as a CIM conceptual framework [2]. We conclude that CIM turned difficult since the language that it uses is the Extensible Markup Language (XML) and such a language is few understandable by humans. Furthermore, such a CIM framework considers a heterogeneity constrained to use resources of distinct technologies as well as does not consider inter-domain communication management. In this paper, we propose a technology-agnostic YANG-based DM for a vertical SDN Management Plane that supports management tasks in a technology-agnostic and heterogeneous environment. Also, we consider the management entities needed to communicate different ASs and, further, our DM supports OAM functions in a reliable and solid way. To test the proposed DM, we evaluate a prototype in an SDN configuration scenario that includes devices supporting diverse technologies. The obtained results provide directions that corroborate the efficiency of our DM in terms of time-response and network traffic.

The remainder of this paper is organized as follows. In Section II we present the related work. In Sections III and IV, we overview our Management Plane and expose from a high-level point of view the YANG-based DM, respectively. In Section V, we present an evaluation of the proposed model.

In Section VI we expose conclusions and future work.

II. RELATED WORK

The systematic review performed for this paper discloses diverse facts: *i)* the investigations that include YANG and CIM [2] [12-13] [16] [17-21] propose solutions for network management, *ii)* few works [22-23] do not employ any modeling language for their approach in network management, both *i* and *ii* highlight the trending fact of using modeling languages in management networks because of their performance, reliability, and security features, *iii)* the most works that implement YANG do not consider SDN [15-16] [17-21], except [13] that uses YANG data models for implementing OFConfig, *iv)* nonetheless the use of CIM in recognized protocols as OpenFlow, CIM studies have less trending interest than YANG studies because the YANG approach is oriented to the Internet networking, *v)* to the best of our knowledge, none work presents a proposal that includes both SDN management and YANG concepts to define the Management Plane, but some works do it separately, for instance, [13] and [20] work with YANG and [2] orients its approach to SDN management, *vi)* as far as we know, few works had considered inter-domain communication in SDN environments but do not consider its management, such as [24] that proposes components to support communication between ASs but it does not consider it from a management perspective, *vii)* YANG is mostly used in real environment implementations [16] [18-19] and CIM in prototypes [2] [12] what exposes the confidence put in YANG for management networks; and *viii)* works with prototypes as result focused on configuration management [2] [12-13] [17] [20-21] [24].

Unlike the above works, in this paper, we model the Management Plane by our DM. These are based on YANG to support SDN management FCAPS+Programming functions [25] in a technology-agnostic and independent-protocol way. Furthermore, our DM also considers the management of inter-ASs communication. For the sake of brevity and space, we present our DM from a high-level point of view by exposing just the entities and the main relationships that between them. The entire YANG-based DM is complete in our GitHub repository [26].

III. VERTICAL MANAGEMENT PLANE

The vertical Management Plane includes elements that allows configuring, controlling and monitoring resources and handling their heterogeneity and inter-domain communication in SDN-based networks. Figure 1 presents a high-level overview of our Management Plane by the four OSI submodels [9]: *(i)* an Information Model for abstracting SDN resources, *(ii)* an Organizational Model for specifying roles and collaboration, *(iii)* a Communication Model for transferring data; and *(iv)* a Functional Model to structure management features including the communication between diverse ASs.

The Management Plane detailed in the AS₁ has a Managing and a Managed section. The Managing section holds elements that allows controlling, configuring and monitoring the networks and the Managed section holds the Data, Control

and Application Planes with the respective elements to be managed. The Management Plane is formed by the Manager, Adapters and Agents entities, the Data Repository, InterAS Data Repository and Management Interfaces. The Manager entity coordinates and deploys Management Services to carry out SDN management tasks. Each one of Management Services deploys a user interface for Network Administrators (NetAdmins) interaction. The Data Repository and the InterAS Data Repository hold a Resource Representation Model (RRM) and serve the Manager to store management instance data belonging to an AS. RRM handles metadata to provide an abstract, technology-neutral characterization of SDN resources. The InterAS Data Repository holds information relevant from other ASs to manage inter-domain communication. The Adapter entities allow a protocol-agnostic communication between the Manager and Agents entities by well-defined Management Interfaces. Each Management Interface connects both corresponding adapter and agent. The Agents entities inside SDN resources act on behalf of Managers.

Figure 1 also represents the entities hosted by the four submodels. The Organizational Model is composed by the Agent, Manager and Adapter entities. Each organizational component is named as its corresponding entity or role. The Manager acts as a manager entity. NetApp, NetInterAS, NOS, NetSlicer, and NetDev Adapters perform an adapter role. In turn, NetApp, NetInterAS, NOS, NetSlicer, and NetDev Agents carry out the agent tasks by performing an agent role. The Communication Model is formed by the User Interface, the Repository Interfaces, the Adapter Interfaces and the Management Interfaces. In order to match each agent with its respective adapter, we establish a Management Interface (MI) per SDN managed resource: NetApp MI, NOS MI, NetSlicer MI, and NetDev MI. The Functional Model classifies the Management Services according the management FCAPS+Programming functions.

In this work, we focus on the DM of the Management Plane composed by RRM, Instance Data and both InterAS Data and Data Repositories. The entire operation of the Management Plane is specified in RRM to achieve an integrated and technology-independent SDN management.

IV. YANG-BASED DATA MODEL FOR A VERTICAL MANAGEMENT PLANE

This paper proposes a YANG-based DM that characterizes the SDN architecture presented in Figure 1. We model the network entities, relationships, and elements by defining the hierarchical structure of YANG module (*i.e.*, YANG statement composed by three types of statements: module header, revision, and definition statements) that interrelates a collection of submodules (*i.e.*, portions of its definitions of a module). Each YANG module defines a hierarchy of data for defining configurations, state data, Remote Procedure Calls and notifications. Modules can import data from other external modules and include data from submodules [27]. Submodules are partial modules that contribute definitions to modules. Each submodule must belong to only one module [28].

Figure 2 introduces our YANG-based DM for the Management Plane. This model is divided into submodules. Each

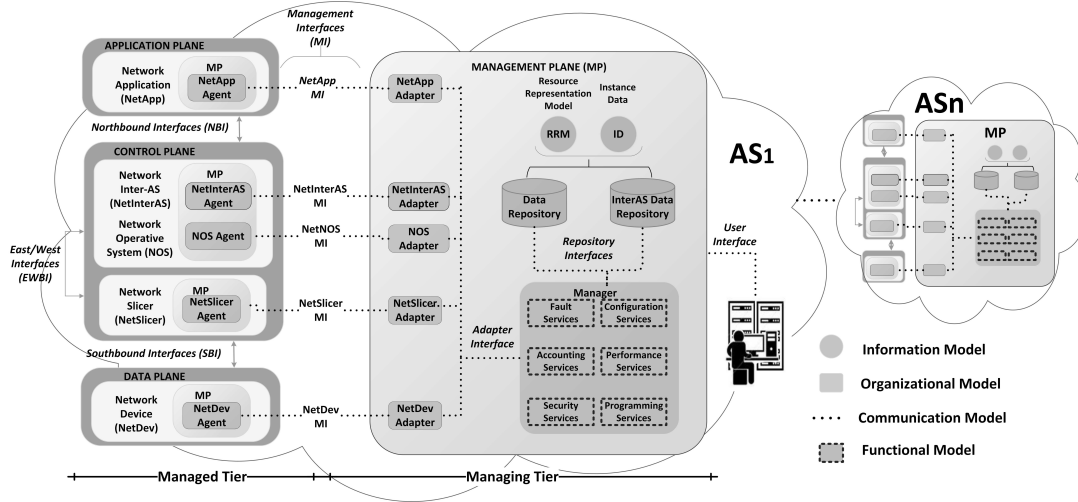


Figure. 1. Vertical Management Plane

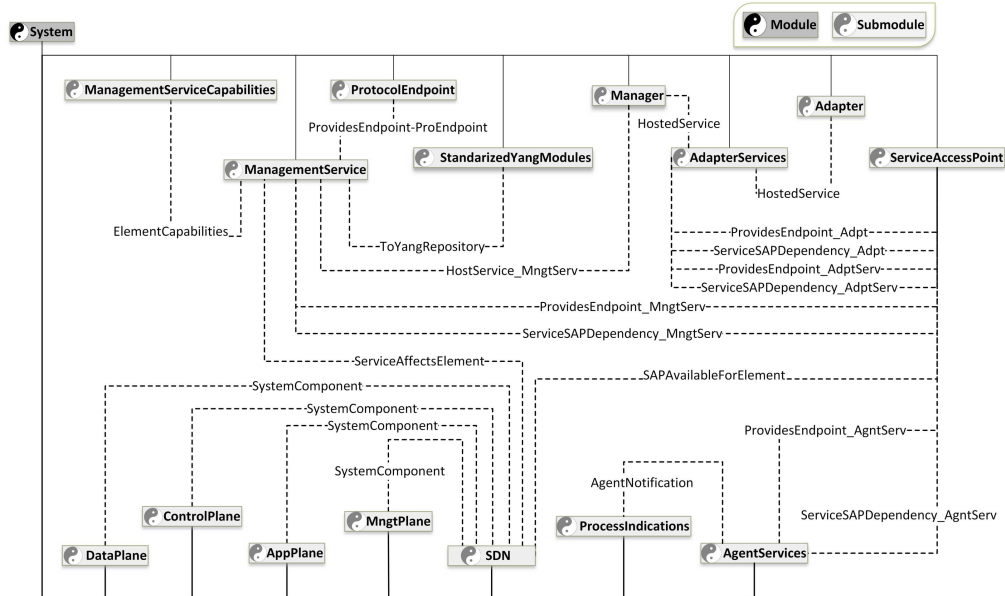


Figure. 2. YANG model for Management Plane

submodule defines the node schema used to model all Planes. The submodules for the Vertical SDN Management Plane are: i) SDN for modeling the SDN network composed by the following Planes: ii) MngtPlane, iii) AppPlane, iv) ControlPlane, and v) DataPlane submodules that are defined to represent each SDN architectural Plane, vi) Manager for representing the system hosting the Management Services, vii) ManagementService for representing Management Services that allow carrying out different SDN management functions, viii) ManagementServiceCapabilities for describing both supported and excluded capabilities in ManagementServices, understanding capabilities as device features, ix) ServiceAccessPoint for defining the communication points between nodes in a data tree

model, it instances communication channels according to the interface required, x) AdapterServices for modeling adapters that parse and forward requests, responses, and notifications between ManagementServices and AgentServices by Management Interfaces defined in the ServiceAccessPoint, xi) Adapter for representing a system that may host the AdapterServices, xii) AgentServices for representing Agents running on SDN managed resources, xiii) ProcessIndication for modeling notifications that may be sent by AgentServices informing about changes in SDN managed resources, xiv) ProtocolEndPoint for modeling the communication point allowing the access of NetAdmins to networks; and xv) StandardizedYANGModules for allowing accessing to well known YANG models that may

be required when modeling a network.

Figure 2 also depicts the relationships between the different submodules that comprise our DM of the Management Plane. The SystemComponent relationships connect the SDN node with the Planes of App, Control, Data, and Management indicating their belonging to SDN. The ServicesAffectsElement relationship between SDN and ManagementService reflects that Management Services have an effect in SDN, such as changing resource behavior and configuring SDN controllers for a set of OpenFlow switches. The SAPAvailableForElement relationship communicates SDN with the ServiceAccessPoint by the management interfaces, it represents that these interfaces provide managing access for SDN. The HostService_MngtServ relationship indicates that the Manager represents the system hosting the SDN Management Services. The ManagementService is related to the ManagementServiceCapabilities submodule by the ElementCapabilities relationship representing the capabilities supported for Management Services.

Relationships tagged as ProvidesEndpoint_“id” have at the end of the name an identifier (id) directly associated with the submodule where is defined (i.e., a YANG “prefix” statement) to identify easily the corresponding submodule connected with the ServiceAccessPoint. These relationships indicate that the nodes related establish their own communication point to access from the other one. Relationships tagged as ServiceSAPDependency_“id” also have at the end of the name an identifier for identifying submodules connected. These relationships imply that both nodes use the ServiceAccessPoint to access the other one. For instance, a Configuration Service and a NetDev Adapter establish a communication using JSON over HTTP. Using this communication, the NetDev Adapter forwards to the Configuration Service a notification from a NetDev Agent that reports about misconfiguration failures. Similarly, the Configuration Service may use the same channel to fix this failure by sending a configuration request to the NetDev Adapter. This Adapter forwards this request to the corresponding NetDev Agent.

The HostedService relationships that connect AdapterServices submodule with the Manager and Adapter indicate that the Adapter Services may be hosted by the Manager or the Adapter. The AgentNotification relationship communicates AgentServices submodule with ProcessIndications submodule and represents the notifications sent by Agents toward ManagementService. In particular, the ProvidesEndpoint_ProEndpoint relationship implies that the ManagementService supplies ProtocolEndPoint users with the corresponding user interface. The ToYANGRepository relationship relates ManagementServices submodule with StandarizedYANGModules submodule to consider well-known YANG modules.

V. EVALUATION

To assess our DM, first, we establish a network management scenario with different SDN technologies. Second, we implement a prototype that includes the DM proposed. Third, we evaluate in terms of time-response and network traffic, the prototype by using the scenario mentioned.

A. Scenario

Let us consider that a Bank deployed an SDN network with NetDevs handled by an NOS₁. Initially, the SDN network offered NetDevs from *Vendor A*. However, there was the need to extend services to the rest of the bank branches for handling more users. In this sense, the NetAdmin decided: *i)* deploy NetDev resources from a *Vendor B* because it offers a better benefit-cost ratio than *Vendor A* and; *ii)* install a NOS₂ that offers better features for handling the entire network. Now, the NetAdmin faces the challenge of configuring heterogeneous NetDevs by NOS₂.

The NetAdmin typically uses *Vendor A* Tool and *Vendor B* Tool to execute specific, complex and individual configuration commands on NetDevs. Instead, we propose an *Integrated Solution* based on our DM to configure the NetDevs in a high-abstraction level. Our *Integrated Solution* allows the NetAdmins to seamlessly configure NetDevs from NOS₂ since this solution hides the complexity of managing heterogeneous SDN-based networks.

In particular, in the test environment, NetDevs provided by *Vendor A* were OVSDB switches. NetDevs from *Vendor B* were OF-CONFIG switches. The NOS₁ was a RYU controller and the NOS₂ was an OpenDayLight controller. The managing operation used during the tests was called *SetController*. This operation is to configure NetDevs with distinct management interfaces (i.e., OVSDB and OF-CONFIG switches) and to define NOS₂ as their new controller.

B. Implementation

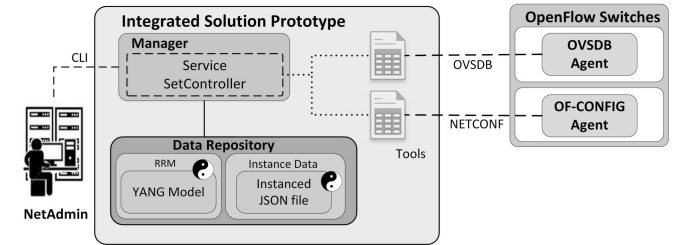


Figure 3. Integrated Solution prototype

Integrated Solution. We built this prototype to perform the *SetController* operation regardless the different configuration interfaces by using our DM. Figure 3 presents the implemented *Integrated Solution* which contains the Data Repository formed by RRM, particularly, the YANG DM, and the Instance Data described in a JSON file. By using a tool called pyang [29], we translated the DM into XSL and XML files to generate a skeleton in JSON format. To carry out the *SetController* operation, the NetAdmin connects by a CLI to the Manager which deploys this operation as a Configuration Service. This Configuration Service invokes tools to change the NOS of the OpenFlow switches based on the instance data from the JSON file.

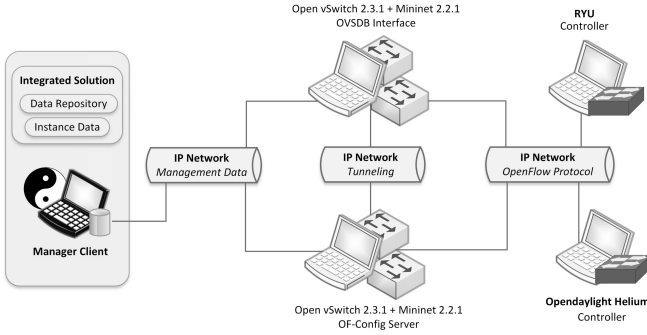


Figure 4. Test environment

C. Test environment

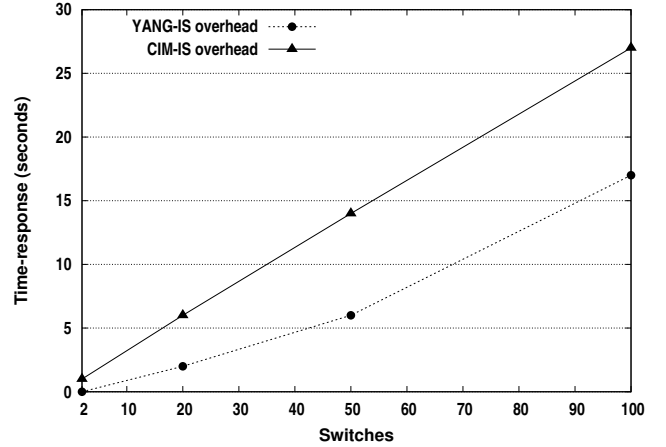
Figure 4 depicts the environment formed by two OpenFlow networks, two OpenFlow controllers and the Manager Client. The Manager Client contains the Data Repository and the Integrated Solution. A tunnel over an IP network interconnected the root switches from each tree topology. The Open vSwitches used the OpenFlow protocol over a second IP network to communicate with a specific OpenFlow controller. A third IP Network transmitted the management data between the OpenFlow networks and the Manager Client. We measured the third IP Network because it represented the management interface.

D. Results and analysis

To evaluate the prototype built, we carried out a comparison in terms of time-response (*i.e.*, time in seconds s that the Manager Client spends when executing the *SetController* operation) and network traffic (*i.e.*, the amount of data in kilobytes KB transmitted and received by the network interface of the Manager Client) with our previous approach [2]. For this comparison we executed the *SetController* operation to measure its time-response and network traffic when configuring 2, 20, 50 and 100 switches per evaluation. The half were OVSDB Switches, and the other were OF-CONFIG Switches. The overhead is the difference between both *Integrated* and *Isolated Solutions*. In all evaluation cases, we took the average values for 30 measurements with a 95% confidence level.

In terms of time, Figure 5 presents the difference between the overhead of the CIM-based *Integrated Solution* (CIM-IS) and the overhead of the YANG-based *Integrated Solution* (YANG-IS), disclosing that the CIM-IS takes longer time than the YANG-IS because the CIM-IS used additional components to build its Manager and execute the Configuration Service (*e.g.*, CIMOM as data repository for a CIM-based IM [30]). In fact, the CIM-IS takes 1.25s and 3.97s more than YANG-IS for 2 switches and more than 20 switches, respectively, to cope with the technology heterogeneity. It is noteworthy that YANG-IS only used the JSON file to execute the Configuration Service.

In terms of network traffic, the comparison of the YANG-IS with the CIM-IS discloses: *i)* as expected, the difference

Figure 5. Comparison between the YANG and CIM *Integrated Solutions*

between the overheads of both solutions is big, that is around 98.2KB and 414KB when the number of configured switches (N_S) are 2 and 20, respectively, *ii)* the CIM-IS generates more network traffic than the YANG-IS (approximate 18KB more per switch when $N_S > 20$), since the YANG-IS only uses a JSON file for instantiating the DM and it is less verbose than the XML language used in the CIM-based solution; and *iii)* both *Integrated Solutions* have a good performance of network traffic while executing the *SetController* operation with a difference between the overheads that increases proportionally with N_S . Also, it is to highlight that the network traffic grows around 124KB and 109KB per switch for the solutions based on CIM and YANG, respectively. Note that CIM-IS needs to connect and disconnect with the CIMOM repository during the configuring and authenticating actions per switch which generates additional traffic. Instead, YANG-IS does not need to communicate with other applications or components.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced a protocol-agnostic YANG-based DM. Such a DM characterizes a vertical SDN Management Plane, handles technological heterogeneity and considers inter-domain communication. The evaluation demonstrated the efficiency of our DM prototype in a realistic scenario, it established the challenge of configuring SDN with heterogeneous technologies by a common interface for the NetAdmin. Our prototype proved to be successful when used to overcoming such a challenge in terms of time-response and network traffic. In fact, our prototype has a shorter time-response and is lesser network traffic generator than CIM-IS.

As future work, we intend to detail all our DM for the Vertical Management Plane and the other SDN planes by YANG statements. Furthermore, we plan to develop more proof-of-concepts of the entire DM.

REFERENCES

- [1] ONF, "Core information model (core model) v1.1," Open Network Foundation, Technical Reference TR-512, November 2015.

- [2] F. E. Solano, A. Ordóñez, L. Z. Granville, and O. M. C. Rendon, "A framework for sdn integrated management based on a cim model and a vertical management plane," *Elsevier Computer Communications*, vol. 102, pp. 150–164, 2017.
- [3] Standardization and O. ITU, "Y.3300: Global information infrastructure, internet protocol aspects and next generation networks - framework of software-defined networking," 2014.
- [4] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-defined networking (sdn): Layers and architecture terminology," Tech. Rep., 2015.
- [5] J. A. Wickboldt, W. P. de Jesus, P. H. Isolani, C. B. Both, J. Rochol, and L. Z. Granville, "Software-defined networking: management requirements and challenges," *IEEE Communications Magazine*, vol. 53, pp. 278–285, 2015.
- [6] D. Kreutz, F. Ramos, P. Esteves Verissimo, and E. Rothenberg, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 1, pp. 14–76, 2015.
- [7] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.
- [8] V. T. Guimarães, R. O. M. Caicedo, G. L. dos Santos, G. Rodrigues, C. M. D. S. Freitas, L. M. R. Tarouco, and L. Z. Granville, "Improving productivity and reducing cost through the use of visualizations for sdn management," in *IEEE ISCC, Messina, Italy*, 2016, pp. 531–538.
- [9] C. Nuangjamnong, S. P. Maj, and D. Veal, "The osi network management model- capacity and performance management," in *IEEE ICMIT*, 2008, pp. 1266–1270.
- [10] Y. T. Lee, "An overview of information modeling for manufacturing systems integration," *National Institute of Standards and Technology*, 1999.
- [11] *RFC 3444 - On the difference between Information Models and Data Models*, Network Working Group Std.
- [12] B. Pinheiro, R. Chaves, C. Eduardo, and A. Abelem, "Cim-sdn: A common information model extension for software-defined networking," *IEEE GLOBECOM, Atlanta, GA USA*, pp. 836–841, 2013.
- [13] R. Narisetty, L. Dane, A. Malishevskiy, D. Gurkan, S. Bailey, S. Narayan, and S. Mysore, "Openflow configuration protocol: Implementation for the of management plane," in *IEEE GREE*. Salt Lake, USA, 2013, pp. 66–67.
- [14] A. S. et al. (2016) Data models and apis. [Online]. Available: <http://openconfig.net/documentation/>
- [15] L. Lhotka and A. Lindem, *A YANG Data Model for Routing Management*, IETF Std., 2016. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-netmod-routing-cfg-25>
- [16] S. Wallin and C. Wikstrom, "Automating network and service configuration using netconf and yang," in *LISA11 (USENIX), Boston, MA USA*, 2011, pp. 22–22.
- [17] A. Lahmadi, E. Nataf, and O. Fester, "Yang-based configuration modeling - the secsip ips case study," in *IFIP/IEEE IM 2011, Lisbon, Portugal*, 2011, pp. 800–811.
- [18] J. Schönwälder, M. Björklund, and P. Shafer, "Network configuration management using netconf and yang," *IEEE Communications Magazine*, vol. 48, pp. 166–173, 2010.
- [19] E. Nataf and O. Fester, "End-to-end yang-based configuration management," in *IEEE NOMS, Osaka, Japan*, 2010, pp. 674–684.
- [20] X. Liu, P. Sarda, and V. Choudhary, "A yang data model for routing information protocol (rip)," Jabil, Ericsson and Huawei Technologies, Tech. Rep., January 2017. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-rtgwg-yang-rip-03>
- [21] E. A. Lindem, Y. Qu, D. Yeung, I. Chen, and J. Zhang, "Routing key chain yang data model," Cisco Systems, Huawei, Arrcus inc, Jabil and Juniper Networks, Tech. Rep., April 2017. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-rtgwg-yang-key-chain-24>
- [22] A. Malishevskiy, D. Gurkan, L. Dane, S. Narisetty, Rajarevanth; Narayan, and S. Bailey, "Openflow-based network management with visualization of managed elements," in *IEEE GREE*, Atlanta, USA, 2014, pp. 73–74.
- [23] Google, "Inter-datacenter wan with centralized te using sdn and openflow," in *Google Inc*, 2012. [Online]. Available: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/customer-case-studies/cs-google-sdn.pdf>
- [24] R. Bennessy, P. Fonseca, E. Mota, and A. Passito, "An inter-as routing component for software-defined networks," in *IEEE NOMS, Maui, Hawaii*, 2012, pp. 138–145.
- [25] *Recommendation M.3400 : TMN management functions*, International Telecommunication Union Std.
- [26] D. M. Casas, A. I. Montoya, and O. M. Caicedo. (2017) Yang-based information model repository. [Online]. Available: <https://github.com/DaniCasas1007AnaMontoya/YANG-InformationModel-for-SDN>
- [27] C. Wildes and K. Koushik, *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*, IETF Std., 2015. [Online]. Available: <http://tools.ietf.org/html/rfc6020>
- [28] M. Björklund, "RFC 7950: The YANG 1.1 data modeling language," 2016. [Online]. Available: <https://tools.ietf.org/html/rfc7950>
- [29] Björklund. (2014) Pyang. [Online]. Available: <http://www.yang-central.org/twiki/pub/Main/YangTools/pyang.1.html>
- [30] DMTF, "Common Information Model (CIM) Infrastructure v2.7.0," Distributed Management Task Force, Tech. Rep. DSP0004, 2012. [Online]. Available: http://www.dmtf.org/sites/default/files/standards/documents/DSP0004_2.7.0.pdf

Machine Learning for Cognitive Network Management

Sara Ayoubi, Noura Limam, Mohammad A. Salahuddin, Nashid Shahriar, Raouf Boutaba, Felipe Estrada-Solano, and Oscar M. Caicedo

The complexity, heterogeneity, and scale of networks have grown far beyond the limits of manual administration. Furthermore, the main cause of outages in many network environments is human error. This has triggered a shift in the design philosophy of network management systems to minimize the role of humans in the control loop.

ABSTRACT

Over the last decade, a significant amount of effort has been invested on architecting agile and adaptive management solutions in support of autonomic, self-managing networks. Autonomic networking calls for automated decisions for management actions. This can be realized through a set of pre-defined network management policies engineered from human expert knowledge. However, engineering sufficiently accurate knowledge considering the high complexity of today's networking environment is a difficult task. This has been a particularly limiting factor in the practical deployment of autonomic systems. ML is a powerful technique for extracting knowledge from data. However, there has been little evidence of its application in realizing practical management solutions for autonomic networks. Recent advances in network softwarization and programmability through SDN and NFV, the proliferation of new sources of data, and the availability of low-cost and seemingly infinite storage and compute resource from the cloud are paving the way for the adoption of ML to realize cognitive network management in support of autonomic networking. This article is intended to stimulate thought and foster discussion on how to defeat the bottlenecks that are limiting the wide deployment of autonomic systems, and the role that ML can play in this regard.

INTRODUCTION

The complexity, heterogeneity, and scale of networks have grown far beyond the limits of manual administration. Furthermore, the main cause of outages in many network environments is human error [1]. This has triggered a shift in the design philosophy of network management systems to minimize the role of humans in the control loop.

In 2001, IBM proposed the autonomic computing initiative. The vision was to have strategies for self-* (i.e., self-configuring, self-healing, self-optimizing, and self-protecting) IT systems, a goal also shared by HP's Adaptive Enterprise and Microsoft's Dynamic Systems. As part of this initiative, IBM proposed an architecture for autonomic computing [2], where autonomic managers maintain a Monitor-Analyze-Plan-Execute (MAPE) over shared knowledge control loop with the managed resources. Since then, several extensions to the MAPE control loop have been proposed, including Foundation-Observe-Com-

pare-Act-Learn-rEason (FOCALE) [1] that is based on Observe-Orient-Decide-Act (OODA) [1]. Essentially, FOCALE offers extensions for knowledge use and learning with dynamic control loops, namely reactive, deliberative and reflective with increasing level of cognitive capabilities. However, IBM's MAPE remains the most widely adopted with many incarnations of its vision proposed for networking, such as *cognitive networks* [1, 3] and *knowledge-driven networking* [4, 5]. In essence, these initiatives advocate for incorporating intelligence and autonomy in network management.

Autonomic networks call for automated decisions for management actions. This can be achieved via policy-based management (PBM) through a set of pre-defined self-* policies engineered from human expert knowledge or derived from high-level policies provided by humans. Activating backup resources upon predicting a fault and steering traffic flows through deep packet inspection (DPI) based on a blacklist are examples of self-healing and self-protection policies, respectively. However, considering the high complexity of today's networking environments, engineering sufficiently accurate knowledge is a cumbersome task. This has prohibited the practical deployment of autonomic systems. In autonomic systems, possible actions should be *learned* from the operating environment, and *reasoned* and *adapted* to changes, while respecting operational goals and requirements.

Machine learning (ML) is a popular technique for extracting knowledge from data. In theory, ML can be used for automating network operations and management. However, there has been little evidence of its application in realizing autonomic networks. Prohibiting factors include the distributed control and vendor-specific nature of legacy network devices, lack of available data, and cost of compute and storage resources. Several technological advances have been made in the last decade to overcome these limitations. The advent of network softwarization and programmability through software-defined networking (SDN) and network functions virtualization (NFV) offers centralized control and alleviates vendor lock-in. The advances in ML along with the proliferation of new sources of data and big data analytics platforms provide abundant data and extract knowledge from them. For instance, recent breakthroughs in deep learning (DL) allow generation of models from raw data without the

Fundamental to ML is feature extraction that determines the best discriminators for learning and inference. Note that learning splits the data into training and validation sets. This split can conform to the 80:20 or 70:30 ratio rule-of-thumb; or follow the k-fold cross validation technique.

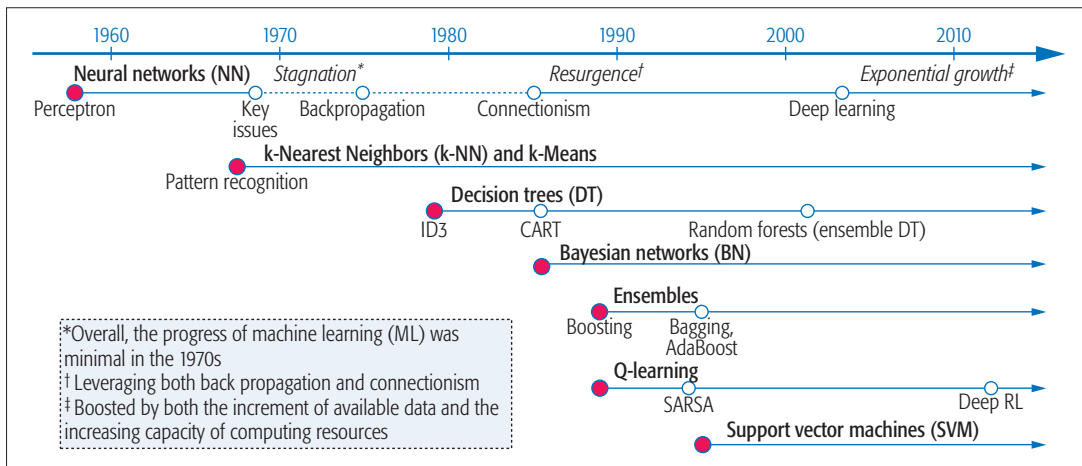


Figure 1. The evolution of machine learning algorithms with key milestones.

need for data labeling. Furthermore, the availability of seemingly *infinite* storage and compute resources through the cloud overcomes the cost of resources. These together provide the environment to realize the vision of autonomic networks. The main contributions of this article are:

- We expose how ML can be used to realize autonomicity in each of the fault, configuration, accounting, performance, and security (FCAPS) [6] management areas.
- We show how ML can be leveraged to realize a cognitive MAPE control loop for network management.
- We discuss the opportunities and challenges pertaining to using ML for the management of autonomic networks.

The remainder of this article is organized as follows. We provide a high-level background on ML in the following section. After that we highlight how ML has been leveraged for network management. We showcase how ML can be used to realize a cognitive control loop, and present a use case illustrating its realization in practice using existing technologies and protocols. Then we present future research directions to facilitate a holistic cognitive management framework.

MACHINE LEARNING

ML goes beyond learning or extracting knowledge to utilizing it and improving it with experience. It has given rise to a plethora of algorithms, as shown in Fig. 1. Essentially, ML is applied to problems that can be solved using *inference* [7] and have large representative training *data*. Fundamental to ML is *feature extraction*, which determines the best discriminators for learning and inference [8]. Note that *learning* splits the data into training and validation sets. This *split* can conform to the 80:20 or 70:30 ratio rule of thumb, or follow the *k*-fold cross-validation technique.

ML is classified into three categories, based on how the learning is achieved [1].

Supervised Learning: uses labeled training datasets to create models that map inputs to corresponding outputs. Typically, this approach is used to solve *classification* and *regression* problems that pertain to predicting discrete or continuous valued outputs, respectively. For example, a classification problem can be to identify an attack as either denial of service (DoS), root-to-local

(R2L), user-to-root (U2R), or probing. A regression problem can be to predict the time of future attacks.

Unsupervised Learning: uses unlabeled training datasets to create models that find dominating structure or patterns in the data. This approach is appropriate for *clustering*, *outliers detection*, and *density estimation* problems. For example, the clustering problem can pertain to grouping different instances of attacks based on their similarities.

Reinforcement Learning (RL): is an iterative process that uses the feedback from the environment to learn the correct sequence of actions to maximize a cumulative reward. Unlike other approaches that are myopic in nature, RL may sacrifice immediate gains for long-term rewards. Hence, RL is best suited for making cognitive choices, such as decision making, planning, and scheduling [9].

Different ML techniques (Fig. 1) belong to one or more of the aforementioned categories. Bayesian networks (BNs) and support vector machines (SVMs) are typically applied in supervised learning. *k*-nearest neighbors (*k*-NN), decision trees (DTs), and neural networks (NNs) have been used in both supervised and unsupervised learning. *k*-means operates only in unsupervised learning. Q-learning, the most prominent RL technique, has recently used NN and deep NN (DNN) to approximate its action-value function (i.e., deep RL).

MACHINE LEARNING FOR NETWORK MANAGEMENT

MACHINE LEARNING FOR FCAPS: WHAT HAS BEEN DONE?

Application of ML to automate network management and close the management loop is a nontrivial task. Table 1 highlights representative ML techniques employed in the literature for the FCAPS management areas to provide some degree of autonomy.

Fault Management: Failure in networks is a norm rather than an exception, and its impact can be quite costly [10]. The slow reaction time and poor accuracy of traditional fault management techniques further increase this cost. This has motivated efforts that leveraged ML for proactive fault prediction. Additional works considered the usage of ML for fault localization and automat-

ed mitigation to minimize downtime and human intervention.

Configuration Management: Operators must implement increasingly sophisticated network policies that have to be translated into constrained low-level configuration commands, and adjusted to changes in network conditions (e.g., intrusions, traffic shifts, performance degradation). As the network state is constantly changing, network managers find themselves constantly configuring the network to adapt to these changes, which is a cumbersome and error-prone process. ML can help automate this process by training models to identify optimal state-action pairs as the network behavior changes over time. A handful of works have showcased the benefits of ML for dynamic resource allocation and service configuration.

Accounting Management: Accounting is tightly coupled with business and control models. These models leverage accounting data in

decision making, service planning, and delivery, and designing tariffs and pricing plans. Therefore, it is essential to ensure the integrity of accounting data by accurate collection of usage data and fraud detection. The use of ML for network accounting management is rather unexplored.

Performance Management: Today's networks typically run a variety of services with different performance requirements to serve an increasing number of users with distinct profiles. Guaranteeing performance is a daunting task. In fact, without the ability to accurately predict network behavior, how can we provide such guarantees? This realization has attracted numerous efforts that have leveraged ML for performance and traffic load prediction, and quality of experience/service (QoE/QoS) correlation for proactive and adaptive network performance management [11].

Security Management: The most commonly employed security approach consists of monitoring the network for patterns of well-known threats. However, this renders the network vulnerable to *zero-day* attacks. This vulnerability is critical as new attacks emerge daily [12]. The need for robust security measures is clear, and the role of ML toward this end has been investigated extensively [13]. Existing efforts have concentrated on using ML for misuse detection in order to learn complex attack patterns from historical data and generate generic rules that allow detecting variations of known attacks. Anomaly detection using ML has also been explored to detect zero-day attacks. This consists of learning patterns of normal behavior and detecting deviations from the norm.

The aforementioned efforts show promising results toward incorporating cognition in network management. However, leveraging ML for the different network management functions alone will not fulfill the vision of cognitive management. In fact, there is a need for a cognitive control loop, detailed below.

C-MAPE: A COGNITIVE CONTROL LOOP

To date, IBM's architecture for autonomic computing [2] is the most influential reference model for autonomic systems and networks. It comprises several layers of autonomic managers. The behavior of each manager is governed by the MAPE control loop that consists of four functions: *monitor*, *analyze*, *plan*, and *execute*. As shown in Fig. 2, the *knowledge* source is orthogonal to every MAPE function. Functions can retrieve data from and/or log created knowledge to the *knowledge* source. For example, the *analyze* function obtains information about the historical behavior of a managed resource and stores the ML models and the analytics it generates in the *knowledge* source.

In [2], we observe that cognition has been restricted to the *analyze* function, which inhibits the ability to achieve closed-loop cognitive network management. In this article, we propose to incorporate cognition at every function in the loop. For example, the *monitor* function should be able to determine what, when, and where to monitor. ML can be leveraged to build this cognition in every function and allow each function to operate in full autonomy. Therefore, we extend IBM's MAPE control loop into a cognitive control loop we call C-MAPE. As illustrated in Fig. 2,

Management area	Management function	Machine learning techniques
Fault	Fault prediction	NN, k-NN, k-Means, DT, BN, SVM
	Fault localization	NN, k-NN, k-Means, DT
	Automated mitigation	BN, SVM
Configuration	Adaptive resource allocation	Q-Learning, Deep
	Adaptive service configuration	Q-Learning
Accounting	—	—
Performance	Traffic load and metrics prediction	(Ensemble) NN, BN, SVM,
	QoE-QoS correlation	DT, BN, SVM, Q-learning
Security	Misuse detection	NN, DT, BN, SVM
	Anomaly detection	(Ensemble) NN, DNN, k-NN, k-means, (Ensemble) DT, Ensemble BN, SVM

Table 1. Sample machine learning techniques used in FCAPS.

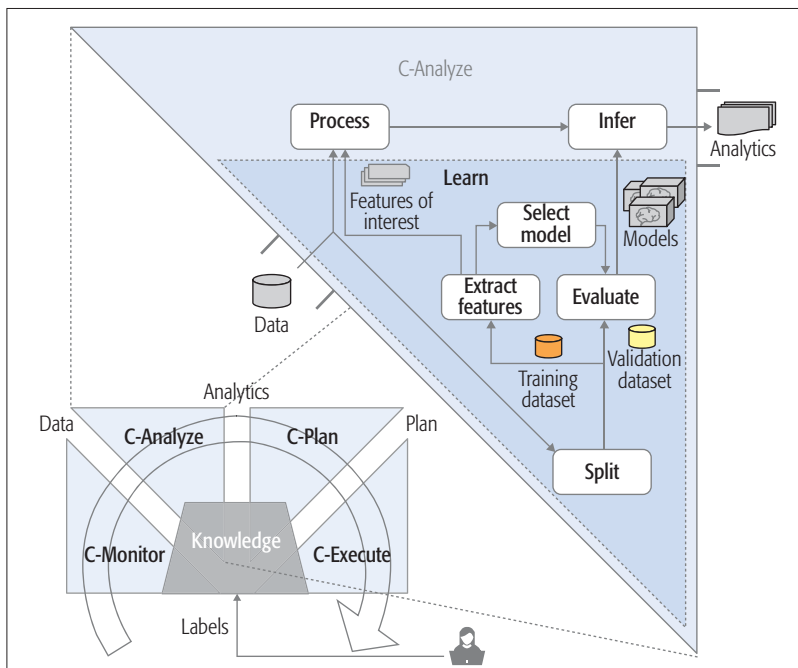


Figure 2. Cognitive control loop for network management.

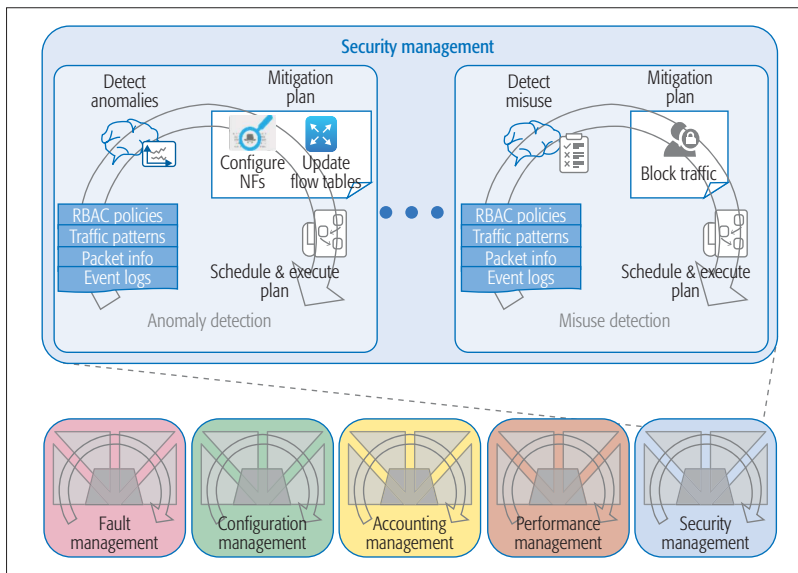


Figure 4. C-MAPE in each function of FCAPS management area.

to install a firewall and configure it to drop packets from the suspicious IP if deemed malicious. Although this use case focuses on C-MAPE for security management, C-MAPE can play a vital role in all areas (e.g., fault, configuration) to offer holistic cognitive network management.

FUTURE RESEARCH DIRECTIONS

MACHINE LEARNING FOR FCAPS: WHAT CAN BE DONE?

Above, we highlighted the previous efforts toward employing ML in FCAPS. These efforts are indeed fundamental toward realizing autonomic network management. As illustrated in Fig. 4, C-MAPE will reside in every management area, and within different functions of every area. However, as we survey previous and ongoing efforts, we observe that the automation of many management tasks has not been explored yet. As a call for action, we identify further research opportunities, where ML can be applied with respect to FCAPS.

Fault Management:

Failure Prevention: ML has been used for proactive failure prediction, leaving the localization and mitigation steps mostly as reactive. However, proactive mitigation combined with fault prediction can help prevent upcoming failures. Since a proactive mitigation approach requires a set of actions to be taken, RL can be a prospective candidate. To select the appropriate mitigation step, the root cause of the predicted fault has to be precisely identified. Existing ML-based localization approaches suffer from poor scalability when analyzing the high-dimensional device log attributes in moderate-size networks. Dimensionality reduction can be leveraged to improve the scalability of fault localization techniques without sacrificing accuracy.

Fault Management in Cloud and Virtualized Environments: The efforts on fault management discussed above focus on single tenant networks. The advent of new technologies, such as multi-tenancy in cloud and virtualization of network functions, magnify the complexity and dimensions of the fault space in a network. For instance, any fail-

ure in the underlying physical resource can propagate to the hosted virtual resources. However, the reverse is not always true. To predict and locate faults in such networks, we can use DNNs, which can model complex multi-dimensional state spaces. A reliable virtual network (VN) embedding algorithm can leverage these predictions to set up a VN. Furthermore, any automated mitigation within a VN should not affect the operations of other coexisting VNs. Here, RL combined with DNNs can learn to optimize mitigation steps.

Configuration Management:

Mapping High-Level Requirements to Low-Level Configurations: Networks are configured to satisfy certain requirements in terms of performance, connectivity, fault tolerance, security, and so on. The gap between high-level requirements and low-level configurations (e.g., resources to be provisioned) is difficult to bridge. RL techniques can be leveraged in this context. The reward for selecting a configuration setting of a given network element can be thought of as the utility of that particular setting in delivering the high-level requirement under a given network condition.

Configuration Verification: Configuration changes (e.g., access control lists, routing tables) should not conflict with high-level requirements; nor should they adversely affect the expected behavior of the network. Formal methods have been used to analyze and verify network configurations [15]. However, these methods are found to be highly complex and are difficult to scale. A growing interest has been shown in applying DL-aided verification, code correction, and theorem proving. Considering the scale and complexity of the configuration parameters in a network, a DL-based verification approach is worth exploring.

Configuration Rollback: After verification, a decision is made to either accept the configuration changes or to revert (some of) them back to a previous stable state. To avoid service disruption and performance degradation, it is essential that the stable state is reached with minimum delay. Assuming that snapshots of stable configurations are logged, the system still has to decide in what order the changes should be applied, while minimizing both disruption and delay. Here, RL can be used to find the optimal rollback strategy, assuming, for instance, that rollbacks are assigned rewards that are inversely related to the incurred disruption and time-to-stable state.

Accounting Management:

Making Accounting Smart: Collecting accurate customer usage yields opportunities for increasing customer experience and resource utilization, and reducing cost of operations. The collected customer usage data can be leveraged by supervised and unsupervised ML models to deduce norms and predict customers' usage habits. These models identify deviations from the norm, triggering misuse detection and root cause analysis for fraudulent activities. Such prediction models can also help service providers to convene smart pricing schemes and smart forecasted bills, provide incentives, and bundle services. Furthermore, these models can facilitate opportunistic, dynamic, and proactive provisioning of resources to improve QoS.

Performance Management:

Adaptive Probing: Obtaining measurements from the network is required for monitoring network behavior. However, the large number of devices in the network, the variety of parameters to measure, and the small time intervals to log data exponentially increase the amount of traffic overhead, resulting in network performance degradation. Regression, mostly based on time series data, can predict the value of the measured parameters to optimize probing. The goal is to set probing rates that keep traffic overhead within a target value, while minimizing performance degradation and providing high prediction accuracy.

Detecting Patterns of Degradation: Poor QoS can be addressed by monitoring network metrics that indicate performance degradation. The challenge is to detect the characteristic patterns of degradation before the quality drops below an acceptable level. This information can be reported to the network administrator or used for autonomic tuning of network parameters to achieve optimum performance. Here, elastic resource allocation can be leveraged to dynamically accommodate user demands for achieving optimum performance while maximizing resource utilization. As aforementioned, supervised learning has been used to predict the value of network performance metrics. However, employing performance prediction for autonomic tuning of the network behavior remains an open challenge.

Security Management:

Reducing Classification Errors: While ML for anomaly detection has received significant interest in the literature, it has not yet been employed in practice. This is mainly due to their high false positive rate that wastes expensive network analyst time. Hence, further efforts are needed to reduce classification errors in anomaly detection. Some promising research directions include the use of alarm post-processing (e.g., correlation, filtering, prioritization) and the correlation of host-based and network-based data traces. The latter allows the detection of threats that fail to be detected at the network level but exhibit anomalous behavior on the host and vice versa. Ensemble learning has also shown encouraging results in terms of its ability to overcome data skew.

Security of Management: An important aspect of management is ensuring that the management interface itself is secured. This is primarily achieved by limiting access to authorized users. However, it cannot guarantee safety against malicious authorized users. Indeed, an authorized user with malicious intent can create havoc. To the best of our knowledge, there has been no work tailored to address the security of management, leaving it as an open research direction.

Autonomic Security Management: Anomaly detection has been extensively explored for its ability to detect new attacks. This is achieved by learning normal behavior and raising an alarm when a deviation from the norm is detected. However, this does not provide any intuition of the attack taking place. Identifying the nature of the attack is fundamental to determine the criticality of the situation and take appropriate mitigation

actions. This is indeed vital to achieve autonomic security management and must be realized in real time to avoid detrimental consequences.

CHALLENGES IN USING MACHINE LEARNING

Representative Datasets: ML is inherently data-driven. Hence, the quality of the data used for training and validation is critical. Non-representative datasets can have a severe impact on the accuracy of the models. Gaining access to representative data is not an easy task mainly due to its sensitive and confidential nature. One possible direction is to encourage sharing of data in the research community. Toward fulfilling this need, we have launched a website (<https://sites.google.com/site/cnetmag/>; accessed 14 June 2017) for the research community to share datasets, tools, and platforms.

Speed vs. Accuracy: Achieving high accuracy often comes at the cost of high computation time for training the model. This is particularly a challenge when dealing with online models. A promising direction to explore to overcome this challenge is the use of ensemble learning and hybrid techniques.

Ground Truth: A key challenge in applying ML is the need for ground truth. For supervised learning, ground truth provides the labels for training. For unsupervised learning it allows the accuracy of the model to be evaluated. However, to obtain the ground truth, one must either manually label the data or synthetically generate it. While the former allows the use of real data traces, the manual labeling process can be highly cumbersome and error-prone. On the other hand, the latter can render unrealistic traffic traces. An interesting research direction worth exploring is the application of active learning [16] to facilitate labeling.

ML Techniques for Networks: Another key challenge with the application of ML in networking is the lack of a “Theory of Networks.” This concern was raised by David Meyer during his talk at Internet Engineering Task Force 97 (IETF97) [17] on machine intelligence and networking. Indeed, without a unified theory, each network has to be learned separately. This could truly hinder the speed of adoption of ML in networking. Furthermore, the currently employed ML techniques in networking have been designed with other applications in mind. An open research direction in this realm is to design ML algorithms tailored for networks [5].

Incremental Learning: Due to the high dynamics of networks, the ML models have to be constantly re-trained to ensure their validity over time. Re-training a model from scratch is computationally expensive and time consuming, particularly for online applications. An interesting research direction is to achieve fast incremental learning, where the model is re-trained with only the new data.

Security of Machine Learning: ML is prone to adversarial attacks [18], also known as mimicry attacks, that aim to confuse learning. For instance, when employing ML for intrusion detection, an adversarial attack can trick the model into misclassifying malicious events as benign by poisoning the training set. In [18], the authors performed a proof of concept to showcase how a learning algorithm (outlier detection) can be manipulat-

The network has come a long way since then with the increasing adoption of SDN, NFV, and Cloud Computing.

These technological advances have rendered the infrastructure more agile and compute and storage resources more abundant than ever before.

ed. They assumed complete knowledge of the ML algorithm in use and its decision boundaries. However, without such knowledge, the vulnerability of ML models remain an open research question. That is, it is unclear what it would take an outsider (without inside information) to pull off an adversarial attack in a blackbox setting. An interesting initiative worth mentioning is Cleverhans (<https://github.com/tensorflow/cleverhans>; accessed 14 June 2017), a useful library that allows to craft adversarial examples. It provides training datasets that can be used to build robust ML models capable of distinguishing legitimate datasets from poisoned ones.

CHALLENGES IN AUTONOMIC NETWORK MANAGEMENT

Orchestration of Cognitive Management Functions:

One key challenge in autonomic network management is how to orchestrate C-MAPE functions within a management area or across management areas. Such coordination is fundamental to attain high-level network policies that ensure the correctness of the system and the stability of the C-MAPE loop. Clearly, an orchestration layer is needed that will sit atop the management areas. Such a layer will need to address three key issues:

- Define valid operating regions of functions and ensure that a function remains within its boundary.
- Enable synchronization among functions so that C-MAPE can converge to a steady state.
- Resolve conflicting policies posed by different functions to ensure that the system behaves correctly.

However, to the best of our knowledge such a cognitive management orchestrator does not exist, and thus remains an open research direction.

Technological Barriers: To attain full autonomy, the management functions must be able to easily interact with the managed resources. Without standardized open interfaces, such autonomous interactions cannot be achieved. With the advent of SDN and NFV, many of these complexities have been alleviated. Although SDN enjoys the advantage of a well defined southbound API between the controller and network devices, the northbound API between the controller and the management applications has yet to mature. In addition, while NFV offers the flexibility to instantiate network functions on the fly, there are no standard APIs to configure their states. These challenges and more demand further efforts toward facilitating and standardizing network configuration and control.

CONCLUSION

More than a decade has passed since the vision of autonomic computing was initially proposed. The gap between the vision's demands and the network capabilities have inhibited the former from being effectuated. However, the network has come a long way since then with the increasing adoption of SDN, NFV, and cloud computing. These technological advances have rendered the infrastructure more agile, and compute and storage resources more abundant than ever before. Motivated by this evolution, coupled with the growing need for enhanced management, this

article presents our preliminary effort to realize a cognitive network management framework using ML. We motivate the major role that ML can play in realizing cognitive network management, and highlight existing efforts that have leveraged ML for performing various management tasks. We follow this discussion with an elucidation of how ML extends the MAPE control loop to realize cognitive network management. We present a use case of a cognitive security manager that can be implemented in practice, and conclude with open research directions to realize a holistic cognitive network management framework.

ACKNOWLEDGMENTS

This work was supported in part by the NSERC Discovery Grants Program (Canada), the Quebec FRQNT postdoctoral research fellowship (Canada), the ELAP scholarship (Canada), and the COLCIENCIAS Scholarship Program No. 647-2014 (Colombia).

REFERENCES

- [1] Q. Mahmoud, *Cognitive Networks: Towards Self-Aware Networks*, Wiley-Interscience, 2007.
- [2] S. R. White et al., "An Architectural Approach to Autonomic Computing," *Proc. Int'l. Conf. Autonomic Computing*, 2004, May 2004, pp. 2–9.
- [3] L. Xu et al., "Cognet: A Network Management Architecture Featuring Cognitive Capabilities," *Proc. Euro. Conf. Networks and Commun.*, June 2016, pp. 325–29.
- [4] D. D. Clark et al., "A Knowledge Plane for the Internet," *Proc. Conf. Applications, Technologies, Architectures, and Protocols for Computer Commun.*, 2003, pp. 3–10.
- [5] A. Mestres et al., "Knowledge-Defined Networking," *ACM SIGCOMM Computer Commun. Review*, vol. 47, no. 3, 2017, pp. 2–10.
- [6] A. Clemm, *Network Management Fundamentals*, Cisco Press, 2006.
- [7] P. Langley and H. A. Simon, "Applications of Machine Learning and Rule Induction," *ACM Commun.*, vol. 38, no. 11, Nov. 1995, pp. 54–64.
- [8] E. Brill et al., "Data-Intensive Question Answering," *TREC*, 2001.
- [9] G. Tesaro, "Reinforcement Learning in Autonomic Computing: A Manifesto and Case Studies," *IEEE Internet Computing*, vol. 11, no. 1, Jan. 2007, pp. 22–30.
- [10] J. Stanganeli, "The High Price of IT Downtime," *Computer Networking*, 2016; online, accessed 14 June 2017.
- [11] N. Bui et al., "A Survey of Anticipatory Mobile Networking: Context-Based Classification, Prediction Methodologies, and Optimization Techniques," *IEEE Commun. Surveys & Tutorials*, vol. 19, no. 3, 3rd qtr. 2017, pp. 1790–1821.
- [12] V. Harrison and J. Pagliery, "Nearly 1 Million New Malware Threats Released Every Day," *CNN Tech.*, 2015; online, accessed 14 June 2017.
- [13] A. L. Buczak and E. Guven, "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection," *IEEE Commun. Surveys & Tutorials*, vol. 18, no. 2, 2016, pp. 1153–76.
- [14] T. Zimmerman and S. Kambhampati, "Learning-Assisted Automated Planning: Looking Back, Taking Stock, Going Forward," *AI Mag.*, vol. 24, no. 2, 2003, pp. 73–96.
- [15] H. Yang and S. S. Lam, "Real-Time Verification of Network Properties Using Atomic Predicates," *IEEE/ACM Trans. Networking*, vol. 24, no. 2, Apr. 2016, pp. 887–900.
- [16] I. Zliobaite et al., "Active Learning with Drifting Streaming Data," *IEEE Trans. Neural Networks and Learning Systems*, vol. 25, no. 1, 2014, pp. 27–39.
- [17] D. Meyer, "Machine Intelligence and Networks," *IETF97*, 2016; online, accessed 14 June 2017.
- [18] M. Barreno et al., "Can Machine Learning Be Secure?" *Proc. ACM Symp. Info., Computer and Commun. Security*, 2006, pp. 16–25.

BIOGRAPHIES

SARA AYOUBI received her M.Sc. in 2012 from the Lebanese American University and her Ph.D. in 2016 from the Concordia Institute for Information and Systems Engineering. She is currently a postdoctoral fellow at the Cheriton School of Computer

Science at the University of Waterloo. She is a co-founder of the Montreal Operations Research Student Chapter. Her research interests are in the fields of operations research, networks, and computer systems.

NOURA LIMAM received her M.Sc. and Ph.D. degrees in computer science from the University Pierre & Marie Curie, Paris VI, in 2002 and 2007, respectively. She is currently a research assistant professor of computer science at the University of Waterloo. She is on the Technical Program Committees and Organization Committees of several IEEE conferences. Her contributions are in the area of network and service management. Her current research interests are in network softwarization and cognitive network management.

MOHAMMAD A. SALAHUDDIN is a postdoctoral fellow at the Cheriton School of Computer Science, University of Waterloo. He received his Ph.D. in computer science from Western Michigan University in 2014. His current research interests include the Internet of Things, content delivery networks, network softwarization, cloud computing, and cognitive network management. He serves as a TPC member for international conferences and is a reviewer for various journals and magazines.

NASHID SHAHRIAR is a Ph.D. candidate at the Cheriton School of Computer Science, University of Waterloo. He received his M.Sc. and B.Sc. degrees in computer science and engineering from Bangladesh University of Engineering and Technology in 2011 and 2009, respectively. His research interests include future network architectures, network virtualization, and optical

networks. He is a recipient of a David R. Cheriton Graduate Scholarship at the University of Waterloo.

RAOUF BOUTABA received his M.Sc. and Ph.D. degrees in computer science from the University Pierre & Marie Curie, Paris, in 1990 and 1994, respectively. He is a professor in the Cheriton School of Computer Science and Associate Dean, Research of the Faculty of Mathematics at the University of Waterloo, and holds an INRIA International Chair at INRIA Nancy. His research interests include network and service management, cloud computing, network virtualization, and network softwarization.

FELIPE ESTRADA-SOLANO is a Ph.D. student of telematics engineering at the University of Cauca, Colombia, and an international visiting student at the Cheriton School of Computer Science at the University of Waterloo. He received his Master's degree in telematics engineering (2016) and his Bachelor's degree in electronics and telecommunications engineering (2010) from the University of Cauca. His topics of interest include network and service management, network virtualization, software-defined networking, machine learning, and big data.

OSCAR M. CAICEDO is a full professor at the Universidad of Cauca, Colombia, where he is a member of the Telematics Engineering Group. He received his Ph.D. degree in computer science (2015) from the Federal University of Rio Grande do Sul, Brazil, and his M.Sc. in telematics engineering (2006) and his degree in electronics and telecommunications engineering (2001) from the University of Cauca. His research interests include network and service management, network virtualization, and software-defined networking.

RESEARCH

Open Access



A comprehensive survey on machine learning for networking: evolution, applications and research opportunities

Raouf Boutaba^{1*}, Mohammad A. Salahuddin¹, Noura Limam¹, Sara Ayoubi¹, Nashid Shahriar¹, Felipe Estrada-Solano^{1,2} and Oscar M. Caicedo²

Abstract

Machine Learning (ML) has been enjoying an unprecedented surge in applications that solve problems and enable automation in diverse domains. Primarily, this is due to the explosion in the availability of data, significant improvements in ML techniques, and advancement in computing capabilities. Undoubtedly, ML has been applied to various mundane and complex problems arising in network operation and management. There are various surveys on ML for specific areas in networking or for specific network technologies. This survey is original, since it jointly presents the application of diverse ML techniques in various key areas of networking across different network technologies. In this way, readers will benefit from a comprehensive discussion on the different learning paradigms and ML techniques applied to fundamental problems in networking, including traffic prediction, routing and classification, congestion control, resource and fault management, QoS and QoE management, and network security. Furthermore, this survey delineates the limitations, give insights, research challenges and future opportunities to advance ML in networking. Therefore, this is a timely contribution of the implications of ML for networking, that is pushing the barriers of autonomic network operation and management.

Keywords: Machine learning, Traffic prediction, Traffic classification, Traffic routing, Congestion control, Resource management, Fault management, QoS and QoE management, Network security

1 Introduction

Machine learning (ML) enables a system to scrutinize data and deduce knowledge. It goes beyond simply learning or extracting knowledge, to utilizing and improving knowledge over time and with experience. In essence, the goal of ML is to identify and exploit hidden patterns in “training” data. The patterns learnt are used to analyze unknown data, such that it can be grouped together or mapped to the known groups. This instigates a shift in the traditional programming paradigm, where programs are written to automate tasks. ML *creates* the program (i.e. *model*) that fits the data. Recently, ML is enjoying renewed interest. Early ML techniques were rigid and incapable of tolerating any variations from the training data [134].

Recent advances in ML have made these techniques flexible and resilient in their applicability to various real-world scenarios, ranging from extraordinary to mundane. For instance, ML in health care has greatly improved the areas of medical imaging and computer-aided diagnosis. Ordinarily, we often use technological tools that are founded upon ML. For example, search engines extensively use ML for non-trivial tasks, such as query suggestions, spell correction, web indexing and page ranking. Evidently, as we look forward to automating more aspects of our lives, ranging from home automation to autonomous vehicles, ML techniques will become an increasingly important facet in various systems that aid in decision making, analysis, and automation.

Apart from the advances in ML techniques, various other factors contribute to its revival. Most importantly, the success of ML techniques relies heavily on data [77]. Undoubtedly, there is a colossal amount of data in today's networks, which is bound to grow further with emerging

*Correspondence: rboutaba@uwaterloo.ca

¹ David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada

Full list of author information is available at the end of the article

networks, such as the Internet of Things (IoT) and its billions of connected devices [162]. This encourages the application of ML that not only identifies hidden and unexpected patterns, but can also be applied to learn and understand the processes that generate the data.

Recent advances in computing offer storage and processing capabilities required for training and testing ML models for the voluminous data. For instance, Cloud Computing offers seemingly infinite compute and storage resources, while Graphics Processing Units [342] (GPUs) and Tensor Processing Units [170] (TPUs) provide accelerated training and inference for voluminous data. It is important to note that a trained ML model can be deployed for inference on less capable devices e.g. smartphones. Despite these advances, network operations and management still remains cumbersome, and network faults are prevalent primarily due to human error [291]. Network faults lead to financial liability and defamation in reputation of network providers. Therefore, there is immense interest in building autonomic (i.e. self-configuring, self-healing, self-optimizing and self-protecting) networks [28] that are highly resilient.

Though, there is a dire need for cognitive control in network operation and management [28], it poses a unique set of challenges for ML. First, each network is unique and there is a lack of enforcement of standards to attain uniformity across networks. For instance, the enterprise network from one organization is diverse and disparate from another. Therefore, the patterns proven to work in one network may not be feasible for another network of the same kind. Second, the network is continually evolving and the dynamics inhibit the application of a fixed set of patterns that aid in network operation and management. It is almost impossible to manually keep up with network administration, due to the continuous growth in the number of applications running in the network and the kinds of devices connected to the network.

Key technological advances in networking, such as network programmability via Software-Defined Networking (SDN), promote the applicability of ML in networking. Though, ML has been extensively applied to problems in pattern recognition, speech synthesis, and outlier detection, its successful deployment for network operations and management has been limited. The main obstacles include what data can be collected from and what control actions can be exercised on legacy network devices. The ability to program the network by leveraging SDN alleviates these obstacles. The cognition from ML can be used to aid in the automation of network operation and management tasks. Therefore, it is exciting and non-trivial to apply ML techniques for such diverse and complex problems in networking. This makes ML in networking an interesting research area, and requires an understanding of the ML techniques and the problems in networking.

In this paper, we discuss the advances made in the application of ML in networking. We focus on traffic engineering, performance optimization and network security. In traffic engineering, we discuss traffic prediction, classification and routing that are fundamental in providing differentiated and prioritized services. In performance optimization, we discuss application of ML techniques in the context of congestion control, QoS/QoE correlation, and resource and fault management. Undoubtedly, security is a cornerstone in networking and in this regard, we highlight existing efforts that use ML techniques for network security.

The primary objective of this survey is to provide a comprehensive body of knowledge on ML techniques in support of networking. Furthermore, we complement the discussion with key insights into the techniques employed, their benefits, limitations and their feasibility to real-world networking scenarios. Our contributions are summarized as follows:

- *A comprehensive view of ML techniques in networking.* We review literature published in peer-reviewed venues over the past two decades that have high impact and have been well received by peers. The works selected and discussed in this survey are comprehensive in the advances made for networking. The key criteria used in the selection is a combination of the year of publication, citation count and merit. For example, consider two papers *A* and *B* published in the same year with citation counts x and y , respectively. If x is significantly larger than y , *A* would be selected for discussion. However, upon evaluating *B*, if it is evidenced that it presents original ideas, critical insights or lessons learnt, then it is also selected for discussion due to its merit, despite the lower citation count.
- *A purposeful discussion on the feasibility of the ML techniques for networking.* We explore ML techniques in networking, including their benefits and limitations. It is important to realize that our coverage of networking aspects are not limited to a specific network technology (e.g. cellular network, wireless sensor network (WSN), mobile ad hoc network (MANET), cognitive radio network (CRN)). This gives readers a broad view of the possible solutions to networking problems across network technologies.
- *Identification of key challenges and future research opportunities.* The presented discussion on ML-based techniques in networking uncovers fundamental research challenges that confront networking and inhibit ultimate cognition in network operation and management. A discussion of these opportunities will motivate future work and push the boundaries of networking.

Though there are various surveys on ML in networking [18, 61, 82, 142, 246, 339], this survey is purposefully different. Primarily, this is due to its timeliness, the comprehensiveness of ML techniques covered, and the various aspects of networking discussed, irrespective of the network technology. For instance, Nguyen and Armitage [339], though impactful, is now dated and only addresses traffic classification in networking. Whereas, Fadlullah et al. [142] and Buczak et al. [82], both state-of-the-art surveys, have a specialized treatment of ML to specific problems in networking. On the other hand, Klaine et al. [246], Bkassiny et al. [61] and Alsheikh et al. [18], though comprehensive in their coverage of ML techniques in networking, are specialized to specific network technology i.e. cellular network, CRN and WSN, respectively. Therefore, our survey provides a holistic view of the applicability, challenges and limitations of ML techniques in networking.

We organize the remainder of this paper as follows. In Section 2, we provide a primer on ML, which discusses different categories of ML-based techniques, their essential constituents and their evolution. Sections 3, 4 and 5 discuss the application of the various ML-based techniques for traffic prediction, classification and routing, respectively. We present the ML-based advances in performance management, with respect to congestion control, resource management, fault management, and QoS/QoE management for networking in Sections 6, 7, 8 and 9. In Section 10, we examine the benefits of ML for anomaly and misuse detection for intrusion detection in networking. Finally, we delineate the lessons learned, and future research challenges and opportunities for ML in networking in Section 11. We conclude in Section 12 with a brief overview of our contributions. To facilitate reading, Fig. 1 presents a conceptual map of the survey, and Table 1 provides the list of acronyms and definitions for ML.

2 Machine learning for networking—a primer

In 1959, Arthur Samuel coined the term “Machine Learning”, as “*the field of study that gives computers the ability to learn without being explicitly programmed*” [369]. There are four broad categories of problems that can leverage ML, namely, *clustering*, *classification*, *regression* and *rule extraction* [79]. In clustering problems, the objective is to group similar data together, while increasing the gap between the groups. Whereas, in classification and regression problems, the goal is to map a set of new input data to a set of discrete or continuous valued output, respectively. Rule extraction problems are intrinsically different, where the goal is to identify statistical relationships in data.

ML techniques have been applied to various problem domains. A closely related domain consists of data analysis for large databases, called data mining [16]. Though, ML techniques can be applied to aid in data mining, the

goal of data mining problems is to critically and meticulously analyze data—its features, variables, invariants, temporal granularity, probability distributions and their transformations. However, ML goes beyond data mining to predict future events or sequence of events.

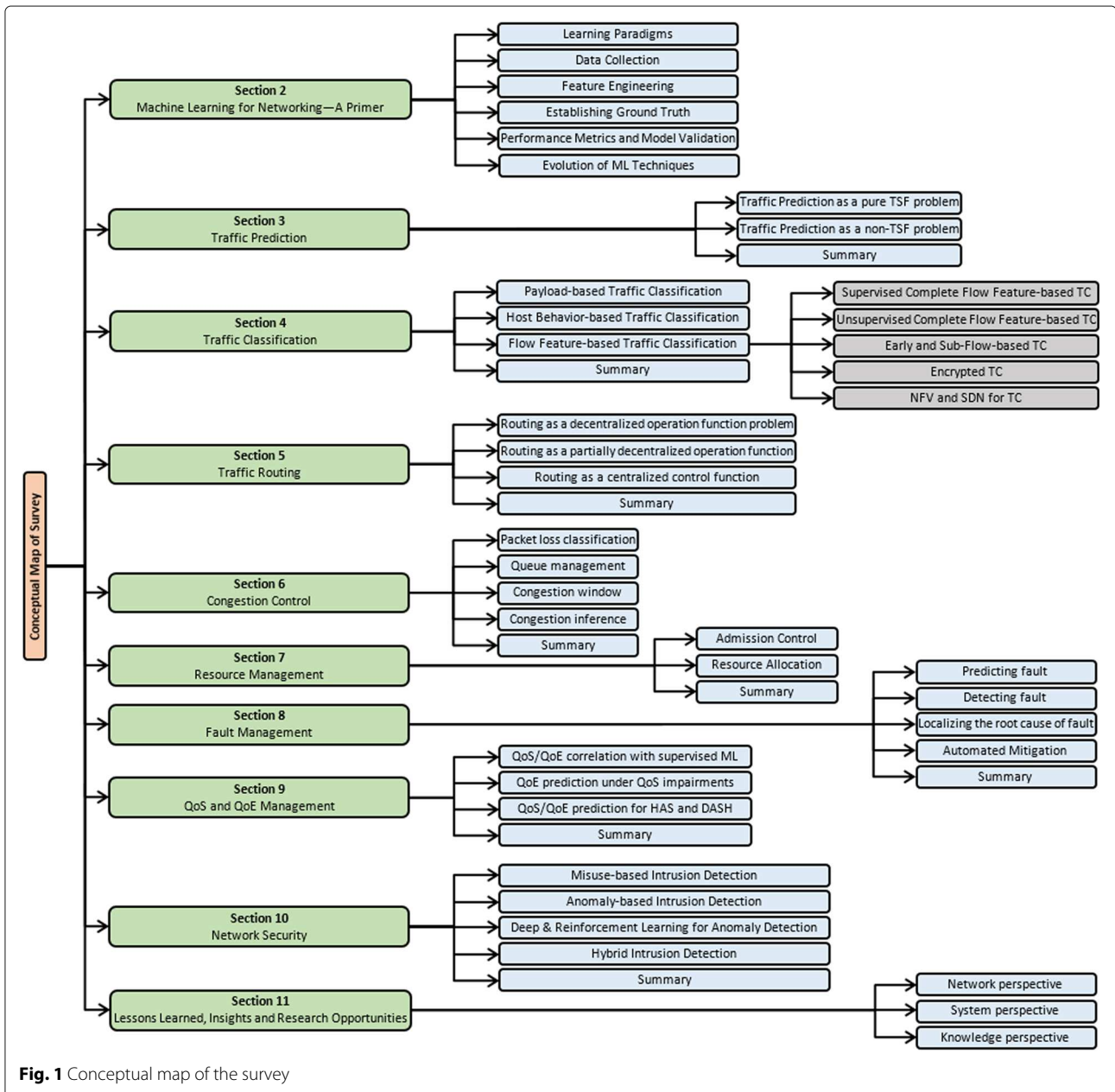
Generally, ML is ideal for inferring solutions to problems that have a large representative dataset. In this way, as illustrated in Fig. 2, ML techniques are designed to identify and exploit hidden patterns in data for (i) describing the outcome as a grouping of data for clustering problems, (ii) predicting the outcome of future events for classification and regression problems, and (iii) evaluating the outcome of a sequence of data points for rule extraction problems. Though, the figure illustrates data and outcome in a two-dimensional plane, the discussion holds for multi-dimensional datasets and outcome functions. For instance, in the case of clustering, the outcome can be a non-linear function in a hyperplane that discriminates between groups of data. Networking problems can be formulated as one of these problems that can leverage ML. For example, a classification problem in networking can be formulated to predict the kind of security attack: Denial-of-Service (DoS), User-to-Root (U2R), Root-to-Local (R2L), or probing, given network conditions. Whereas, a regression problem can be formulated to predict of when a future failure will transpire.

Though there are different categories of problems that enjoy the benefits of ML, there is a generic approach to building ML-based solutions. Figure 3 illustrates the key constituents in designing ML-based solutions for networking. *Data collection* pertains to gathering, generating and, or defining the set of data and the set of classes of interest. *Feature engineering* is used to reduce dimensionality in data and identify discriminating features that reduce computational overhead and increase accuracy. Finally, ML techniques carefully analyze the complex inter- and intra-relationships in data and learn a model for the outcome.

For instance, consider an example of Gold values over time, as illustrated in Fig. 2c. Naively, a linear regression model, shown as a best-fit line through the historical data, can facilitate in predicting future values of Gold. Therefore, once the ML model is built, it can be deployed to deduce outcomes from new data. However, the outcomes are periodically validated, since they can drift over time, known as concept drifting. This can be used as an indicator for incremental learning and re-training of the ML model. In the following subsections, we discuss each of the components in Fig. 3.

2.1 Learning paradigms

There are four learning paradigms in ML, *supervised*, *unsupervised*, *semi-supervised* and *reinforcement learning*. These paradigms influence data collection, feature



engineering, and establishing ground truth. Recall, the objective is to infer an outcome, given some dataset. The dataset used in constructing the ML model is often denoted as training data and labels are associated with training data if the user is aware of the description of the data. The outcome is often perceived as the identification of membership to a class of interest.

There are two schools of thought on the methodology for learning; *generative* and *discriminative* [333]. The basis for the learning methodologies is rooted in the famous Bayes' theorem for conditional probability and

the fundamental rule that relates joint probability to conditional probability. Bayes' theorem is stated as follows. Given two events A and B , the conditional probability is defined as

$$P(A | B) = \frac{P(B | A) \times P(A)}{P(B)},$$

which is also stated as

$$posterior = \frac{likelihood \times prior}{evidence}.$$

Table 1 List of acronyms for machine learning

AdaBoost	Adaptive Boosting
AIWPSO	Adaptive Inertia Weight Particle Swarm Optimization
BN	Bayesian Network
BNN	Bayesian Neural Network
BP	BackPropagation
CALA	Continuous Action-set Learning Automata
CART	Classification and Regression Tree
CMAC	Cerebellar Model Articulation Controller
DBN	Deep belief Network
DBSCAN	Density-based Spatial Clustering of Applications with Noise
DE	Differential Evolution
DL	Deep Learning
DNN	Deep Neural Network
DQN	Deep Q-Network
DT	Decision Tree
EM	Expectation Maximization
EMD	Entropy Minimization Discretization
FALA	Finite Action-set Learning Automata
FCM	Fuzzy C Means
FNN	Feedforward Neural Network
GD	Gradient Descent
HCA	Hierarchical Clustering Analysis
HMM	Hidden Markov Model
HNN	Hopfield Neural Network
ID3	Iterative Dichotomiser 3
k-NN	k-Nearest Neighbor
KDE	Kernel Density Estimation
LDA	Linear Discriminant Analysis
LSTM	Long Short-term Memory
LVQ	Learning Vector Quantization
MART	Multiple Additive Regression Tree
MaxEnt	Maximum Entropy
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multi-layer Perceptron
NB	Naïve Bayes
NBKE	Naïve Bayes with Kernel Estimation
NN	Neural Network
OLS	Ordinary Least Squares
PCA	Principal Component Analysis
PNN	Probabilistic Neural Network
POMDP	Partially Observable Markov Decision Process
RandNN	Random Neural Network
RBF	Radial Basis Function
RBFNN	Radial Basis Function Neural Network

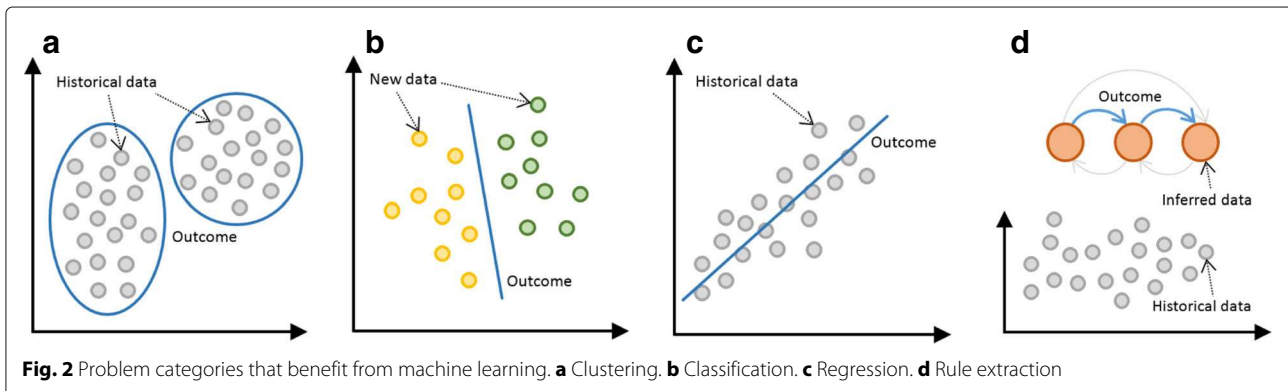
Table 1 List of acronyms for machine learning (*Continued*)

RBM	Restricted Boltzman Machines
REPTree	Reduced Error Pruning Tree
RF	Random Forest
RIPPER	Repeated Incremental Pruning to Produce Error Reduction
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SARSA	State-Action-Reward-State-Action
SGBost	Stochastic Gradient Boosting
SHLLE	Supervised Hessian Locally Linear Embedding
SLP	Single-Layer Perceptron
SOM	Self-Organizing Map
STL	Selt-Taught Learning
SVM	Support Vector Machine
SVR	Support Vector Regression
TD	Temporal Difference
THAID	THeta Automatic Interaction Detection
TLFN	Time-Lagged Feedforward Neural Network
WMA	Weighted Majority Algorithm
XGBoost	eXtreme Gradient Boosting

The joint probability $P(A, B)$ of events A and B is $P(A \cap B) = P(B | A) \times P(A)$, and the conditional probability is the normalized joint probability. The generative methodology aims at modeling the joint probability $P(A, B)$ by predicting the conditional probability. On the other hand, in the discriminative methodology a function is learned for the conditional probability.

Supervised learning uses labeled training datasets to create models. There are various methods for labeling datasets known as ground truth (cf., Section 2.4). This learning technique is employed to “learn” to identify patterns or behaviors in the “known” training datasets. Typically, this approach is used to solve *classification* and *regression* problems that pertain to predicting discrete or continuous valued outcomes, respectively. On the other hand, it is possible to employ semi-supervised ML techniques in the face of partial knowledge. That is, having incomplete labels for training data or missing labels. Unsupervised learning uses unlabeled training datasets to create models that can discriminate between patterns in the data. This approach is most suited for *clustering* problems. For instance, outliers detection and density estimation problems in networking, can pertain to grouping different instances of attacks based on their similarities.

Reinforcement learning (RL) is an agent-based iterative process for modeling problems for decision making.



Generally, learning is based on exemplars from training datasets. However, in RL there is an agent that interacts with the external world, and instead of being taught by exemplars, it learns by exploring the environment and exploiting the knowledge. The actions are rewarded or

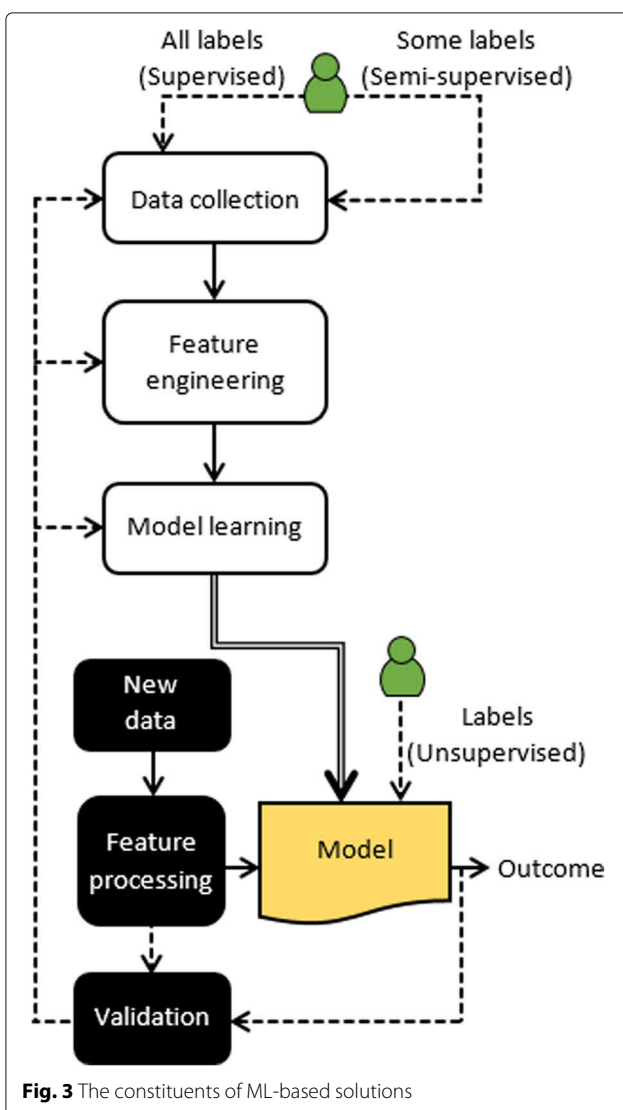
penalized. Therefore, the training data in RL constitutes a set of state-action pairs and rewards (or penalties). The agent uses feedback from the environment to learn the best sequence of actions or “policy” to optimize a cumulative reward. For example, *rule extraction* from the data that is statistically supported and not predicted. Unlike, generative and discriminative approaches that are myopic in nature, RL may sacrifice immediate gains for long-term rewards. Hence, RL is best suited for making cognitive choices, such as decision making, planning and scheduling [441].

It is important to note that there is a strong relationship between the training data, problem and the learning paradigm. For instance, it is possible that due to lack of knowledge about the training data, supervised learning cannot be employed and other learning paradigms have to be employed for model construction.

2.2 Data collection

ML techniques require representative data, possibly without bias, to build an effective ML model for a given networking problem. Data collection is an important step, since representative datasets vary not only from one problem to another but also from one time period to the next. In general, data collection can be achieved in two phases—offline and online [460]. Offline data collection allows to gather a large amount of historical data that can be used for model training and testing. Whereas, real-time network data collected in the online phase can be used as feedback to the model, or as input for re-training the model. Offline data can also be obtained from various repositories, given it is relevant to the networking problem being studied. Examples of these repositories include Waikato Internet Traffic Storage (WITS) [457], UCI Knowledge Discovery in Databases (KDD) Archive [450], Measurement and Analysis on the WIDE Internet (MAWI) Working Group Traffic Archive [474], and Information Marketplace for Policy and Analysis of Cyber-risk & Trust (IMPACT) Archive [202].

An effective way to collect both offline and online data is by using monitoring and measurement tools. These tools



provide greater control in various aspects of data collection, such as data sampling rate, monitoring duration and location (e.g. network core *vs.* network edge). They often use network monitoring protocols, such as Simple Network Management Protocol (SNMP) [208], Cisco Net-Flow [100], and IP Flow Information Export (IPFIX) [209]. However, monitoring can be active or passive [152]. Active monitoring injects measurement traffic, such as probe packets in the network and collects relevant data from this traffic. In contrast, passive monitoring collects data by observing the actual network traffic. Evidently, active monitoring introduces additional overhead due to bandwidth consumption from injected traffic. Whereas, passive monitoring eliminates this overhead, at the expense of additional devices that analyze the network traffic to gather relevant information.

Once data is collected, it is decomposed into training, validation (also called development set or the “dev set”), and test datasets. The training set is leveraged to find the ideal parameters (e.g. weights of connections between neurons in a Neural Network (NN)) of a ML model. Whereas, the validation set is used to choose the suitable architecture (e.g. the number of hidden layers in a NN) of a ML model, or choose a model from a pool of ML models. Note, if a ML model and its architecture are pre-selected, there is no need for a validation set. Finally, test set is used to assess the unbiased performance of the selected model. Note, validation and testing can be performed using one of two methods—holdout or *k*-fold cross-validation. In the holdout method, part of the available dataset is set aside and used as a validation (or testing) set. Whereas, in the *k*-fold cross-validation, the available dataset is randomly divided into *k* equal subsets. Validation (or testing) process is repeated *k* times, with *k* − 1 unique subsets for training and the remaining subset for validating (or testing) the model, and the outcomes are averaged over the rounds.

A common decomposition of the dataset can conform to 60/20/20% among training, validation, and test datasets, or 70/30% in case validation is not required. These rule-of-thumb decompositions are reasonable for datasets that are not very large. However, in the era of big data, where a dataset can have millions of entries, other extreme decompositions, such as 98/1/1% or 99/0.4/0.1%, are also valid. However, it is important to avoid skewness in the training datasets, with respect to the classes of interest. This inhibits the learning and generalization of the outcome, leading to model over- and/or under-fitting. In addition, both validation and testing datasets should be independent of the training dataset and follow the same probability distribution as the training dataset.

Temporal and spatial robustness of ML model can be evaluated by exposing the model to training and validation datasets that are temporally and spatially distant. For

instance, a model that performs well when evaluated with datasets collected a year after being trained or from a different network, exhibits temporal and spatial stability, respectively.

2.3 Feature engineering

The collected raw data may be noisy or incomplete. Before using the data for learning, it must go through a pre-processing phase to clean the data. Another important step prior to learning, or training a model, is feature extraction. These features act as discriminators for learning and inference. In networking, there are many choices of features to choose from. Broadly, they can be categorized based on the level of granularity.

At the finest level of granularity, packet-level features are simplistically extracted or derived from collected packets, e.g. statistics of packet size, including mean, root mean square (RMS) and variance, and time series information, such as *hurst*. The key advantage of packet-level statistics is their insensitivity to packet sampling that is often employed for data collection and interferes with feature characteristics [390]. On the other hand, Flow-level features are derived using simple statistics, such as mean flow duration, mean number of packets per flow, and mean number of bytes per flow [390]. Whereas, connection-level features from the transport layer are exploited to infer connection oriented details. In addition to the flow-level features, transport layer details, such as throughput and advertised window size in TCP connection headers, can be employed. Though these features generate high quality data, they incur computational overhead and are highly susceptible to sampling and routing asymmetries [390].

Feature engineering is a critical aspect in ML that includes feature selection and extraction. It is used to reduce dimensionality in voluminous data and to identify discriminating features that reduce computational overhead and increase accuracy of ML models. Feature selection is the removal of features that are irrelevant or redundant [321]. Irrelevant features increase computational overhead with marginal to no gain in accuracy, while redundant features promote over-fitting. Feature extraction is often a computationally intensive process of deriving extended or new features from existing features, using techniques, such as entropy, Fourier transform and principal component analysis (PCA).

Features selection and extraction can be performed using tools, such as NetMate [21] and WEKA [288]. However, in this case, the extraction and selection techniques are limited by the capability of the tool employed. Therefore, often specialized filter, embedded, and wrapper-based methods are employed for feature selection. Filtering prunes out the training data after carefully analyzing the dataset for identifying the irrelevant and redundant

features. In contrast, wrapper-based techniques take an iterative approach, using a different subset of features in every iteration to identify the optimal subset. Whereas, embedded methods combine the benefits of filter and wrapper-based methods, and perform feature selection during model creation. Examples of feature selection techniques include, colored traffic activity graphs (TAG) [221], breadth-first search (BFS) [496], L1 Regularization [259], backward greedy feature selection (BGFS) [137], consistency-based (CON) and correlation-based feature selection (CFS) [321, 476]. It is crucial to carefully select an ideal set of features that strikes a balance between exploiting correlation and reducing/eliminating over-fitting for higher accuracy and lower computational overhead.

Furthermore, it is important to consider the characteristics of the task we are addressing while performing feature engineering. To better illustrate this, consider the following scenario from network traffic classification. One variant of the problem entails the identification of a streaming application (e.g. Netflix) from network traces. Intuitively, average packet-size and packet inter-arrival times are representative features, as they play a dominant role in traffic classification. Average packet size is fairly constant in nature [492] and packet inter-arrival times are a good discriminator for bulk data transfer (e.g. FTP) and streaming applications [390]. However, average packet size can be skewed by intermediate fragmentation and encryption, and packet inter-arrival times and their distributions are affected by queuing in routers [492]. Furthermore, streaming applications often behave similar to bulk data transfer applications [390]. Therefore, it is imperative to consider the classes of interest i.e. applications, before selecting the features for this network traffic classification problem.

Finally, It is also essential to select features that do not contradict underlying assumptions in the context of the problem. For example, in traffic classification, features that are extracted from multi-modal application classes (e.g. WWW) tend to show a non-Gaussian behavior [321]. These relationships not only become irrelevant and redundant, they contradict widely held assumptions in traffic classification, such as feature distributions being independent and following a Gaussian distribution. Therefore, careful feature extraction and selection is crucial for the performance of ML models [77].

2.4 Establishing ground truth

Establishing the ground truth pertains to giving a formal description (i.e. labels) to the classes of interest. There are various methods for labeling datasets using the features of a class. Primarily, it requires hand-labeling by domain experts, with aid from deep packet inspection (DPI) [462, 496], pattern matching (e.g. application signatures) or unsupervised ML techniques (e.g. Auto-Class using EM) [136].

For instance, in traffic classification, establishing ground truth for application classes in the training dataset can be achieved using application signature pattern matching [140]. Application signatures are built using features, such as average packet size, flow duration, bytes per flow, packets per flow, root mean square packet size and IP traffic packet payload [176, 390]. Average packet size and flow duration have been shown to be good discriminators [390]. Application signatures for encrypted traffic (e.g. SSH, HTTPS) extract the signature from unencrypted handshakes. However, these application signatures must be kept up-to-date and adapted to the application dynamics [176].

Alternatively, it is possible to design and rely on statistical and structural content models for describing the datasets and infer the classes of interest. For instance, these models can be used to classify a protocol based on the label of a single instance of that protocol and correlations can be derived from unlabeled training data [286]. On the other hand, common substring graphs capture structural information about the training data [286]. These models are good at inferring discriminators for binary, textual and structural content [286].

Inadvertently, the ground truth drives the accuracy of ML models. There is also an inherent mutual dependency on the size of the training data of one class of interest on another, impacting model performance [417]. The imbalance in the number of training data across classes, is a violation of the assumptions maintained by many ML techniques, that is, the data is independent and identically distributed. Therefore, typically there is a need to combat class imbalance by applying under-, over-, joint-, or ensemble-sampling techniques [267]. For example, uniform weighted threshold under-sampling creates smaller balanced training sets [222].

2.5 Performance metrics and model validation

Once an ML model has been built and the ground truth has been ascertained, it is crucial to gauge the performance of the ML model that will describe, predict, or evaluate outcomes. However, it is important to realize that there is no way to distinguish a learning algorithm as the “best” and it is not fair to compare error rates across a whole variety of applications [16]. The performance metrics can be used to measure the different aspects of the model, such as reliability, robustness, accuracy, and complexity. In this section, we discuss the validation of the ML models with respect to accuracy (cf., Table 2), which is a critical aspect in the applicability of the model for networking problems. Moreover, the accuracy is often used as a feedback for incremental learning [389], to increase model robustness and resilience in a dynamic environment.

Table 2 Performance metrics for accuracy validation

Metric	Description
Mean Absolute Error (MAE)	Average of the absolute error between the actual and predicted values. Facilitates error interpretability.
Mean Squared Error (MSE)	Average of the squares of the error between the actual and predicted values. Heavily penalizes large errors.
Mean Absolute Prediction Error (MAPE)	Percentage of the error between the actual and predicted values. Not reliable for zero values or low-scale data.
Root MSE (RMSE)	Squared root of MSE. Represents the standard deviation of the error between the actual and predicted values.
Normalized RMSE (NRMSE)	Normalized RMSE. Facilitates comparing different models independently of their working scale.
Cross-entropy	Metric based on the logistic function that measures the error between the actual and predicted values.
Accuracy	Proportion of correct predictions among the total number of predictions. Not reliable for skewed class-wise data.
True Positive Rate (TPR) or recall	Proportion of actual positives that are correctly predicted. Represents the sensitivity or detection rate (DR) of a model.
False Positive Rate (FPR)	Proportion of actual negatives predicted as positives. Represents the significance level of a model.
True Negative Rate (TNR)	Proportion of actual negatives that are correctly predicted. Represents the specificity of a model.
False Negative Rate (FNR)	Proportion of actual positives predicted as negatives. Inversely proportional to the statistical power of a model.
Received Operating Characteristic (ROC)	Curve that plots TPR versus FPR at different parameter settings. Facilitates analyzing the cost-benefit of possibly optimal models.
Area Under the ROC Curve (AUC)	Probability of confidence in a model to accurately predict positive outcomes for actual positive instances.
Precision	Proportion of positive predictions that are correctly predicted.
F-measure	Harmonic mean of precision and recall. Facilitates analyzing the trade-off between these metrics.
Coefficient of Variation (CV)	Intra-cluster similarity to measure the accuracy of unsupervised classification models based on clusters.

Let us consider the accuracy validation of ML models for prediction. Usually, this accuracy validation undergoes an error analysis that computes the difference between the actual and predicted values. Recall, a prediction is an outcome of ML models for classification and regression problems. In classification, the common metrics for error analysis are based on the logistic function, such as binary and categorical cross-entropy—for binary and multi-class classification, respectively. In regression, the conventional error metrics are Mean Absolute Error (MAE) and Mean Squared Error (MSE). Both regression error metrics disregard the direction of under- and over-estimations in the predictions. MAE is simpler and easier to interpret than MSE, though MSE is more useful for heavily penalizing large errors.

The above error metrics are commonly used to compute the cost function of ML-based classification and regression models. Computing the cost function of the training and validation datasets (cf., Section 2.2) allow diagnosing performance problems due to high bias or

high variance. High bias refers to a simple ML model that poorly maps the relations between features and outcomes (under-fitting). High variance implies an ML model that fits the training data but does not generalize well to predict new data (over-fitting). Depending on the diagnosed problem, the ML model can be improved by going back to one of the following design constituents (cf., Fig. 3): (i) data collection, for getting more training data (only for high variance), (ii) feature engineering, for increasing or reducing the set of features, and (iii) model learning, for building a simpler or more complex model, or for adjusting a regularization parameter.

After tuning the ML model for the training and validation datasets, the accuracy metrics for the test dataset are reported as the performance validation of the model. Regression models often use MAE or MSE (i.e. error metrics) to report the performance results. Other error metrics commonly used in the literature to gauge the accuracy of regression models include Mean Absolute Prediction Error (MAPE), Root MSE (RMSE), and Normalized RMSE

(NRMSE). MAPE states the prediction error as a percentage, while RMSE expresses the standard deviation of the error. Whereas, NRMSE allows comparing between models working on different scales, unlike the other error metrics described.

In classification, the conventional metric to report the performance of an ML model is the accuracy. The accuracy metric is defined as the proportion of true predictions T for each class $C_i \forall i = 1...N$ among the total number of predictions, as follows:

$$Accuracy = \frac{\sum_{i=1}^N T_{C_i}}{Total\ Predictions}$$

For example, let us consider a classification model that predicts whether an email should go to the spam, inbox, or promotion folder (i.e. multi-class classification). In this case, the accuracy is the sum of emails correctly predicted as spam, inbox, and promotion, divided by the total number of predicted emails. However, the accuracy metric is not reliable when the data is skewed with respect to the classes. For example, if the actual number of spam and promotion emails is very small compared to inbox emails, a simple classification model that predicts *every* email as inbox will still achieve a high accuracy.

To tackle this limitation, it is recommended to use the metrics derived from a confusion matrix, as illustrated in Fig. 4. Consider that each row in the confusion matrix represents a predicted outcome and each column represents the actual instance. In this manner, True Positive (TP) is the intersection between correctly predicted outcomes for the actual positive instances. Similarly, True Negative (TN) is when the classification model correctly predicts an actual negative instance. Whereas, False Positive (FP) and False Negative (FN) describe incorrect predictions for negative and positive actual instances, respectively. Note, that TP and TN correspond to the true predictions for

the positive and negative classes, respectively. Therefore, the accuracy metric can also be defined in terms of the confusion matrix:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

The confusion matrix in Fig. 4 works for a binary classification model. Therefore, it can be used in multi-class classification by building the confusion matrix for a specific class. This is achieved by transforming the multi-class classification problem into multiple binary classification subproblems, using the *one-vs-rest* strategy. For example, in the email multi-class classification, the confusion matrix for the spam class sets the positive class as spam and the negative class as the rest of the email classes (i.e. inbox and promotion), obtaining a binary classification model for spam and not spam email.

Consequently, the True Positive Rate (TPR) describing the number of correct predictions is inferred from the confusion matrix as:

$$TPR\ (Recall) = \frac{TP}{TP + FN}$$

The converse, False Positive Rate (FPR) is the ratio of incorrect predictions and is defined as:

$$FPR = \frac{FP}{FP + TN}$$

Similarly, True Negative Rate (TNR) and False Negative Rate (FNR) are used to deduce the number of correct and incorrect negative predictions, respectively. The terms *recall*, *sensitivity*, and *detection rate (DR)* are often used to refer to TPR. In contrast, a comparison of the TPR versus FPR is depicted in a Received Operating Characteristics (ROC) graph. In a ROC graph, where TPR is on the y-axis and FPR is on the x-axis, a good classification model will yield a ROC curve that has a highly positive gradient. This implies high TP for a small change in FP. As the gradient gets closer to 1, the prediction of the ML model deteriorates, as the number of correct and incorrect predictions is approximately the same. It should be noted that a classification model with a negative gradient in the ROC curve can be easily improved by flipping the predictions from the model [16] or swapping the labels of the actual instances.

In this way, multiple classification models for the same problem can be compared and can give insight into the different conditions under which one model outperforms another. Though the ROC aids in visual analysis, the area under the ROC curve (AUC), ideally 1, is a measure of the probability of confidence in the model to accurately predict positive outcomes for actual positive instances. Therefore, the precision of the ML model can be formally defined as the frequency of correct predictions for actual positive instances:

		Actual instance	
		Positive (P)	Negative (N)
Predicted outcome	P	True Positive (TP)	False Positive (FP)
	N	False Negative (FN)	True Negative (TN)

Fig. 4 Confusion matrix for binary classification

$$\text{Precision} = \frac{TP}{TP + FP}$$

The trade-off between recall and precision values allows to tune the parameters of the classification models and achieve the desired results. For example, in the binary spam classifier, a higher recall would avoid missing too many spam emails (lower FN), but would incorrectly predict more emails as spam (higher FP). Whereas, a higher precision would reduce the number of incorrect predictions of emails as spam (lower FP), but would miss more spam emails (higher FN). F-measure allows to analyze the trade-off between recall and precision by providing the harmonic average, ideally 1, of these metrics:

$$F\text{-measure} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The Coefficient of Variation (CV) is another accuracy metric, particularly used for reporting the performance of unsupervised models that conduct classification using clusters (or states). CV is a standardized measure of dispersion that represents the intra-cluster (or intra-state) similarity. A lower CV implies a small variability of each outcome in relation to the mean of the corresponding cluster. This represents a higher intra-cluster similarity and a higher classification accuracy.

2.6 Evolution of machine learning techniques

Machine learning is a branch of artificial intelligence whose foundational concepts were acquired over the years from contributions in the areas of computer science, mathematics, philosophy, economics, neuroscience, psychology, control theory, and more [397]. Research efforts during the last 75 years have given rise to a plethora of ML techniques [15, 169, 397, 435]. In this section, we provide a brief history of ML focusing on the techniques that have been particularly applied in the area of computer networks (cf., Fig. 5).

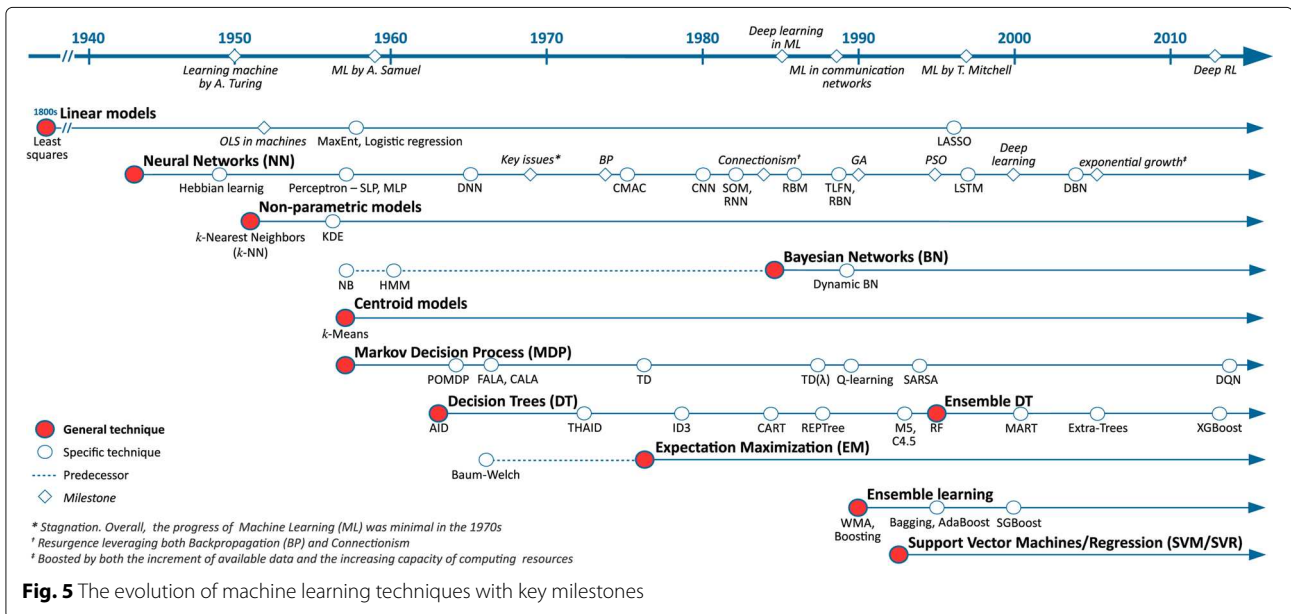
The beginning of ML dates back to 1943, when the first mathematical model of NNs for computers was proposed by McCulloch [302]. This model introduced a basic unit called artificial neuron that has been at the center of NN development to this day. However, this early model required to manually establish the correct weights of the connections between neurons. This limitation was addressed in 1949 by Hebbian learning [184], a simple rule-based algorithm for updating the connection weights of the early NN model. Like the neuron unit, Hebbian learning greatly influenced the progress of NN. These two concepts led to the construction of the first NN computer in 1950, called SNARC (Stochastic Neural Analog Reinforcement Computer) [397]. In the same year, Alan Turing proposed a test –where a computer tries to fool

a human into believing it is also human– to determine if a computer is capable of showing intelligent behavior. He described the challenges underlying his idea of a “*learning machine*” in [449]. These developments encouraged many researchers to work on similar approaches, resulting in two decades of enthusiastic and prolific research in the ML area.

In the 1950s, the simplest linear regression model called Ordinary Least Squares (OLS) –derived from the least squares method [266, 423] developed around the 1800s– was used to calculate linear regressions in electro-mechanical desk calculators [168]. To the best of our knowledge, this is the first evidence of using OLS in computing machines. Following this trend, two linear models for conducting classification were introduced: Maximum Entropy (MaxEnt) [215, 216] and logistic regression [105]. A different research trend centered on pattern recognition exposed two non-parametric models (i.e. not restricted to a bounded set of parameters) capable of performing regression and classification: *k*-Nearest Neighbors (*k*-NN) [147, 420] and Kernel Density Estimation (KDE) [388], also known as Parzen density [349]. The former uses a distance metric to analyze the data, while the latter applies a kernel function (usually, Gaussian) to estimate the probability density function of the data.

The 1950s also witnessed the first applications of the Naïve Bayes (NB) classifier in the fields of pattern recognition [97] and information retrieval [297]. NB, whose foundations date back to the 18th and 19th centuries [43, 261], is a simple probabilistic classifier that applies Bayes’ theorem on features with strong independence assumptions. NB was later generalized using KDE, also known as NB with Kernel Estimation (NBKE), to estimate the conditional probabilities of the features. In the area of clustering, Steinhaus [422] was the first to propose a continuous version of the to be called *k*-Means algorithm [290], to partition a heterogeneous solid with a given internal mass distribution into *k* subsets. The proposed centroid model employs a distance metric to partition the data into clusters where the distance to the centroid is minimized.

In addition, the Markov model [159, 296] (elaborated 50 years earlier) was leveraged to construct a process based on discrete-time state transitions and action rewards, named Markov Decision Process (MDP), which formalizes sequential decision-making problems in a fully observable, controlled environment [46]. MDP has been essential for the development of prevailing RL techniques [435]. Research efforts building on the initial NN model flourished too: the modern concept of perceptron was introduced as the first NN model that could learn the weights from input examples [387]. This model describes two NN classes according to the number of layers: Single-Layer Perceptron (SLP), an NN with one input layer



and one output layer, and Multi-Layer Perceptron (MLP), an NN with one or more hidden layers between the input and the output layers. The perceptron model is also known as Feedforward NN (FNN) since the nodes from each layer exhibit directed connections only to the nodes of the next layer. In the remainder of the paper, MLP-NNs and NNs in general, will be denoted by the tuple $(input_nodes, hidden_layer_nodes^+, output_nodes)$, for instance a $(106, 60, 40, 1)$ MLP-NN has a 160-node input layer, two hidden layers of 60 and 40 nodes respectively, and a single node output layer.

By the end of the 1950s, the term “*Machine Learning*” was coined and defined for the first time by Arthur Samuel (cf., Section 2), who also developed a checkers-playing game that is recognized as the earliest self-learning program [401]. ML research continued to flourish in the 1960s, giving rise to a novel statistical class of the Markov model, named Hidden Markov Model (HMM) [426]. An HMM describes the conditional probabilities between hidden states and visible outputs in a partially observable, autonomous environment. The Baum-Welch algorithm [41] was proposed in the mid-1960s to learn those conditional probabilities. At the same time, MDP continued to instigate various research efforts. The partially observable Markov decision process (POMDP) approach to finding optimal or near-optimal control strategies for partially observable stochastic environments, given a complete model of the environment, was first proposed by Cassandra et al. [25] in 1965, while the algorithm to find the optimal solution was only devised 5 years later [416]. Another development in MDP was the learning automata—officially published in 1973 [448]—, a Reinforcement Learning (RL) technique that continuously updates the

probabilities of taking actions in an observed environment, according to given rewards. Depending on the nature of the action set, the learning automata is classified as Finite Action-set Learning Automata (FALA) or Continuous Action-set Learning Automata (CALA) [330].

In 1963, Morgan and Sonquis published Automatic Interaction Detection (AID) [323], the first regression tree algorithm that seeks sequential partitioning of an observation set into a series of mutually exclusive subsets, whose means reduces the error in predicting the dependent variable. AID marked the beginning of the first generation of Decision Trees (DT). However, the application of DTs to classification problems was only initiated a decade later by Morgan and Messenger’s Theta AID (THAID) [305] algorithm.

In the meantime, the first algorithm for training MLP-NNs with many layers—also known as Deep NN (DNN) in today’s jargon—was published by Ivakhnenko and Lapa in 1965 [210]. This algorithm marked the commencement of the Deep Learning (DL) discipline, though the term only started to be used in the 1980s in the general context of ML, and in the year 2000 in the specific context of NNs [9]. By the end of the 1960s, Minsky and Papertkey’s *Perceptrons* book [315] drew the limitations of perceptrons-based NN through mathematical analysis, marking a historical turn in AI and ML in particular, and significantly reducing the research interest for this area over the next several years [397].

Although ML research was progressing slower than projected in the 1970s [397], the 1970s were marked by milestones that greatly shaped the evolution of ML, and contributed to its success in the following years. These include the Backpropagation (BP) algorithm, the

Cerebellar Model Articulation Controller (CMAC) NN model [11], the Expectation Maximization (EM) algorithm [115], the to-be-referred-to as Temporal Difference (TD) learning [478], and the Iterative Dichotomiser 3 (ID3) algorithm [373].

Werbos's application of BP –originally a control theory algorithm from the 1960s [80, 81, 233]– to train NNs [472] resurrected the research in the area. BP is to date the most popular NN training algorithm, and comes in different variants such as Gradient Descent (GD), Conjugate Gradient (CG), One Step Secant (SS), Levenberg-Marquardt (LM), and Resilient backpropagation (Rp). Though, BP is widely used in training NNs, its efficiency depends on the choice of initial weights. In particular, BP has been shown to have slower speed of convergence and to fall into local optima. Over the years, global optimization methods have been proposed to replace BP, including Genetic Algorithms (GA), Simulated Annealing (SA), and Ant Colony (AC) algorithm [500]. In 1975, Albus proposed CMAC, a new type of NN as an alternative to MLP [11]. Although CMAC was primarily designed as a function modeler for robotic controllers, it has been extensively used in RL and classification problems for its faster learning compared to MLP.

In 1977, in the area of statistical learning, Dempster et al. proposed EM, a generalization of the previous iterative, unsupervised methods, such as the Baum-Welch algorithm, for learning the unknown parameters of statistical HMM models [115]. At the same time, Witten developed an RL approach to solve MDPs, inspired by animal behavior and learning theories [478], that was later referred to as Temporal Difference (TD) in Sutton's work [433, 434]. In this approach the learning process is driven by the changes, or differences, in predictions over successive time steps, such that the prediction at any given time step is updated to bring it closer to the prediction of the same quantity at the next time step.

Towards the end of the 1970s, the second generation of DTs emerged as the Iterative Dichotomiser 3 (ID3) algorithm was released. The algorithm, developed by Quinlan [373], relies on a novel concept for attribute selection based on entropy maximization. ID3 is a precursor to the popular and widely used C4.5 and C5.0 algorithms.

The 1980s witnessed a renewed interest in ML research, and in particular in NNs. In the early 1980s, three new classes of NNs emerged, namely Convolutional Neural Network (CNN) [157], Self-Organizing Map (SOM) [249], and Hopfield network [195]. CNN is a feedforward NN specifically designed to be applied to visual imagery analysis and classification, and thus require minimal image preprocessing. Connectivity between neurons in CNN is inspired by the organization of the animal visual cortex –modeled by Hubel in the 1960s [200, 201]–, where the visual field is divided between neurons, each responding

to stimuli only in its corresponding region. Similarly to CNN, SOM was also designed for a specific application domain; dimensionality reduction [249]. SOMs employ an unsupervised competitive learning approach, unlike traditional NNs that apply error-correction learning (such as BP with gradient descent).

In 1982, the first form of Recurrent Neural Network (RNN) was introduced by Hopfield. Named after the inventor, Hopfield network is an RNN where the weights connecting the neurons are bidirectional. The modern definition of RNN, as a network where connections between neurons exhibit one or more than one cycle, was introduced by Jordan in 1986 [226]. Cycles provide a structure for internal states or memory allowing RNNs to process arbitrary sequences of inputs. As such, RNNs are found particularly useful in Time Series Forecasting (TSF), handwriting recognition and speech recognition.

Several key concepts emerged from the 1980s' *connectionism movement*, one of which is the concept of *distributed representation* [187]. Introduced by Hinton in 1986, this concept supports the idea that a system should be represented by many features and that each feature may have different values. Distributed representation establishes a many-to-many relationship between neurons and (*feature,value*) pairs for improved efficiency, such that a (*feature,value*) input is represented by a pattern of activity across neurons as opposed to being locally represented by a single neuron. The second half of 1980s also witnessed the increase in popularity of the BP algorithm and its successful application in training DNNs [263, 394], as well as the emergence of new classes of NNs, such as Restricted Boltzmann Machines (RBM) [413], Time-Lagged Feedforward Network (TLFN) [260], and Radial Basis Function Neural Network (RBFNN) [260].

Originally named Harmonium by Smolensky, RBM is a variant of Boltzmann machines [2] with the restriction that there are no connections within any of the network layers, whether it is visible or hidden. Therefore, neurons in RBMs form a bipartite graph. This restriction allows for more efficient and simpler learning compared to traditional Boltzmann machines. RBMs are found useful in a variety of application domains such as dimensionality reduction, feature learning, and classification, as they can be trained in both supervised and unsupervised ways. The popularity of RBMs and the extent of their applicability significantly increased after the mid-2000s as Hinton introduced in 2006 a faster learning method for Boltzmann machines called Contrastive Divergence [186] making RBMs even more attractive for deep learning [399]. Interestingly, although the use of the term “deep learning” in the ML community dates back to 1986 [111], it did not apply to NNs at that time.

Towards the end of 1980s, TLFN –an MLP that incorporates the time dimension into the model for conducting TSF [260]–, and RBFNN –an NN with a weighted set of Radial Basis Function (RBF) kernels that can be trained in unsupervised and supervised ways [78]– joined the growing list of NN classes. Indeed any of the above NNs can be employed in a DL architecture, either by implementing a larger number of hidden layers or stacking multiple simple NNs.

In addition to NNs, several other ML techniques thrived during the 1980s. Among these techniques, Bayesian Network (BN) arose as a Directed Acyclic Graph (DAG) representation model for the statistical models in use [352], such as NB and HMM –the latter considered as the simplest dynamic BN [107, 110]–. Two DT learning algorithms, similar to ID3 but developed independently, referred to as Classification And Regression Trees (CART) [76], were proposed to model classification and regression problems. Another DT algorithm, under the name of Reduced Error Pruning Tree (REPTree), was also introduced for classification. REPTree aimed at building faster and simpler tree models using information gain for splitting, along with reduced-error pruning [374].

Towards the end of 1980s, two TD approaches were proposed for reinforcement learning: TD(λ) [433] and Q-learning [471]. TD(λ) adds a discount factor ($0 \leq \lambda \leq 1$) that determines to what extent estimates of previous state-values are eligible for updating based on current errors, in the policy evaluation process. For example, TD(0) only updates the estimate of the value of the state preceding the current state. Q-learning, however, replaces the traditional state-value function of TD by an action-value function (i.e. Q-value) that estimates the utility of taking a specific action in specific states. As of today, Q-learning is the most well-studied and widely-used model-free RL algorithm. By the end of the decade, the application domains of ML started expanding to the operation and management of communication networks [57, 217, 289].

In the 1990s, significant advances were realized in ML research, focusing primarily on NNs and DTs. Bio-inspired optimization algorithms, such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), received increasing attention and were used to train NNs for improved performance over the traditional BP-based learning [234, 319]. Probably one of the most important achievements in NNs was the work on Long Short-Term Memory (LSTM), an RNN capable of learning long-term dependencies for solving DL tasks that involve long input sequences [192]. Today, LSTM is widely used in speech recognition as well as natural language processing. In DT research, Quinlan published the M5 algorithm in 1992 [375] to construct tree-based multivariate linear models analogous to piecewise linear functions. One well-known variant of the M5 algorithm is M5P, which aims at building

trees for regression models. A year later, Quinlan published C4.5 [376], that builds on and extends ID3 to address most of its practical shortcomings, including data overfitting and training with missing values. C4.5 is to date one of the most important and widely used algorithms in ML and data mining.

Several techniques other than NN and DT also prospered in the 1990s. Research on regression analysis propounded the Least Absolute Selection and Shrinkage Operator (LASSO), which performs variable selection and regularization for higher prediction accuracy [445]. Another well-known ML technique introduced in the 1990s was Support Vector Machines (SVM). SVM enables plugging different kernel functions (e.g. linear, polynomial, RBF) to find the optimal solution in higher-dimensional feature spaces. SVM-based classifiers find a hyperplane to discriminate between categories. A single-class SVM is a binary classifier that deduces the hyperplane to differentiate between the data belonging to the class against the rest of the data, that is, *one-vs-rest*. A multi-class approach in SVM can be formulated as a series of single class classifiers, where the data is assigned to the class that maximizes an output function. SVM has been widely used primarily for classification, although a regression variant exists, known as Support Vector Regression (SVR) [70].

In the area of RL, SARSA (State-Action-Reward-State-Action) was introduced as a more realistic, however less practical, Q-learning variation [395]. Unlike Q-learning, SARSA does not update the Q-value of an action based on the maximum action-value of the next state, but instead it uses the Q-value of the action chosen in the next state.

A new emerging concept called *ensemble learning* demonstrated that the predictive performance of a single learning model can be improved when combined with other models [397]. As a result, the poor performance of a single predictor or classifier can be compensated with ensemble learning at the price of (significantly) extra computation. Indeed the results from ensemble learning must be aggregated, and a variety of techniques have been proposed in this matter. The first instances of ensemble learning include Weighted Majority Algorithm (WMA) [279], boosting [403], bootstrap aggregating (or bagging) [75], and Random Forests (RF) [191]. RF focused explicitly on tree models and marked the beginning of a new generation of ensemble DT. In addition, some variants of the original boosting algorithm were also developed, such as Adaptive Boosting (AdaBoost) [153] and Stochastic Gradient Boosting (SGBost) [155].

These advances in ML facilitated the successful deployment of major use cases in the 1990s, particularly, handwriting recognition [419] and data mining [3]. The latter represented a great shift to data-driven ML, and since then it has been applied in many areas (e.g., retail,

finance, manufacturing, medicine, science) for processing huge amounts of data to build models with valuable use [169]. Furthermore, from a conceptual perspective, Tom Mitchell formally defined ML: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ” [317].

The 21st century began with a new wave of increasing interest in SVM and ensemble learning, and in particular ensemble DT. Research efforts in the field generated some of the most widely used implementations of ensemble DT as of today: Multiple Additive Regression Trees (MART) [154], extra-trees [164], and eXtreme Gradient Boosting (XGBoost) [93]. MART and XGBoost are respectively a commercial and open source implementation of Friedman’s Gradient Boosting Decision Tree (GBDT) algorithm; an ensemble DT algorithm based on gradient boosting [154, 155]. Extra-trees stands for *extremely randomized trees*, an ensemble DT algorithm that builds random trees based on k randomly chosen features. However instead of computing an optimal split-point for each one of the k features at each node as in RF, extra-trees selects a split-point randomly for reduced computational complexity.

At the same time, the popularity of DL increased significantly after the term “deep learning” was first introduced in the context of NNs in 2000 [9]. However, the attractiveness of DNN started decreasing shortly after due to the experienced difficulty of training DNNs using BP (e.g. vanishing gradient problem), in addition to the increasing competitiveness of other ML techniques (e.g. SVM) [169]. Hinton’s work on Deep Belief Networks (DBN), published in 2006 [188], gave a new breath and strength to research in DNNs. DBN introduced an efficient training strategy for deep learning models, which was further used successfully in different classes of DNNs [49, 381]. The development in ML –particularly, in DNNs– grew exponentially with advances in storage capacity and large-scale data processing (i.e. Big Data) [169]. This wave of popularity in deep learning has continued to this day, yielding major research advances over the years. One approach that is currently receiving tremendous attention is Deep RL, which incorporates deep learning models into RL for solving complex problems. For example, Deep Q-Networks (DQN) –a combination of DNN and Q-learning– was proposed for mastering video games [318]. Although the term Deep RL was coined recently, this concept was already discussed and applied 25 years ago [275, 440].

It is important to mention that the evolution in ML research has enabled improved learning capabilities which were found useful in several application domains, ranging from games, image and speech recognition, network operation and management, to self-driving cars [120].

3 Traffic prediction

Network traffic prediction plays a key role in network operations and management for today’s increasingly complex and diverse networks. It entails forecasting future traffic and traditionally has been addressed via time series forecasting (TSF). The objective in TSF is to construct a regression model capable of drawing accurate correlation between future traffic volume and previously observed traffic volumes.

Existing TSF models for traffic prediction can be broadly decomposed into statistical analysis models and supervised ML models. Statistical analysis models are typically built upon the generalized autoregressive integrated moving average (ARIMA) model, while majority of learning for traffic prediction is achieved via supervised NNs. Generally, the ARIMA model is a popular approach for TSF, where autoregressive (AR) and moving average (MA) models are applied in tandem to perform auto-regression on the differenced and “stationarized” data. However, with the rapid growth of networks and increasing complexity of network traffic, traditional TSF models are seemingly compromised, giving rise to more advanced ML models. More recently, efforts have been made to reduce overhead and, or improve accuracy in traffic prediction by employing features from flows, other than traffic volume. In the following subsections, we discuss the various traffic prediction techniques that leverage ML and summarize them in Table 3.

3.1 Traffic prediction as a pure TSF problem

To the best of our knowledge, Yu et al. [489] were the first to apply ML in traffic prediction using MLP-NN. Their primary motive was to improve *accuracy* over traditional AR methods. This was supported by rigorous independent mathematical proofs published in the late eighties and the early nineties by Cybenko [106], Hornik [196], and Funahashi [158]. These proofs showed that SLP-NN approximators, which employed sufficient number of neurons of continuous sigmoidal activation type (a constraint introduced by Cybenko and relaxed by Hornik), were universal approximators, capable of approximating any continuous function to any desired accuracy.

In the last decade, different types of NNs (SLP, MLP, RNN, etc.) and other supervised techniques have been employed for TSF of network traffic. Eswaradass et al. [141] propose a MLP-NN-based bandwidth prediction system for Grid environments and compare it to the Network Weather Service (NWS) [480] bandwidth forecasting AR models for traffic monitoring and measurement. The goal of the system is to forecast the available bandwidth on a given path by feeding the NN with the minimum, maximum and average number of bits per second used on that path in the last epoch (ranging from 10 to 30 s). Experiments on the *dotresearch.org* network and the

Table 3 Summary of TSF and non-TSF-based traffic prediction

Ref.	ML Technique	Application (approach)	Dataset (availability)	Features	Output (training)	Evaluation Settings	Results ^{ab}
NBP [141]	Supervised: • MLP-NN (offline)	End-to-end path availability prediction (TSF)	NSF TeraGrid dataset (N/A)	Max, Min, Avg load observed in past 10 s ~ 30 s	Available bandwidth on a end-to-end path in future epoch	Number of features= 3 MLP-NN: (N/A)	MSE = 8%
Cortez et al. [104]	Supervised: • NNE trained with Rp (offline)	Link load and traffic volume prediction in ISP networks (TSF)	SNMP traffic data from 2 ISP nets, traffic on a transatlantic link • aggregated traffic in the ISP backbone (N/A)	Traffic volume observed in past few minutes~several days	Expected traffic volume	Number of features= 6 ~ 9 5 NNs NNE: • all SLPs for dataset1 • 1 hidden layer MLPs with 6 ~ 8 neurons for dataset2	1h lookahead: • MAPE = 1.43% ~ 5.23% 1h ~ 24h lookahead: • MAPE = 6.34% ~ 23.48%
Bermolen et al. [52]	Supervised: • SVR (offline)	Link load prediction in ISP networks (TSF)	Internet traffic collected at the POP of an ISP network (N/A)	Link load observed at τ time scale	Expected link load	Number of features= d samples with $d = 1..30$ Number of support vectors: • varies with d (e.g. ~ 320 for $d = 10$)	RMSE < 2 for $\tau = 1ms$ and $d = 5$ • $\approx$ AR • 10% less than MA
Chabaa et al. [86]	Supervised: MLP-NN with different training algorithms (GD, CG, SS, LM, Rp) (offline)	Network traffic prediction (TSF)	1000 points dataset (N/A)	Past measurements	Expected traffic volume	Number of features (N/A) MLP-NN: • 1 hidden layer	LM: • RMSE= 0.0019 RPE = 0.0230% Rp: • RMSE= 0.0031 RPE= 0.0371% MSE = 0.006 on normalized data 50% less than BP
Zhu et al. [500]	Supervised: MLP-NN with PSO-ABC (offline)	Network traffic prediction (TSF)	2-week hourly traffic measurements (N/A)	N past days hourly traffic volume	Expected next-day hourly traffic volume	Number of features= 5 MLP-NN (5, 11, 1) PSO-ABC: • 30 particles of length=66	RRMSE= 4% ~ 10% for $k = 1 \sim 40$
Li et al. [274]	Supervised: MLP-NN (offline)	Traffic volume prediction on an inter-DC link (Regression)	6-week inter-DC traffic dataset from Baidu • SNMP counters data collected every 30 s • Top-5 applications traffic data collected every 5 min (N/A)	Level-N wavelet transform used to extract time and frequency features from total and elephant traffic volumes time series	$k \times 30$ -s ahead expected traffic volume	Number of wavelets: • $N = 10$ Number of features= $k \times 120$ for $N = 10$ 1 hidden layer MLP-NN	
Chen et al. [94]	Supervised: • KBR • LSTM-RNN (offline)	Inferring future traffic volume based on flow statistics (regression)	Network traffic volume and flow count collected every 5 min over a 24-week period (public)	Flow count	Expected traffic volume	Number of features: • 1 feature (past sample) LSTM-RNN: (N/A)	RNN • MSE > 0.3 on normalized data • 0.05 higher than KBR • twice as much as RNN fed with traffic volume time series
Poupart et al. [365]	Supervised: • GPR • oBMM • MLP-NN (offline)	Early flow-size prediction and elephant flow detection (classification)	3 university and academic networks datasets with over three million flows each (public)	• source IP • destination IP • source port • destination port • protocol • server vs. client • size of 3 first packets	Flow size class: elephant vs. non-elephant	Number of features: • 7 features MLP-NN: (106,60,40,1)	GPR: • TPR > 80% • TNR > 80% oBMM: • TPR and TNR $\approx$ 100% on one dataset • TPR < 50% on other datasets MLP-NN: • TPR > 80% • lowest TNR < 80%

^a Average values. Results vary according to experimental settings^b Accuracy metrics: mean square error (MSE), relative prediction error (RPE), mean absolute prediction error (MAPE), average root mean square error (RMSE)

40 gigabit/s NSF TeraGrid network datasets show that the NN outperforms the NWS bandwidth forecasting models with an error rate of up to 8 and 121.9% for MLP-NN and NWS, respectively. Indeed the proposed NN-based forecasting system shows better learning ability than NWS's. However, no details are provided for the characteristics of the MLP employed in the study, nor the time complexity of the system compared to NWS.

Cortez et al. [104] choose to use a NN ensemble (NNE) of five MLP-NN with one hidden layer each. Resilient backpropagation (Rp) training is used on SNMP traffic data collected from two different ISP networks. The first data represents the traffic on a transatlantic link, while the second represents the aggregated traffic in the ISP backbone. Linear interpolation is used to complete missing SNMP data. The NNE is tested for *real-time* forecasting (online forecasting on a few-minute sample), *short-term* (one-hour to several-hours sample), and *mid-term* forecasting (one-day to several-days sample). The NNE is compared against AR models of traditional Holt-Winters, double Holt-Winters seasonal variant to identify repetitions in patterns at fixed time periods, and ARIMA. The comparison amongst the TSF methods show that in general the NNE produces the lowest MAPE for both datasets. It also shows that in terms of time and computational complexity, NNE outperforms the other methods with an order of magnitude, and is well suited for real-time forecasting.

The applicability of NNs in traffic prediction instigated various other efforts [86, 500] to compare and contrast various training algorithms for network traffic prediction. Chabaa et al. [86] evaluate the performance of various BP training algorithms to adjust the weights in the MLP-NN, when applied to Internet traffic time series. They show superior performance, with respect to RMSE and RPE, of the Levenberg-Marquardt (LM) and the Resilient backpropagation (Rp) algorithms over other BP algorithms.

In contrast to the local optimization in BP, Zhu et al. [500] propose a hybrid training algorithm that is based on global optimization, the PSO-ABC technique [98]. It is an artificial bee colony (ABC) algorithm employing particle swarm optimization (PSO), an evolutionary search algorithm. The training algorithm is implemented with a (5, 11, 1) MLP-NN. Experiments on a 2 weeks hourly traffic measurement dataset show that PSO-ABC has higher prediction accuracy than BP, with an MSE of 0.006 and 0.011, respectively, on normalized data and has stable prediction performance. Furthermore, the hybrid PSO-ABC has a faster training time than ABC or PSO.

On the other hand, SVM has a low computational overhead and is more robust to parameter variations (e.g. time scale, number of samples) in general. However, they are usually applied to classification rather than TSF. SVM and its regression variant, SVR, are scrutinized for their

applicability to traffic prediction in [52]. Bermolen et al. [52] consider the prospect of applying SVR for link load forecasting. The SVR model is tested on heterogeneous Internet traffic collected at the POP of an ISP network. At 1sec timescale, the SVR model shows a slight improvement over an AR model in terms of RMSE. A more significant 10% improvement is achieved over a MA model. Most importantly, SVR is able to achieve 9000 forecast per sec with 10 input samples, and shows the potential for real-time operation.

3.2 Traffic prediction as a non-TSF problem

In contrast to TSF methods, network traffic can be predicted leveraging other methods and features. For instance, Li et al. [274] propose a frequency domain based method for network traffic flows, instead of just traffic volume. The focus is on predicting incoming and outgoing traffic volume on an inter-data center link dominated by elephant flows. Their models incorporate FNN, trained with BP using simple gradient descent and wavelet transform to capture both the time and frequency features of the traffic time series. Elephant flows are added as separate feature dimensions in the prediction. However, collecting all elephant flows at high frequencies is more expensive than byte count for traffic volume. Therefore, elephant flow information is collected at lower frequencies and interpolated to fill in the missing values, overcoming the overhead for elephant flow collection.

The dataset contains the total incoming and outgoing traffic collected in 30 s intervals using SNMP counters on the data center (DC) edge routers and inter-DC link at Baidu over a six-week period. The top 5 applications account for 80% of the total incoming and outgoing traffic data, which is collected every 5 min and interpolated to estimate missing values at the 30 s scale. The time series is decomposed using level 10 wavelet transform, leading to 120 features per timestamp.

Thus, k -step ahead predictions, feed $k \times 120$ features into the NN and show a relative RMSE (RRMSE) ranging from 4 to 10% for the NN-Wavelet transformation model as the prediction horizon varies from 30 s to 20 min. Evidently, wavelet transformation reduces the average prediction errors for different prediction horizons by 5.4 and 2.9% for incoming and outgoing traffic, respectively. In contrast, the linear ARIMA model has prediction error of approximately 8.5 and 6.9% for incoming and outgoing traffic, respectively. The combined NN and wavelet transform model reduces the peak inter-DC link utilization, i.e. the ISP's billed utilization, by about 9%. However, the model does not seem to be fully considering the features related to the elephant flow, which may explain the inexplicable good performance of the 0-interpolation, a simple method that fills zeros for all unknown points.

Chen et al. [94], investigate the possibility of reducing cost of monitoring and collecting traffic volume, by inferring future traffic volume based on flow count only. They propose a HMM to describe the relationship between the flow count, flow volume and their temporal dynamic behavior. The Kernel Bayes Rule (KBR) and RNN with LSTM unit is used to predict future traffic volume based on current flow count. A normalized dataset, with mean=0 and standard deviation=1, consists of network traffic volumes and corresponding flow counts collected every 5 min over a 24-week period [391]. The RNN shows a prediction MSE of 0.3 at best, 0.05 higher than KBR and twice as much as the prediction error of an RNN fed with traffic volume instead of flow count. Therefore, though the motive was to promote flow count based traffic prediction to overcome the cost of monitoring traffic volume, the performance is compromised.

Poupart et al. [365] explore the use of different ML techniques for flow size prediction and elephant flow detection. These techniques include gaussian processes regression (GPR), online bayesian moment matching (oBMM) and a (106, 60, 40, 1) MLP-NN. Seven features are considered for each flow, including source IP, destination IP, source port, destination port, protocol, server versus client (if protocol is TCP), and the size of the first three data packets after the protocol/synchronization packets.

The datasets consist of three public datasets from two university networks [50] and an academic building at Dartmouth College [251] with over three million flows each. Elephant flow detection is based on a flow size threshold that varies from 10KB to 1MB. The experiments show noticeable discrepancies in the performance of the approaches with varying datasets. Although oBMM outperforms all other approaches in one dataset with an average TPR and TNR very close to 100%, it fails miserably in the other datasets with an average TPR below 50% for one dataset. In the latter dataset, oBMM seems to suffer the most from class imbalance. As the detection flow size threshold increases, less flows are tagged as elephant flows, creating class imbalance in the training dataset and leading to lower TPR. However, it is worth noting that oBMM outperforms all other approaches in terms of average TNR in all 3 datasets. On the other hand, NN and GPR, have an average TPR consistently above 80%. Although NN outperforms GPR in terms of robustness to class imbalance by looking at the consistency of its TPR with varying flow size threshold, it has the lowest average TNR of below 80% in all datasets.

The motive for flow size prediction in [365], is to discriminate elephant flows from mice flows in routing to speed up elephant flow completion time. Presumably, mice flows are routed through Equal-cost multi-path routing, while elephant flows are routed through the least congested path. The performance of the routing policy

combined with GPR and oBMM for elephant flow prediction is tested with a varying subset of features. According to the authors, GPR improves the completion time by 6.6% in average for 99% of elephant flows when only the first packet header information is considered. A 14% improvement is observed when the size of the three first packets is used along with the header information. It is also noticed that considering the size of the first three packets alone leads to over 13.5% improvement, regardless of whether GPR or oBMM is used, with a very slight impact on the completion time of mice flows.

3.3 Summary

Supervised NNs (including MLP and RNN) have been successfully applied to traffic prediction, as shown in Table 9. TSF approaches, such as [52, 86, 104, 141, 500], where NNs are used to infer traffic volumes from past measured volumes, show high long-term and short-term prediction accuracy at low complexity with limited number of features and limited number of layers and neurons.

Unfortunately, TSF approaches are restrictive in general. In fact they are only possible if past observations on the prediction variable are made. For instance, in order to predict the traffic for a particular flow f on link l , there must be a counter on link l actively measuring the traffic for that particular flow f , which can be challenging on very high speed links. Because it might not be possible to have the appropriate counter in place, or because it might be technically difficult to conduct measurements at the required speed or granularity, non-TSF approaches can be useful.

Non-TSF approaches were investigated in [94, 274, 365] to infer traffic volumes from flow count and packet header fields. Although higher prediction error rates are experienced, these rates remain relatively low not only for NNs but also for other supervised learning techniques, such as GPR and oBMM. According to [365] a more complex MLP-NN (in terms of number of layers and neurons) might be required to achieve better accuracy in a non-TSF setting, and the performance of supervised learning techniques varies when tested on different datasets. This motivates the need for ensemble learning.

It is envisaged that as the applicability of ML techniques for traffic prediction increases, traffic prediction will improve with respect to computational overhead and accuracy. Furthermore, learning will enable automation of various network operation and management activities, such as network planing, resource provisioning, routing optimization, and SLA/QoS management.

4 Traffic classification

Traffic classification is quintessential for network operators to perform a wide range of network operation and management activities. These include capacity planning, security and intrusion detection, QoS and service

differentiation, performance monitoring, and resource provisioning, to name a few. For example, an operator of an enterprise network may want to prioritize traffic for business critical applications, identify unknown traffic for anomaly detection, or perform workload characterization for designing efficient resource management schemes that satisfy diverse applications performance and resource requirements.

Traffic classification requires the ability to accurately associate network traffic to pre-defined classes of interest. These classes of interest can be classes of applications (e.g. HTTP, FTP, WWW, DNS and P2P), applications (e.g. Skype [310], YouTube [488] and Netflix [331]), or class of service [390]. A class of service, for instance based on QoS, encompasses all applications or classes of applications that have the same QoS requirements. Therefore, it is possible that applications that apparently behave differently, belong to the same class of service [462].

Generally, network traffic classification methodologies can be decomposed into four broad categories that leverage port number, packet payload, host behavior or flow features [31, 244]. The classical approach to traffic classification simply associates Internet Assigned Numbers Authority (IANA) [207] registered port numbers to applications. However, since it is no longer the de facto, nor, does it lend itself to learning due to trivial lookup, it is not in the scope of this survey. Furthermore, relying solely on port numbers has been shown to be ineffective [125, 228, 320], largely due to the use of dynamic port negotiation, tunneling and misuse of port numbers

assigned to well-known applications for obfuscating traffic and avoiding firewalls [54, 109, 176, 286]. Nevertheless, various classifiers leverage port numbers in conjunction with other techniques [31, 56, 244, 417] to improve the performance of the traffic classifiers. In the following subsections, we discuss the various traffic classification techniques that leverage ML and summarize them in Tables 4, 5, 6, 7 and 8.

4.1 Payload-based traffic classification

Payload-based traffic classification is an alternate to port-based traffic classification. However, since it searches through the payload for known application signatures, it incurs higher computation and storage costs. Also, it is cumbersome to manually maintain and adapt the signatures to the ever growing number of applications and their dynamics [138]. Furthermore, with the rise in security and privacy concerns, payload is often encrypted and its access is prohibited due to privacy laws. This makes it non-trivial to infer a signature for an application class using payload [54, 138].

Haffner et al. [176] reduce the computational overhead by using only the first few bytes of unidirectional, unencrypted TCP flows as binary feature vectors. For SSH and HTTPS encrypted traffic, they extract features from the unencrypted handshake that negotiate the encryption parameters of the TCP connection. They use NB, AdaBoost and MaxEnt for traffic classification. AdaBoost outperforms NB and MaxEnt, and yields an overall precision of 99% with an error rate within 0.5%.

Table 4 Summary of Payload* and Host Behavior†-based Traffic Classification

Ref.	ML Technique	Dataset	Features	Classes	Evaluation	
					Settings	Results
Haffner et al. [176]*	Supervised NB, AdaBoost, MaxEnt	Proprietary	Discrete byte encoding for first n bytes of unidirectional flow	FTP, SMTP, POP3, IMAP, HTTPS, HTTP, SSH	$n = 64 - 256$ bytes	Overall error rate < 0.51%, precision > 99%, recall > 94%
Ma et al. [286]*	Unsupervised HCA	Proprietary: U. Cambridge, UCSD	Discrete byte encoding for first n bytes of unidirectional flow	FTP, SMTP, HTTP, HTTPS, DNS, NTP, NetBIOS, SrvLoc	$n = 64$ bytes, distance metric: PD = 250, MP = 150, CSG = 12%	Error rate: PD $\leq 4.15\%$, MP $\leq 9.97\%$, CSG $\leq 6.19\%$
Finamore et al. [146]*	Supervised SVM	Tstat [439]; NAPA-WINE [268]; Proprietary: ISP network	Statistical characterization of first N bytes of each packet a window of size C , divided into G groups of b consecutive bits	eMule, BitTorrent, RTP, RTCP, DNS, P2P-TV (PPLive, Joost, SopCast, TVAnts), Skype, Background	$C = 80, N = 12, G = 24, b = 4$	Average TP = 99.6%, FP < 1%
Schatzmann et al. [404]†	Supervised SVM	Proprietary: ISP network	Service proximity, activity profiles, session duration, periodicity	Mail, Non-Mail	N/A	Average accuracy = 93.2%, precision = 79.2%
Bermolan et al. [53]†	Supervised SVM	Proprietary: campus network, ISP network	Packet count exchanged between peers in duration ΔT	PPLive, TVAnts, SopCast, Joost	$\Delta T = 5$ s, SVM distance metric $R = 0.5$	Worst-case TPR $\approx 95\%$, FPR < 0.1%

N/A: Not available

Table 5 Summary of supervised flow feature-based traffic classification

Ref.	ML Technique	Dataset	Features	Classes	Evaluation	
					Settings	Results
Roughan et al. [390]	Supervised k -NN	Proprietary: univ. networks, streaming service	Packet-level and flow-level features	Telnet, FTP-data, Kazaa, RealMedia Streaming, DNS, HTTPS	$k = 3$, number of QoS classes = 3, 4, 7	Error rate: 5.1% (4), 2.5% (3), 9.4% (7); (#): number of QoS Classes
Moore and Zuev [321]	Supervised NBKE	Proprietary: campus network	Baseline and derivative packet-level features	BULK, WWW, MAIL, SERVICES, DB, P2P, ATTACK, MULTIMEDIA	N/A	Accuracy upto 95%, TPR upto 99%
Jiang et al. [218]	Supervised NBKE	Proprietary: campus network	Baseline and derivative flow-level features	WWW, email, bulk, attack, P2P, multimedia, service, database, interaction, games	N/A	Average accuracy $\approx$ 91%
Park et al. [347]	Supervised REPTree, REPTree-Bagging	NLANR [457]	Packet-level, flow-level and connection-level features	WWW, Telnet, Messenger, FTP, P2P, Multimedia, SMTP, POP, IMAP, DNS, Services	Burst packet threshold = 0.007s	Accuracy $\geq$ 90% (features $\geq$ 7)
Zhang et al. [496]	Supervised BoF-NB	WIDE [474], proprietary: ISP network	Packet-level and flow-level features from unidirectional flows	BT, DNS, FTP, HTTP, IMAP, MSN, POP3, SMTP, SSH, SSL, XMPP	Aggregation rule = <i>sum</i> , BoF size	Accuracy 87-94%, F-measure = 80%
Zhang et al. [497]	Supervised RF, Unsupervised k -Means (BoF-based, RTC)	KEIO [474], WIDE [474], proprietary: ISP network	Packet-level and flow-level features from unidirectional flows	FTP, HTTP, IMAP, POP3, RAZOR, SSH, SSL, UNKNOWN / ZERO-DAY (BT, DNS, SMTP)	N/A	RTC upto 15% and 10% better in flow and byte accuracy, respectively, than second best F-measure = 0.91 (before update), 0.94 (after update)
Auld et al. [26]	Supervised BNN	Proprietary	Packet-level and flow-level features	ATTACK, BULK, DB, MAIL, P2P, SERVICE, WWW	Number of features = 246, hidden layers = 0-1, 0-30 nodes in the hidden layer, output = 10	Accuracy > 99%, 95% with temporally distant training and testing datasets
Sun et al. [431]	Supervised PNN	Proprietary: campus networks	Packet-level and flow-level features	P2P, WEB, OTHERS	Number of features = 22	Accuracy = 87.99%; P2P: TPR = 91.25%, FPR = 1.36%; WEB: TPR = 98.74%, FPR = 27.7%
Este et al. [140]	Supervised SVM	LBNL [262], CAIDA [451], proprietary: campus network	Packet payload size	HTTP, SMTP, POP3, HTTPS, IMAPS, BitTorrent, FTP, MSN, eDonkey, SSL, SMB, Kazaa, Gnutella, NNTP, DNS, LDAP, SSH	Number of support vectors cf., [140]	TP > 90% for most classes
Jing et al. [223]	Supervised FT-SVM	Proprietary [270, 321]	A subset of 12 from 248 features [321]	BULK, INTERACTIVE, WWW, MAIL, SERVICES, P2P, ATTACK, GAME, MULTIMEDIA, OTHER	SVM parameters automatically chosen	Accuracy up to 96%, error ratio $\downarrow$ 2.35 times, avg. computation cost $\downarrow$ 7.65 times
Wang et al. [464]	Supervised multi-class SVM, unbalanced binary SVM	Proprietary: univ. network	Flow-level and connection-level features	BitTorrent, eDonkey, Kazaa, p2p	N/A	Accuracy 75-99%

N/A: Not available

Table 6 Summary of unsupervised flow feature-based traffic classification

Ref.	ML Technique	Dataset	Features	Classes	Evaluation	
					Settings	Results
Liu et al. [283]	Unsupervised <i>k</i> -Means	Proprietary: campus network	Packet-level and flow-level features	WWW, MAIL, P2P, FTP (CONTROL, PASV, DATA), ATTACK, DATABASE, SERVICES, INTERACTIVE, MULTIMEDIA, GAMES	$k = 80$	Average accuracy $\approx 90\%$, minimum recall = 70%
Zander et al. [492]	Unsupervised AutoClass	NLANR [457]	Packet-level and flow-level features	AOL Messenger, Napster, Half-Life, FTP, Telnet, SMTP, DNS, HTTP	Intra-class homogeneity (<i>H</i>)	Mean accuracy = 86.5%
Erman et al. [136]	Unsupervised AutoClass	Univ. Auckland [457]	Packet-level and flow-level features	HTTP, SMTP, DNS, SOCKS, IRC, FTP (control, data), POP3, LIMEWIRE, FTP	N/A	Accuracy = 91.2%
Erman et al. [135]	Unsupervised DBSCAN	Univ. Auckland [457], proprietary: Univ. Calgary	Packet-level and flow-level features	HTTP, P2P, SMTP, IMAP, POP3, MSSQL, OTHER	$eps = 0.03$, $minPts = 3$, number of clusters = 190	Overall accuracy = 75.6%, average precision > 95% (7/9 classes)
Erman et al. [138]	Unsupervised <i>k</i> -Means	Proprietary: univ. network	Packet-level and flow-level features from unidirectional flows	Web, EMAIL, DB, P2P, OTHER, CHAT, FTP, STREAMING	$k = 400$	Server-to-client: Avg. flow accuracy = 95%, Avg. byte accuracy = 79%; Web: precision = 97%, recall = 97%; P2P: precision = 82%, recall = 77%

N/A: Not available

Their ML models are scalable and robust due to the use of partial payloads, and unidirectional flows and diverse usage patterns, respectively. The unidirectional flows circumvent the challenges due to asymmetric routing. In comparison to campus or enterprise networks, residential network data offer an increased diversity, with respect to, social group, age and interest with less spatial and temporal correlation in usage patterns. Unfortunately, performance of AdaBoost traffic classifier deteriorates with noisy data [176] and their approach requires a priori knowledge about the protocols in the application classes.

Ma et al. [286] show that payload-based traffic classification can be performed without any a priori knowledge of the application classes using unsupervised clustering. They train their classifiers based on the label of a single instance of a protocol and a list of partially correlated protocols, where a protocol is modeled as a distribution of sessions. Each session is a pair of unidirectional flow distributions, one from the source to the destination and another from the destination to the source. For tractability, the sessions are assumed to be finite and a protocol model is derived as a distribution on n byte flows, rather than pair of flows.

In product distribution (PD) protocol model, the n byte flow distribution is statistically represented as a product of n independent byte distributions, each describing the distribution of bytes at a particular offset in the flow. Similarly, in the Markov process (MP) protocol model, nodes are labeled with unique byte values and the edges are weighted with a transition probability, such that the sum of all egress transition probabilities from a node is one. A random walk through the directed graph identify discriminator strings that are not tied to a fixed offset. In contrast, the common substring graphs (CSG) capture structural information about the flows using longest common subsequence. A subsequence in a series of common substrings that capture commonalities including the fixed offsets in statistical protocol modeling.

Finally, the authors perform agglomerative (bottom-up) hierarchical clustering analysis (HCA) to group the observed protocols and distinguish between the classes of interest. They employ weighted relative entropy for PD and MP, and approximate graph similarity for CSG, as the distance metric. In evaluation, the PD-based protocol models resulted in the lowest total misclassification error, under 5%. Thus, there is a high invariance at fixed

Table 7 Summary of Early^{*}, Sub-flow[†]-based and Encrypted[‡] flow feature-based traffic classification

Ref.	ML Technique	Dataset	Features	Classes	Evaluation	
					Settings	Results
Bernaille et al. [55] [*]	Unsupervised <i>k</i> -Means	Proprietary: univ. network	Packet size and direction of first <i>P</i> packets in a flow	eDonkey, FTP, HTTP, Kazaa, NNTP, POP3, SMTP, SSH, HTTPS, POP3S	$P = 5, k = 50$	Accuracy > 80%
TIE [108, 121] [*]	Supervised J48 DT, <i>k</i> -NN, Random Tree, RIPPER, MLP, NB	Proprietary: Univ. Napoli campus network	Payload size stats and inter-packet time stats of first <i>N</i> packets, bidirectional flow duration and size, transport protocol	BitTorrent, SMTP, Skype2Skype, POP, HTTP, SOULSEEK, NBNS, QQ, DNS, SSL RTP, EDONKEY	$N = 1...10$	Overall accuracy = 98.4% with BKS (J48, Random Tree, RIPPER, PL) combiner, $N = 10$
Nguyen et al. [337] [†]	Supervised NB, C4.5 DT	Proprietary: home network, univ. network, game server	Inter-packet arrival time statistics, inter-packet length variation statistics, IP packet length statistics of <i>N</i> consecutive packets	Enemy Territory (online game), VoIP, Other	$N = 25$	C4.5 DT: Enemy Territory - recall* = 99.3%, prec.* = 97%; VoIP - recall* = 95.7%, precision* = 99.2% NB: Enemy Territory - recall* = 98.9%, prec.* = 87%, VoIP - recall* = 99.6%, precision* = 95.4% * median
Erman et al. [137] [*]	Semi-supervised <i>k</i> -Means	Proprietary: Univ. Calgary	Number of packets, average packet size, total bytes, total header bytes, total payload bytes (caller to callee and vice versa)	P2P, HTTP, CHAT, EMAIL, FTP, STREAMING, OTHER	$k = 400$, 13 layers, packet milestones (number of packets) in layers are separated exponentially (8, 16, 32, ...)	Flow accuracy > 94%, byte accuracy 70-90%
Li et al. [270] [*]	Supervised C4.5 DT, C4.5 DT with AdaBoost, NBKE	Proprietary	A subset of 12 from 248 features [321] of first <i>N</i> packets	WEB, MAIL, BULK, Attack, P2P, DB, Service, Interactive	$N = 5$	C4.5 DT: Accuracy > 99%; Attack is an exception with moderate-high recall
Jin et al. [222] [*]	Supervised AdaBoost	Proprietary: ISP network, labeled as in [176]	Lowsrcport, highsrcport, duration, mean packet size, mean packet rate, toscout, tcpflags, dstinnet, lowdstport, highdstport, packet, byte, tos, numtosbytes, srcinnet	Business, chat, DNS, FileSharing, FTP, Games, Mail, Multimedia, NetNews, SecurityThreat, VoIP, Web	Number of binary classifiers (<i>k</i>): TCP = 12, UDP = 8	Error rate: TCP = 3%, UDP = 0.4%
Bonfiglio et al. [69] [‡]	Supervised NB, Pearson's χ^2 test	Proprietary: univ. network, ISP network	Message size, average inter-packet gap	Skype	NB decision threshold $B_{min} = -5$, $\chi^2(Thr) = 150$	NB $\wedge \chi^2$: UDP - E2E - FP = 0.01%, FN = 29.98% E2O - FP = 0.0%, FN = 9.82% (univ. dataset); E2E - FP = 0.01%, FN = 24.62% E2O - FP = 0.11%, FN = 2.40% (ISP dataset) TCP - negligible FP
Alshammari et al. [17] [‡]	Supervised AdaBoost, SVM, NB, RIPPER, C4.5 DT	AMP [457], MAWI [474], DARPA99 [278], proprietary from Univ. Dalhousie	Packet size, packet inter-arrival time, number of packets, number of bytes, flow duration, protocol (forward and backward direction)	SSH, Skype	N/A	C4.5 DT: SSH - DR = 95.9%, FPR = 2.8% (Dalhousie), DR = 97.2%, FPR = 0.8% (AMP), DR = 82.9%, FPR = 0.5% (MAWI) Skype - DR = 98.4%, FPR = 7.8% (Dalhousie)

Table 7 Summary of Early^{*}, Sub-flow[†]-based and Encrypted[‡] flow feature-based traffic classification (Continued)

Ref.	ML Technique	Dataset	Features	Classes	Evaluation	
					Settings	Results
Shbair et al. [409] [‡]	Supervised C4.5 DT, RF	Synthetic trace	Statistical features from encrypted payload and [253] (client to server and <i>vice versa</i>)	Service (number of services): Uni-lorraine.fr (15), Google.com (29), akamihd.net (6), Googlevideo.com (1), Twitter.com (3), Youtube.com (1), Facebook.com (4), Yahoo.com (19), Cloudfront.com (1)	Provider (15), (29), (6), (3), (1), (4), (19), (1)	N/A RF (service provider): precision = 92.6%, recall = 92.8%, F-measure = 92.6% RF (service): accuracy in 95-100% for majority of service providers > 100 connections per HTTPS service

N/A: Not available

offsets in binary and textual protocols, such as DNS and HTTP, respectively. Though, the CSG resulted in a higher misclassification error, approximately 7%, it performed best for SSH encrypted traffic. However, it is important to realize that encryption often introduces randomness in the payload. Hence, techniques such as in Ma et al.

[286] that search for keywords at fixed offsets will suffer in performance.

Techniques that rely on capturing the beginning of flows [176, 286] are infeasible for links with high data rates where sampling is often employed. Finamore et al. [146] overcome this limitation by extracting signatures

Table 8 Summary of NFV^{*} and SDN[†]-based traffic classification

Ref.	ML Technique	Dataset	Features	Classes	Evaluation	
					Settings	Results
He et al. [182] [*]	Supervised k -NN, Linear-SVM, Radial-SVM, DT, RF, Extended Tree, AdaBoost, Gradient-AdaBoost, NB, MLP	KDD [42]	Protocol, network service, source bytes, destination bytes, login status, error rate, connection counts, connection percentages (different services among the same host, different hosts among the same service)	Attack types from [450]	Dynamic selection of classifier and features to collect	Accuracy = 95.6%
Amaral et al. [19] [†]	Supervised RF, SGBost, XGBost	Proprietary: enterprise network	Packet size (1 to N packets), packet timestamp (1 to N packets), inter-arrival time (N packets), source/destination MAC, source/destination IP, source/destination port, flow duration, packet count byte count	BitTorrent, Dropbox, Facebook, Web Browsing (HTTP), LinkedIn, Skype, Vimeo, YouTube	$N = 5$	RF: Accuracy 73.6-96.0% SGBost: Accuracy 71.2-93.6% XGBost: Accuracy 73.6-95.2%
Wang et al. [462] [†]	Semi-supervised Laplacian-SVM	Proprietary: univ. network	Entropy of packet length, average packet length (source to destination and <i>vice versa</i>), source port, destination port, packets to respond from source to destination, minimum length of packets from destination to source, packet inactivity degree from source to destination, median of packet length from source to destination for the first N packets	Voice/video conference, streaming, bulk data transfer, interactive	$N = 20$, Laplacian-SVM parameters $\lambda = 0.00001 - 0.0001$, $\sigma = 0.21 - 0.23$	Accuracy > 90%

N/A: Not available

from any point in a flow. In light of the rise in streaming applications, they focus on analyzing packet payload to extract signature of applications over long-lived UDP traffic. In essence, to extract application signatures, the authors employ Pearson's Chi-square (χ^2) test to capture the level of randomness in the first N bytes of each packet divided into G groups of b consecutive bits within a window of C packets. The randomness is evaluated based on the distance between the observed values and a reference uniform distribution. The signatures are then used to train a SVM classifier that distinguishes between the classes of interest with an average TP of 99.6%. However, FP are more sensitive to the window size, and reduce below 5% only for window sizes over 80.

Despite the disadvantages of payload-based classification techniques, the payload-based classifiers achieve high accuracy and are often employed to establish ground truth [55].

4.2 Host behavior-based traffic classification

This technique leverages the inherent behavioral characteristics of hosts on the network to predict the classes of interest. It overcomes the limitations of unregistered or misused port numbers and encrypted packet payload, by moving the observation point to the edge of the network and examining traffic between hosts (e.g. how many hosts are contacted, by which transport protocol, how many different ports are involved). These classifiers rely on the notion that applications generate different communication patterns. For example, a P2P host may contact several different peers using a different port number for each peer. While, a webserver may be contacted by different clients on the same port.

Schatzmann et al. [404] exploit correlations across protocols and time, to identify webmail traffic over HTTPS. They exploit the following features: (i) service proximity—webmail servers tend to reside in the same domain or subnet as SMTP, IMAP, and POP servers, that are identifiable using port numbers [243], (ii) activity profiles—irrespective of the protocol (i.e. IMAP, POP, webmail), users of a mail server share distinct daily and weekly usage habits, (iii) session duration—users of a webmail service spend more time on emails than other web pages and tend to keep the web client open for incoming messages, and (iv) periodicity—webmail traffic exhibit periodic patterns due to application timers (e.g. asynchronous checking for new message from AJAX-based clients). The authors show that these features act as good discriminators for a SVM classifier to differentiate between webmail and non-webmail traffic. Using 5-fold cross validation, the classifier achieves an average accuracy of 93.2% and a precision of 79.2%. The higher FN is attributed to the inability of the classifier to distinguish between VPN and webmail servers.

The data exchanged amongst P2P applications is highly discriminative [53]. For example, a P2P application may establish long flows to download video content from a few peers. Whereas, another P2P application may prefer to use short flows to download fixed size video chunks from many peers in parallel. Therefore, Bermolan et al. [53] leverage this behavior to derive P2P application signatures from the packets and bytes exchanged between peers in small time windows. Formally, the application signature is the probability mass function (PMF) of the number of peers that send a given number of packets and bytes to a peer in the time interval ΔT .

These signatures are used to train a SVM classifier with a Gaussian kernel function and exponential binning strategy, with a rejection threshold (distance metric) of 0.5, to discriminate between applications belonging to the P2P-TV class (i.e. PPLive, TVAnts, SopCast, Joost). The authors evaluate the sensitivity of parameters to optimize their settings in order to guarantee the best performance, that is higher TPR and lower FPR. The classifier results in a worst-case TPR of about 95%, with FPR well below 0.1%. Also, temporal and spatial portability of signatures is validated with marginal degradation in performance.

However, the accuracy of the host behavior-based traffic classification strongly depends on the location of the monitoring system, especially since the observed communication pattern may be affected by routing asymmetries in the network core [229].

4.3 Flow Feature-based traffic classification

In contrast to payload-based and host behavior-based traffic classifiers, flow feature-based classifiers have a different perspective. They step back and consider a communication session, which consists of a pair of complete flows. A complete flow is a unidirectional exchange of consecutive packets on the network between a port at an IP address and another port at a different IP address using a particular application protocol [100]. It is identified with the quintuple $\langle srcIP, destIP, srcPort, destPort, protocol \rangle$. For example, a complete flow in an online game session would consist of all sequential packets sent from source s to destination d (e.g. host to game server). Therefore, a complete flow includes all packets pertaining to session setup, data exchange and session tear-down. A sub-flow is a subset of a complete flow and can be collected over a time window in an on-going session. A feature is an attribute representing unique characteristic of a flow, such as packet length, packet inter-arrival time, flow duration, and number of packets in a flow. Flow feature-based technique uses flow features as discriminators to map flows to classes of interest.

In essence, flow feature-based traffic classification exploits the diversity and distinguishable characteristics of the traffic footprint generated by different applications

[31, 467]. It has the potential to overcome numerous limitations of other techniques, such as unregistered port numbers, encrypted packet payload, routing asymmetries, high storage and computational overhead [55, 138, 176, 347]. However, it remains to be evaluated if flow feature-based classifiers can achieve the accuracy of payload-based classifiers [176, 286]. The corresponding traffic classification problem can be defined as follows: given a set of flows $X = \{x_1, x_2, x_3, \dots, x_{|X|}\}$, such that X consists of either complete or sub-flows, and a set of classes of interest $Y = \{y_1, y_2, y_3, \dots, y_{|Y|}\}$, find the mapping $g(X) \rightarrow Y$. This mapping can be used to classify previously unseen flows. ML is an ideal tool for finding this mapping automatically.

4.3.1 Supervised complete flow feature-based traffic classification

One of the earliest works in network traffic classification using ML is from Roughan et al. [390]. They employ k -NN and Linear Discriminant Analysis (LDA) to map network traffic into different classes of interest based on QoS requirements. Their traffic classification framework uses statistics that are insensitive to application protocol. The authors employ both packet-level and flow-level features. However, they observe that the average packet size and flow duration act as good discriminators, hence used these in their preliminary evaluation.

In their evaluation, k -NN outperforms LDA with the lowest error rate of 5.1 and 9.4% for four and seven class classification, respectively. They notice that often streaming applications behave very similar to bulk data transfer applications. Therefore, either a prioritization rule is necessary to break the tie, or extended/derivative features must be employed to act as good discriminators. In their extended evaluation, the authors employ inter-arrival variability to distinguish between streaming and bulk data transfer applications.

On the other hand, flow features were also leveraged in Moore and Zuev [321] that extend NB with Kernel Estimation (NBKE) to overcome the limitations that make it impractical for network traffic classification. Though, NB classifiers have been commonly used in classification, they have two fundamental assumptions, (i) probability of occurrence of each feature being independent from the occurrence of another feature, and (ii) probability distribution of a feature following a Gaussian distribution. Both of these assumptions are unrealistic for traffic classification and lead to poor accuracy. However, NBKE is a feasible alternate that generalizes NB and overcomes the Gaussian distribution approximation assumption.

Features are extracted from the header of packets in TCP flows using Fast Correlation-Based Filter (FCBF) to address the first assumption. In this way, NBKE with FCBF achieves a classification accuracy of upto 95% and TPR of

upto 99%. It also achieves temporal stability by classifying new flows collected twelve months later with an accuracy of 93% and TPR of 98%. Moreover, it also outperforms NB with respect to training time. However, it incurs increased inference time, especially for classifying unknown flows [347].

Realizing the need for lightweight traffic classification, Jiang et al. [218] further reduce the complexity of KE by employing symmetric uncertainty and correlation measures for feature selection, derived from flows rather than packets. In this manner, NBKE can still be used to classify flows with an accuracy of 91.4%. Though, the classification accuracy is lower than the NBKE in [321], the techniques of varying sampling and application aware feature selection increases its applicability for online classification. Generally, when training time is important, NBKE classifiers are preferred over tree-based approaches, such as C4.5 DT and NB tree [347, 476]. However, DT performs better than NBKE, with respect to execution time and space in memory [347]. Unfortunately, DT suffers from overfitting with noisy data, which deteriorates performance [347].

It is not always possible to collect bidirectional flows due to routing asymmetries [138, 347]. However, it is possible to derive components of the feature vector for an application class given a priori knowledge [347], or estimate the missing statistics [138]. In addition, filtering can be leveraged to reduce the dimensionality of the feature space and the training time. Park et al. [347] employ supervised tree-based classifiers on unidirectional flows and compare them against NBKE using WEKA [288]. They exploit the faster classification time and low memory storage requirements of DT to employ Reduced Error Pruning Tree (REP-Tree) for classification. REPTree finds a sub-optimal tree that minimizes classification error. In addition, the Bagging ensemble is used to classify flows using majority rule to aggregate multiple REPTree predictions. Recall, that P2P bulk data transfer and streaming applications often behave similar to each other [390]. Therefore, the authors in [347] employ a burst feature to better discriminate between such classes. The burst feature is based on packet inter-arrival statistics and a predetermined threshold that dictates whether packets are exhibiting “bursty” behavior. Evidently, bulk data transfer applications exhibit higher burst than streaming applications.

Though, it was presumed that Bagging will outperform REPTree, both classifiers exhibited similar performance. REPTree achieves over 90% accuracy in classification of unidirectional flows and plateaus at seven features. This is in contrast to NBKE, where the classification accuracy deteriorates dramatically with increasing number of features. Though, the accuracy of REPTree is sensitive to packet sampling, the degradation is limited if the same sampling rate is used for both training and testing data.

Evidently, supervised learning yields high classification accuracy, due to a priori information about the characteristics of the classes of interest. However, it is infeasible to expect complete a priori information for applications, since often network operators do not even know all the applications that are running in the network. Therefore, Zhang et al. [496] present a traffic classification scheme suitable for a small set of supervised training dataset. The small labeled training set can be trivially hand-labeled based on the limited knowledge of the network operators. They use *discretized* statistical flow features and Bag-of-Flow (BoF)-based traffic classification. A BoF consists of discretized flows (i.e. correlated flows) that share the destination IP address, destination port and transport layer protocol.

Traditional NB classifiers are simple and effective in assigning the test data a posterior conditional probability of belonging to a class of interest. The BoF-based traffic classification leverages NB to generate predictions for each flow, and aggregate the predictions using rules, such as sum, max, median, majority, since flows are correlated. The F-measure, used to evaluate per class performance, of BoF-NB outperforms NB irrespective of the aggregation rule. For example, the F-measure of BoF-NB with the sum rule is over 15 and 10% higher than NB for DNS and POP3, respectively. Evidently, the accuracy increases and error sensitivity decreases as the size of BoFs increase, due to the growth in BoF intra-diversity [496].

Zhang et al. [497] propose a scheme, called Robust Traffic Classification (RTC). They combine supervised and unsupervised ML techniques for the classification of previously unknown zero-day application traffic, and improving the accuracy of known classes in the presence of zero-day application traffic. Their motivation is that unlabeled data contains zero-day traffic. The proposed RTC framework consists of three modules, namely unknown discovery, BoF-based traffic classification, and system update.

The unknown discovery module uses k -Means to identify zero-day traffic clusters, and RF to extract zero-day samples. The BoF module guarantees the purity of zero-day samples, which classifies correlated flows together, while the system update module complements knowledge by learning new classes in identified zero-day traffic. RTC is novel in its ability to reflect realistic scenarios using correlated flows and identify zero-day applications. Therefore, even with small labeled training datasets, RTC can achieve a higher flow and byte accuracy of up to 15% and 10%, respectively, in comparison to the second best technique.

The accuracy of traffic classification can be increased to over 99%, by using the discriminative MLP-NN classifier to assign membership probabilities to flows. Auld et al. [26] employ hyperbolic tangent for activation function

and a softmax filter to ensure that activation to output generates a positive, normalized distribution over the classes of interest. Their MLP-NN with Bayesian trained weights (BNN) also increases the temporal accuracy of the classifier to 95%. The increase in accuracy is primarily achieved due to the ability to reject predictions. Though the NN with Bayesian weights attain very high performance, it comes at the cost of high compute and storage overhead. Furthermore, some employed features, such as effective bandwidth based on entropy and fourier transform of packet inter-arrival time are computationally intensive, inhibiting its use for online classification. The authors purport that their Bayesian trained weights are robust and efficient, and require only zero or one hidden layer.

On the other hand, Probabilistic Neural Network (PNN) uses Bayes inference theory for classification. Sun et al. [431] leverage PNN that requires no learning processes, no initial weights, no relationship between learning and recalling processes, and the difference between inference and target vectors are not used to modify weights. They employ an activation function that is typical in radial basis function networks and filter out mice flows. Elephant versus mice flows is a prevalent problem in traffic classification, since there is often a lack of representative data for the short-lived mice flows. Often, these flows are discarded for efficient classification. The authors detect mice flows as those flows that contain less than 10 packets and the duration is less than 0.01s. They show that PNN outperforms RBFNN, a feed forward neural network with only two layers and a typical non-linear RBF activation function.

In contrast, Este et al. [140] use single-class SVM followed by multi-class SVM for traffic classification. They consider “semantically valid” bidirectional TCP flows, while ignoring short flows. A grid is maintained to keep track of the percentage of vectors of training sets that are correctly and incorrectly classified as a class. To reduce the overhead in the grid search, they randomly select a small number of flows from the training set to satisfactorily train both single and multi-class classifiers to classify using the first few packets payload sizes. The Multi-class stage is only resorted to, if the single-class stage is unable to clearly identify the application classes. The authors apply their technique to different datasets, with TP of over 90% and low FP for most classes. However, the performance is compromised for encrypted traffic, where ground truth is established using unreliable port-based labeling.

A traffic classification problem with more than two classes, naïvely transforms the SVM into N *one-vs-rest* binary subproblems, resulting in a higher computation cost for a large number of classes. However, Jing et al. [223] propose a SVM based on tournaments for

multi-class traffic classification. In the tournament design of SVM, in each round the candidate classes are randomly organized into pairs, where one class of each pair is selected by a binary SVM classifier, reducing the candidate classes by half. This limits the number of support vectors, which is now based only on the two classes in the pair in contrast to using the entire training dataset. This *all-vs-all* approach to multi-class classification in SVM results in a much lower computational cost for classification. The tournament in [223] results in only one candidate class being left as the classified class.

It is important to note that it is possible that the most appropriate class is eliminated, resulting in higher misclassification. To overcome this, a fuzzy policy is used in the tournament. It allows competing classes to proceed to the next round without being eliminated, if neither class has a clear advantage over the other. However, if two classes are continually paired against each other, the fuzzy rule will break the tie. Unfortunately, this special handling results in higher computational cost. The authors compare their proposed basic tournament and fuzzy tournament (FT-SVM) schemes with existing SVM [270] and [140]. The FT-SVM scheme achieves a high overall accuracy of up to 96%, reduces classification error ratio by up to 2.35 times, and reduces average computation cost by up to 7.65 times.

Traditional SVM and multi-class SVM fall short in efficiency for large datasets. Therefore, Wang et al. [464] use multi-class SVM along with an unbalanced binary SVM to perform statistics-based app-level classification for P2P traffic. Unlike the typical approach of decomposing a multi-class problem into multiple binary classification problems and using one-vs-all approach, the authors employ the *all-together* approach. They leverage NetFlow to collect TCP flows on the edge router of their campus network. In the classification process, unknown flows go through the unbalanced binary model first. Only if identified as P2P, they go through a weighted multi-class model. The unbalanced binary SVM model is built using non-P2P and N types of P2P traffic to help decrease FP (i.e. misclassification of non-P2P traffic as P2P). Whereas, the weighted multi-class model is trained using N types of P2P traffic, giving more weight to data traffic than control/signaling traffic. The proposed scheme correctly classifies atleast 75% and atmost 99% of the entire P2P traffic with generally low misclassification.

4.3.2 Unsupervised complete flow feature-based traffic classification

It is not always possible to apply supervised learning on network traffic, since information about all applications running in the network is rarely available. An alternate is unsupervised learning, where the training data is not labeled. Therefore, the classes of interest are unknown. In

this case, ML techniques are leveraged to learn about similarities and patterns in data and generate clusters that can be used to identify classes of interest. Therefore, clustering essentially identifies patterns in data and groups them together. In hard clustering, an unknown data point must belong to a single cluster, whereas, in soft clustering, a data point can be mapped into multiple different clusters.

Hard clustering often relies on distance and similarity metrics to select a cluster that most closely resembles a data point. Liu et al. [283] employ hard clustering using k -Means with unsupervised training and achieve an accuracy of up to 90%. They use complete TCP flows and log transformation of features to extract and approximate features to a normal distribution, disposing of any noise and abnormality in data. However, it is unrealistic to apply hard clustering for membership, since flow features from applications, such as HTTP and FTP can exhibit high similarity [303]. Therefore, it is impractical to assume a cluster can accurately represent an application [492]. Hence, a fine-grained view of applications is often necessary by employing soft clustering and assigning an unknown data point to a set of clusters. EM is an iterative and probabilistic soft clustering technique, which guesses the expected cluster(s) and refines the guess using statistical characteristics, such as mean and variance.

McGregor et al. [303] employ EM to group flows with a certain probability and use cross-validation to find the optimal number of clusters. To refine the clusters, they generate classification rules from the clusters and use the rules to prune features that have insignificant impact on classification and repeat the clustering process. Though, promising preliminary results indicate stable clusters and an alternative to disaggregate flows based on traffic types, very limited results are provided.

Similarly, AutoClass is an EM approximation approach employed in Zander et al. [492] that automatically creates clusters from unlabeled training datasets. The boundaries of the classes are improved using an intra-class homogeneity metric, defined as the largest fraction of flows belonging to one application in the class. Their objective is to maximize the mean of intra-class homogeneities and achieve a good separation between different application data. On average, the intra-class homogeneity improves as number of features increase. It eventually plateaus at approximately 0.85-0.89, which implies that it is possible to achieve a good separation between classes without using the full set of features. However, this is still computationally expensive.

Erman et al. [136] uncover that this computational overhead can be reduced by training with half of the clusters, since majority of flows were grouped into these clusters. The trade-off between an intra-class homogeneity metric [492] versus iterative clustering remains to be investigated. The suitability of unsupervised learning in traffic

classification reached an milestone when AutoClass was employed to achieve an accuracy of over 91%, higher than the 82.5% accuracy of the supervised NBKE [136]. Thus, it became possible to identify previously unknown applications. Unfortunately, the training time of AutoClass is magnitudes higher than the training time of NBKE [321].

In contrast to EM-based unsupervised clustering, density-based clustering has been found to have a significantly lower training time. Furthermore, density-based spatial clustering of applications with noise (DBSCAN) has the ability to classify noisy data in contrast to k -Means and AutoClass. It differs from conventional clustering, as it exploits the idea of a core object and objects connected to it. An object that does not belong in the neighborhood of a core object and is not a core object itself, is noise. Noisy objects are not assigned to any clusters. The neighborhood around an object is demarcated by epsilon (ϵ). An object is determined to be the core of the cluster when the number of objects in its neighborhood exceeds minimum number of points ($minPts$). In this manner, data points are evaluated to be core points, neighbors of core, or noise, and assigned to clusters or discarded accordingly.

Erman et al. [135] leverage DBSCAN with transport layer statistics to identify application types. Flows are collected from TCP-based applications, identifying flow start and end based on the TCP three way handshake and FIN/RST packets, respectively. They employ logarithms of features due to their heavy-tail distribution and deduce similarity based on Euclidean distance. The authors use WEKA [288] and Cluster 3.0 [194] for evaluating DBSCAN, AutoClass and k -Means clustering and found that AutoClass outperforms DBSCAN. However, training time of DBSCAN is magnitudes lower than AutoClass, 3 min vs. 4.5 h. Furthermore, its non-spherical clusters are more precise than the spherical clusters of k -Means. Uniquely, DBSCAN has the ability to classify data into the smallest number of clusters, while the accuracy of k -Means is dependent on the cluster size.

Erman et al. [138] extend their previous work [136] to employ unidirectional TCP flows to classify network traffic using k -Means. The motivation for unidirectional flows is justified, since it may not always be possible to collect bidirectional flows due to routing asymmetries [138, 347]. Therefore, the authors analyze the effectiveness of flows in one direction. To this end, they divide their dataset into three sets, consisting of server-to-client flows, client-to-server flows, and random flows that have a combination of both. The beginning and ending of the TCP flows are identified using SYN/SYNACK packets and FIN/RST packets, respectively. Whereas, a cluster is labeled to the traffic class with the majority of flows. As the number of clusters increase, it is possible to identify applications with a finer granularity.

It is observed that server-to-client flows consistently exhibit the highest average classification accuracy of 95 and 79% for flows and bytes, respectively. However, the random flows attain an average accuracy of 91 and 67% for flows and bytes, respectively. Whereas, the client-to-server flows show the average classification accuracy of 94 and 57% for flows and bytes, respectively. Also, the ML model using the server-to-client dataset has precision and recall values of 97% for Web traffic, while the P2P traffic had a precision of 82% and a recall of 77%. It is apparent that features from traffic in the server-to-client direction act as a good discriminators to classify unidirectional TCP flows. Furthermore, many network application's payload size is much higher in the server-to-client direction.

4.3.3 Early and sub-flow-based traffic classification

Relying on the completion of a flow for traffic classification not only surmounts to extensive classifier training time and memory overhead, but also delays time-sensitive classification decisions. Therefore, Bernaille et al. [55] leverage the size and direction of the first few P packets from an application's negotiation phase (i.e. during TCP connection setup) for early traffic classification. They inspect packet payload to establish flow labels and employ unsupervised k -Means clustering to model the application classes. The authors empirically deduce the optimal P and number of clusters (k) that strikes a balance between behavior separation and complexity. They represent a flow in a P -dimensional space, where the p^{th} coordinate is the size of the p^{th} packet in the flow and compute a behavioral similarity between flows using Euclidean distance between their spatial representations.

In the online classification phase, the distance between the spatial representation of a new flow and the centroid of all the clusters is computed. The flow is classified to the cluster with the minimum distance, hence to the dominant application class in the cluster. This approach achieves a classification accuracy exceeding 80% with $P = 5$ and $k = 50$ for most application classes. However, if different application classes exhibit similar behavior during the application's negotiation phase, their flows can be easily misclassified. For example, POP3 is always misclassified as NNTP and SMTP, the dominant application classes in corresponding clusters. However, this issue can be resolved by leveraging port number as a feature during cluster composition [56], which increases the classification accuracy of POP3 to over 90%.

Key advantages of the traffic classification approach in [55] is the ability to classify the same set of application classes from another network, since network packet sizes are similar across different traces. Furthermore, as their approach does not depend on packet payload, it is suitable for classifying encrypted traffic. Though, the packet

size may increase due to encryption method used, it can be adjusted for based on a simple heuristic [56]. However, a fundamental requirement is to extract the first few packets during the negotiation phase of a TCP connection. Though simplistic, it is not always possible, especially in networks that use packet sampling. Furthermore, it is impractical for networks that have high load or miss packet statistics.

The classification accuracy of stand-alone classifiers that perform early classification, can be significantly improved by combining the outputs of multiple classifiers using combination algorithms [108, 121]. Donato et al. [121] propose Traffic Identification Engine (TIE), a platform that allows the development and evaluation of ML (and non-ML) classification techniques as plugins. Furthermore, TIE capitalizes on complementarity between classifiers to achieve higher accuracy in online classification. This is realized by using classifier output fusion algorithms, called combiners, including NB, majority voting (MV), weighted majority voting (WMV), Dempster-Shafer (D-S) [484], Behavior-Knowledge Space (BKS) method [199], and Wernecke (WER) method [473]. Note, BKS-based combiners overcome the independent classifier assumption of the other combiners [108]. However, due to the reliance of classifiers on the first few packets, it inherits the limitations of [56].

The authors [108, 121] evaluate the accuracy of the stand-alone classifiers and the combiners. They extract features for ML from flows in proprietary dataset, which is split into 20% classifier training set, 40% classifier and combiner validation set, and 40% classifier and combiner test set. The authors label the dataset using a ground truth classifier e.g. payload-based (PL) classifier. In stand-alone mode, the J48 classifier achieves the highest overall accuracy of 97.2%. Combining the output of J48 with other classifiers (i.e. Random Tree, RIPPER, PL) using BKS method, increases the overall accuracy to 98.4%, when considering first 10 packets per bidirectional flow. Most notably, an average gain in accuracy of 42% is achieved when extracting features from only the first packet, which is significant for online classification. However, higher accuracies are achieved when PL classifier is considered by the combiners in the pool of classifiers, thus increasing computational overhead.

The objective of Nguyen et al. [337] is to design a traffic classifier that can perform well, irrespective of missing statistics, using a small number of most recent packets, called sub-flows [336]. These sub-flows are derived by using a small sliding window over each flow, of N consecutive packets and a step fraction S , to start at packet 0 and slide across the training dataset in steps of N/S . Parameters N and S are critically chosen based on the memory limitations and the trade-off between classification time and accuracy.

To ensure high accuracy of the classifier it is imperative to identify and select sub-flows that best capture the distinctive statistical variations of the complete flows. To this end, the authors manually identify sub-flow starting positions based on protocol knowledge. They leverage Assistance of Clustering Technique (ACT) [338] to automate the selection of sub-flows using unsupervised EM clustering to establish ground truth. To account for directionality, Synthetic Sub-flow Pairs (SSP) are created for every sub-flow recorded with forward and backward features swapped, both labeled as the same application class [335].

Finally, the authors in [337] use NetMate [21] to compute feature values and employ supervised NB and C4.5 DT for traffic classification using WEKA [288]. Both classifiers perform well when evaluated with missing flow start, missing directionality, or 5% independent random packet loss. However, the C4.5 DT classifier performs better with 99.3% median recall and 97% median precision, and achieved 95.7% median recall and 99.2% median precision, for sub-flow size $N = 25$ packets for Enemy Territory and VoIP traffic, respectively. The authors also evaluate a real-world implementation of their approach, called DIFFUSE, for online traffic classification. DIFFUSE achieves a high accuracy of 98–99% for Enemy Territory (online game) and VoIP traffic replayed across the network, while monitoring one or more 1 Gb/s links. Despite the high accuracy, this technique lacks in flexibility, since the classifier can only recognize the application classes that were known a priori.

Erman et al. [137] propose a semi-supervised TCP traffic classification technique that partially overcomes a limitation of [55] and [337]—a priori knowledge of application class or protocol. They have the following objectives: (i) use a small number of labeled flows mixed with a large number of unlabeled flows, (ii) accommodate both known and unknown applications and allow iterative development of classifiers, and (iii) integrate with flow statistics collection and management solutions, such as Bro [350] and NetFlow [100], respectively.

To accomplish this, the authors employ backward greedy feature selection (BGFS) [173] and k -Means clustering to group similar training flows together, while using Euclidean distance as the similarity metric. Here, the objective is to use the patterns hidden in flows to assign them together in clusters, without pre-determined labels. Note, a small set of pre-determined labels are assigned to clusters using maximum likelihood, mapping clusters to application classes. While, other clusters remain unlabeled, accommodating for new or unknown applications. Thus, unknown flows are assigned to an unlabeled cluster. This gives network operators the flexibility to add new unlabeled flows and improve classifier performance by allowing identification of application classes that were

previously unknown. The authors establish ground truth using payload-based signature matching with hand classification for validation.

In offline classification, the authors in [137] achieve over 94% flow accuracy with just two randomly labeled flows per cluster, amongst a mix of 64,000 unlabeled flows and $k = 400$. For real-time classification, the authors leverage a layered classification approach, where each layer represents a packet milestone, that is, the number of packets a flow has sent or received within a pre-defined sliding window. Each layer uses an independent model to classify ongoing flows based on statistics available at the given milestone.

Though, the model is trained with flows that have reached each specific packet milestone, previously assigned labels are disregarded upon reclassification. A significant increase in the average distance of new flows to their nearest cluster mean is indicative of the need for retraining, which could be achieved by incremental learning. This approach not only has a small memory footprint, it allows to update the model and potentially improve classification performance [137]. The authors integrate their layered online classification in Bro and achieve byte accuracies in the 70–90% range. Furthermore, the classifier remains fairly robust over time for different traces.

Similar to [137, 337], Li et al. [270] classify TCP traffic into application classes, including unknown application classes using a few packets in a flow. Their approach uniquely trains the ML model for the application class “Attack”, that enables early detection and classification of anomalous traffic. They employ C4.5 DT to achieve high accuracy for online classification and reduce complexity in the number of features, by using correlation-based filtering. They perform their evaluations on WEKA [288], and find C4.5 DT to outperform C4.5 DT with AdaBoost and NBKE.

The classification accuracy of C4.5 DT with 0.5% randomly selected training flows, exceed 99% for most classes except Attack, which exhibits moderate-high recall. This is because Attack is a complex application class that shows no temporal stability and its characteristics dynamically change over time. However, it may be possible to overcome this by iterative retraining of the classifier, either by using approach similar to [137] or introducing rules (e.g. based on port numbers or flow metrics) in the DT to increase temporal stability of the classifier. Furthermore, the use of port numbers in conjunction with other features results in a slightly higher accuracy. However, this leaves the classifier vulnerable to issues in port-based classification.

In contrast to the semi-supervised and unsupervised techniques for TCP and UDP traffic classification, Jin et al. [222] employ a supervised approach. They classify network traffic using complete flows, while achieving high

accuracy, temporal and spatial stability, and scalability. For accuracy and scalability, their system offers two levels of modularity, partitioning flows and classifying each partition. In the first level, domain knowledge is exploited to partition a flow into m non-overlapping partitions based on flow features, such as protocol or flow size. Second, each partition can be classified in parallel, leveraging a series of k -binary classifiers. Each binary classifier, i , assigns a likelihood score that is reflective of the probability of the flow belonging to the i^{th} traffic class. Eventually, the flow is assigned to the class with the highest score.

They design and leverage weighted threshold sampling and logistic calibration to overcome the imbalance of training and testing data across classes. Though, non-uniform weighted threshold sampling creates smaller balanced training sets, it can distort the distribution of the data. This may violate the independent and identically distributed assumption held by most ML algorithms, invalidating the results of the binary classifiers. Therefore, logistic calibrators are trained for each binary classifier and used at runtime to adjust the prediction of the binary classifiers.

The authors in [222] evaluate their system with respect to spatial and temporal stability, classification accuracy, and training and runtime scalability. With training and testing data collected two months apart from two different locations, result in low error rates of 3 and 0.4% for TCP and UDP traffic, respectively. However, with a larger time difference between training and testing data collection, the error rates increase to 5.5 and 1.2% for TCP and UDP traffic, respectively. By employing collective traffic statistics [221] via colored traffic activity graphs (TAGs) improves the accuracy for all traffic classes, reducing the overall error rate by 15%.

This diminishes the need for frequent retraining of the classifiers. Their system also provides flexible training configuration. That is, given a training time budget it can find the suitable amount of training data and iterations of the ML algorithm. It took the system about two hours to train the classifiers resulting in the reported error rates. Furthermore, their system on a multi-core machine using multi-threads, was able to handle 6.5 million new flows arriving per minute.

4.3.4 Encrypted traffic classification

Various applications employ encryption, obfuscation and compression techniques, that make it difficult to detect the corresponding traffic. Bonfiglio et al. [69] perform controlled experiments to reverse engineer the structure of Skype messages between two Skype clients (E2E) and between a Skype client and a traditional PSTN phone (E2O). The proposed framework uses three technique to identify Skype traffic, with a focus on voice calls, regardless of the transport layer protocol, TCP or UDP. The

first technique uses a classifier based on the Pearson's χ^2 test that leverages the randomness in message content bits, introduced by the Skype encryption process, as a signature to identify Skype traffic. Whereas, the second technique is based on NB classifier that relies on the stochastic characteristics of traffic, such as message size and average inter-packet gap, to classify Skype traffic over IP. The third technique uses DPI to create a baseline payload-based classifier.

In the evaluation, the NB classifier is effective in identifying all voice traffic, while the χ^2 classifier accurately identifies all Skype traffic over UDP and all encrypted or compressed traffic over TCP. Jointly, NB and χ^2 classifier outperform the classifiers in isolation by detecting Skype voice traffic over UDP and TCP with nearly zero FP. However, higher FNs are noticeable in comparison to the isolated classifiers, as the combination disregards video and data transfers, and correctly identify only those Skype flows that actually carry voice traffic.

The identification of Skype traffic at the flow level is also addressed in Alshammari et al. [17] by employing supervised AdaBoost, Repeated Incremental Pruning to Produce Error Reduction (RIPPER), SVM, NB and C4.5 DT classifiers. Additionally, these classifiers are used to identify Secure Shell (SSH) encrypted traffic. The authors use flow-based statistical features extracted using Net-Mate [21] and leverage WEKA [288] to train the classifiers using a sampled dataset for SSH, non-SSH, and Skype, non-Skype traffic. The trained models are applied to complete datasets to label flows as SSH, non-SSH, Skype and non-Skype.

In the evaluation, C4.5 DT outperform the other classifiers for the majority of datasets. For SSH traffic, it achieves 95.9% DR and 2.8% FPR on the Dalhousie dataset, 97.2% DR and 0.8% FPR on the AMP dataset, and 82.9% DR and 0.5% FPR on the MAWI dataset. Furthermore, when trained and tested across datasets (i.e. across networks), it achieves 83.7% DR and 1.5% FPR. Hence, it generalizes well from one network to another. The C4.5 DT classifier also performed well for Skype traffic with 98.4% DR and 7.8% FPR in the Dalhousie dataset. However, secure communication in SSH and HTTPS sessions can contain a variety of applications, identification of which may be needed for granularity. Unfortunately, Alshammari et al. [17] do not detect the precise applications within a secure session.

This problem is addressed by Shbair et al. [409] by adopting a hierarchical classification to identify the service provider (e.g. google.com, dropbox.com), followed by the type of service (e.g. maps.google.com, drive.google.com) that are encapsulated in TLS-based HTTPS sessions. They start with the reconstruction of TLS connections from the HTTPS traces and label them using the Server Name Identification (SNI) field, creating

the service provider-service hierarchy. These labeled connections are used to build: (i) a classifier to differentiate between service providers, and (ii) a classifier for each service provider to differentiate between their corresponding services. This hierarchical approach reduces the effort required to retrain the classifiers in the event of an addition of a new service. They use statistical features extracted over encrypted payload with CFS and employ C4.5 DT and RF classifiers.

In the evaluation, RF performs better in comparison to C4.5 DT with a precision of 92.6%, recall of 92.8%, and F-measure of 92.6%, to classify service providers with selected features. Furthermore, the accuracy of service classification is between 95-100% for majority of the providers. Thus, asserting the benefit of a hierarchical approach to traffic classification. Also, overall accuracy of the system across both levels is 93.10% with a degradation of less than 20% over a period of 23 weeks without retraining.

4.3.5 NFV and SDN for traffic classification

Recent advances in network paradigms, such as Network Functions Virtualization (NFV) and SDN enable flexible and adaptive techniques for traffic classification. The efforts discussed in this subsection present contrasting approaches for traffic classification using ML in software-defined and virtualized networks.

It is well-known that the performance of classifiers vary significantly based on the type of flow features used. Furthermore, flows inherently exhibit specific characteristics of network applications and protocols. Therefore, finding the ideal set of features is fundamental to achieve efficiency in traffic classification. In a preliminary effort, He et al. [182] propose a NFV-based traffic-driven learning framework for traffic classification, called ν TC. ν TC consists of a controller, and a set of ML classifiers and feature collectors as virtual network functions (VNFs). Their objective is to dynamically select the most effective ML classifiers and the most cost-efficient flow features, by leveraging a controller and a group of VNFs, for traffic classification. The ν TC framework strives to achieve a balance between classification accuracy and speed, and the choice of features have a significant impact on these criteria. Therefore, it is critical to determine the most suitable classifier and dynamically adjust feature collection for a given flow protocol (e.g. TCP, UDP, ICMP).

The cost of extracting different features vary from one another. The same holds true for the execution of classifiers. Therefore, it is important to: (i) identify whether a feature should be collected on the data plane or the control plane, and (ii) have a centralized view of network resources while selecting the appropriate classifier. The controller in ν TC is responsible for maintaining the ML models from offline training, and selecting the most

suitable classifier and flow features to collect by chaining the corresponding VNFs at runtime. It also monitors the load on the VNFs for scaling resources, if necessary. The adaptive selection of features and classifiers in ν TC, based on the flow protocol, result in an accuracy of 95.6%. However, the performance overhead of adaptive selection of classifiers and features, and chaining of corresponding VNFs in ν TC is not discussed. Furthermore, fine-grained classification and corresponding results are missing.

SDN offers built-in mechanisms for data collection via the OpenFlow (OF) protocol. Amaral et al. [19] harness SDN and OF to monitor and classify TCP enterprise network traffic. They leverage ML to extract knowledge from the collected data. In their architecture, a SDN application collects flow statistics from controlled switches and pro-actively installs a flow entry to direct all packets to the controller. For TCP traffic, the controller skips the TCP control packets, and stores the features of sizes, timestamps, MAC and IP addresses, and port numbers for the first five packets, along with their inter-arrival times. Then, the controller installs a flow entry with an idle timeout for local processing at the switch. Upon timeout, flow features of packet count, byte count and duration are collected at the controller.

The collected features are pruned using PCA and adjusted to eliminate high variability and scaling effects. However, the use of port numbers as a feature leaves the classifiers susceptible to issues in port-based classification. Nevertheless, the authors evaluate three ensemble ML classifiers, namely RF, Stochastic Gradient Boosting (SGBoost) and Extreme Gradient Boosting (XGBoost). The results exhibit high accuracy for some application classes (e.g. Web Browsing), while poor performance for others (e.g. LinkedIn). The authors do not provide justifications for the performance of the classifiers. However, this can be attributed to the fairly small training dataset used in their evaluation.

In contrast, Wang et al. [462] propose a framework to classify network traffic to QoS classes rather than applications. They assume that applications with similar QoS requirements exhibit similar statistical properties. This allows for equal treatment of different applications having similar QoS requirements. Their framework consists of two components: (i) traffic identification component that resides in switches at the network edge, to detect QoS-significant (i.e. elephant or long-lived) flows, and (ii) QoS aware traffic classification engine in the SDN controller that leverages DPI (for offline labeling) and semi-supervised ML to map long-lived flows to QoS classes. A significant number of flows remain unlabeled due to limited information on all possible/existing applications, thus calling for semi-supervised learning.

Similar to [137], periodic retraining of classifier is required to cater to new applications. The Laplacian-SVM

classifier employed uses flow features from the first twenty packets to classify flows into QoS classes. Furthermore, they employ forward feature selection to reduce the number of features to nine from the initial sixty features. In the evaluation, the accuracy of classifying long-lived flows to QoS classes exceed 90%. However, the performance of the proposed framework is not evaluated, especially in light of the entropy-based features used for traffic classification.

4.4 Summary

Traditionally, Internet traffic has been classified using port numbers, payload and host-based techniques. Port-based techniques are unreliable and antiquated, largely due to the use of dynamic port negotiation, tunneling and misuse of port numbers assigned to well-known applications for obfuscating traffic and avoiding firewalls [54, 109, 176, 286]. In contrast, payload-based techniques are designed to inspect application payload. Though, they are computationally intensive and complicated due to encryption, supervised and unsupervised ML has been successfully applied for traffic classification with high accuracy. Generally, unencrypted handshake payload is used for traffic classification, which is infeasible for high data rate links. On the other hand, long-lived UDP traffic lends itself to supervised payload-based traffic classification, where payload is inspected randomly in an observation window [146]. However, it is not widely applicable and is highly sensitive to the observation window size. Similarly, host-based traffic classification is highly susceptible to routing asymmetries.

In contrast to these myopic approaches, flow feature-based traffic classification techniques inspect the complete communication session, which includes all consecutive, unidirectional packets in the network. This is the most widely studied technique for traffic classification that leverages both supervised and unsupervised ML. In supervised learning, various kernel estimation, NN and SVM-based ML techniques have been employed to achieve high accuracy. Though, traditional kernel estimation techniques are simple and effective, their underlying assumptions are unrealistic and infeasible for traffic classification. In this light, NBKE has been explored for traffic classification, but NN-based traffic classification has shown higher accuracy with probabilistic and, or Bayesian trained weights. Similarly, traditional and multi-class SVM have been applied jointly to increase the accuracy of traffic classification and its applicability to large datasets [464].

Rarely do network operators have complete information about all the applications in their network. Therefore, it is impractical to expect complete a priori knowledge about all applications for traffic classification. Therefore, unsupervised ML techniques have been explored for practical traffic classification using flow features. For traffic

classification with unsupervised ML, both hard and soft clustering techniques have been investigated. Since flow features from applications can exhibit high similarity, it is unrealistic to apply hard clustering for fine-grained traffic classification. On the other hand, soft clustering achieves the required granularity with density-based clustering techniques, which also has a lower training time than EM-based soft clustering technique.

Complete flow feature-based traffic classification has been shown to achieve high spatial and temporal stability, high classification accuracy, and training and runtime scalability [222]. However, it requires extensive memory for storage and delays time sensitive classifier decisions. Flow feature-based traffic classification can be achieved using only a small number of packets in a flow, rather than the complete flow. These sub-flows can be extended synthetically [55] or derived from a small sliding window over each flow [337]. Sub-flow-based traffic classification achieves high accuracy using fast and efficient C4.5 DT classifier with correlation-based filtering. Similar to payload-based traffic classification, encryption can also complicate flow feature-based traffic classification. However, it is possible to circumvent these challenges. For instance, a hierarchical method that identifies service provider followed by type of service, using statistical features from encrypted payload has been highly accurate and temporally stable [409].

Undoubtedly, supervised ML lends itself to accuracy in traffic classification, while unsupervised techniques are more robust. Consequentially, joint application of supervised and unsupervised ML for traffic classification [137, 497] has demonstrated success. Not only are semi-supervised classifiers resilient, they can be easily adapted for zero-day traffic or retrained for increased accuracy against previously unknown applications. Recent advances in networking increase the opportunities in traffic classification with SDN- and NFV-based identification of applications and classes of QoS. Though, some preliminary work in this area has achieved high accuracy, more scrutiny is required with respect to their resilience, temporal and spatial stability, and computational overhead. Most importantly, it is imperative to assess the feasibility of these technologies for time sensitive traffic classification decisions.

5 Traffic routing

Network traffic routing is fundamental in networking and entails selecting a path for packet transmission. Selection criteria are diverse and primarily depend on the operation policies and objectives, such as cost minimization, maximization of link utilization, and QoS provisioning. Traffic routing requires challenging abilities for the ML models, such as the ability to cope and scale with complex and dynamic network topologies, the ability to learn

the correlation between the selected path and the perceived QoS, and the ability to predict the consequences of routing decisions. In the existing literature, one family of ML techniques has dominated research in traffic routing, Reinforcement Learning.

Recall, RL employs learning agents to explore, with no supervision, the surrounding environment, usually represented as a MDP with finite states, and learn from trial-and-error the optimal action policy that maximizes a cumulative reward. RL models are as such defined based on a set of states $\mathcal{S}$, a set of actions per state $\mathcal{A}(s_t)$, and the corresponding rewards (or costs) r_t . When $\mathcal{S}$ is associated with the network, a state s_t represents the status at time t of all nodes and links in the network. However, when it is associated with the packet being routed, s_t represents the status of the node holding the packet at time t . In this case, $\mathcal{A}(s_t)$ represents all the possible next-hop neighbor nodes, which may be selected to route the packet to a given destination node. To each link or forwarding action within a route may be associated an immediate static or dynamic reward (respectively cost) r_t according to a single or multiple reward (respectively cost) metrics, such as queuing delay, available bandwidth, congestion level, packet loss rate, energy consumption level, link reliability, retransmission count, etc.

At routing time, the cumulative reward, i.e. the total reward accumulated by the time the packet reaches its destination, is typically unknown. In Q-learning, a simple yet powerful model-free technique in RL, an estimate of the *remaining* cumulative reward, also known as *Q-value*, is associated with each state-action pair. A Q-learning agent learns the best action-selection policy by greedily selecting at each state the action a_t with highest expected Q-value $\max_{a \in \mathcal{A}(s_t)} Q(s_t, a)$. Once the action a_t is executed and the corresponding reward r_t is known, the node updates the Q-value $Q(s_t, a_t)$ accordingly as follows:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a) \right)$$

α ($0 < \alpha \leq 1$) and γ ($0 \leq \gamma \leq 1$) denote the learning rate and discount factor respectively. The closer α is to 1, the higher is the impact of the most recently learned Q-value. While higher γ values make the learning agent aim for longer-term high rewards. Indeed, the greedy action-selection approach is only optimal if the learning agent knows the current Q-values of all possible actions. The agent can then *exploit* this knowledge to select the most rewarding action. If not, an ϵ -greedy approach may be used such that with probability ϵ the agent chooses to *explore* a random action rather than choosing deterministically the one with highest Q-value.

Though, RL is gaining a lot of attention these days, its application in network traffic routing dates back to the early 1990s. Boyan and Littman's [71, 280] seminal work

introduced Q-routing, a straight-forward application of the Q-learning algorithm to packet routing. In Q-routing, a router x learns to map a routing policy, such as routing to destination d via neighbor y , to its Q-value. The Q-value is an estimate of the time it will take for the packet to reach d through y , including any time the packet would have to spend in node x 's queue plus the transmission time over the link x, y . Upon reception of the packet, y sends back to x the new estimated remaining routing delay, and x adjusts accordingly its Q-value based on a learning rate. After convergence of the algorithm, optimal routing policies are learned.

Q-routing does not require any prior knowledge of the network topology or traffic patterns. However, experiments on a 36-node network demonstrated the Q-routing outperforms the shortest path first routing algorithm in terms of average packet delivery time. It was also found that, although Q-routing does no exploration or fine-tuning after policies and Q-values are learned, it still outperforms in a dynamically changing network topology, a full-echo Q-routing algorithm where the policy is dynamically adjusted to the current estimated time to destination. In fact, under heavy load, the full-echo Q-routing algorithm constantly changes the routing policy creating bottlenecks in the network. On the contrary, the original Q-routing shows better stability and robustness to topology changes under higher loads.

Since then the application of Q-learning to packet routing has attracted immense attention. A number of research efforts from late 1990s and early 2000s, built on and proposed improvements to Q-learning, resulting in three main research directions: (a) improving the performance of Q-routing to increase learning and convergence speed [96, 254], (b) leveraging the low complexity of Q-learning and devising Q-learning-inspired algorithms adapted to the specificities of the network (e.g. energy-constrained networks) and/or routing paradigm (e.g. multicast routing [430]), and (c) enforcing further collaboration between the routing learning agents to achieve complex global performance requirements [424, 479].

In 1996, a memory-based Q-learning algorithm called predictive Q-routing (PQ-routing) is proposed to keep past experiences to increase learning speed. PQ-routing keeps past best estimated delivery times to destination via each neighboring node y and reuses them in tandem with more current ones. In 1997, Kumar et al. apply dual reinforcement Q-routing (DRQ-Routing) to minimize packet delivery time [254]. DRQ-Routing integrates dual reinforcement learning [167] with Q-routing, so that nodes along the route between the source and the destination receive feedbacks in both directions (i.e. from both the up-stream and down-stream nodes). Both PQ-routing and DRQ-Routing are fully distributed as in Q-routing, and use only local information plus the feedbacks received

from neighboring nodes. While PQ-routing shows better performance than Q-routing at lower-loads, DRQ-routing converges faster and the protocol shows better overall performance at the cost of slightly increased communication overhead due to backward rewards.

The problem of ML-based multicast routing was first addressed by Sun et al. [430] in the context of MANETs. Q-MAP, a Q-learning-based algorithm, was proposed to find and build the optimal multicast tree in MANETs. In Q-MAP, Q-values are associated with different upstream nodes and the best Q-values are disseminated directly from the sinks to the nodes thus making exploration of routes unnecessary, while speeding up the convergence of the learning process. Indeed an exploration-free approach eventually leads to maximum routing performance since only actions with maximum Q-values are selected, however it reduces the protocol to a static approach that is insensitive to topology changes.

The traditional single-agent RL model, which is greedy in nature, provides local optimizations regardless of the global performance. Therefore, it is not sufficient to achieve global optimizations such as network lifetime maximization or network-wide QoS provisioning. Multi-Agent Reinforcement Learning (MARL) entails that, in addition to learning information from the environment, each node exchanges local knowledge (i.e. state, Q-value, reward) and decisions (i.e. actions) with other nodes in the network in order to achieve global optimizations. This helps the nodes to consider not only their own performance, but also the one of their neighbors and eventually others, in selecting the routing policy. Generally, this comes at the price of increased complexity as the state is a joint state of all the learning agents, and the transitions are the result of the joint action of all the agents in the system. Q-routing and Q-routing-inspired approaches like PR-routing and DRQ-Routing do use a form of MARL, as Q-values are exchanged between neighboring nodes. This form of MARL is *soft* in that it is easy to implement and has low communication and computational complexity as opposed to the general more complex form of MARL like in [124, 425].

Team-partitioned opaque-transition reinforcement learning (TPOT-RL), proposed by Stone and Veloso for the RoboCup-1998 (Robot Soccer World Cup II) [425], is the first fully collaborative MARL technique to be applied to packet routing [424]. Routing was used by the authors as a proof-of-concept of the applicability of their algorithm to real world problems. However, in practice this algorithm has high computational complexity considering the very large number of states to be explored, and high communication overhead as every routed packet is acknowledged back by the sink along the path from the source for reward computation.

These early works paved the way to a decade of continued and prolific research in the area. While existing research studies preeminently consider routing as a decentralized operation function, and as such distribute the learning function across the routing nodes, works like [276, 461] take a centralized and a partially decentralized approach respectively. In the following we discuss representative works in the area and summarize them in Table 9.

5.1 Routing as a decentralized operation function

RL when applied in a fully distributed fashion, turns each routing node into a learning agent that makes local routing decisions from information learned from the environment. Routing nodes can take their decisions either independently or through collaboration in a multi-agent system fashion.

In [151], Forster et al. use a Q-learning approach in a multicast routing protocol, called FROMS (Feedback Routing for Optimizing Multiple Sinks). The goal of FROMS is to route data efficiently, in terms of hop count, from one source to many mobile sinks in a WSN by finding the optimal shared tree. Like in [430], a FROMS node is a learning agent that runs Q-learning to incrementally learn the real costs of different possible routes. Its state is updated with every data packet that needs to be routed, and the set of actions is defined by the possible next hop neighbors (n_i) and their route to the sinks ($\text{hops}_{D_p}^{n_i}$). Rewards are received back from the upstream nodes and used to update the Q-values of the corresponding actions. However, unlike [430], next-hop neighbors are selected using a variant of the ϵ – greedy algorithm, such that the routing algorithm alternates between an exploration phase and a greedy exploitation phase. FROMS shows up to 5 times higher delivery rates than the popular directed diffusion algorithm [205] in the presence of node failure, and 20% less network overhead per packet due to route aggregation.

Arroyo-Valles et al. [24] propose Q-probabilistic routing (Q-PR), a localization-aware routing scheme for WSNs that applies Q-learning to achieve a trade-off between packet delivery rate, expected transmission count (ETX), and network lifetime. A node's decision as to drop a packet or forward it to one of the neighbors is a function of the energy cost at transmission and reception, packet priority, and the ETX to the sink through the neighbor. A node greedily chooses among its next-hop candidate neighbors the one that minimizes the cost of the route to the sink, which is estimated by the Q-value of the nodes. It updates its Q-value every time it relays a packet, and broadcast it so it is received by its neighbors. Experimental evaluations are carried out through simulations with over 50 different topologies of connected networks.

Q-PR is compared to the greedy perimeter stateless routing algorithm (GPSR) and the Expected progress-Face-Expected progress (EFE), both localization-aware routing algorithms. Results show that Q-PR as well as EFE outperform GPSR in terms of successful delivery rate (over 98% against 75.66%). Moreover, Q-PR shows lower number of retransmission retries and acknowledgements (on average over 50% and 40% less than GPSR and EFE respectively). Thus the Q-PR algorithm preserves better the lifetime of the WSN ($3\times$ and $4\times$ more than GPSR and EFE respectively). However the algorithm requires that each node maintains locally a number of information regarding each of its neighbors. These include the distance between the nodes, the distance of the neighbor to the sink, the delivery probability between nodes, the estimated residual energy at the neighbor, and the 2-hop neighbors. This hampers the scalability of the approach.

Hu and Fei [197] propose QELAR, a model-based variant of the Q-routing algorithm, to provide faster convergence, route cost reduction, and energy preservation in underwater WSNs. In QELAR, rewards account for both the packet transmission energy (incurred for forwarding the packet to the neighbor node) and the neighbor node's residual energy. Taking into account the residual energy helps achieve a balanced energy distribution among nodes by avoiding highly utilized routes (hotspots). A model representation for each packet is adopted such that the state is defined as per which node holds the packet. Next-hop nodes are selected greedily based on their expected Q-values. The latter are maintained by the node along with corresponding transition probabilities learned at runtime. Each time a node forwards a packet, it appends its Q-value along with its energy level.

QELAR is evaluated and compared against the vector-based forwarding protocol (VBF) through simulations with 250 mobile sensor nodes uniformly deployed in a 3D space. Results show that QELAR is 25% more energy efficient than VBF. The lifetime of the network is 25% ~ 30% higher with QELAR in the presence of failures and network partition compared with VBF with comparable transmission range, which makes QELAR more robust to faults. Whereas, both show comparable routing efficiency and delivery rates. On the other hand, further research could be pursued to study the convergence speed of the model-based learning algorithm of QELAR compared to the model-free Q-learning when appropriate learning rate and discount factor are used.

In [277] Lin and Schaar address the problem of routing delay-sensitive applications in the more general context of multi-hop wireless ad hoc networks. They rely on a n -step temporal difference (TD) [433] learning method, and aim at reducing the frequency of message exchange, and thus the communication overhead without jeopardizing the convergence speed. The routing protocol is evaluated

Table 9 Summary of RL-based decentralized, partially decentralized, and centralized routing models

Ref.	Technique (selection)	Application (network)	Dataset	Features ^a	Action set	Evaluation Settings ^a	Improvement ^b
AdaR [461]	Partially decentralized LSP (ϵ -greedy)	Unicast routing (WSN)	Simulations -400 sensors -20 data sources -1 sink	State: $\mathcal{N}_i^k$ Reward: function of - node load - residual energy - hop cost to sink - link reliability	Next-hop nodes to destination	$S = \#nodes$ $A = \#neighbors$	Compared to Q-learning: - Faster convergence (by 40 episodes) - Less sensitive to initial parameters
FROMS [151]	Q-learning (variant of ϵ -greedy)	Multicast routing (WSN)	Omnet++ Mobility Framework with 50 random topologies -50 nodes -5 sources -45 sinks	State: $(\mathcal{N}_i^k, D_k)$ Reward: function of hop cost	$\{a_1 \dots a_m\}$ $a_k = (\mathcal{N}_i^k, D_k)$ $\mathcal{N}_i^k = \text{next hop along the path to sink } D_k$	$S = \#nodes$ $A = \#neighbors$	Compared to directed diffusion: - up to 5x higher delivery rate $\approx 20\%$ lower overhead
Q-PR [24]	Variant of Q-learning (ϵ -greedy)	Localization-aware routing to achieve a trade-off between packet delivery rate, ETX, and network lifetime (WSN)	Simulations -50 different topologies -100 nodes	State: $\mathcal{N}_i$ Reward: function of - distance($\mathcal{N}_i, \mathcal{N}_j$) - distance($\mathcal{N}_j, d$) - energy at $\mathcal{N}_j$ - ETX - $\mathcal{N}_i$'s neighbors for any neighbor $\mathcal{N}_j$ and destination	Next-hop nodes to destination	$S = \#nodes$ $A = \#neighbors$	Delivery rate: 25% more than GPSR Network lifetime 3x more than GPSR 4x more than EFE
Ref.	Technique (selection)	Application (network)	Dataset	Features ^a	Action set	Evaluation Settings ^a	Improvement ^b
Xia et al. [482]	DRQ-learning (greedy)	Spectrum-aware routing (CRN)	OMNET++ simulations - stationary multi-hop CRN - 10 nodes - 2 PUs	State: $\mathcal{N}_i$ Reward: # available channels between current node and next-hop node	Next-hop nodes to destination	$S = \#nodes$ $A = \#neighbors$	Compared to Q-routing: 50% faster at lower activity level Compared to Q-routing and SP-routing: - lower converged end-to-end delay
QELAR [197]	Model-based Q-learning (greedy)	Distributed energy-efficient routing (underwater WSN)	Simulations (ns-2) -250 sensors in $500^3 m^3$ space -100m transmission range - fixed source/sink - 1m/s maximum speed for intermediate nodes	State: $\mathcal{N}_i$ Reward: function of the residual energy of the node receiving the packet and the energy distribution among its neighbor nodes.	Next-hop nodes to destination U packet withdrawal	$S = \#nodes$ $A = 1 + \#neighbors$	Compared to Q-learning: - Faster convergence (40 episodes less) - Less sensitive to initial parameters

Table 9 Summary of RL-based decentralized, partially decentralized, and centralized routing models (Continued)

Lin et al. [277]	n -step TD (greedy)	Delay-sensitive application routing (multi-hop wireless ad hoc networks)	Simulations 2 users transmitting video sequences to the same destination node. 3 ~ 4-hops wireless network	State: current channel states and queue sizes at the nodes in each hop Reward: goodput at destination	Next-hop nodes to destination	$S = n_q^N \times n_c^H$ $A = (N_h^2)^{H-1} \times N_h$ $N = \#nodes$ $N_h = \#nodes$ at hop h $H = \#hops$ $n_q = \#queue$ states $n_c = \#channel$ states	Complexity $\approx 2 \times 10^8$ for the 3-hop network. With 95% less information exchanges. ~ 10% higher PSNR. ~ slightly slower convergence (+1 ~ 2sec)
d-AdaptOR [59]	Q-learning with adaptive learning rate (ϵ -greedy)	Opportunistic routing (multi-hop wireless ad hoc networks)	Simulations on Qual-Net with 36 randomly placed wireless nodes in a 150m x 150m	State: $\mathcal{N}_i$ Reward: - fixed negative transmission cost - is receiver is not the destination - fixed positive reward if receiver is the destination 0 if packet is withdrawn	Next-hop nodes to destination U packet withdrawal	$S = \#nodes$ $A = 1 + \#neighbors$	After convergence ($\approx 300sec$) - ETX comparable to a topology-aware routing algorithm > 30% improvement over greedy-SR, greedy EXOR and SRCR with a single flow - Improvement decreases with # flows
QAR [276]	Centralized SARSA (ϵ -greedy)	QoS-aware adaptive routing (SDN)	Sprint GIP network trace-driven simulations [418] 25 switches, 53 links	State: $\mathcal{N}_i$ Reward: function of delay, loss, throughput	Next-hop nodes to destination	$S = \#nodes$ $A = \#neighbors$	Compared to Q-learning with QoS-awareness: Faster convergence time (20 episodes less)

^a $\mathcal{N}_i$: node i ; D_k : sink k ; S : number of state variables; A : number of possible actions per state; #: number of^b Average values. Results vary according to experimental settings.

in a simulated multi-hop network with 2 sources transmitting videos to a same destination node. Results show that by reducing the frequency of message exchange by 95% (from every 1ms to every 20ms), the goodput and effective data rate are increased by over 40%, and the video quality, calculated in terms of peak signal-to-noise ratio (PSNR), is increased by 10%. The convergence time seems to be only slightly affected ($1 \sim 2\text{sec}$). This is an interesting finding considering the bandwidth that can be saved and the interferences that can be avoided by spacing information exchanges.

Bhorkar et al. also address the problem of routing in multi-hop wireless ad hoc networks. They propose d-AdaptOR [59], a distributed adaptive opportunistic routing protocol which minimizes the average packet routing cost. d-AdaptOR is based on Q-learning with adaptive learning rate.

In opportunistic routing, instead of pre-selecting a specific relay node at each packet transmission as in traditional routing, a node broadcasts the data packet so that it is overheard by multiple neighbors. Neighbors who successfully acknowledge the packet form the set of candidate relays. The node will then choose among the candidate relays the one that will be forwarding the packet to destination. This property is an opportunity for the Q-learner to receive from the candidate relays their *up-to-date* Q-values. Traditionally, in Q-learning action selection is based on older, previously received Q-values.

Routing in d-AdaptOR consists of four main steps: (1) the sender transmits the data packet, (2) neighbors acknowledge the packet while sending its Q-value, the estimated cumulative cost-aware packet delivery reward, (3) the sender selects a routing action, either a next-hop relay or the termination of packet transmission, based on the outcome of the previous step using an ϵ -greedy selection rule (4) after the packet is transmitted, the sender updates its own Q-value at a learning rate that is specific to the selected next-hop relay. The learning rate is adjusted using a counter that keeps track of the number of packets received from that neighbor node. The higher the value of the counter, the higher is the convergence rate, though at the expense of Q-values fluctuations. Indeed the value of the counter depends also on the frequency of explorations. Further research could be pursued to investigate the optimal exploration-exploration strategy and the effects of different strategies on the convergence rate.

d-AdaptOR performance was investigated on the QualNet simulator using a random network benchmark consisting of 36 randomly placed wireless nodes. Simulations show that d-AdaptOR consistently outperforms existing adaptive routing algorithms, in terms of number of retransmissions per packet. Further study could be pursued to investigate the added value of node-specific

learning rates in Q-value computation, compared to the traditional node-oblivious learning rate that is more efficient in terms of storage and computation.

Xia et al. [482] apply a spectrum-aware DRQ-routing approach in cognitive radio networks. In CRNs, the availability of a channel is dynamic, and is dependent on the activity level of the primary user (PU). The purpose of the routing scheme is to enable a node to select a next-hop neighbor node with higher estimate of total number of available channels up to destination. Indeed, higher number of available channels reduces channel contention, and hence reduces the MAC layer delay. However, relying on the total number of available channels along the path to destination can lead to very poor results in practice. The dual DRQ-routing approach was tested through simulations on a tailored stationary (non-mobile) multi-hop network topology with 10 cognitive radio nodes and 2 PUs operating on different channels. DRQ-routing was also compared against spectrum-aware Q-routing and spectrum-aware shortest path routing (SP-routing) at different activity levels. Simulation results show that after convergence, DRQ-routing minimizes end-to-end delay, is faster to converge than Q-routing (50% faster at lower activity level), and that it significantly reduces end-to-end delay compared to SP-routing at higher activity levels. However, although the nodes are not mobile and the topology is fixed, the convergence time at a 2 packet/s activity level is around 700sec which implies that 1400 periods have elapsed before DRQ-routing has converged. As the activity level reaches 2.75packet/s, over 3000 periods are necessary for DRQ-routing to converge. These numbers are quite significant, but that is not surprising considering that a discount factor of 1 was used.

Elwhishi et al. [133] propose a Collaborative Reinforcement Learning (CRL) -based routing scheme for delay tolerant networks. CRL is an extension to RL introduced by Dowling et al. in 2004 for solving system-wide optimization problems in decentralized multi-agent systems with no global state [123], and was first applied to routing in MANETs by the same authors in [124]. Routing schemes for delay tolerant networks are characterized by the lack of end-to-end aspect, and each node explores network connectivity through finding a new link to a next-hop neighbor node when a new packet arrives, which must be kept in the buffer while a link is formed. SAMPLE, the proposed routing mechanism, selects a reliable next-hop neighbor node while taking into account three factors; two factors relevant to the channel availability (node mobility and congestion level), and a factor relevant to the buffer utilization (remaining space in the buffer). These are learned through feedback exchange among agents. Tested with different network topologies and mobility models, SAMPLE shows better performance

than the traditional AODV and DSR routing algorithms in terms of packet delivery ratio and throughput.

5.2 Routing as a partially decentralized operation function

In [461] Wang et al. present AdaR, a routing mechanism for WSNs based on a centralized implementation of the model-free Least Squares Policy Iteration (LSPI) RL technique [258]. AdaR uses an offline learning procedure, and is claimed to converge to the fixed point routing policy faster than the traditional Q-learning. The algorithm takes into account the node's load, its residual energy, and hop count to the sink, as well as the reliability of the links. The algorithm runs in learning episodes. The base station is the learning agent, while the routing nodes are passive in terms of learning. However, actions are selected by the routing nodes in a decentralized fashion based on the Q-values assigned by the base station, and the ϵ -greedy selection algorithm. During each episode, the current Q-values are used to select a route to the base station. At each hop, the full hop information is appended to the packet and is used by the base station to calculate immediate rewards. When the base station has received enough information (the required number of packets is undefined), it calculates the new Q-values of the nodes offline, and disseminates them via a network-wide broadcast.

AdaR is tested on a simulated WSN with varying node residual energy and link reliability. Results show that the algorithm converges faster than Q-learning; a routing success rate of $\sim 95\%$ with a low deviation was reached even before the 5th learning episode, whereas, it took 40 episodes for Q-learning to reach comparable success rates. This can be explained by Q-learning's initial Q-values and the selected learning rate ($\alpha = 0.5$). Appropriate initial Q-values and higher learning rate would have helped Q-learning converge faster. In fact, the authors show that Q-learning is more sensitive to the initial choice of Q-values than AdaR. Indeed AdaR has some useful properties, like taking into account different routing cost metrics and having faster convergence time. However, this comes at the price of higher computational complexity, and communication overhead due to the growing size of the packets at each hop and the broadcasting of Q-values, which also makes it more sensitive to link failures and node mobility.

5.3 Routing as a centralized control function

More recently, a centralized SARSA with a softmax policy selection algorithm has been applied by Lin et al. [276] to achieve QoS-aware adaptive routing (QAR) in SDN. Although a multi-layer hierarchical SDN control plane is considered by the authors, the proposed SARSA-based routing algorithm is not specific to such an architecture,

and is meant to run on any controller that has global visibility of the different paths and links in the network.

For each new flow, the first packet is transmitted by the switch to the controller. The controller implicitly recognizes the QoS requirements of the flow, calculates the optimal route using the SARSA-based QAR algorithm, and accordingly updates the forwarding tables of the switches along the path. The QoS requirements consist in what metric to minimize/maximize (delay, loss, throughput, etc.). They are used to control the weight of each metric in the reward function.

It is suggested that the controller iterates the SARSA algorithm until convergence, which in practice results in delayed routing. The question is, how long is the delay and how suitable is the solution for real-time traffic. Also the impact of routing new flows on the QoS of other flows in the network is overlooked. If the flow is an elephant flow, it may congest the links and severely impact the QoS of flows with tight delay requirements.

5.4 Summary

The low computational and communication requirements of traditional RL algorithms, in particular Q-learning, and their ability to perform well at finding an optimal solution and adapting to changes in the environment, have motivated their—reportedly successful—application to traffic routing in a variety of network settings, as shown in Table 9.

Different approaches have been considered in applying RL to the traffic routing problem. These approaches vary in terms of: (i) level of distribution of the learning capability, and (ii) level of collaboration among multiple learners. Clearly, different approaches lend themselves more naturally to different network topologies and utility functions. For instance, in SDN [276] as well as WSN, the existence of a central node—the controller in SDN and the sink in WSN, respectively—allows for centralized learning. Whereas, routing in wireless ad hoc networks calls for decentralized RL [59, 277] where the learning capability is distributed among the routing nodes.

For the nodes to select the optimal routing policy, they need to evaluate different routing policies (actions) against a given utility function (reward). Rewards can be calculated in a central node, such as a sink or base station like in AdaR [461]. Alternatively, rewards are locally estimated by the nodes, which requires the nodes to exchange information. The nature and the amount of information, as well as the dissemination process, vary according to the utility function, as shown in Table 9. Indeed utility functions such as QoS provisioning, load balancing and network lifetime maximization, as in Q-PR [24], QELAR [197, 277], require more information to be disseminated

at the cost of an increased complexity and communication overhead.

It is also important to notice that learners are very loosely coupled in most recently adopted decentralized RL approaches, where routers tend to select routing policies in an asynchronous, independent, very soft MARL fashion. Clearly, MARL aims at coordinating learning agents in order to achieve the optimal network-wide performance. This should further enhance the routing performance. However, several challenges arise from MARL. In fact, the difficulty of defining a good global learning goal, the overhead for an agent to coherently coordinate with other learning agents, and the longer convergence time can be prohibitive when applying MARL to realistic problem sizes. Indeed, there is a need for understanding the trade-off between benefits and overhead when applying MARL, particularly in resource-constrained and dynamic wireless networks where coordination has eventually a lot to offer.

6 Congestion control

Congestion control is fundamental to network operations and is responsible for throttling the number of packets entering the network. It ensures network stability, fairness in resource utilization, and acceptable packet loss ratio. Different network architectures deploy their own set of congestion control mechanisms. The most well-known congestion control mechanisms are those implemented in TCP, since TCP along with IP constitute the basis of the current Internet [13]. TCP congestion control mechanisms operate in the end-systems of the network to limit the packet sending rate when congestion is detected. Another well-known congestion control mechanism is queue management [72] that operates inside the intermediate nodes of the network (e.g. switches and routers) to complement TCP. There have been several improvements in congestion control mechanisms for the Internet and evolutionary network architectures, such as Delay-Tolerant Networks (DTN) and Named Data Networking (NDN). Despite these efforts, there are various shortcomings in areas such as packet loss classification, queue management, Congestion Window (CWND) update, and congestion inference.

This section describes several research works that demonstrate the potential of applying ML to enhance congestion control in different networks. Majority of the techniques have been applied to TCP/IP networks. It is important to note that the first ML-based approaches for congestion control were proposed in the context of asynchronous transfer mode (ATM) networks [175, 264, 284, 437]. However, we exclude these works from the survey because, to the best of our knowledge, this type of network has a low impact on present and future networking research interests [177].

6.1 Packet loss classification

In theory, TCP works well regardless of the underlying transmission medium, such as wired, wireless, and optical. In practice the standard TCP congestion control mechanism has been optimized for wired networks. However, the major problem in TCP is that it recognizes and handles all packet losses as network congestion, that is buffer overflow. Hence, performing unjustified congestion control when a loss is due to other reasons, such as packet reordering [150], fading and shadowing in wireless networks [130], and wavelength contention in optical networks [214]. As a consequence, TCP unnecessarily reduces its transmission rate at each detected packet loss, lowering the end-to-end throughput.

Therefore, the TCP throughput for wireless networks can be improved by accurately identifying the cause of packet loss [34, 62, 490] and reducing the TCP transmission rate only when congestion is detected. However, TCP congestion control has no mechanism for identifying the cause of packet loss. We term this problem as packet loss classification and various efforts have been made to propose solutions to this problem. In general, the solutions for packet loss classification fall in two broad categories, depending on where the solution is implemented in the network, that is, at intermediate nodes or in end-systems. The former requires additional implementation at the intermediate nodes that either hide the error losses from the sender [32, 33], or communicate to the sender extra statistics about the network state, such as congestion notification [483] and burst acknowledgment (ACK) [490]. It is important to mention that hiding error losses may violate TCP end-to-end principle as it may require splitting the TCP connection by sending an ACK to the sender before the packet arrives at the receiver [129].

In the latter approach, end-systems are complemented with solutions, such as TCP-Veno [156] and TCP-Westwood [463]. These leverage information available at end-systems, such as inter-arrival time (IAT), round-trip time (RTT), and one-way delay, to distinguish causes of packet loss and aid TCP congestion control mechanism. However, it has been shown that it is difficult to perform a good classification using simple tests, such as the ones implemented by TCP-Veno and TCP-Westwood, on these metrics, since they lack correlation to the cause for packet loss [60].

Therefore, various ML-based solutions have been proposed for packet loss classification in end-systems for different networks, such as hybrid wired-wireless [38, 129, 130, 163, 282], wired [150], and optical networks [214]. Generally, the classifier is trained offline, leveraging diverse supervised and unsupervised ML algorithms for binary classification. The majority of these techniques use the metrics readily available at end-systems, and evaluate their classifier on synthetic data on network simulators,

such as ns-2 [203]. We delineate the proposed ML-based solutions for packet loss classification in Table 10 and discuss these techniques in this subsection.

Liu et al. [282] proposed, to the best of our knowledge, the first approach using ML for inferring the cause of packet loss in hybrid wired-wireless networks. Particularly, they distinguish between losses due to congestion and errors in wireless transmission. They employ EM to train a 4-state HMM based on *loss pair* RTT values, that is RTT measured before a packet loss. The Viterbi algorithm [455] is applied on the trained HMM to infer the cause of packet loss. The resultant ML-based packet loss classifier exhibits greater flexibility and superiority over TCP-Vegas [73]. Since, TCP-Vegas has been shown to outperform non-ML-based packet loss classifiers [60], the ML-based solution of [282] was fundamental in creating a niche and instigating the feasibility of ML-based solutions for packet loss classification problems. However, the authors assume that the RTT values never change during measurement. This is an unrealistic assumption since a modification in the return path changes the RTT values without affecting the cause of packet loss. Thus, affecting the correlation between RTT and cause of packet loss.

Barman and Matta [38] use EM on a 2-state HMM and consider discrete delay values to improve the accuracy of the above packet loss classifier, though at the expense of a higher computational cost. This work substitutes the Viterbi algorithm with a Bayesian binary test that provides comparable accuracy, while being computationally efficient. However, this ML-based packet loss classifier, unlike others, requires support from the network to obtain one of its input features, the estimated probability of wireless loss. Furthermore, [38, 282] evaluate their packet loss classifiers on simple linear topologies, which is far from realistic network topologies.

In contrast, El Khayat et al. [129, 130, 163] simulate more than one thousand random hybrid wired-wireless topologies for collecting a dataset of congestion and wireless error losses. The authors compute 40 input features from this dataset by using information that is only available at end-systems, including one-way delay and IAT of packets preceding and succeeding a packet loss. Several supervised ML algorithms are leveraged to build packet loss classifiers using these features. All the classifiers achieve a much higher classification accuracy than non-ML solutions, such as TCP-Veno and TCP-Westwood. In particular, Boosting DT with 25 trees provide the highest accuracy and the second fastest training time. It is important to realize that the training time of DT is the fastest, with a small reduction in accuracy of less than 4% compared to Boosting DT. Therefore, in case of computational constraints, DT achieves the best balance between accuracy and training time.

The authors continue on to improve TCP with the Boosting DT classifier, which exhibit throughput gains over the standard TCP-NewReno [185] and TCP-Veno. The results also show that the improved TCP can maintain a fair link share with legacy protocols (i.e. TCP-friendly). Their ML-based packet loss classifier is flexible and enables the selection between TCP throughput gain and fairness without retraining the classifier.

On the other hand, Fonseca and Crovella [150] focus on detecting the presence of packet loss by differentiating Duplicated ACKs (DUPACK) caused by congestion losses and reordering events. Similar to [282], they employ loss pair RTT as an input feature, however, to infer the network state and not the state of a single TCP connection. Thus, avoiding the poor correlation between RTT and the cause of packet loss. The authors construct a Bayesian packet loss classifier that achieves up to 90% detection probability with a false alarm of 20% on real wired network datasets from the Boston University (BU) and Passive Measure Analysis (PMA) [1]. The performance is superior for the BU dataset due to the poor quality of RTT measurements in the PMA dataset. In addition, the authors adapt an analytic Markov model to evaluate a TCP variant enhanced with the Bayesian packet loss classifier, resulting in a throughput improvement of up to 25% over the standard TCP-Reno.

In the context of optical networks, Jayaraj et al. [214] tackle the classification of congestion losses and contention losses in Optical Burst Switching (OBS) networks. The authors collect data by simulating the National Science Foundation Network (NSFNET) with OBS modules and derive a new feature from the observed losses, called the number of burst between failures (NBBF). They construct two ML-based packet loss classifiers by applying EM for both HMM and clustering. These classifiers integrate two TCP variants that keep a low control overhead for providing better performance (e.g. higher throughput and fewer timeouts) over the standard TCP-NewReno [185] and TCP-SACK [299], and Burst-TCP [490] for OBS networks. The TCP variant using EM for clustering perform slightly better than EM for HMM, as the former produce states (clusters) with a higher degree of similarity, while requiring a similar training time.

6.2 Queue management

Queue management is a mechanism in the intermediate nodes of the network that complements TCP congestion control mechanisms. Specifically, queue management is in charge of dropping packets when appropriate, to control the queue length in the intermediate nodes [72]. The conventional technique for queue management is Drop-tail, which adopts the First-In-First-Out (FIFO) scheme to handle packets that enter a queue. In Drop-tail, each queue establishes a maximum length for accepting

Table 10 Summary of packet loss classification using offline training at end-systems of the network

Ref.	ML Technique	Network	Dataset	Features	Classification	Evaluation Settings	Results
Liu et al. [282]	Unsupervised: • EM for HMM	Hybrid wired and wireless	Synthetic data: • ns-2 simulation • 4-linear topology Data distribution: • Training = 10k	• Loss pair RTT	• Congestion loss • Wireless loss	• 4-state HMM • Gaussian variables • Viterbi inference	HMM accuracy ^a : • .44 — 98%
Barman and Matta [38]	Unsupervised: • EM for HMM	Hybrid wired and wireless	Synthetic data: • ns-2 simulation • Topology: - 4-linear - Dumbbell	• Loss pair delay • Loss probabilities: - Congestion - Wireless (nw) nw: network support	• Congestion loss • Wireless loss	• 2-state HMM • Gaussian variables • Bayesian inference • Discretized values: - 10 symbols	HMM accuracy ^a : • .92 — 98%
El Khayat et al. [129, 130, 163]	Supervised: • Boosting DT • DT • RF • Bagging DT • Extra-trees • MLP-NN • k-NN	Hybrid wired and wireless	Synthetic data: • Simulation in: - ns-2 - BRTE • > 1k random topologies Data distribution: • Training = 25k • Testing = 10k	40 features applying avg, stdev, min, and max on parameters: • One-way delay • IAT And on packets: • 3 following loss • 1 before loss • 1/2 before RTT [130] finds that adding the number of losses is insignificant	• Congestion loss • Wireless loss	Ensemble DT: • 25 trees NN: • 40 input neurons • 2 hidden layers with 30 neurons • 1 output neuron • LMA ^b learning k-NN: • k = 7	AUC (%) ^c : • .98.40 • .94.24 • .98.23 • .97.96 • .98.13 • .97.61 • .95.41
Fonseca and Crovella [150]	Supervised: • Bayesian	Wired	Real data: • PMA project • BU Web server	• Loss pair RTT	• Congestion loss • Reordering	• Gaussian variables • 0 to 3 historic samples	In PMA: • TPR = 80% • FPR = 40% In BU: • TPR = 90% • FPR = 20%
Jayaraj et al. [214]	Unsupervised: • EM for HMM • EM-clustering	Optical	Synthetic data: • ns-2 simulation • NSFNET topology Data distribution: • Training = 25k • Testing = 15k	• Number of bursts between failures	• Congestion loss • Contention loss	HMM: • 8 states • Gaussian variables • Viterbi inference • 26 EM iterations Clustering: • 8 clusters • 24 EM iterations	CV ^c : • .0.16 — 0.42 • .0.15 — 0.28 HMM accuracy ^a : • .86 — 96%

^aVaries according to HMM prior estimates and network simulation settings (e.g. loss rate, error model, delay, traffic)^bLevenberg-Marquardt Algorithm (LMA)^cRespectively to the list of elements in the column ML technique

incoming packets. When the queue becomes full, the subsequent incoming packets are dropped until the queue becomes available again. However, the combination of Drop-Tail with the TCP congestion avoidance mechanism leads to TCP synchronization that may cause serious problems [68, 72]: (i) inefficient link utilization and excessive packet loss due to a simultaneous decrease in TCP rate, (ii) unacceptable queuing delay due to a continuous full queue state; and (iii) TCP unfairness due to a few connections that monopolize the queue space (i.e. *lock-out* phenomenon).

Active Queue Management (AQM) is a proactive approach that mitigates the limitations of Drop-tail by dropping packets (or marking them for drop) before a queue becomes full [72]. This allows end-systems to respond to congestion before the queue overflows and intermediate nodes to manage packet drops. Random Early Detection (RED) [148] is the earliest and most well known AQM scheme. RED continually adjusts a dropping (marking) probability according to a predicted congestion level. This congestion level is based on a pre-defined threshold and a computed average queue length. However, RED suffers from poor responsiveness, fails to stabilize the queue length to a target value, and its performance (w.r.t. link utilization and packet drop) greatly depends on its parameter tuning, which has not been successfully addressed [269]. Many AQM schemes have been proposed to improve these shortcomings [4]. However, they rely on fixed parameters that are insensitive to the time-varying and nonlinear network conditions.

For this reason, significant research has been conducted to apply ML for building an effective and reliable AQM scheme, which is capable of intelligently managing the queue length and tuning its parameters based on network and traffic conditions. The proposals presented in this survey conduct online training in the intermediate nodes of the network and evaluate their solutions by simulating diverse network topologies, mostly in ns2, using characteristics of wired networks. As highlighted in Table 11, these AQM schemes apply different supervised techniques for TSF [160, 179, 212, 498] and reinforcement-based methods for deducing the increment in the packet drop probability [298, 427, 428, 485, 499]. It is important to note that in this section we use the term *increment* to refer to a small positive or negative change in the value of the packet drop probability. The accuracy results depict the quality of the ML technique, for either correctly predicting future time series values or stabilizing the queue length. In addition, the computational complexity of these AQM schemes depend on the learning algorithm employed and the elements that constitute the ML component. For example, the NN structure and its complementing components. In the following, we discuss these ML-based AQM schemes.

PAQM [160], to the best of our knowledge, is the first approach using ML for improving AQM. Specifically, PAQM used OLS on time series of traffic samples (in bytes) for predicting future traffic volume. Based on such predictions, PAQM dynamically adjusted the packet dropping probability. The proposed OLS method relies on the normalized least mean square (NLMS) algorithm to calculate the linear minimum mean square error (LMMSE). Through simulations, the authors demonstrated that their linear predictor achieves a good accuracy, enabling PAQM to enhance the stability of the queue length when compared to RED-based schemes. Therefore, PAQM is capable of providing high link utilization while incurring low packet loss. Similarly, APACE [212] configure the packet dropping probability by using a similar NLMS-based OLS on time series of queue lengths to predict the current queue length. Simulations show that APACE is comparable to PAQM in terms of prediction accuracy and queue stability, while providing better link utilization with lower packet loss and delay under multiple bottleneck links. However, these NLMS-based predictors have a high computational overhead that is unjustified in comparison to a simpler predictor based on, for instance, a low pass filter.

To address these shortcomings, α _SNFAQM [498] was proposed to predict future traffic volume by applying the BP algorithm to train a neuro-fuzzy hybrid model using NN and fuzzy logic, called α _SNF. This α _SNF predictor uses time series of traffic samples and the predicted traffic volume as features. Then, α _SNFAQM leverage the predicted traffic volume and the instantaneous queue length to classify the network congestion as either severe or light. On this basis, α _SNFAQM decides to either drop all packets, drop packets with probability, or drop none. Simulations demonstrate that the α _SNF predictor slightly exceeds the accuracy of the NLMS-based predictor and incurs lower computational overhead. Furthermore, α _SNFAQM achieves smaller and more stable queue length than PAQM and APACE, while providing comparable goodput. However, α _SNFAQM produce more packet drops in order to notify the congestion earlier.

Similarly, to keep a low computational overhead, NN-RED [179] apply an SLP-NN on time series of queue length to predict a future queue length. The predicted queue length is compared to a threshold to decide if packet dropping is needed for preventing severe congestion. The SLP-NN is trained using the least mean square (LMS) algorithm (*a.k.a.*, delta-rule), which is marginally less complex than NLMS. Basic simulations exhibit that NN-RED outperforms RED and Drop-tail in terms of queuing delay, dropped packets, and queue stability. However, this work lacks comparison of NN-RED with similar approaches, such as PAQM, APACE, and α _SNFAQM, in terms of performance and computational overhead.

Table 11 Summary of AQM schemes with online training in the intermediate nodes of a wired network

Ref.	ML Technique	Multiple Bottleneck ^a	Synthetic data from ns-2 simulation	Features	Output (<i>action-set for RL</i>)	Evaluation	
						Settings	Results
PAQM [160]	Supervised: · OLS	✓	Topology: · 6-linear · Arbitrary dumbbell Time = 50s	· Traffic volume (bytes)	TSF: · Traffic volume	· NMLS algorithm based on LMMSE	Accuracy: · 90 – 92.3%
APACE [212]	Supervised: · OLS	✓	Topology: · Dumbbell (1-sink) · 6-linear Time = 40s	· Queue length	TSF: · Queue length	· NMLS algorithm based on LMMSE	Accuracy: · 92%
α _SNFAQM [498]	Supervised: · MLP-NN	–	Topology: · Dumbbell (1-sink) Time = 300s	· Traffic volume · Predicted traffic volume	TSF: · Traffic volume	· 2 input neurons · 2 hidden layers with 3 neurons · 1 output neuron	Accuracy: · 90 – 93%
NN-RED [179]	Supervised: · SLP-NN	–	Topology: · Dumbbell Time = 900s	· Queue length	TSF: · Queue length	· 1+N input neurons (N past values) · 0 hidden layers · 1 output neuron · Delta-rule learning	N/A
DEEP BLUE [298]	Reinforcement: · Q-learning · ϵ -greedy	–	Topology: · Dumbbell Time = 50s <i>OPNET simulator instead of ns-2</i>	States: · Queue length · Packet drop prob. Reward: · Throughput · Queuing delay	Decision making: · Increment of the packet drop probability (<i>finite: 6 actions</i>)	· N/A states · 6 actions · ϵ -greedy ASS ^b	Optimal packet drop probability: · Outperforms BLUE [144]
Neuron [428]	PID Reinforcement: · PIDNN	✓	Topology: · Dumbbell Time = 100s	· Queue length error	Decision making: · Increment of the packet drop probability (<i>continuous</i>)	· 3 input neurons · 0 hidden layers · 1 output neuron · Hebbian learning · 1 PID component	QL _{Acc} error ^c : · 7.15 QL _{Jit} : · 20.18
AN-AQM [427]	Reinforcement: · PIDNN	✓	Topology: · Dumbbell · 6-linear Time = 100s	· Queue length error · Sending rate error	Decision making: · Increment of the packet drop probability (<i>continuous</i>)	· 6 input neurons · 0 hidden layers · 1 output neuron · Hebbian learning · 2 PID components	QL _{Acc} error ^c : · 6.44 QL _{Jit} : · 22.61
FAPIDNN [485]	Reinforcement: · PIDNN	✓	Topology: · Dumbbell Time = 60s	· Queue length error	Decision making: · Increment of the packet drop probability (<i>continuous</i>)	· 3 input neurons · 0 hidden layers · 1 output neuron · 1 PID component · 1 fuzzy component	QL _{Acc} error ^c : · 3.73 QL _{Jit} : · 31.8
NRL [499]	Reinforcement: · SLP-NN	✓	Topology: · Dumbbell Time = 100s	· Queue length error · Sending rate error	Decision making: · Increment of the packet drop probability (<i>continuous</i>)	· 2 input neurons · 0 hidden layers · 1 output neuron · RL learning	QL _{Acc} error ^c : · 38.73 QL _{Jit} : · 128.84

^aSpecifies if the approach was evaluated for multiple bottleneck links (✓) or simply for a single bottleneck link (–)^bAction Selection Strategy (ASS)^cValue computed using RMSE on the results presented in [269] for different network conditions

On the other hand, DEEP BLUE [298] focus on addressing the limitations of BLUE [144], an AQM scheme proposed for improving RED. BLUE suffers from inaccurate

parameter setting and is highly dependent on its parameters. DEEP BLUE addresses these problems by introducing a fuzzy Q-learning (FQL) approach that learns to select

the appropriate increment (*actions*) for achieving the optimal packet drop probability. The features for inferring the FQL *states* are the current queue length and packet drop probability. Whereas, the *reward* signal adopts a linear combination of the throughput and queuing delay. The authors use the OPNET simulator to show that DEEP BLUE improves BLUE in terms of queue stabilization and dropping policy. In addition, the authors mention that DEEP BLUE only generates a slight surplus of storage and computational overhead over BLUE, though no evaluation results are reported.

Other NN-based AQM schemes adopt an RL approach for deciding the proper increment of the packet drop probability. Neuron PID [428] uses a Proportional-Integral-Derivative (PID) controller that incorporates an SLP-NN to tune the controller parameters. Specifically, the SLP-NN receives three terms from the PID component and updates their weights by applying the associative Hebbian learning. The three terms of this PID-based SLP-NN (PIDNN) are computed from the queue length error, which is the difference between the target and current queue lengths. The latter represents the reward signal of the PID control loop. It is important to note that the PID component includes a transfer function that increases the computational overhead of the PIDNN, when compared to a simple SLP-NN.

AN-AQM [427] extends Neuron PID by including another PID component. Therefore, the SLP-NN of AN-AQM receives three terms more from the second PID component for updating their weights. The three terms of the second PID component are generated from the sending rate error, which is the mismatch between the bottleneck link capacity and the queue input rate. The latter serves as the reward signal for the PID control loop. This modification improves the performance of the PIDNN in more realistic scenarios. However, it incurs a higher computational overhead, due to an additional PID transfer function and the increase in the number of input neurons. Similarly, FAPIDNN [485] adopts a fuzzy controller to dynamically tune the learning rate of a PIDNN. As in Neuron PID, FAPIDNN includes only one PID component to calculate the three terms from the queue length error. However, the fuzzy controller of FAPIDNN also adds computational complexity. Alternatively, NRL [499] directly uses an SLP-NN—without a PID or fuzzy component—that relies on a reward function to update the learning parameters. This reward function is computed from the queue length error and the sending rate error.

Li et al. [269] carry out extensive simulations on ns-2 to perform a comparative evaluation of the above NN-based AQM schemes (i.e. Neuron PID, AN-AQM, FAPIDNN, and NRL) and AQM schemes based on RED and Proportional-Integral (PI) controllers. For a single bottleneck link, the results demonstrate that the

NN-based schemes outperform the RED/PI schemes in terms of queue length accuracy (QL_{Acc}) and queue length jitter (QL_{Jit}), under different network settings. Where, QL_{Acc} is the difference between the average queue length and a target value, while QL_{Jit} is the standard deviation of the average queue length. However, the NN-based schemes result in a higher packet drop than PI schemes. For multiple bottleneck links, one of the PI schemes (i.e. IAPI [429]) present better QL_{Acc} and packet drop, yet producing higher QL_{Jit} . When comparing only NN-based schemes, FAPIDNN provides the best QL_{Acc} , while Neuron PID has the least QL_{Jit} and packet drop. Nevertheless, AN-AQM is superior in these performance metrics for realistic scenarios involving UDP traffic noise.

6.3 Congestion window update

CWND is one of the TCP per-connection state variables that limits the amount of data a sender can transmit before receiving an ACK. The other state variable is the Receiver Window (RWND), which is a limit advertised by a receiver to a sender for communicating the amount of data it can receive. The TCP congestion control mechanisms use the minimum between these state variables to manage the amount of data injected into the network [13]. However, TCP was designed based on specific network conditions and assumes all losses as congestion (cf., Section 6.1). Therefore, TCP in wireless lossy links unnecessarily lowers its rate by reducing CWND at each packet loss, negatively affecting the end-to-end performance. Furthermore, the CWND update mechanism of TCP is not suitable for the diverse characteristics of different network technologies [30, 122]. For example, networks with a high Bandwidth-Delay Product (BDP), such as satellite networks, require a more aggressive CWND increase. Whereas, networks with a low BDP, such as Wireless Ad hoc Networks (WANET), call for a more conservative CWND increase.

The challenge of properly updating CWND in resource-constrained wireless networks, like WANET and IoT, is difficult. This is due to their limited bandwidth, processing, and battery power, and their dynamic network conditions [271, 380]. In fact, the deterministic nature of TCP is more prone to cause higher contention losses and CWND synchronization problems in WANET, due to node mobility that continuously modifies the wireless multi-hop paths [29, 379]. Several TCP variations, such as TCP-Vegas and TCP-Westwood, have been proposed to overcome these shortcomings. However, the fixed rule strategies used by such TCP variations are inadequate for adapting CWND to the rapidly changing wireless environment. For example, TCP-Vegas fails to fully utilize the available bandwidth in WANETs, as its RTT-based rate estimate is incorrect under unstable network conditions

[219]. Furthermore, methods for improving TCP-Vegas (e.g. Vegas-W [119]) are still insufficient to account for such variability, as their operation relies on the past network conditions rather than present or future.

As summarized in Table 12, this survey reviews several approaches based on RL that have been proposed to cope with the problems of properly updating CWND (or sending rate) according to the network conditions. Some of these approaches are particularly designed for resource-constrained networks, including WANETs [29, 219, 379, 380] and IoT [271], while others address a wider range of network architectures [30, 122, 477], such as satellite, cellular, and data center networks. Unless otherwise stated, the RL component conducts online training in the end-systems of the network to decide the increment for updating CWND. Although some approaches may apply the same RL technique, they differ in either the defined action-set (i.e. finite or continuous) or the utilization of a function approximation.

The evaluation of these RL-based approaches rely on synthetic data generated from multiple network topologies simulated in tools, such as GloMoSim, ns-2, and ns-3. A couple of these approaches [29, 122] also include experimental evaluation. The performance results show the improvement ratio of each RL-based approach against the best TCP implementation baseline. For example, if an approach is compared to TCP-Reno and TCP-NewReno, we present the improvement over the latter, as it is an enhancement over the former. It is important to note that an optimal CWND update reduces the number of packets lost and delay, and increases the throughput and fairness. Therefore, the selected improvement metrics allow to measure the quality of the RL component for deciding the best set of actions to update CWND.

To the best of our knowledge, TCP-FALA [380] is the first RL-based TCP variant that focuses on CWND adaptation in wireless networks, particularly in WANETs. TCP-FALA introduces a CWND update mechanism that applies FALA to learn the congestion state of the network. On receipt of a packet, it computes five states using IATs of ACKs, and distinguishes DUPACKs to compute the states in a different way. Each state corresponds to a single action that defines the increment for updating CWND. The probabilities for each possible action are continually updated, which are used by TCP-FALA for stochastically selecting the action to be executed. Such stochastic decision facilitates in adapting to changing network conditions and prevents CWND synchronization problem. Simulations in GloMoSim demonstrate that TCP-FALA experiences lower packet loss and higher throughput than standard TCP-Reno in different network conditions. However, the limited size of the action-set makes difficult mapping the range of responses provided by the network to the appropriate actions. In addition, WANETs require a

much finer update of the CWND due to their constrained bandwidth.

To overcome this limitation, Learning-TCP [29, 379] extends TCP-FALA by employing CALA for enabling a finer and more flexible CWND update. Instead of separately calculating probabilities for each action, Learning-TCP continually updates an action probability distribution, which follows a normal distribution and requires less time to compute. Similar to TCP-FALA, Learning-TCP uses IATs of ACKs for computing the states, though without distinguishing DUPACKs and reducing the number of states to two. Several simulations in ns-2 and GloMoSim show that both Learning-TCP and TCP-FALA outperform standard TCP-NewReno in terms of packet loss, goodput, and fairness. Furthermore, the simulations demonstrate that Learning-TCP is superior to TCP-FALA and TCP-FeW [329] (a non-ML TCP variant enhanced for WANETs) with respect to these performance metrics. Whereas, TCP-FALA only achieves better fairness than TCP-FeW. The authors also provide experimental results that are consistent with the simulations.

TCP-GVegas [219] also focuses on updating CWND in WANETs. It improves TCP-Vegas by combining a grey model and a Q-learning model. The grey model predicts the real throughput of the next stage, while the Q-learning model adapts CWND to network changes. This Q-learning model uses the three stages of CWND changes (defined by TCP-Vegas) as the states and the throughput as the reward. The state is determined from CWND, RTT, and actual and predicted throughput. The action-set is continuous and limited by a range computed from RTT, throughput, and a pre-defined span factor. TCP-GVegas adopts an ϵ -greedy strategy for selecting the optimal action that maximizes the quality of the state-action pair. Simulations in ns-2 reveal that TCP-GVegas outperforms TCP-NewReno and TCP-Vegas in terms of throughput and delay for different wireless topologies and varying network conditions. However, TCP-GVegas has higher computational and storage overhead compared to standard TCP and TCP-Vegas. In fact, TCP-GVegas even has a higher computational and storage overhead than TCP-FALA and Learning-TCP, though a more thorough performance evaluation is required to determine the trade-off between these RL-based TCP variants.

In the similar context of resource constrained networks, TCPLearning [271] apply Q-learning for updating CWND in IoT networks. This Q-learning model computes the states by using a 10-interval discretization of each of the following four features: IAT of ACKs, IAT of packets sent, RTT, and Slow Start Threshold (SSThresh). TCPLearning defines a finite action-set that provides five increments for updating CWND and a selection strategy based on ϵ -greedy. The reward for each state-action pair is calculated from the throughput and RTT. To cope with the

Table 12 Summary of decision making of the increment for updating CWND by using online training at end-systems of the network

Ref.	RL Technique	Network	Synthetic Dataset	Features	Action-set (action selection)	Evaluation	
						Settings	Results ^a
TCP-FALA [380]	FALA	WANET	GloMoSim simulation: · Topology: - Random - Dumbbell	States and reward: · IAT of ACKs (distinguish ACKS and DUPACKs)	Finite: · 5 actions (stochastic)	· 1 input feature · 5 states · 5 actions	To TCP-NewReno ^b : · Packet loss = 66% · Goodput = 29% · Fairness = 20% To TCP-FeW ² : · Packet loss = -5% · Goodput = -10% · Fairness = 12%
Learning-TCP [29, 379]	CALA	WANET	Simulation: · ns2 and GloMoSim · Topology: - Chain - Random node - Grid Experimental: · Linux-based · Chain topology	States and reward: · IAT of ACKs	Continuous: · Normal action probability distribution (stochastic)	· 1 input feature · 2 states · ∞ actions	To TCP-FeW: · Packet loss = 37% · Goodput = 13% · Fairness = 23% To TCP-FALA: · Packet loss = 28% · Goodput = 36% · Fairness = 14%
TCP-GVegas [219]	Q-learning	WANET	ns-2 simulation: · Topology: - Chain - Random	States: · CWND · RTTz · Throughput Reward: · Throughput	Continuous: · Range based on RTT, throughput, and a span factor (ϵ -greedy)	· 3 input features · 3 states · N/A actions	To TCP-Vegas: · Throughput = 60% · Delay = 54%
FK-TCP Learning [271]	FKQL	IoT	ns-3 simulation: · Dumbbell topology: - Single source/sink - Double source/sink	States: · IAT of ACKs · IAT of packets sent · RTT · SSThresh Reward: · Throughput · RTT	Finite: · 5 actions (ϵ -greedy)	· 5 input features · 10k states · 5 actions · FK approx: · 100 prototypes	To TCP-NewReno: · Throughput = 34% · Delay = 12% To TCP Learning based on pure Q-learning: · Throughput = -1.5% · Delay = -10%
UL-TCP [30]	CALA	Wireless: · Single-hop: - Satellite - Cellular - WLAN · Multi-hop: - WANET	ns-2 simulation: · Single-hop dumbbell · Multi-hop topology: - Chain - Random - Grid	States and reward: · RTT · Throughput · RTO CWND	Continuous: · Normal action probability distribution (stochastic)	· 3 input features · 2 states · ∞ actions	For single-hop, to ATL: · Packet loss = 51% · Goodput = -14% · Fairness = 53% For multi-hop, similar to Learning-TCP
Remy [477]	Own (offline training)	· Wired · Cellular	ns-2 simulation: · Wired topology: - Dumbbell - Datacenter · Cellular topology	States: · IAT of ACKs · IAT of packets sent · RTT Reward: · Throughput · Delay	Continuous with 3-dimensions: · CWND multiple · CWND increment · Time between successive sends (ϵ -greedy)	· 4 input features · (16k) ³ states · 100 ³ actions · 16 network configurations	To TCP-Cubic: · Throughput = 21% · Delay = 60% To TCP-Cubic/SFQ-CD: · Throughput = 10% · Delay = 38%
PCC [122]	Own	· Wired · Satellite	Experimental: · GENI · Emulab · PlanetLab	States: · Sending rate Reward: · Throughput · Delay · Loss rate	Finite: · 2 actions of the increment for updating sending rate (not CWND) (gradient ascent)	· 3 input features · 4 states · 2 actions	To TCP-Cubic: · Throughput = 21% · Delay = 60%

^a Average value of improvement ratio. Results vary according to the configured network parameters (e.g. topology, mobility, traffic)^b Based on the results from the simulated and experimental evaluations in [29]

memory restrictions of IoT devices, the authors use two function approximation methods: tile coding [435] and Fuzzy Kanerva (FK) [481]. The latter significantly reduces the memory requirements, hence, is incorporated in a modification of TCP Learning, called FK-TCP Learning.

Specifically, FK-TCP Learning with a set of 100 prototypes, needs only 1.2% (2.4KB) of the memory used by TCP Learning based on pure Q-learning (200KB), for storing 50,000 state-action pairs. Furthermore, basic simulations in ns-3 reveal that FK-TCP Learning improves

the throughput and delay of TCP-NewReno, while being marginally inferior to TCP Learning.

The approaches above are specifically designed for resource constrained networks, hence, restricting their applicability. For example, TCP-FALA and Learning-TCP estimate the congestion state from IATs of ACKs, which are prone to fluctuations in single-hop-wireless networks with high and moderate BDP, such as satellite networks, cellular networks, and WLAN. UL-TCP [30] address this gap by modifying Learning-TCP to compute the two congestion states from three different network features: RTT, throughput, and CWND at retransmission timeout (RTO). Simulations of single-hop-wireless networks in ns-2 show that UL-TCP achieves significantly better packet loss and fairness than TCP-NewReno and TCP-ATL [10] (a non-ML TCP variant designed for single-hop-wireless networks). However, UL-TCP is slightly inferior in terms of goodput than TCP-ATL. It is important to note that, unlike UL-TCP, TCP-ATL requires additional implementation in the intermediate nodes of the network. For multi-hop-wireless networks (i.e. WANETs), UL-TCP is compared to TCP-NewReno and TCP-FALA, and exhibit similar results to Learning-TCP. However, UL-TCP is slightly more complex than Learning-TCP due to the storage and usage of more parameters for computing the states.

Remy [477] and PCC [122] went further by introducing congestion control mechanisms that learn to operate in multiple network architectures. Remy designed an RL-based algorithm that is trained offline under many simulated network samples. It aims to find the best rule map (i.e. RemyCC) between the network state and the CWND updating actions to optimize a specific objective function. The simulated samples are constructed based on network assumptions (e.g. number of senders, link speeds, traffic model) given at design time—along with the objective function—as prior knowledge to Remy. The generated RemyCC is deployed in the target network without further learning to update CWND according to the current network state and the rule map. Several tests on simulated network topologies, such as cellular and data center networks, reveal that most of the generated RemyCCs provide a better balance between throughput and delay, in comparison to the standard TCP and its many enhanced variants, including TCP-Vegas, TCP-Cubic [174], and TCP-Cubic over Stochastic Fair Queuing [304] with Controlled-Delay AQM [340] (SFQ-CD). However, if the target network violates the prior assumptions or if the simulated samples incompletely consider the parameters of the target network, the performance of the trained RemyCC may degrade.

To tackle this uncertainty, PCC [122] avoids network assumptions and proposes an online RL-based algorithm that continually selects the increment for updating the

sending rate, instead of CWND, based on a utility function. This utility function aggregates performance results (i.e. throughput, delay, and loss rate) observed for new sending rates during short periods of time. The authors emulate various network topologies, such as satellite and data center networks, on experimental testbeds for evaluating their proposal. The results demonstrate that PCC outperforms standard TCP and other variants specially designed for particular networks, such as TCP-Hybla [83] for satellite networks and TCP-SABUL [172] for inter-data center networks.

6.4 Congestion inference

Network protocols adapt their operation based on estimated network parameters that allow to infer the congestion state. For example, some multicast and multipath protocols rely on predictions of TCP throughput to adjust their behavior [238, 316], and the TCP protocol computes the retransmission timeout based on RTT estimations [22]. However, the conventional mechanisms for estimating these network parameters remain inaccurate, primarily because the relationships between the various parameters are not clearly understood. This is the case of analytic and history-based models for predicting the TCP throughput and the Exponential Weighted Moving Average (EWMA) algorithm used by TCP for estimating RTT.

For the aforementioned reasons, several ML-based approaches have addressed the limitations of inferring the congestion in various network architectures by estimating different network parameters: throughput [238, 316, 371], RTT [22, 128], and mobility [309] in TCP-based networks, table entries rate in NDNs [230], and congestion level in DTNs [412]. As depicted in Table 13, the majority of these proposals apply diverse supervised learning techniques, mostly for prediction. While, the one focused on DTN uses Q-learning for building a congestion control mechanism. The location of the final solution and the training type (i.e. online or offline) differ throughout the proposals, as well as the dataset and tools used for evaluating them. Similarly, the accuracy column shows a variety of metrics mainly due to the lack of consistency from the authors for evaluating the quality of their ML-based components for correctly predicting a specific parameter.

El Khayat et al. [238] apply multiple supervised learning techniques for predicting the TCP throughput in a wired network. From the different features used in the learning phase for building the ML models, the authors find that the Timeout Loss Rate (TLR) adds significant improvement in prediction accuracy. This is because TLR helps to discriminate two types of losses: triple duplicates and timeout. The ML models are trained and tested using synthetic data collected from ns-2 simulations. MLP-NN achieve the lowest accuracy error, followed by MART with

Table 13 Summary of congestion inference from the estimation of different network parameters

Ref.	ML Technique	Network (location)	Dataset	Features	Output	Evaluation	
						Settings	Results ^{ab}
El Khayat et al. [238]	Supervised: · MLP-NN · MART · Bagging DT · Extra-trees (offline)	Wired (end-system)	Synthetic data: · ns-2 simulation · > 1k random topologies Data distribution: · Training = 18k · Testing = 7.6k	· Packet size · RTT: avg, min, max, stdev · Session loss rate · Initial timeout · Packets ACK at once · Session duration · TLR	Prediction: · Throughput	Ensemble DT: · 25 trees NN: N/A	MSE (10^{-3}) ^c : · 0.245 · 0.423 · 0.501 · 0.525
Mirza et al. [316]	Supervised: · SVR (offline)	Multi-path wired (end-system)	Synthetic data: · Laboratory testbed · Dumbbell multi-path topology · RON testbed	· Queuing delay · Packet loss · Throughput	Prediction: · Throughput	· 2 input features · RBF kernel	Rate of predictions with RPE $\leq 10\%$: · Lab: 51% · RON: 87%
Quer et al. [371]	Supervised: · BN (offline)	WLAN (access point)	Synthetic data: · ns-3 simulation · Star topology Data distribution: · Training = 40k · Testing = 10k	· MAC-TX · MAC-RTX · MAC contention window · CWND · CWND status · RTT · Throughput	Prediction: · Throughput	DAG: · 7 vertices · 6 edges	Using MAC-TX: · NRMSE = 0.37 Using all features: · NRMSE = 0.27
Mezzavilla et al. [309]	Supervised: · BN (offline)	WANET (end-system)	Synthetic data: · ns-3 simulation · Topology: · (not mentioned)	· MAC-TX · MAC-RTX · Slots before TX · Queue TX packets · Missing entries in IP table	Classification: · Static · Mobile	DAG: · 6 vertices · 5 edges	Using MAC-TX and MAC-RTX: · Precision = 0.88 · Recall = 0.91
Fixed-Share Experts [22]	Supervised: · WMA (online)	· WANET · Wired · Hybrid wired and wireless (end-system)	Synthetic data: · QualNet simulation · Topology: · Random WANET · Dumbbell wired Real data: · File transfer · Wired and WLAN	· RTT	Prediction: · RTT	· 1 input feature · 100 experts · Simple experts	MAE (ticks): · Synthetic data (ticks of 500ms): = 0.53 · Real data (ticks of 4ms): = 2.95
SENSE [128]	Supervised: · WMA (online)	Hybrid wired and wireless (end-system)	Real data: · Dataset from [22]	· RTT	Prediction: · RTT	· 1 input feature · 100 experts · EWMA experts	MAE (ticks of 4ms): = 1.55
ACCPndn [230]	Supervised: · TLFN · PSO · GA (online)	NDN (controller node)	Synthetic data: · ns-2 simulation · Topology: · DFN · SWITCH Data distribution: · Training = 70% · Validation = 15% · Testing = 15%	· PIT entries rate	Prediction: · PIT entries rate	· R input neurons · 2 hidden layers with R neurons · R output neurons R : number of contributing routers	MSE: · PSO-GA = 2.23 · GA-PSO = 3.25 · PSO = 4.05 · GA = 5.65 · BP = 7.27
Smart-DTN-CC [412]	Reinforcement: · Q-learning · Boltzmann · WoLF (online)	DTN (node)	Synthetic data: · ONE simulation: · Random topology	States: · Input rate · Output rate · Buffer space Reward: · State transition	Decision-making: · Action to control the congestion (finite action-set: 12 actions)	· 3 input features · 4 states · 12 actions	Improvement to CCC: · Delivery ratio = 53% · Delay = 95%

^a Average values. Results vary according to the configured network parameters (e.g. topology, mobility, traffic)^b Error metrics: MAE, MSE, NRMSE, and Relative Prediction Error (RPE)^c Respectively to the list of elements in the column ML technique

25 trees. Both methods are recommended by the authors when learning from the full feature set. In addition, the authors demonstrate that all their ML-based predictors are more TCP-friendly than conventional throughput analytic models, like SQRT [300] and PFTK [343], which are generally used by multicast and real-time protocols.

Mirza et al. [316] also focus on throughput prediction, however, for multi-path wired networks, such as multi-homed and wide-area overlay networks. The authors train and test a supervised time series regression model using SVR on synthetic data collected from two distinct testbeds: authors laboratory deployment and the Resilient Overlay Networks (RON) project [332]. They also include a confidence interval estimator that triggers retraining if the predicted throughput falls outside the interval. The results reveal that the SVR model yields more predictions with a relative prediction error (RPE) of at least 10% than a simple history-based predictor. Moreover, the evaluation show that using active measurement tools for computing the model features, provide predictions as accurate as relying on ideal passive measurements. This is important because it is difficult to correctly collect passive measurements in real wide-area paths.

Another approach for predicting TCP throughput is proposed by Quer et al. [371]. Their ML solution resides in the access point of a WLAN, instead of the end-systems as in the above approaches. The authors apply BN for constructing a DAG that contains the probabilistic structure between the multiple features that allow predicting the throughput. A simplified probabilistic model is derived from the constructed DAG by using a subset of the features for inference. The training and testing of the BN model rely on synthetic data collected from ns-3 simulations. The results demonstrate that for a good amount of training samples (≥ 1000), this model provides a low prediction error. Furthermore, the authors exhibit that the prediction based only on the number of MAC Transmissions (MAC-TX) achieves a comparable error—and sometimes even lower—than using the full set of features.

A similar BN-based approach is proposed by Mezzavilla et al. [309], for classifying the mobility of the nodes in WANETs as either static or mobile. The DAG is built from a fewer number of features, therefore, reducing its number of vertices and edges. As in [371], the authors derive a simplified probabilistic model from the DAG by using two features for inference: MAC-TX and MAC Retransmissions (MAC-RTX). The results reveal that the simplified BN model achieves a good accuracy for classifying mobility in WANETs, when varying the radio propagation stability. This mobility classifier was used to implement a TCP variant that outperforms TCP-NewReno in terms of throughput and outage probability.

Fixed-Share Experts [22] and SENSE [128] concentrate on a different challenge, i.e. predicting RTT for estimating

the congestion state at the end-systems of the network. Both are based on the WMA ensemble method and conduct online training for TFS. It is important to note that WMA uses the term *experts* to refer to algorithms or hypotheses that form the ensemble model. Particularly, SENSE extends Fixed-Share Experts by adding: (i) EWMA equations with different weights as experts, (ii) a meta-learning step for modifying experts penalty regarding recent past history, and (iii) a level-shift for adapting to sudden changes by restarting parameter learning. The two RTT predictors are trained and tested on real data collected from file transfers in a hybrid wired-wireless network. Only Fixed-Share Experts is evaluated on synthetic data collected from QualNet simulations. The results on real data show that SENSE achieves a lower prediction error—measured in ticks of 4ms—than Fixed-Share Experts for predicting RTT. For synthetic data, Fixed-Share Experts provide a lower prediction error in comparison to real data even with a higher tick value of 500ms. In terms of complexity, it is important to mention that SENSE requires more computational resources than Fixed-Share Experts, due to the adoption of EWMA as experts and the meta-learning step.

Finally, other works apply ML techniques to build novel congestion control mechanisms for non-TCP-based networks. ACCPndn [230] propose to include a TLFN into a controller node for predicting the rate of entries arriving to the Pending Interest Table (PIT) of NDN routers. The controller node gathers historical PIT entries rate from contributing routers and sends the prediction back to the corresponding router. The defined TLFN consists of two hidden layers between the input and output layers. The number of neurons for each layer corresponds to the number of contributing routers. To improve the parameter tuning of the TLFN trained using BP, the authors introduce a hybrid training algorithm that combines two optimization methods: PSO and GA. Various tests on synthetic data collected from ns-2 simulations demonstrate that the TLFN trained with PSO-GA provides a lower prediction error than the TLFN with other training algorithms, such as GA-PSO, GA or PSO only, and BP. Additionally, ACCPndn incorporate fuzzy decision-making in each router that uses the predicted PIT entries rate to proactively respond to network congestion. This congestion control mechanism considerably outperforms other NDN congestion control protocols, such as NACK [487] and HoBHIS [392], in terms of packet drop and link utilization.

Smart-DTN-CC [412] is another congestion control mechanism based on ML for DTN nodes. In particular, Smart-DTN-CC applies Q-learning for adjusting the congestion control behavior to the operating dynamics of the environment. Four congestion states are computed from information locally available at each node. The actions are

selected from a finite set of 12 actions based on Boltzmann or Win-or-Learn Fast (WoLF) strategies. The reward of the state-action pairs depend on the transition between the states caused by a specific action. Simulations in the Opportunistic Network Environment (ONE) tool show that Smart-DTN-CC achieves higher delivery ratio and significantly lower delay than existing DTN congestion control mechanisms, such as CCC [265] and SR [406].

6.5 Summary

In the current Internet, TCP implements the most prevalent congestion control mechanism. TCP degrades the throughput of the network when packet losses are due to reasons other than congestion. Therefore, identifying the cause of packet loss can improve TCP throughput. Table 10 summarizes various solutions that have leveraged ML for classifying packet losses at the end-systems of different network technologies. In hybrid wired-wireless networks, the unsupervised EM for HMM and various supervised techniques were used to differentiate wireless losses (e.g. fading and shadowing) from congestion. In wired networks, a supervised Bayesian classifier was proposed to distinguish DUPACKs caused by reordering from the ones due to congestion. In optical networks, the unsupervised algorithm EM was employed on HMM training and clustering for classifying contention and congestion losses.

TCP variants built upon these ML-based classifiers outperform standard and diverse non-ML TCP versions (e.g. TCP-Veno and Burst-TCP). The majority of the ML-based classifiers were tested using synthetic data collected from simulations. EM-based classifiers simulate simpler topologies. Only the Bayesian classifier was evaluated on real data, though the small number of losses in the data negatively affects the results. In addition, all the classifiers perform binary classification of packet losses. Therefore, it would be interesting to explore an ML-based classifier that distinguishes between multiple causes of packet loss.

The other well-known congestion control mechanism is queue management. Several variations of AQM schemes (e.g. RED) have been proposed to overcome the TCP synchronization problem. However, these schemes suffer from poor responsiveness to time-varying and nonlinear network conditions. Therefore, different AQM schemes have integrated ML for better queue length stabilization and parameter tuning in changing network traffic conditions. As depicted in Table 11, half of the ML-based AQM schemes apply supervised OLS and NN for predicting future time series values of either traffic volume or queue length. The predicted values are used to dynamically adjust the packet drop probability. The other half of ML-based schemes employ reinforcement-based methods for deducing the increment in the packet drop probability.

All these ML-based AQM schemes improve and speed up the queue stabilization over non-ML AQM schemes for varying network conditions. However, the evaluation was based only on simulations of wired networks, though including single and multiple bottleneck topologies. Additionally, none of the ML-based AQM schemes have considered providing a fair link share among senders and have not been tested under coexisting legacy schemes in other bottleneck links in the network.

Another shortcoming in TCP is that its CWND update mechanism does not fit the distinct characteristics of different networks. For example, while satellite networks demand an aggressive CWND increase, WANETs perform better under a conservative approach. Table 12 outlines several solutions that have used RL techniques to appropriately update CWND according to the network conditions. Half of these ML-based approaches apply FALA, CALA, or Q-learning (including the FK function approximation) on resource-constrained networks (i.e. WANET and IoT). Whereas, the other half either use CALA or an own RL design on a wider range of network architectures, including satellite, cellular, and data center networks.

TCP variants built upon these ML-based CWND updating mechanisms perform better in terms of throughput and delay than standard and non-ML TCP versions particularly enhanced for specific network conditions (e.g. TCP-FeW and TCP-Cubic). Some of the ML-based TCP also show improvements in packet loss and fairness. The evaluation has been only based on synthetic data collected from simulations and experimental testbeds. In this case, it would be interesting to explore other ML techniques rather than RL for properly updating CWND.

TCP, as well as some multicast and multipath protocols, infer the congestion state from estimated network parameters (e.g. RTT and throughput) to adapt their behavior. However, such estimation remains imprecise mainly because of the difficulty in modeling the relationships between the various parameters. As summarized in Table 13, several solutions have leveraged ML for inferring congestion in different network architectures by estimating diverse network parameters. In the context of TCP-based networks, various supervised techniques were employed to predict the throughput in wired and WLAN networks. A supervised BN was also built to classify the node mobility in WANETs, while the ensemble WMA was used for predicting RTT in WANETs and hybrid wired-wireless networks. In the context of evolutionary network architectures, a supervised TLFN predicted the rate of entries arriving to the PIT of NDN routers, whereas Q-learning was employed in DTN nodes to select the proper action for the corresponding congestion state.

All these ML-based estimators outperform the accuracy of conventional estimation mechanisms (e.g. EWMA

and history-based). The evaluation was mostly performed using synthetic data collected from simulations and laboratory testbeds. Only the WMA-based estimators collected real data to test their approaches.

As final remarks, note that the majority of the ML-based solutions rely on synthetic data to conduct the evaluation. However, synthetic data rely on simulations that may differ from real conditions. Therefore, there is a need to collect data from real networks to successfully apply and evaluate ML-based solutions. In some cases, such as in queue management, real collected data might not be enough to accomplish a realistic evaluation because the solutions impact the immediate network conditions. Therefore, recent networking technologies, like SDN and NFV, might support the evaluation in real networks. In addition, despite some ML-based solutions work on the same problem and report similar evaluation metrics, there is still the need of establishing a common set of metrics, data, and conditions that facilitate their comparison in terms of performance and complexity.

7 Resource management

Resource management in networking entails controlling the vital resources of the network, including CPU, memory, disk, switches, routers, bandwidth, AP, radio channels and its frequencies. These are leveraged collectively or independently to offer services. Naïvely, network service providers can provision a fixed amount of resources that satisfies an expected demand for a service. However, it is non-trivial to predict demand, while over and under estimation can lead to both poor utilization and loss in revenue. Therefore, a fundamental challenge in resource management is predicting demand and dynamically provisioning and reprovisioning resources, such that the network is resilient to variations in service demand. Despite the widespread application of ML for load prediction and resource management in cloud data centers [367], various challenges still prevail for different networks, including cellular networks, wireless networks and ad hoc networks. Though, there are various challenges in resource management, in this survey, we consider two broad categories, admission control and resource allocation.

Admission control is an indirect approach to resource management that does not need demand prediction. The objective in admission control is to optimize the utilization of resources by monitoring and managing the resources in the network. For example, new requests for compute and network resources are initiated for a VoIP call or connection setup. In this case, admission control dictates whether the new incoming request should be granted or rejected based on available network resources, QoS requirements of the new request and its consequence on the existing services utilizing the resources in the network. Evidently, accepting a new request generates

revenue for the network service provider. However, it may degrade the QoS of existing services due to scarcity of resources and consequentially violate SLA, incurring penalties and loss in revenue. Therefore, there is an imminent trade-off between accepting new requests and maintaining or meeting QoS. Admission control addresses this challenge and aims to maximize the number of requests accepted and served by the network without violating SLA.

In contrast, resource allocation is a decision problem that actively manages resources to maximize a long-term objective, such as revenue or resource utilization. The underlying challenge in resource allocation is to adapt resources for long-term benefits in the face of unpredictability. General model driven approaches for resource allocation have fallen short in keeping up with the velocity and volume of the resource requests in the network. However, resource allocation is exemplar for highlighting the advantages of ML, which can learn and manage resource provisioning in various ways.

7.1 Admission control

As shown in Table 14, Admission control has leveraged ML extensively in a variety of networks, including ATM networks [95, 189, 190], wireless networks [8, 36, 359], cellular networks [66, 67, 281, 372, 458], ad hoc networks [452], and next generation networks [311]. To the best of our knowledge, Hiramatsu [189] was the first to propose NN based solutions controlling the admission of a service requesting resources for a basic call setup in ATM networks. He demonstrated the feasibility of NN based approaches for accepting or rejecting requests, for resilience to dynamic changes in network traffic characteristics. However, there were unrealistic underlying assumptions. First, all calls had similar traffic characteristics, that is, single bit or multi bitrate. Second, cell loss rate was the sole QoS parameter.

Later, Hiramatsu [190] overcome these limitations, by integrating admission control for calls and link capacity control in ATM networks using distributed NNs. The NNs could now handle a number of bit-rate classes with unknown characteristics and adapt to the changes in traffic characteristics of each class. Cheng and Chang [95] use a congestion-status parameter, a cell-loss probability, and three traffic parameters, including peak bitrate, average bitrate, and mean peak-rate duration, to achieve a 20% improvement over Hiramatsu. To reduce the dimensionality of the feature space, they transform peak bitrate, average bitrate, and mean peak-rate duration of a call into a unified metric.

Piamrat et al. [359] propose an admission control mechanism for wireless networks based on subjective QoE perceived by end-users. This is in contrast to leveraging quantitative parameters, such as bandwidth, loss and

Table 14 Summary of ML-based Admission Control

Ref.	ML Technique	Network	Dataset	Features	Output	Evaluation Settings	Results
Hiramatsu [189, 190]	Supervised: NN	ATM	Simulation	- Link capacity - Observed call generation rate	Call loss rate	2-10-1 ^a	Improved call loss rate
Cheng and Chang [95]	Supervised: MLP-NN	ATM	Simulation	- Congestion-status - Cell-loss probability - Peak bitrate - Average bitrate - Mean peak-rate duration	Acceptance or rejection	30-30-1 ^a	20% system utilization improvement over [189]
Piamrat et al. [359]	Supervised: RandNN	Wireless	Videos (distorted) generated by streaming application	Codec, bandwidth, delay, and jitter	loss, MOS	N/A	N/A
Baldo et al. [36]	Supervised: MLP-NN	Wireless LAN	ns-3 simulator and testbed	Link load and frame loss	Service quality	9-10-1 ^a	98.5% (offline) 92% (online)
Liu et al. [281]	Supervised: MLP-NN	Cellular (CDMA)	Simulation of cellular networks	- Network environment - User behavior - Call class - Action	GoS	5-10-1 ^a	Performs better than the static algorithms
Bojovic et al. [66]	Supervised: MLP-NN	Cellular (LTE)	ns-3 network simulator	- Application throughput - Average packet error rate - Average size of packet data unit	QoS fulfillment ratio	N/A	Accuracy: 86%
Vassil et al. [452]	Supervised: Probabilistic RBFNN	Ad hoc networks	Pamvotis simulator	- Network throughput - Packet generation rate	Average packet delays	N/A	Correctness: 77% - 88% (Probabilistic RBFNN) Others do not converge
Ahn et al. [8]	Un-Supervised: HNN network	Wireless network	Simulation	Usable QoS levels	QoS assignment matrix for each connection	N/A	Minimized connection blocking and dropping probabilities
Blenk et al. [63]	Supervised: RNN	VN	Simulation	Different graph features	Acceptance or rejection of VN	18 different Recurrent NNs	89% - 98%
Bojovic et al. [67]	Supervised: NN	Cellular (LTE) network	ns-3 simulator	Channel quality indicator	R-factor	Two layers with Number of nodes in the hidden layer: 10 and 20	Accuracy: 98% (BN)
Quer et al. [372]	Supervised: BN	Wireless LAN	ns-3 simulator	Link Layer conditions	Voice call quality	Nodes: 9, Links: 14	Accuracy: 95%
Mignanti et al. [311]	RL: Q-learning	NGN	OMNET simulator	- States - Environment state based on number of active connections of each traffic class	- Action - Accept or reject (greedy)	Not provided	10%-30% better than a greedy approach
Wang et al. [458]	RL: Q-learning	LTE femtocell networks	Simulation	- States - Queue length of handoff and new calls	- Action - Maintain, degrade, or upgrade proportion levels		Reduction in blocking probability
Tong et al. [446]	RL: Q-learning	Multimedia networks	Simulation	- States - The number of calls of each class - Call arrival or termination event - QoS and capacity constraints	- Action - Accept or reject or no action	K x 2, where K is number of constraints	Improvement in rejection rates
Marbach et al. [295]	RL: TD(0)	Integrated service networks	Simulation	- States - The number of calls of each class - Routing path of each active call	- Action - Accept with a route or reject	States 1.4 x 10 ²⁵⁶	2.2% improvement in rewards

latency. To do so, they first choose configuration parameters, such as codec, bandwidth, loss, delay, and jitter, along with their value ranges. Then, the authors synthetically distort a number of video samples by varying the chosen parameters. These distorted video samples are evaluated by human observers who provide a mean opinion score (MOS) for each sample. The configurations and corresponding MOSs are used to construct the training and testing datasets for a Random Neural Network (RandNN), which predicts MOSs in real-time without human interaction. Though, they evaluate their admission control mechanism for user satisfaction and throughput based metrics, no accuracy or error analysis is reported for the RandNN.

Baldo et al. [36] propose a ML-based solution using MLP-NN to address the problem of user driven admission control for VoIP communications in a WLAN. In their solution, a mobile device gathers measurements on the link congestion and the service quality of past voice calls. These measurements are used to train the MLP-NN to learn the relationship between the VoIP call quality and the underlying link layer, thus inferring whether an access point can satisfactorily sustain the new VoIP call. The authors report 98.5% and 92% accuracy for offline and online learning, respectively.

On the other hand, Liu et al. [281] propose a self-learning call admission control mechanism for Code Division Multiple Access (CDMA) cellular networks that have both voice and data services. Their admission control mechanism is built atop a novel learning control architecture (e.g., adaptive critic design) that has only one controller module, namely, a critic network. The critic network is trained with an 3:6:1 MLP-NN that uses inputs such as network environment (e.g. total interference received at the base station), user behavior (e.g. call type—new or hand off call), call class (e.g. voice, data), and the action to accept or reject calls. The output is the Grade of Service (GoS) measure. The MLP-NN is retrained to adapt to changes in the admission control requirements, user behaviors and usage patterns, and the underlying network itself. Through simulation of cellular networks with two classes of services, the authors demonstrate that their admission control mechanism outperforms non-adaptive admission control mechanisms, with respect to GoS, in CDMA cellular networks.

In contrast, Bojovic et al. [66] design an ML-based radio admission control mechanism to guarantee QoS for various services, such as voice, data, video and FTP, while maximizing radio resource utilization in long term evolution (LTE) networks. In their mechanism, the MLP-NN is trained using features, such as application throughput, average packet error rate, and average size of payload. The MLP-NN is then used to predict how the admission of a new session would affect the QoS of all sessions to come. Using a LTE simulator, it is shown that MLP-NN

can achieve up to 86% accurate decisions provided it has been trained over a relatively long period of time. Despite its high accuracy, a critical disadvantage of MLP-NN is over-fitting, thus it fails to generalize in the face of partial new inputs.

Vassiss et al. [452] propose an adaptive and distributed admission control mechanism for variable bitrate video sessions, over ad hoc networks with heterogeneous video and HTTP traffic. Unlike previous admission control approaches that only consider the new request, this mechanism takes into account the QoS constraints of all the services in the network. The authors evaluate five different NNs, namely MLP, probabilistic RBFNN, learning vector quantization network (LVQ) –a precursor to SOM–, HNN, and SVM network. Using network throughput and packet generation rates of all nodes prior to starting each session and the average packet delays of those sessions as the training and validation data, respectively, they found probabilistic RBFNN to always converge with a success rate between 77 and 88%.

Similarly, Ahn et al. [8] propose a dynamic admission control algorithm for multimedia wireless networks based on unsupervised HNN. In this mechanism, new or hand-off connections requesting admission are only granted admission if the bandwidth of the corresponding cell is sufficient to meet the bandwidth required for their best QoS level. Otherwise, the QoS levels of the existing connections are degraded to free up some bandwidth for the new or hand-off connection requesting admission. The compromised QoS levels of existing connections and the QoS levels of the new or hand-off connections are then computed using a hardware-based HNN that permits real-time admission control. Most importantly, the HNN does not require any training, and can easily adapt to dynamic network conditions. This admission control mechanism achieves significant gains in ATM networks, in terms of minimizing the blocking and dropping probabilities and maximizing fairness in resource allocation.

Recently, Blenk et al. [63] employ RNN for admission control for the online virtual network embedding (VNE) problem. Before running a VNE algorithm to embed a virtual network request (VNR), the RNN predicts the probability whether VNR will be accepted by the VNE algorithm based on the current state of the substrate and the request. This allows the VNE algorithm to process only those requests that are accepted by the RNN, thus reducing the overall runtime and improving the system performance. The RNN is trained with new representations of substrate networks and VNRs that are based on topological and network resource features. To obtain a compact representation, the authors apply PCA on a set of feature vectors and select features that are sensitive to high load, including number of nodes, spectral radius, maximum effective eccentricity, average neighbor degree,

number of eigenvalues, average path length, and number of edges. A total of 18 different RNNs are trained offline using a supervised learning algorithm and a dataset generated through simulation of two VNE algorithms, namely Shortest Distance Path and Load Balanced. These RNNs achieve accuracies between 89% and 98%, demonstrating that this admission control mechanism can learn from the historical performances of VNE algorithms.

However, a potential disadvantage of NN based systems is that the confidence of the predicted output is unknown. As a remedy, a BN can predict the probability distribution of certain network variables for better performance in admission control [67, 372]. Specifically, Bojovic et al. [67] compare NN and BN models by applying them for admission control of calls in LTE networks. Both models are trained to learn the network behavior from the observation of the selected features. Upon arrival of an incoming VoIP call and assuming that the call is accepted, these two models are used to estimate the R-factor [206] QoS metric. A major difference between NN and BN is that NN can directly predict the value of the R-factor, while BN provides a distribution over its possible values. In NN, if the estimated R-factor is greater or smaller than a QoS threshold, the call is accepted or rejected, respectively. In contrast, the BN model accepts a call if the probability of the R-factor exceeding a threshold is greater than a probability threshold, or drops it otherwise. This gives the admission control mechanism additional flexibility to choose the probability threshold that allows to meet different system requirements by opportunistically tuning these thresholds. Through a simulation of macro cell LTE admission control scenario in ns-3, the BN model shows less FPs and FNs compared to NN.

Similarly, Quer et al. [372] develop an admission control mechanism for VoIP calls in a WLAN. They employ BN to predict the voice call quality as a function of link layer conditions in the network, including the fraction of channel time occupied by voice and background best effort traffic, estimated frame error probabilities of voice and background traffic, and R-factor representing the posteriori performance. The BN model is built upon four phases, (i) a structure learning phase to find qualitative relationships among the variables, (ii) a parameter learning phase to find quantitative relationships, (iii) the design of an inference engine to estimate the most probable value of the variable of interest, and (iv) an accuracy verification to obtain the desired level of accuracy in the estimation of the parameter of interest. The authors evaluate the BN model via ns-3 based simulation of a WLAN, having both VoIP and TCP traffic, and show an accuracy of 95%.

Besides NN and BN, the admission control problem has also been formulated as an MDP [311, 458]. Traditionally, dynamic programming (DP) is used to solve a MDP. However, DP suffers from two limitations in the

context of admission control. First, it expects the number of states in the MDP to be in polynomial order, which is seldom the case in real networks. Second, DP requires explicit state transition probabilities, which are non-trivial to determine a priori. Therefore, RL, that can handle MDP problems with very large state spaces and unknown state transition probabilities, has been successfully applied to solve MDP-based admission control problems in networking.

Mignanti et al. [311] employ Q-learning to address admission control for connections in next generation networks. In their approach, when a connection request arrives, the Q-values of accepting and rejecting the request are computed. The request is accepted or rejected depending on whether the Q-value for acceptance or rejection is higher. Similarly, Q-learning has been used to allocate guard channels as part of the admission control mechanism for new calls in the LTE femtocell networks [458]. It is important to realize that allocating a guard channel for a new or hand-off call can raise the blocking probability. Therefore, Q-learning has to find the optimal policy that minimizes the cumulative blocking probability.

RL has also been leveraged for more complex problems that pertain to admission control with routing [295, 446]. In such problems, when a request is admitted, a route has to be established such that each link in the route meets the QoS requirements of the request. Therefore, RL-based solutions discussed earlier for admission control, with only two possible actions, are infeasible for admission control with routing. Here, the action space consists of selecting a route from a predefined set of routes in the network. Tong et al. [446] formulate this problem as a semi-MDP, and leverage Q-learning to define policies for route selection, such that the revenue is maximized and QoS requirements of the requests are met. In the formulation, they consider two important classes of QoS constraints, (i) state dependent constraint (e.g. capacity constraint) that is a function of only the current state, and (ii) past dependent constraint (e.g. fairness constraint) that depends on statistics over the past history. Since a detailed specification of a network state is computationally intractable [446], they exploit statistical independence of the links in the network for developing a decentralized RL training and decision making algorithm. In this approach, each link in the network performs Q-learning locally using only the link state information, instead of network state information. The authors evaluate their approach to admission control with routing via simulation of a network with 4 nodes and 12 links. The results show significant improvement over heuristic based algorithms.

Similarly, Marbach et al. [295] use RL to construct a dynamic admission control with routing policy for new calls in integrated service networks. As the traditional

DP-based models for admission control with routing are computationally intractable, Marbach et al. [295] propose an approximation architecture consisting of an MLP with internal tunable weights that can be adjusted using TD(0). However, TD(0) has a slow rate of convergence, hence the authors integrate it with decomposition approach to represent the network as a set of decoupled link processes. This allows to adopt a decentralized training and decision making, which not only significantly reduce training time, but also achieve sophisticated admission control with routing policies that are otherwise difficult to obtain via heuristics approaches.

7.2 Resource allocation

Recall that the challenge in resource allocation lies in predicting demand variability and future resource utilization. ML-based techniques can be leveraged to learn the indicators that can aid in resource allocation as summarized in Table 15. The most suitable ML-based approach for the resource allocation decision problem is RL. The primary advantage of RL is that it can be deployed without any initial policies, and it can learn to adapt to the dynamic demands for a reactive resource allocation. For instance, Tesauro [442] use decompositional RL to allocate and reallocate data center server resources to two different workloads, a web-based time-varying transactional workload and a non-web-based batch workload. Since the impact of a resource allocation decision is Markovian, the RA problem benefits largely from an MDP-based formulation. However, the state and action space of an MDP grows exponentially and leads to the dimensionality problem.

To address this problem, the authors in [442] propose a decompositional formula of RL for composite MDPs. The decompositional RL uses a localized version of SARSA(0) algorithm to learn a local value function based on local state and local resource allocation of a request instead of global knowledge. Vengerov [454] go further in applying RL to the allocation of multiple resource types (e.g. CPU, memory, bandwidth), using fuzzy rules where some or all the fuzzy categories can overlap. Whereas, Mao et al. [294] use DNN to approximate functions in large scale RL task in order to develop a multi-resource cluster scheduler. Most recently, Pietrabissa et al. [361] propose a scalable RL based solution to the MDP problem for resource allocation using policy reduction mechanism proposed in [360] and state aggregation that combines lightly loaded states into one single state.

More specifically, Baldo et al. [35] and Bojovic et al. [65] optimize network resource allocation. Baldo et al. [35] use a supervised MLP-NN for real-time characterization of the communication performance in wireless networks and optimize resource allocation. On the other hand, Bojovic et al. [65] use MLP-NN to select the AP that will provide the best performance to a mobile user

in IEEE 802.11 WLAN. In their proposals, each user collects measurements from each AP, such as signal to noise ratio (SNR), probability of failure, business ratio, average beacon delay, and number of detected stations. These metrics are used to describe different APs and train a two layer MLP-NN. The output of the MLP-NN is the downlink throughput, which is a standard performance metric used by mobile clients. The MLP-NN is trained rigorously with different configuration parameters to result in the lowest normalized RMSE (NRMSE). Finally, the MLP-NN is deployed to select the AP that will yield the optimal throughput in different scenarios and evaluated on EXTREME testbed [364]. Undoubtedly, the ML-based AP selection for network resource allocation, outperforms AP selection mechanisms based on the signal to noise ratio (SNR), the load based scheme and the beacon delay scheme, especially in dynamic environments.

Similarly, Adeel et al. [6] leverage RNN to build an intelligent LTE-Uplink system that can optimize radio resource allocation based on user requirements, surrounding environments, and equipment's ability. In particular, their system can allocate the optimal radio parameters to serving users and suggest the acceptable transmit power to users served by adjacent cells for inter-cell-interference coordination. To analyze the performance of RNN, three learning algorithms are analyzed, namely GD, adaptive inertia weight particle swarm optimization (AIWPSO), and differential evolution (DE). One RNN is trained and validated using each of the above learning algorithms with a dataset of 6000 samples. The dataset is synthetically generated by executing multiple simulations of the LTE environment using a SEAMCAT simulator. Evaluation results show that AIWPSO outperforms the other learning algorithms, with respect to accuracy (based on MSE). However, AIWPSO's better accuracy is achieved at the expense of longer convergence time due to extra computational complexity. Unfortunately, [6] does not evaluate the effectiveness of resource allocation for the proposed LTE system. However, the analysis of the learning algorithms can provide valuable insights in applying ML to similar networking problems.

Though, admission control and resource allocation have been studied separately, Testolin et al. [443] leverage ML to address them jointly for QoE-based video requests in wireless networks. They combine unsupervised learning, using stochastic RNN, also known as RBM, with supervised classification using a linear classifier, to estimate video quality in terms of the average Structural SIMilarity (SSIM) index. The corresponding module uses video frame size that is readily available at the network layer to control admission and resource provisioning. However, the relationship between video frame size and SSIM in non-linear and RBM extracts an abstract representation of the features that describe the video. The linear classifier

Table 15 Summary of ML-based Resource Allocation

Ref.	ML Technique	Network	Dataset	Features	Output	Evaluation Settings	Results
Baldo et al. [35]	Supervised: • MLP-NN	Wireless networks	Simulation data generated using ns-Miracle simulator	• Signal to noise ratio • Received frames • Error-prone frames • Idle time	• Throughput • Delay • Reliability	2 layers with 6 neurons in the hidden layer	Very good accuracy
Bojovic [65]	Supervised: • MLP-NN	Wireless LAN	Synthetic data generated using testbed	• Signal to noise ratio • Probability of failure • Business ratio • Average beacon delay • Number of detected stations	• Throughput of an access point	2 layers with varying number of nodes in the hidden layer, maximum number of epochs, and learning rate	NRMSE = 8%
Adeel et al. [6]	RNN with GD, AIWPSO, and DE	Cellular network	Synthetically generated using a SEAMCAT LTE simulator	• Signal to interference noise ratio • Inter-cell-interference • Modulation/coding schemes • Transmit power	Throughput	5-8-1 ^a	Mean square error • AIWPSO: 8.5×10^{-4} • GD: 1.03×10^{-3} • DE: 9.3×10^{-4}
Testolin et al. [443]	Supervised: • Linear classifier Unsupervised: • RNN	Wireless networks	38 video clips taken from CIF	• Video frame size	• Quality level of each video in terms of the average SSIM index	32 visible units with a varying number of hidden units	RMSE < 3%
Mijumbi et al. [312]	RL • Q-learning (ϵ -greedy and softmax)	VNs	Simulation on ns-3 and real Internet traffic traces	States • Percentages of allocated and unused resources in substrate nodes and links	Actions • Increase or decrease the percentages of allocated resource	2 ⁹ states, 9 actions	Improved the acceptance ratio
Mijumbi et al. [313]	Supervised: • FNN	VNF chains	VoIP traffic traces	• Dependency of resource requirements of each VNF on its neighbor VNFCs • Historical local VNF resource utilization	• Resource requirements of each VNFC	2 NNs for each VNFC	Accuracy ~90%
Shi et al. [410]	Supervised: • MDP • BN	VNF chains	Simulation data generated using WorkflowSim	• Historical resource usage	• Future resource reliability	Running time for MDP: $O(t^{v+1})$, where t and v stand for the number of NFV component tasks and the number of VMs, respectively	Better than other greedy methods in terms of cost

^a Number of neurons at the input layer, hidden, and output layers, respectively

maps the abstractions to the SSIM coefficients, which are leveraged to accept or reject new video requests, and to adapt resource provisioning to meet the network resource requirements. The authors report a RMSE of below 3% using videos from a pool of 38 video clips with different data rates and durations.

Virtualization of network resources through NFV and virtual networks brings forward a new dimension to the resource allocation problem, that is, provisioning virtual resources sitting on top of physical resources. To leverage the benefits of virtualization, Mijumbi et al. [312] propose a dynamic resource management approach for virtual networks (VNs) using distributed RL that dynamically and opportunistically allocates resources to virtual nodes and links. The substrate network is modeled as a decentralized system, where multiple agents use Q-learning on each substrate node and link. These agents learn the optimal policy to dynamically allocate substrate network resources to virtual nodes and links. The percentage of allocated and unused resources (e.g. queue size, bandwidth) in substrate nodes or links represent the states of Q-learning, with two explicit actions to increase or decrease the percentage of allocated resource. A biased learning policy is exploited with an initialization phase to improve the convergence rate of Q-learning. This Q-learning based action selection approach for resource allocation outperforms ϵ -greedy and softmax in ns-3 simulation with real Internet traffic traces. Furthermore, in comparison to static allocation, the proposed method improve the ratio of accepting VNs without affecting their QoS.

In addition, Mijumbi et al. [312] use FNN to predict future resource requirements for each VNF component (VNFC) in a service function chain [313]. Each VNFC is modeled using a pair of supervised FNNs that learn the trend of resource requirements for the VNFC by combining historical local VNFC resource utilization information with the information collected from its neighbors. The first FNN learns the dependence of the resource requirements for each of the VNFCs, which is used by the second FNN to forecast the resource requirements for each VNFC. The predictions are leveraged to spin-up and configure new VNFCs or deallocate resources to turn off VNFCs. Evaluation based on real-time VoIP traffic traces on a virtualized IP Multimedia Subsystem (IMS) reveals a prediction accuracy of approximately 90%.

In contrast, Shi et al. [410] use BN to predict future resource reliability, the ability of a resource to ensure constant system operation without disruption, of NFV components based on historical resource usage of VNFC. The learning algorithm is triggered when an NFV component is initially allocated to resources. As time evolves, the BN is continuously trained with resource reliability responses and transition probabilities of the BN are updated, resulting in improved prediction accuracy. The

predictions are leveraged in an MDP to dynamically allocate resources for VNFCs. Using WorkflowSim simulator, the authors demonstrate that the proposed method outperforms greedy methods in terms of overall cost.

7.3 Summary

As evident from Tables 14 and 15, the ML-based resource management schemes studied in this paper can be broadly classified into two groups—supervised learning-based and RL-based. Application of unsupervised techniques in resource management is rather unexplored, with the exception of a few works. In addition, MLP-NN, though applied with a variety of parameter settings, is the most popular supervised technique, while Q-learning dominates the choice of RL-based approaches. Furthermore, other works have leveraged BN techniques, to introduce the flexibility of having a probability distribution rather than individual values produced by NN-based approaches. However, MLP-NNs offer better scalability than BN and RL, since the number of neurons in different layers of an MLP-NN can be tuned based on the problem dimension. Whereas, the number of states in RL can explode very quickly in a moderate size network. In the past, several techniques, such as decomposition, decentralization, and approximation have been used to deal with the dimensionality issue of applying RL. Recently, RL combined with deep-learning has been shown as a promising alternative [294] that can be leveraged to tackle various resource management problems in practical settings. Nonetheless, NN-based supervised approaches exhibit steady performance in terms of both accuracy and convergence.

Although the ML-based resource management schemes studied in this paper differ in terms of the feature sets, they either predict one or more QoS metrics of interest or generate an acceptance/rejection decision for an incoming request, based on a QoS estimation. The ML-based resource management approaches also exhibit a similarity regarding the network and dataset. The focus of the majority of approaches is on wireless networks, where resource contention is more profound than wired networks. Due to the lack of real-life traces, these approaches adopt different methods to simulate the network of interest and produce training and testing data. Therefore, more research is needed that can evaluate the performance of the proposed ML techniques in real networks and with real data.

8 Fault management

Fault management involves detection, isolation, and correction of an abnormal condition of a network. It requires network operators and administrators to have a thorough knowledge of the entire network, its devices and all the applications running in the network. This is

an unrealistic expectation. Furthermore, recent advances in technology, such as virtualization and softwarization makes today's network monumental in size, complexity and highly dynamic. Therefore, fault management is becoming increasingly challenging in today's networks.

Naïve fault management is reactive and can be perceived as a cyclic process of detection, localization and mitigation of faults. First, fault detection jointly correlates various different network symptoms to determine whether one or more network failures or faults have occurred. For example, faults can occur due to reduced switch capacity, increased rate of packet generation for a certain application, disabled switch, and disabled links [37]. Therefore, the next step in fault management is localization of the root cause of the fault(s), which requires pinpointing the physical location of the faulty network hardware or software element, and determining the reason for the fault. And lastly, fault mitigation aims to repair or correct the network behaviour. In contrast, fault prediction is proactive and aims to prevent faults or failures in the future by predicting them and initiating mitigation procedures to minimize performance degradation. ML-based techniques have been proposed to address these challenges and promote cognitive fault management in the areas of fault prediction, detection, localization of root cause, and mitigation of the faults. In the following subsections, we describe the role ML has played in these prominent challenges for fault management.

8.1 Predicting fault

One of the fundamental challenges in fault management is fault prediction to circumvent upcoming network failures and performance degradation. One of the first ML-based approaches for detecting anomalous events in communication networks is [301]. This approach performs fault prediction by continuously learning to distinguish between normal and abnormal network behaviors and triggering diagnostic measures upon the detection of an anomaly. The continuous learning enables adaptation of the fault prediction and diagnostic measure to the network dynamics without explicit control. Although the work in [301] leverages ML in fault prediction, it does not mention any specific technique. On the other hand, BNs have been widely used in communication and cellular networks to predict faults [193, 247, 248].

In a BN, the normal behavior of a network and deviations from the normal are combined in the probabilistic framework to predict future faults in communication and cellular networks. However, one shortcoming of the system in [193] is that it cannot predict impact on network service deterioration. Nevertheless, a common drawback of BN is that they are not sensitive to temporal factors, and fail to model IP networks that dynamically evolve over time. For such networks, Ding et al. [118] apply dynamic

BN to model both static and dynamic changes in managed entities and their dependencies. The dynamic BN model is robust in fault prediction of a network element, localization of fault and its cause and effect on network performance.

Snow et al. [414] use a NN to estimate the dependability of a 2G wireless network that is used to characterize availability, reliability, maintainability, and survivability of the network. Though the NN is trained vigorously with analytical and empirical datasets, it is limited, due to the wireless network topology having a fixed topology. This is far from reality. Furthermore, network fault predictions are tightly coupled with wireless link quality. Therefore, in Wang et al. [466], the estimation of the link quality in WSNs is postulated as a classification problem, and solved by leveraging supervised DT, rule learner, SVM, BN, and ensemble methods. The results reveal that DTs and rule learners achieve the highest accuracy and result in significant improvement in data delivery rates.

A daunting and fundamental prerequisite for fault prediction is feature selection. It is non-trivial to extract appropriate features from an enormous volume of event logs of a large scale or distributed network system [285]. Therefore, feature selection and dimensionality reduction are imperative for accurate fault prediction. Wang et al. [466] propose to employ local over the global features, as local features can be collected without costly communications in a wireless network. In contrast, Lu et al. [285] use a manifold learning technique called Supervised Hessian Locally Linear Embedding (SHLLE), to automatically extract the failure features and generate failure prediction. Based on an empirical experiment, the authors show that SHLLE outperforms the feature extraction algorithm, such as PCA, and classification methods, including k -NN and SVM.

Pellegrini et al. [355] propose an ML-based framework to predict the remaining time to failure (RTTF) of applications. Their framework is application-agnostic, that is, it is applicable to scenarios where a sufficient number of observations of the monitored phenomena can be collected in advance. The framework uses different ML techniques for building prediction models, namely linear regression, M5P, REPTree, LASSO, SVM, and Least-Square SVM, allowing network operators to select the most suitable technique based on their needs. In addition, other ML techniques can be easily integrated in the framework. The ML techniques in the framework are compared for a multi-tier e-commerce web application running on a virtualized testbed, and show that the REPTree and M5P outperform the other ML techniques for predicting RTTF. It is essential to note that the model has a high prediction error when the network system is temporally far from the occurrence of the failure. However, as the network system approaches the time of the occurrence of the failure, the

number of accumulated anomalies increase and the model is able to predict the RTTF with a high accuracy.

Wang et al. [469] present a mechanism for predicting equipment failure in optical networks using ML-based techniques and TSF. The operational states of an equipment are built by leveraging physical indicators, such as input optical power, laser bias current, laser temperature offset, output optical power, environmental temperature, and unusable time. A double-exponential smoothing time series algorithm uses the historical data from time $t - n$ to time $t - 1$ to predict the values of the physical indicators at a future time instance $t + T$. This is accomplished by using a kernel function and penalty factor in an SVM to model non-linear relationships and reduce misclassification, respectively. The enhanced SVM accomplish an accuracy of 95% in predicting equipment failure based on real data from an optical network operator.

Most recently, Kumar et al. [255] explore the applicability of a wide range of regression and analytical models to predict inter-arrival time of faults in a cellular network. They analyze time-stamped faults over a period of one month from multiple base stations of a national mobile operator in USA. The authors observe that current networks barely reside in a healthy state and patterns of fault occurrence is non-linear. In a comparison of the different ML-based techniques for fault prediction, they show that DNN with autoencoders outperform other ML techniques, including autoregressive NN, linear and non-linear SVM, and exponential and linear regression. An autoencoder is a variant of NN that consists of an encoder and a decoder and used for dimensionality reduction. The autoencoder is pre-trained on the testing data and then converted into a traditional NN for computing prediction error. The pre-training of each layer in an unsupervised manner allows for better initial weights, and results in higher prediction accuracy.

8.2 Detecting fault

Unlike fault prediction, fault detection is reactive and identifies and, or classifies a failure *after* it has occurred, using network symptoms, performance degradation, and other parameters. Rao [382] propose fault detection for cellular networks that can detect faults at different levels, base station, sector, carrier, and channel. They employ a statistical hypothesis testing framework which combines parametric, semi-parametric, and non-parametric test statistics to model expected behavior. In parametric and semi-parametric statistical tests, a fault is detected when significant deviations from the expected activity is observed. In the case of non-parametric statistical tests, where the expected distribution is not known a-priori, the authors use a combination of empirical data and statistical correlations to conduct the hypothesis test. The test is dependent on a threshold value that is initially set through

statistical analysis of traffic patterns. However, improper threshold settings may lead to high FPs and FNs. Hence, the threshold should be adapted to changing traffic patterns due to spatial, temporal, and seasonal effects. In the background, an open loop routine continuously learns and updates the threshold, in an adjustable period of time. However, the time for learning may be large for certain applications that may impact fault detection time.

Baras et al. [37] implement a reactive system to detect and localize the root cause of faults for X.25 protocol, by combining an NN with an expert system. Performance data, such as blocking of packets, queue sizes, packet throughput from all applications, utilization of links connecting subnetworks, and packet end-to-end delays, are used to train a RBFNN for various faults. The output of the NN is a fault code that represents one of the various fault scenarios. A classifier leverages the aggregated output of the NN to determine the current status of the network as normal or faulty. The detection phase is repeated until a confidence of K out of M is achieved, which activates the expert system to collect and deduce the location and cause of the fault.

Recently, Adda et al. [5] build a real-time fault detection and classification model using k -Means, Fuzzy C Means (FCM), and EM. They leverage SNMP to collect information from the routers, switches, hubs, printers and servers in an IP network of a college campus. The authors select 12 features that exhibit sensitivity to the behavior of network traffic [370], and use the traffic patterns to form clusters that represent normal traffic, link failure, server crash, broadcast storm and protocol error. Their evaluation results reveal that though k -Means and EM are relatively faster than FCM, FCM is more accurate.

Moustapha and Selmic [324] detect faulty nodes in a WSN using RNN. The nodes in the RNN hidden layers model sensor nodes in WSN, while the weights on the edges are based on confidence factors of the received signal strength indicators (RSSI). Whereas, the output of the RNN is an approximation of the operation of the WSN. Fault detection is achieved by identifying discrepancies between approximated and real WSN values. The RNN successfully detect faults, without early false alarms, for a small scale WSN with 15 sensors and synthetically introduced faults.

Recall, supervised fault detection requires models to be trained with normal and failure-prone datasets. However, Hajji [178] propose an unsupervised fault detection mechanism for fast detection of anomalies in LAN through traffic analysis. They design a parametric model of network traffic, and a method for baselining normal network operations using successive parameter identification, instead of EM. The fault detection problem is formulated as a change point problem that observes the baseline random variable and raises an alarm as soon as the variable

exceeds an expected value. Experimental evaluation validate the fault detection mechanism in real-time on a real network with high detection accuracy.

Recently, Hashmi et al. [181] use different unsupervised algorithms, such as k -Means, FCM, Kohonens SOM, Local Outlier Factor, and Local Outlier Probabilities, to detect faults in a broadband service provider network that serves about 1.3 million customers. For this purpose, they analyze a real network failure log (NFL) dataset that contains status of customer complaints, along with network generated alarms affecting a particular region during a certain time. The selected data spans a duration of 12 months and contains about 1 million NFL data points from 5 service regions of the provider. The collected NFL dataset has 9 attributes, out of which 5 are selected for the analysis: (i) fault occurrence date, (ii) time of the day, (iii) geographical region, (iv) fault cause, and (v) resolution time. At first, k -Means, FCM and Kohonens SOM clustering techniques are applied to cluster the NFL dataset that is completely unlabeled. Afterwards, density-based outlier determination algorithms, such as Local Outlier Factor, and Local Outlier Probabilities, are used on the clustered data to determine the degree of anomalous behavior for every SOM node. The evaluation results show that SOM outperforms k -Means and FCM in terms of error metric. Furthermore, Local Outlier Probabilities algorithm applied on SOM is more reliable in identifying the spatio-temporal patterns linked with high fault resolution times.

8.3 Localizing the root cause of fault

The next step in fault management is to identify the root cause and physically locate the fault to initiate mitigation. This minimizes the mean time to repair in a network that does not deploy a proactive fault prediction mechanism. Chen et al. [91, 92] use DTs and clustering to diagnose faults in large network systems. The DTs are trained using a new learning algorithm, MinEntropy [91], on datasets of failure prone network traces. To minimize convergence time and computational overhead, MinEntropy uses an early stopping criteria and follows the most suspicious path in the DT. Chen et al. [91] complement the DT with heuristics, to correlate features with the number of detected failures to aid in feature selection and fault localization. MinEntropy is validated against actual failures observed for several months on eBay [127]. For single fault cases, the algorithm identifies more than 90% of the faults with low FPRs. In contrast, Chen et al. [92] employ clustering to group the successes and failures of requests. A faulty component is detected and located by analyzing the components that are only used in the failed requests. In addition to the single fault cases, the clustering approach can also locate faults occurring due to interactions amongst multiple components,

with a high accuracy and relatively low number of false positives.

Ruiz et al. [393] use a BN to localize and identify the most probable cause of two types of failures, the tight filtering and inter-channel interference, in optical networks. They discretize the continuous real-valued features of Quality of Transmission (QoT), such as received power and pre-forward error correction bit error rate (pre-FEC BER) for categories. The authors use these categories and type of failures to train the BN, which can identify the root cause of the failure at the optical layer when a service experiences excessive errors. The BN achieves high accuracy of 99.2% on synthetically generated datasets.

Similarly, Khanafer et al. [237] develop an automated diagnosis model for Universal Mobile Telecommunications System (UMTS) networks using BN. The core elements of the diagnosis model are the causes and symptoms of faults. The authors consider two types of symptoms, i.e., alarms and Key Performance Indicators (KPI). To automatically specify KPI thresholds, they investigated two different discretization methods, an unsupervised method called Percentile-based Discretization (PBD) and a supervised method called Entropy Minimization Discretization (EMD). The performances of the two discretization methods are evaluated on a semi-dynamic UMTS simulator that allows the generation of a large amount of causes and symptoms data required to construct the diagnosis model. As EMD technique outperforms PBD by a large margin in the simulation study, the authors analyze the diagnosis model consisting of BN and EMD in a real UMTS network, utilizing alarms and KPIs extracted from an operations and maintenance center. Using a 3-fold cross-validation test, the correct faults are diagnosed in 88.1% of the cases. In the remaining cases, the diagnosis is incorrect for the first cause but correct for the second, and the diagnosis model converges from around 100 data points.

Kiciman and Fox [241] propose PinPoint for fault detection and localization that requires no a priori knowledge of the faults. The models capture the runtime path of each request served by the network and delineates it as the causal path in the network. It exploits the paths to extract two low-level behaviors of the network, the path shape and the interaction of the components. Using the set of previous path shapes modeled as a Probabilistic Context-Free Grammar (PCFG), it builds a dynamic and self-adapting reference model of the network. Therefore, fault prediction is a search for anomalies against the reference model. Pinpoint uses DT with ID3 to correlate the anomaly to its probable cause in the network. The DT is converted to an equivalent set of rules by generating a rule for each path from the root of the tree to a leaf. PinPoint ranks the rules, based on the number of paths classified

as anomalous, to identify the hardware and, or software components that are correlated with the failures.

Johnsson et al. [225] use discrete state-space particle filtering to determine the locations of performance degradations in packet switched networks. Their approach is based on active network measurements, probabilistic inference, and change detection in the network. They define a PMF to define the location of faulty components in the network. It is a lightweight fault detection and isolation mechanism, which is capable of automatically detecting and identifying the location of the fault in simulation of different sized tree topologies. It is imperative to realize that time to fault localization is dependent on precise position of the fault in the topology. This is because the links closer to the root are measured more often in comparison to links close to the leaf nodes. Hence, the filter is able to learn the positions close to the root. In addition, the algorithm minimizes false positives or false negatives for the chosen parameter values.

Barreto et al. [40] develop an unsupervised approach to monitor the condition of cellular networks using competitive neural algorithms, including Winner-Take-All (WTA), Frequency-Sensitive Competitive Learning (FSCL), SOM, and Neural-Gas algorithm (NGA). The model is trained on state vectors that represent the normal functioning of a CDMA2000 wireless network. Global and local normality profiles (NPs) are built from the distribution of quantization errors of the training state vectors and their components, respectively. The overall state of the cellular network is evaluated using the global NP and the local NPs are used to identify the causes of faults. Evidently, the joint use of global and local NPs is more accurate and robust than applying these methods in isolation.

8.4 Automated mitigation

Automated mitigation improves fault management by minimizing and, or eliminating human intervention, and reducing downtime. For proactive fault prediction, automated mitigation involves gathering information from the suspected network elements to help find the origin of the predicted fault. For building this information base, a fault manager may either actively poll selected network elements, or rely on passive submission of alarms from them. In both cases, actions should be selected carefully since frequent polling wastes network resources, while too many false alarms diminish the effectiveness of automated mitigation. On the other hand, in the case of reactive fault detection, automated mitigation selects a workflow for troubleshooting the fault. Therefore, the fundamental challenge in automated mitigation is to select the optimal set of actions or workflow in a stochastic environment.

He et al. [183] address this fundamental challenge for proactive fault management using a POMDP, to formulate

the trade-off between monitoring, diagnosis, and mitigation. They assume partial observability, to account for the fact that some monitored observations might be missing or delayed in a communication network. They propose an RL algorithm to obtain approximate solutions to the POMDP with large number of states representing real-life networks. The authors devise a preliminary policy where the states are completely observable. Then, they fine-tune this policy by updating the belief space and transition probabilities in the real world, where the states are incompletely observed.

In contrast, for reactive fault detection, Watanabe et al. [470] propose a method for automatically extracting a workflow from unstructured trouble tickets to troubleshoot a network fault. A trouble ticket contains free-format texts that provide a complete history of troubleshooting a failure. The authors use supervised NB classifier to automatically classify the correct labels for each sentence of a trouble ticket and remove unrelated sentences. They propose an efficient algorithm to align the same actions described with different sentences by using multiple sequence alignment. Furthermore, clustering is used to find the actions that have different mitigation steps depending on the situation. This aid the operators in selecting the appropriate next action. Using real trouble tickets, obtained from an enterprise service, the authors report a precision of over 83%.

8.5 Summary

As summarized in Tables 16, 17 and 18, most of the ML-based fault management approaches use different supervised learning techniques that depend on training data to predict/detect/locate faults in the network. However, a common challenge faced by these techniques is the scarcity of fault data generated in a production network. While both normal and fault data are easily available for a test or simulated network, only normal data with infrequent faults are routinely available for a production network. Although injecting faults can help produce the required data [285], it is unrealistic to inject faults in the production network just for the sake of generating training data. On the other hand, synthetic data generated in a test or simulated network may not perfectly mimic the behavior of a production network. Such limitations increase the probability of ML techniques being ill-trained in an unfamiliar network setting. As a remedy, some approaches leverage unsupervised techniques that rely on detecting changes in network states instead of using labeled fault data. However, unsupervised techniques can take longer time to converge than supervised approaches, potentially missing any fault occurring before the convergence. Therefore, a potential research direction can be to explore the applicability of semi-supervised and RL-based techniques for fault management.

Table 16 Summary of ML-based fault prediction

Ref.	ML Technique	Network (location)	Dataset	Features	Output (training)	Evaluation Settings	Results
Hood et al. [193] [248]	Supervised: BN	Campus network	Data collected from router	Management information base (MIB) variables for following network functions · Interface group · IP group · UDP group	Predict network health	500 samples for each of 14 MIB variables of the 3 network functions	Predict approximately 8 min before fault occurrence
Kogeda et al. [248]	Supervised: BN	Cellular network	Simulation with fault injection	·Power ·Cell ·Transmission	Faulty or not	4 nodes each with 3 states	Confidence level of 99.8%
Snow et al. [414]	Supervised: NN (MLP)	Wireless network	Generated using discrete time event simulation	·Mean time to failure ·Mean time to restore ·Time Profile-Run Time	Dependability of a network ·Survivability ·Availability ·Failed components ·Reportable outages	14 inputs, 10 and 5 nodes in the first and second hidden layer, respectively	Closely approximates reportable outages
Wang et al. [466]	Supervised: DT (J4.8) Rule learners (JRip) BN Ensemble	Wireless sensor network	Generated using sensor network testbed	·Received strength ·Send and forward buffer sizes ·Channel load assessment ·Forward and backward	Link quality estimation	10-fold cross validation was used with 5000 samples	Accuracy · 82% for J4.8 ·80% for JRip
Lu et al. [285]	Manifold learning: SHLE	Distributed systems	Generated from a testbed of a distributed environment with a file transfer application	System performance ·interface group ·IP group ·TCP group ·UDP group	Prediction of network, CPU, and memory failures	Not provided	· Precision: 0.452 · Recall: 0.456 · False positive rate: 0.152
Pellegrini et al. [355]	Different ML methods: ·Linear Regression ·MSP · REP-Tree · LASSO · SVM Least-Square SVM	Multi-tier e-commerce web application	Generated from a testbed of a virtual architecture	Different system performance	Remaining Time to Failure (RTTF)	Not provided	Soft error mean absolute error · Linear regression: 137.600 · MSP: 79.182 · REP-Tree: 69.832 · LASSO as a Predictor: 405.187 · SVM: 132.668 · Least-Square SVM: 132.675
Wang et al. [469]	Supervised: Double-exponential smoothing (DES) and SVM	Optical network	Real data collected from an optical network of a telecommunications operator	Indicators In Board Data: ·Input Optical Power · Laser Bias Current · Laser Temperature · Off-set · Output Optical Power · Environmental Temperature · Unusable Time	Predicting equipment failure	10-fold cross-validation was used to test model accuracy	DES with SVM · Prediction accuracy: 95%
Kumar et al. [255]	Unsupervised: DNN with Autoencoders	Cellular Network	Fault data from one of the national mobile operators of USA for a month	Historical data of fault occurrence and their inter-arrival times	Prediction of inter-arrival time of faults	10 neurons in the hidden layer	DNN with autoencoders · NRMSE: 0.122092 · RMSE: 0.504425

Table 17 Summary of ML-based Fault Detection

Ref.	ML Technique	Network (location)	Dataset	Features	Output (training)	Evaluation	
						Settings	Results
Rao [382]	Statistical learning	Cellular network	Data collected from real cellular networks	Mobile user call load profile	Detect faults at -Base station level -Sector level -Carrier level -Channel level	Not provided	Bounded probability of false alarm
Baras et al. [37]	A combination of NN (radial basis functions)	Cellular network (X.25 protocol)	Simulation with OPNET	For each fault scenario -Blocking of packets -Queue sizes -Packet throughput -Utilization on links connecting subnetworks -Packet end-to-end delays	Detect one of the fault scenarios -Reduced switch capacity -Increased packet generation rate of a certain application -Disabled switch -Disabled links	Varying number of hidden nodes between 175 and 230	Different rates of errors
Adda et al. [5]	Supervised: - k -Means - FCM - EM	IP network of a school campus	Obtained from a network with heavy and light traffic scenarios	12 variables of interface (IF) category collected through SNMP	Fault classes: - Normal traffic - Link failure traffic - Server crash - Broadcast storm - Protocol error	Not provided	Precision for heavy scenario in router dataset - k -Means = 40 - FCM = 85 - EM = 40
Moustapha and Selmic [324]	Supervised: - RNN	Wireless sensor network	Collected from a simulated sensor network	- Previous outputs of sensor nodes - Current and previous output samples of neighboring sensor nodes	Approximation of the output of the sensor node	8-10-1 ^a	Constant error smaller than state-of-the-art
Hajji [178]	Unsupervised change detection method	Local area networks	Collected from a real network using remote monitoring agents	Baseline random variable	An alarm as soon as an anomaly occurs	Time to detect : 50 s to 17 min	Accuracy: 100% alarm rate: 0.12 alarms per hour
Hashmi et al. [181]	Supervised: - k -Means - FCM - SOM	Broadband service provider network	1 million NFL data points from 5 service regions	- Fault occurrence date - Time of the day - Geographical region - Fault cause - Resolution time	Identify the spatio-temporal patterns linked with high fault resolution times	SOM on a 15x15 network grid for 154 epochs	Sum of squared errors: - k -Means = 2156788 - FCM = 2822823 - SOM = 1136

^a Number of neurons at the input layer, hidden, and output layers, respectively

Table 18 Summary of ML-based fault localization

Ref.	ML Technique	Network (location)	Dataset	Features	Output (training)	Evaluation	Results
						Settings	
Chen et al. [91]	DT (C4.5)	Network systems	Snapshots of logs from eBay	A complete request trace ·Request type ·Request name ·Pool ·Host ·Version ·Status of each request	Different faulty elements	10 one hour snapshots with 14 faults in total	·Precision: 92% ·Recall: 93%
Ruiz et al. [393]	BN	Optical network	Synthetically generated time series	Quality of Transmission (QoT) parameters ·Received power ·Pre-forward error correction bit error rate (pre-FEC BER)	Detect one of the two fault scenarios ·Tight filtering ·Inter-channel interference	5,000 and 500 time series for training and testing, respectively	Accuracy: 99.2%
Khanafer et al. [237]	BN and EMD	Cellular network	Synthetically generated from a simulated and a real UMTS network	·Causes of faults ·Symptoms, i.e., alarms and KPIs	Identify the cause of the fault	77 and 42 faulty cells for training and testing, respectively	Accuracy: 88.1%
Kiciman and Fox [241]	Supervised: ·DT (ID3)	Three-tier enterprise applications	Generated using small testbed platform	Paths classified as normal or anomalous	Hardware and software components that are correlated with the failures	Three different DTs were evaluated	Correctly detect 89% to 96% of major failures
Johnsson et al. [225]	Unsupervised: discrete state-space particle filtering	IP network	Discrete event simulator	·Active network measurements ·Probabilistic inference ·Change detection	Probability mass function indicating the location of the faulty components	Operations per filter: $O(G)$ where $ G $ is the number of edges in a graph G	Found the location of faults and performance degradations in real time
Barreto et al. [40]	Unsupervised: ·Winner-Take-All (WTA) ·Frequency-Sensitive Competitive Learning (FSCL) ·SOM ·Neural-Gas algorithm (NGA)	Cellular network	Simulation study	State vectors representing the normal functioning of a network	State vector causing the abnormality	400 vectors were used for training and 100 vectors were used for testing	False alarm: ·WTA: 12.43 ·FSCL: 10.20 ·SOM: 8.75 ·NGA: 9.50

The ML-based fault management approaches surveyed in this paper focus on a variety of networks. Consequently, the fault scenarios studied in these approaches vary greatly as they depend both on the layer (e.g. physical, link, or IP layer) and the type (e.g. cellular, wireless, local area network) of the network. The same holds for feature set and output of these schemes, as both features and outputs depend on the fault scenario of a particular network. In addition, the evaluation settings adopted by these approaches lack uniformity. Therefore, a pairwise comparison between the evaluation results of two approaches in any of the Tables 16, 17 and 18 may be misleading. Nonetheless, it is clear that ML techniques can aid the cumbersome and human centric fault management process, by either predicting faults in advance, or narrowing down the cause or location of the fault that could not be avoided in the first place.

9 QoS and QoE management

The knowledge about the impact of network performance on user experience is crucial, as it determines the success, degradation or failure of a service. User experience assessment has attracted a lot of attention. In early works, there was no differentiation between user experience and network QoS. User experience was then measured in terms of network parameters (e.g. bandwidth, packet loss rate, delay, jitter), and application parameters, such as bitrate for multimedia services. While monitoring and controlling QoS parameters is essential for delivering high service quality, it is more crucial, especially for service providers, to evaluate service quality from the user's perspective.

User QoE assessment is complex as individual experience depends on individual expectation and perception. Both are subjective in nature, and hard to quantify and measure. QoE assessment methods went through different stages this last decade, from subjective testing to engagement measurement through objective quality modeling. Subjective testing, where users are asked to rate or assign opinion scores averaged into a mean opinion score (MOS), has been and is still widely used. Subjective testing is simple and easy to implement, and the MOS metric is easy to compute. However because one cannot force users to rate a service and rate it objectively, MOS scores can be unfair and biased, and are subjected to outliers. Objective quality models, such as the video quality metric (VQM) [362], the perceptual evaluation of speech quality (PESQ) metric [386] and the E-model [51] for voice and video services, were proposed to objectively assess service quality by human beings and infer more "fair" and unbiased MOS. Full-reference (FR) quality models, like PESQ and VQM, compute quality distortion by comparing the original signal against the received one. They are as such accurate, but at the expense of a high computational effort. On the contrary, no-reference (NR) models like E-model try to assess

the quality of a distorted signal without any reference to the original signal. They are more efficient to compute, however they may be less accurate. More recently, measurable user *engagement* metrics, such as service time and probability of return, have emerged from data-driven QoE analysis. Such metrics are found to draw more directly the impact of user quality perception to content providers; business objectives.

Statistical and ML techniques have been found useful in linking QoE to network- and application-level QoS, and understanding the impact of the latter on the former. Linear and non-linear regression (e.g. exponential, logarithmic, power regression) was used to quantify the individual and collective impact of network- and application- level QoS parameters (e.g. packet loss ratio, delay, throughput, round-trip time, video bitrate, frame rate, etc.) on the user's QoE. In the literature, simple-regression models with a single feature are most dominant [145, 240, 383, 408], although the collective impact of different QoS parameters was also considered [23, 132].

Simple regression: In [408], Shaikh et al. study existing correlation between different network-level QoS parameters and MOS in the context of a web surfing. They show that a correlation does exist and that among 3 forms of regression (linear, exponential, and logarithmic), linear regression renders the best correlation coefficient between QoE and packet loss rate, exponential regression captures the correlation between QoE and file download time with highest accuracy, whereas logarithmic regression is the best fit for linking QoE to throughput.

Reichl et al. [383], in alignment with the Weber-Fechner law from the field of psychophysics, use logarithmic regression to quantify the correlation between available bandwidth and mobile broadband service users' MOS.

In [145], Fiedler et al. test the IQX hypothesis according to which QoE and QoS parameters are connected through an exponential relationship. Their experiment validates the IQX hypothesis for VoIP services, where PESQ-generated MOS is expressed as a function of packet loss, and reordering ratio caused by jitter. For web surfing, exponential mappings are shown to outperform a previously published logarithmic function.

Steven's power law from the field of psychophysics, according to which there is a power correlation between the magnitude of a physical stimulus and the intensity or strength that people feel, was applied by Khorsandroo et al. [239, 240] to find a power mapping function between MOS and packet loss ratio. A comparative study shows that the proposed power correlation is outperformed by the logarithmic correlation from [383].

Multi-parameter regression: In order to grasp the impact of the global network condition on the QoE, Elko-tob et al. [132] propose to map MOS to a set of QoS parameters (e.g. packet loss rate, frame rate, bandwidth, round trip time and jitter) as opposed to a single one. This idea was further promoted by Aroussi et al. [23] who propose a generic exponential correlation model between QoE and several QoS parameters based on the IQX hypothesis.

More complex regression and classification models based on supervised and unsupervised ML techniques (including deep learning) were also proposed and tested against real-life and trace-driven datasets. We report below on the characteristics of surveyed models and their performance in terms of accuracy, generally measured in terms of MSRE, and linearity, generally measured in terms of Pearson correlation coefficient (PCC), all summarized in Tables 19 and 20.

9.1 QoE/QoS correlation with supervised ML

In [235, 236], Khan et al. propose an Adaptive Neural Fuzzy Inference System (ANFIS)-based model to predict streamed video quality in terms of MOS. They also investigate the impact of QoS on end-to-end video quality for H.264 encoded video, and in particular the impact of radio link loss models in UMTS networks. A combination of physical and application layer parameters is used to train both models. Simulation results show that both models give good prediction accuracy. However, the authors conclude that the choice of parameters is crucial in achieving good performance. The proposed models in this paper need to be validated by more subjective testing. Other works like [501] have also used the ANFIS approach to identify the causal relationship between the QoS parameters that affect the QoE and the overall perceived QoE.

MLP-NNs are also reported to efficiently estimate the QoE by Machado et al. [287], who adopt a methodology that is similar to Khan et al. [235]. In this work, QoE is estimated by applying an MLP over network-related features (delay, jitter, packet loss, etc.) as well as video-related features (type of video, e.g. news, football, etc.). Different MLP models are generated for different program-generated QoE metrics, including Peak-Signal-to-Noise-Ratio (PSNR), MOS, Structural SIMilarity (SSIM) [468], and VQM. A synthetic video streaming dataset of 565 data points is created with EvalVid integrated to NS-2, and the models are trained over 70% of the database for parameter fine-tuning. It is observed that different QoE metrics can lead to very different model parameters. For instance, for the estimated MOS metric, best results are achieved by a single hidden-layer MLP with 10 neurons trained over 2700 epochs. Whereas for SSIM, 2 hidden layers with, respectively, 12 and 24 neurons trained over 1800 epochs are needed to achieve similar results. With

a MSE of ≈ 0.01 , the MOS-MLP model outperforms the other models. Nevertheless, with appropriate configuration all the models are able to predict the QoE with very high accuracy.

In [328], Mushtaq et al. apply six ML classifiers to model QoE/QoS correlation, namely NB, SVM, k -NN, DT, RF and NN. A dataset is generated from a controlled network environment where streamed video traffic flows through a network emulator and different delay, jitter, and packet loss ratio are applied. Opinion scores are collected from a panel of viewers and MOS are calculated. ML models are fed with nine features related to the viewers, network condition and the video itself, namely, viewer gender, frequency of viewing, interest, delay, jitter, loss, conditional loss, motion complexity and resolution. A 4-fold cross-validation is performed to estimate the performance of the models. Results show that DT and RF perform slightly better than the other models with a mean absolute error of 0.126 and 0.136 respectively, and a TPR of 74% and 74.8% respectively. The parameters of the models are not disclosed, and neither is the significance of the selected features in particular the viewer-related ones, whose usefulness and practicality in real-life deployment are questionable.

In [89] Charonyktakis et al. develop a modular user-centric algorithm MLQoE based on supervised learning to correlate the QoE and network QoS metrics for VoIP services. The algorithm is modular in that it trains several supervised learning models based on SVR, single hidden layer MLP-NN, DT, and GNB, and after cross-validation, it selects the most accurate model. The algorithm is user-centric in that a model is generated for each user, which makes it computationally costly and time consuming. 3 datasets are generated synthetically with calls established in 3 different testbeds under different network conditions: during handover (dataset 1), in a network with heavy UDP traffic (dataset 2), in a network with heavy TCP traffic (dataset 3). OMNET++ and a VoIP tool are used in this matter. The QoE of the received calls are assessed through both subjective testing (user-generated MOS) and objective measurement (PESQ and E-model). The no-reference ML models are trained with 10 network metrics (including average delay, packet loss, average jitter, and more) to output predicted MOS. The accuracy of the MLQoE model in predicting MOS and the accuracies of pure SVR, NN, DT and GNB models are further compared against the full-reference PSEQ's, the no-reference E-model's, as well as the predictive accuracies of the single-feature simple-regression WFL [383] and IQX [145] models. Experiments show that, in terms of mean absolute error MAE, the supervised learning models generally outperform E-model and even the full-reference PESQ, only one exception is observed with dataset 2. It also shows that there is no single ML model that outperforms

Table 19 Summary of supervised ML-based QoS/QoE correlation models

Ref.	ML Technique	Application (approach)	Dataset (availability)	Features	Output	Evaluation Settings	Results ^{ab}
Khan et al. [235, 236]	ANFIS	Impact of network and application-level QoS on MPEG4 video streaming over wireless mobile networks (NR regression)	Simulations with Evalvid and ns — 2 • MPEG4 video source • 3 video types • variable network conditions • mobile video streaming client • PSNR-generated MOS	Video type Application-related: frame rate, send bitrate network-level: link bandwidth, packet error rate	MOS	Number of features = 5 5-layer ANFIS-NN: fuzzy layer, product layer, normalized layer, defuzzy layer, total output layer	For slight/gentle/rapid motion video type: • RMSE = 0.15/0.18/0.56 • $R^2 = 0.7/0.8/0.75$ (on normalized data) Outperformed by a simple regression model [235]
Machado et al. [287]	MLP-NN	Impact of QoS and video features over QoE (FR/NR regression)	Simulations on Evalvid integrated to NS — 2 • 3 video types (slight, gentle, rapid motion) • 565 data points • MOS, PSNR, SSIM and VQM generated by Evalvid and the VQMT tool	Delay, jitter, total/I/P/B frame loss • not clear if type of video is considered	A model is created for each output • MOS • PSNR • SSIM • VQM	Number of features = 6 ~ 7 (-10,1) MOS-MLP (-10,1) PSN-MLP (-12,24,1) SSIM-MLP (-10,1) VQM-MLP	MOS-MLP • MSE ≈ 0.01 PSNR-MLP • MSE ≈ 0.14 SSIM-MLP • MSE ≈ 0.01 VQM-MLP MSE • MSE ≈ 0.3 (on normalized data)
Mushtaq et al. [328]	DT, RF, NB, SVM, k-NN, and NN	Impact of QoS, video features and viewer features over QoE (NR classification)	Collected from streaming videos over QoS-controlled emulated network, and MOS collected from a panel of viewers	network-level: • delay, jitter, packet loss, etc. application-related: • resolution type of video: • motion complexity viewer-related: • gender, interest, etc.	MOS	Number of features = 9 k-NN (k = 4) Other settings (N/A)	RF • MAE = 0.136 • TP = 74.8% DT • MAE = 0.126 • TP = 74% NB • MAE ≈ 0.23 • TP ≈ 57% SVM • MAE = 0.26 • TP ≈ 61% 4-NN • MAE ≈ 0.2 • TP = 49% NN • MAE ≈ 0.18 • TP ≈ 65% (on normalized data)

Table 19 Summary of supervised ML-based QoS/QoE correlation models (Continued)

Ref.	ML Technique	Application (approach)	Dataset (availability)	Features	Output	Evaluation Settings	Results ^{ab}
MLQoE [89]	SVR, MLP-NN, DT, and GNB	modular user-centric correlation of QoE and network QoS metrics for VoIP services (NR regression)	3 datasets of VoIP sessions under different network conditions generated with OMNET++: during handover (dataset 1), in a network with heavy UDP traffic (dataset 2), in a network with heavy TCP traffic (dataset 3) QoE assessed with user-generated MOS and program-generated PESQ and E-model QoE	network-related: delay, jitter, packet loss, etc.	MOS	Number of features= 10 MLP-NN (10, 2 ~ 5, 1) Gaussian, linear, and polynomial kernel SVR	SVR • MAE ₁ = 0.66 • MAE ₂ = 0.65 • MAE ₃ = 0.47 MLP-NN • MAE ₁ = 0.75 • MAE ₂ = 0.68 • MAE ₃ = 0.53 DT • MAE ₁ = 0.73 • MAE ₂ = 0.55 • MAE ₃ = 0.5 GNB • MAE ₁ = 0.69 • MAE ₂ = 0.68 • MAE ₃ = 0.53 (on normalized data)
Dermibilek et al. [114]	RF, BG, and DNN	Correlation of QoE and network and application QoS metrics for video streaming services (NR regression)	INRS dataset, including user-generated MOS on audiovisual sequences encoded and transmitted with varying video and network parameters, and other pub (public [112])	network-related: delay, jitter, packet loss, etc. application-related: video frame rates, quantization parameters, filters, etc.	MOS	Number of features: • RF ₁ , BG ₁ = 34 • RF ₂ , BG ₂ = 5 • DNN ₂₁ , DNN ₂₂ = 5 RF, BG tree size= 200 Number of hidden layers: • DNN ₂₁ = 1 • DNN ₂₂ hidden= 20	RF ₁ • RMSE= 0.340 • PCC= 0.930 RF ₂ • RMSE= 0.340 • PCC= 0.930 BG ₁ • RMSE= 0.345 • PCC= 0.928 BG ₂ • RMSE= 0.355 • PCC= 0.925 DNN ₂₁ • RMSE= 0.403 • PCC= 0.909 DNN ₂₂ • RMSE= 0.437 • PCC= 0.894 (on normalized data)

^a Average values. Results vary according to experimental settings^b Accuracy metrics: mean square error (MSE), mean absolute error (MAE), root MSE (RMSE), correlation coefficient (R), Pearson correlation coefficient (PCC)

Table 20 Summary of ML-based QoS/QoE prediction models for HAS and DASH

Ref.	ML Technique (training)	Application (approach)	Dataset (availability)	Features	Output	Evaluation Settings	Results ^{a,b}
CS2P [432]	Supervised: HMM (offline)	Throughput prediction for midstream bitrate adaptation in HAS clients to improve the QoE for video streaming (regression)	iQYI dataset consisting of 20 million sessions covering 3 million unique clients IPs and 18 server IPs 87 ISPs	Throughput samples	Throughput 1 ~ 10 periods ahead	HMM model per cluster of similar sessions: . Number of states= 6 . Number of samples= 100 SVM, GBR single model for all sessions: . Number of features=6	MAE= 7% (on normalized data) . up to 50% more accurate than SVR, GBR and HMM with no clustering . 3.2% improvement on overall QoE . 10.9% improved bitrate over MPC
Claeys et al. [102]	Reinforcement learning: Q- Learning (online)	Video quality adaptation in a HAS client to maximize QoE under varying network conditions (rule extraction)	ns-3 simulation based on TCP streaming sessions in Norway's Telenor 3G/HSDPA mobile wireless network dataset. (public [384])	State: . client buffer filling level . client throughput level Reward: QoE as function of . targeted quality level . span between current and targeted video quality level . rebuffering level	Finite action set of $N = 7$ possible video quality levels (softmax selection)	Improvement compared to Microsoft MSS: 9.12% higher estimated MOS .1665% lower standard deviation	$S = (N+1) \frac{B_{max}}{T_{seg}+1}$.A = N

^a Average values. Results vary according to experimental settings^b Evaluation metrics: mean absolute error (MAE); S: number of state variables; A: number of possible actions per state

all others; while the SVR model has the lowest MAE with dataset 1 (0.66), DT achieves the best result with dataset 2 (0.55) and GBN with dataset 3 (0.43). MLQoE further outperforms the WFL-model and the IQX-model with a MAE improvement of $18 \sim 42\%$. Indeed this motivates the need for a modular ML-based QoE prediction algorithm. However, further research could be pursued to study the correlation between the performance of the different ML-models and the way the QoS parameters evolve in each of the 3 datasets.

Another subset of ML techniques are considered by Demirbilek et al. [114] and used to develop no-reference models to predict QoE for audiovisual services. These techniques include: decision tree ensemble methods (RF and BG), and deep learning (DNN). Genetic programming (GP) is also considered and compared against the ML techniques. All models are trained and validated through 4 $\sim$ 10-fold cross-validation on the INRS dataset [113]. The dataset includes user-generated MOS on audiovisual sequences encoded and transmitted with varying video frame rates, quantization parameters, filters and network packet loss rates. 34 no-reference application- and network-level features are considered. Experiments with different feature sets show that, apart from the DNN model, all models perform better with the complete set of features, and hence do not require feature processing. On the contrary the DNN model performs better when trained only with 5 independent features, namely: video frame rate, quantization, noise reduction, video packet loss rate, and audio packet loss rate. Also, the one-hidden layer DNN model outperforms the model with 20 hidden layers in terms of RMSE (0.403 vs. 0.437) and PCC (0.909 vs. 0.894). The conducted experiments also show that all models perform quite well and that the RF model with complete set of features performs the best (lowest RMSE 0.340 and highest PCC 0.930). The video packet loss rate seems to be the most influential feature on the RF model. The model is further trained on other publicly available audiovisual datasets and still performs well. However it is not compared to the other models, which would be useful to confirm or infirm the supremacy of RF.

9.2 QoE prediction under QoS impairments

In [453], Vega et al. propose an unsupervised deep learning model based on Restricted Boltzmann Machines (RBMs) for real-time quality assessment of video streaming services. More precisely, the model is intended to infer the no-reference features of the received video from only a subset of those features that the client extracts in real-time fashion. 10 video-related features are extracted: one related to the bit stream, five to the frame, two to the inter-frame and the last two to the content. Network QoS parameters are not considered in the feature set, however the impact of the network conditions is studied in

the paper based on two synthetic network-impaired video datasets, namely ReTRiEVED (for general condition networks) and LIMP (for extremely lossy networks). It is observed that the PCC between the VQM of the received video and the *bit rate* feature is the highest amongst the ten features, under network delay, jitter and throughput impairments. However, it is the *blur ratio* that correlates the most with VQM under severe packet loss condition. A discrepancy between video types was also recorded. This eventually motivated the need for one RBM model (with different feature set) per video type and network impairment, which raised the number of devised models to 32. Video-type and network-condition specific RBMs (with 100 hidden neurons) eventually shows a better performance than the single video-type and network-condition oblivious model on the ReTRiEVED dataset, according to the authors, which contradicts the results shown on the tables. Assuming that there is improvement, the practicality and overhead of the multi-RBMs solution are yet to be evaluated. In fact, delay, jitter, and throughput impairments are treated as if they were independent conditions and a condition-specific model is created. In practice, however impairments are correlated and happen together. Therefore, if the client has to assess the quality of the streamed video, it will also have to find out what impairment there is prior to selecting the appropriate predictor.

9.3 QoS/QoE prediction for HAS and DASH

Recently, the widespread adoption of HTTP adaptive streaming (HAS) drove increasing interest in developing QoE/QoS-aware HAS clients. Data-driven approaches, in particular ML, have been employed mainly in two different ways: (1) to predict changes in network QoS, namely throughput, and trigger adaptation mechanism to reduce rebuffering time [432], and (2) to select appropriate adaptation action [102].

It has been shown in recent work [432] that accurate throughput prediction can significantly improve the QoE for adaptive video streaming. ML has been widely used in throughput prediction in general as shown in Section 3. In the particular context of adaptive video streaming, Sun et al. propose in [432] the Cross Session Stateful Predictor (CS2P), a throughput prediction system to help with bitrate selection and adaptation in HAS clients. CS2P uses HMM for modeling the state-transition evolution of throughput, one model per session *cluster*, where sessions are clustered according to common features (e.g. ISP, region). The system is testing with a video provider (iQIYI) dataset consisting of 20 million sessions covering 3 million unique client IPs, 18 server IPs, and 87 ISPs. The HMM model is trained offline via the expectation maximization algorithm, and 4-fold cross-validation is used for tuning the number of states (6 states in total). Online prediction provides an estimate of the throughput

1 ~ 10 epochs ahead using maximum likelihood estimation. Throughput is continuously monitored and the model is updated online accordingly. Midstream throughput prediction experiments show that the model achieves 7% median absolute normalized prediction error (~ 20% 75th-percentile error) reducing the median prediction error by up to 50% compared to history-based predictors (last sample, harmonic mean, AR) as well as other ML-based predictors (SVR, GBR, and HMM trained on all sessions as opposed to the session cluster). It is also shown that CS2P achieves 3.2% improvement on overall QoE and 10.9% higher average bitrate over state-of-art Model Predictive Control (MPC) approach, which uses harmonic mean for throughput prediction. The authors claim that SVR and GBR perform poorly when trained on the session cluster. This might be due to the smaller size of the session cluster dataset, but requires further investigation.

In [102] (that extends [103] - [357]), a Q-learning-based HAS client is proposed to dynamically adjust to the current network conditions, while optimizing the QoE. Adaptation is assumed at the segment level; the quality level (e.g. bitrate) of the next video segment may go higher or lower depending on network conditions. States are defined as a combination of the client buffer filling level and throughput level. $B_{max}/T_{seg} + 1$ different buffer filling levels are considered where B_{max} denotes the maximum client buffer size in seconds, and T_{seg} the segment duration in seconds. Whereas $N + 1$ throughput levels are considered, ranging between 0 and the client link capacity, where N is the number of quality levels. The reward function to be maximized is a measure of the QoE, calculated on the basis of the targeted segment quality level, the span between the current and targeted quality level, and the rebuffering level (which may result in video freezes).

The model is trained and tested on $NS - 3$ with 10-min different video sequences (6 in total), split into 2sec segments each encoded at $N = 7$ different bitrates. The algorithm is trained for 400 episodes of streaming each of the video sequences over a publicly available 3G network bandwidth trace [384, 385]. The authors claim that the Q-learning client achieves in average 9.12% higher estimated MOS (program-generated), with 16.65% lower standard deviation, than the traditional Microsoft ISS Smooth Streaming (MSS) client. Similar performance is recorded when alternating between 2 video sequences every 100 streaming episodes. However, shifting to a random new video sequence after convergence time was not investigated.

9.4 Summary

Research in QoS/QoE provisioning has been leveraging ML for both prediction and adaptation, as shown in Tables 19 and 20. Clearly, research has been dominated by works on predicting QoE based on video-level and

network-level features. As such, a number of different QoS/QoE correlation models have been proposed in the literature for different media types (e.g. voice, video and image) ranging from simple regression models to NNs, including SVM, DT, RF, etc. For each media type, different QoE assessment methods and metrics have been used (e.g. MOS, VQM), each with its own set of computational and operational requirements. The lack of a common, standard QoE measure makes it difficult to compare the efficiency of different QoS/QoE prediction and correlation models. In addition, there is a lack of a clear quantitative description of the impact of network QoS on QoE. This impact is poorly understood and varies from one scenario to another. While some find it sufficient to correlate the QoE to a single network QoS parameter [145, 240, 383, 408], e.g. delay or throughput, others argue that multiple QoS parameters impact the QoE and need to be considered in tandem as features in a QoE/QoS correlation model [89, 102, 114, 235, 287, 328, 432]. Still others consider QoS as a confounding parameter and build different QoE assessment models for different network QoS conditions [453].

This motivates the need for an efficient methodology for QoE/QoS correlation, based on a combination of quantifiable subjective and objective QoS measures and outcomes of service usage. This calls for the identification of the influential factors of QoE for a given type of service and understanding their impact on user's expectation and satisfaction. QoE measures, such as MOS, and user engagement metrics are very sensitive to contextual factors. Though, contextual information undoubtedly influences QoE and is necessary to develop relevant QoE optimization strategies, it can raise privacy concerns.

Results depicted in Table 19 show that supervised learning techniques, such as NNs, SVR, DT and RF have consistent low MOS prediction errors. According to [89], RF is a better classifier than NN when it comes to predicting MOS. Table 20 also shows that using ML in HAS and DASH for prediction and adaptation, using supervised learning and RL, can improve QoE. However, this still needs to be validated in a real-world testbed.

10 Network security

Network security consists of protecting the network against cyber-threats that may compromise the network's availability, or yield unauthorized access or misuse of network-accessible resources. Undoubtedly, businesses are constantly under security threats [231], which not only costs billions of dollars in damage and recovery [227], but could also have a detrimental impact to their reputation. Therefore, network security is a fundamental cornerstone in network operations and management.

It is undeniable that we are now facing a cyber arms race, where attackers are constantly finding clever ways

to attack networks, while security experts are developing new measures to shield the network from known attacks, and most importantly zero-day attacks. Examples of such security measures include:

- *Encryption of network traffic*, especially the payload, to protect the integrity and confidentiality of the data in the packets traversing the network.
- *Authorization using credentials*, to restrict access to authorized personnel only.
- *Access control*, for instance, using security policies to grant different access rights and privileges to different users based on their roles and authorities.
- *Anti-viruses*, to protect end-systems against malwares, e.g. Trojan horse, ransom-ware, etc.
- *Firewalls*, hardware or software-based, to allow or block network traffic based on pre-defined set of rules.

However, encryption keys and login credentials can be breached, exposing the network to all kinds of threats. Furthermore, the prevention capabilities of firewalls and anti-viruses are limited by the prescribed set of rules and patches. Hence, it is imperative to include a second line of defense that can detect early symptoms of cyber-threats and react quickly enough before any damage is done. Such systems are commonly referred to as Intrusion Detection/Prevention Systems (IDS/IPS). IDSs monitor the network for signs of malicious activities and can be broadly classified into two categories—Misuse- and Anomaly-based systems. While the former rely on signatures of known attacks, the latter is based on the notion that intrusions exhibit a behavior that is quite distinctive from normal network behavior. Hence, the general objective of anomaly-based IDSs is to define the “normal behavior” in order to detect deviations from this norm.

When it comes to the application of ML for network security, through our literature survey we have found that the majority of works have focused on the application of ML for intrusion detection. Here, intrusion detection refers to detecting any form of attacks that may compromise the network e.g. probing, phishing, DoS, DDoS, etc. This can be seen as a classification problem. While there is a body of work on host-based intrusion detection (e.g. malware and botnet detection), we do not delve into this topic, as most of these works utilize traces collected from the end-host (sometimes in correlation with network traces). Concretely, in our discussion, we focus on network-based intrusion detection and we classify the works into three categories, namely misuse, anomaly, and hybrid network IDSs.

Previous surveys [82, 161, 447] looked at the application of ML for cyber-security. However, [161, 447] cover the literature between 2000-2008, leaving out a decade of work. More recently, [82] looked at the application of Data Mining and ML for cyber-security intrusion detection.

The proposed taxonomy consists of the different ML techniques with a sample of efforts that apply the corresponding technique. Our discussion is different, as we focus on ML-based approaches with a quantitative analysis of existing works (Tables 21, 22 23, 24 and 25). Furthermore, we survey efforts related to SDN and reinforcement learning, which have been recently published.

10.1 Misuse-based intrusion detection

Misuse-based IDSs consist of monitoring the network and matching the network activities against the expected behavior of an attack. The key component of such a system is the comprehensiveness of the attack signatures. Typically, the signatures fed to a misuse-IDS rely on expert knowledge [84]. The source of this knowledge can either be human experts, or it can be extracted from data. However, the huge volume of generated network traces renders manual inspection practically impossible. Furthermore, attack signatures extracted by sequentially scanning network traces will fail to capture advanced persistent threats or complex attacks with intermittent symptoms. Intruders can easily evade detection if the signatures rely on a stream of suspicious activities by simply inserting noise in the data.

In light of the above, ML became the tool of choice for misuse-based IDSs. Its ability to find patterns in big datasets, fits the need to learn signatures of attacks from collected network traces. Hence, it comes as no surprise to see a fair amount of literature [20, 84, 90, 252, 322, 344, 354, 402, 421] that rely on ML for misuse-detection. These efforts are summarized in Table 21. Naturally, all existing works employ supervised learning, and the majority perform the detection offline. Note, we classify all work that use normal and attack data in their training set as misuse-detection.

The earliest work that employed ML for misuse detection is [84]. It was among the first to highlight the limitations of rule-based expert systems, namely that they (i) fail to detect variants of known attacks, (ii) require constant updating, and (iii) fail to correlate between multiple individual instances of suspicious activities if they occur in isolation. Following the success of NN in the detection of computer viruses, the application of NN for misuse detection as an alternative to rule-based systems is proposed. The advantages of NN are its ability to analyze network traces in a less structured-manner (as opposed to rule-based systems), and to provide prediction in the form of a probability. The latter can enable the detection of variants of known attacks. For evaluation, training and testing dataset are generated using *RealSecure*TM—a tool that monitors network data and compares it against signatures of known attacks. For attack dataset, *InternetScanner*TM [368] and Satan Scanner [143] tools are used to generate port scans and syn-flood attacks on the monitored

Table 21 Summary of ML-based Misuse Detection

Ref.	ML Technique	Dataset	Features	Evaluation	
				Settings	Results
Cannady [84]	Supervised NN (offline)	Normal: RealSecure Attack: [143, 368]	TCP, IP, and ICMP header fields and payload	-1 Layer MLP: 9, 2 -Sigmoid function -Number of nodes in hidden layers determined by trial & error	DR: 89%-91% Training + Testing runtime: 26.13 hrs
Pfahringier [358]	Supervised Ensemble of C5 DTs (offline)	KDD Cup [257]	all 41 features	-Two-processor (2x300Mhz) -512M memory, 9 GB disc Solaris OS 5.6 -10-folds cross-validation	DR Normal: 99.5% DR Probe: 83.3% DR DoS: 97.1% DR U2R: 13.2% DR R2L: 8.4% Training: 24 h
Pan et al. [344]	Supervised NN and C4.5 DT (offline)	KDD Cup [257]	all 41 features	-29,313 training data records -111,858 testing data records -1 Layer MLP: 70-14-6 -NN trained until MSE = 0.001 or # Epochs = 1500 -Selected attacks for U2L and R2L -After-the-event analysis	DR Normal : 99.5% DR DoS: 97.3% DR Probe (Satan): 95.3% DR Probe (Portsweep): 94.9% DR U2R: 72.7% DR R2L: 100% ADR: 93.28% FP: 0.2%
Moradi et al. [322]	Supervised NN (offline)	KDD Cup [257]	35 features	-12,159 training data records -900 validation data records -6,996 testing data records -Attacks: SYN Flood and Satan -2 Layers MLP: 35 35 35 3 -1 Layer MLP: 35 45 35 -ESVM Method	2 Layers MLP DR: 80% 2 Layers MLP Training time > 25 hrs 2 Layers MLP w/ ESVM DR: 90% 2 Layers MLP w/ ESVM Training time < 5 hrs 1 Layers MLP w/ ESVM DR: 87%
Chebrolu et al. [90]	Supervised BN and CART (offline)	KDD Cup [257]	Feature Selection using Markov Blanket and Gini rule	-5,092 training data records -6,890 testing data records - AMD Athlon 1.67 GHz processor with 992 MB of RAM	DR Normal: 100% DR Probe: 100% DR DoS: 100% DR U2R: 84% DR R2L: 99.47% Training BN time: 11.03 ~ 25.19 sec Testing BN time: 5.01 ~ 12.13 sec Training CART time : 0.59 ~ 1.15 sec Testing CART time: 0.02~ 0.13 sec
Amor et al. [20]	Supervised NB (offline)	KDD Cup [257]	all 41 features	-494,019 training data records -311,029 testing data records -Pentium III 700 Mhz processor	DR Normal: 97.68% PCC DoS: 96.65% PCC R2L: 8.66% PCC U2R: 11.84% PCC Probing: 88.33%
Stein et al. [421]	Supervised C4.5 DT (offline)	KDD Cup [257]	GA-based feature selection	-489,843 training data records -311,029 testing data records -10-fold cross validation -GA ran for 100 generations	Error rate DoS: 2.22% Error rate Probe: 1.67% Error rate R2L: 19.9% Error rate U2R: 0.1%
Paddabachigari et al. [354]	Supervised Ensemble of SVM, DT, and SVM-DT Offline	KDD Cup [257]	all 41 features	5,092 training data records 6,890 testing data records AMD Athlon, 1.67 GHz processor with 992 MB of RAM -Polynomial kernel	DR Normal: 99.7% DR Probe: 100% DR DoS: 99.92% DR U2R: 68% DR R2L: 97.16% Training time: 1~ 19 sec Testing time: 0.03~ 2.11 sec
Sangkatsanee et al. [402]	Supervised C4.5 DT (online)	Normal: Reliability Lab Data 2009 (RLD09) Attack: [341, 444, 475]	TCP, UDP, and ICMP header fields	-55,000 training data records -102,959 testing data records -12 features -2.83 GHz Intel Pentium Core2 Quad 9550 processor with 4 GB RAM and 100 Mbps LAN -Platform used: Weka V.3.6.0	DR Normal: 99.43% DR DoS: 99.17% DR Probe: 98.73% Detection speed: 2~ 3 sec

Table 21 Summary of ML-based Misuse Detection (*Continued*)

Ref.	ML Technique	Dataset	Features	Evaluation	
				Settings	Results
Miller et al. [314]	Supervised Ensemble MPML (<i>Offline</i>)	NSL-KDD [438]	all 41 features	-125,973 training records -22,544 testing records -3 NBs trained w/ 12, 9, 9 features -Platform used Weka [288]	TP: 84.137% FP: 15.863%
Li et al. [272]	Supervised TCM K-NN (<i>Offline</i>)	KDD Cup [257]	all 41 features 8 features selected using Chi-square	-Intel Pentium 4, 1.73 GHz, 1 GB RAM, Windows XP Professional - Platform Weka [288] -49,402 training records -12,350 testing records -K = 50	41 features: TP 99.7% 41 features: FP 0% 8 features: TP 99.6% 8 features: FP 0.1%

^aDetermined empirically, Mean Square Error (MSE), Percentage Correct Classification (PCC), Average Detection Rate (ADR), Early Stop Validation Method (ESVM)

host. Results show that the NN is able to correctly identify normal and attack records 89–91% of the time.

In 1999, the KDD cup was launched in conjunction with the KDD'99 conference. The objective of the contest was the design of a classifier that is capable of distinguishing between normal and attack connections in a network. A dataset was publicly provided for this contest [257], and since then became the primary dataset used in ML-based intrusion detection literature. It consists of 5 categories of attacks, including DoS, probing, user-to-root (U2R) and root-to-local (R2L), in addition to normal connections. The top three contestants employed DT-based solutions [421]. The winner of the contest [358] used an ensemble of 50 times 10 C5 DTs with a mixture of bagging and boosting [377]. The results of the proposed method are presented in Table 21. Clearly, the proposed approach performs poorly for U2R and R2L attack categories. The authors do mention that many of the decisions were pragmatic and encouraged more scientific endeavors. Subsequently, an extensive body of literature emerged for ML-based intrusion detection using the KDD'99 dataset, in efforts to improve on these results, where some use the winners' results as a benchmark.

For instance, Moradi et al. [322] investigate the application of NN for multi-class classification using the KDD'99 dataset. Specifically, the authors focused on DoS and probing attacks. As opposed to the work of [84], two NNs were trained: one with a single hidden layer and the second with two hidden layers, to increase the precision of attack classification. They leverage the *Early Stopping Validation Method* [366] to reduce training and validation time of the NN to less than 5 hours. As expected, the NN with 2 hidden layers achieves a higher accuracy of 91%, compared to the 87% accuracy of the NN with a single hidden layer.

Amor et al. [20] compare NB and DT also using KDD'99 dataset, and promote NB's linear training and

classification times as a competitive alternative to DT. NB is found to be 7 times faster in learning and classification than DT. For whole attacks, DT shows a slightly higher accuracy over NB. However, NB achieves better accuracy for DoS, R2L, and probing attacks. Both NB and DT perform poorly for R2L and U2R attacks. In fact, Sabhnani and Serpen [398] expose that no classifiers can be trained successfully on the KDD dataset to perform misuse detection for U2R or R2L attack categories. This is due to the deficiencies and limitations of the KDD dataset rather than the inadequacies of the proposed algorithms.

The authors found via multiple analysis techniques that the training and testing datasets represent dissimilar hypothesis for the U2R and R2L attack categories; so if one would employ any algorithm that attempts to learn the signature of these attacks using the training dataset is bound to perform poorly on the testing dataset. Yet, the work in [344] reports surprisingly impressive detection accuracy for U2R and R2L. Here, a hybrid of BP NN with C4.5 is proposed, where BP NN is used to detect DoS and probing attacks, and C4.5 for U2R and R2L. For U2R and R2L only a subcategory of attacks is considered (yielding a total of 11 U2R connections out of more than 200 in the original dataset and ~2000 out of more than 15000 for R2L connections). *After-the-event* analysis is also performed to feed C4.5 with new rules in the event of misclassification.

Other seminal works consider hybrid and ensemble methods for misuse detection [90, 354, 421]. The goal of ensemble methods is to integrate different ML techniques to leverage their benefits and overcome their individual limitations. When applied to misuse detection, and more specifically for the KDD'99 dataset, these work focused on looking at which ML technique works best for a class of connections. For instance, Peddabachigari et al. [354] propose an IDS that leverages an ensemble of DT, SVM with polynomial kernel based function, and hybrid DT-SVM to detect various different cases of misuse. Through

Table 22 Summary of ML for flow feature-based anomaly detection

Ref.	ML Technique	Dataset	Features	Evaluation	
				Settings	Results
Kayacik et al. [232]	Unsupervised Hierarchical SOM (Offline)	KDD Cup [257]	6 TCP features	-494,021 training records -311,029 records in test set 1 -4,898,431 records in test set 2 -Platforms: SOM-Toolbox [12] & SOM PAK [250] -3-level SOM w/ # Epochs: 4000	DR Test-set 1: 89% FP Test-set 1: 4.6% DR Test-set 2: 99.7% FP Test-set 2: 1.7%
Kim et al. [242]	Supervised SVM (Offline)	KDD Cup [257]	selected using GA	Training set: kddcup.data.gz [257] Testing set: corrected.gz [257] -Detect only DoS attacks -10-fold cross validation -GA ran for 20 generations	DR w/ Neural Kernel: 99% DR w/ Radial Kernel: 87% DR w/ Inverse Multi-Quadratic Kernel: 77%
Jiang et al. [220]	Unsupervised Improved NN (Offline)	KDD Cup [256, 257]	all 41 features	-40,459 training records -429,742 testing records -Cluster Radius Thresh $r=[0.2-0.27]$	DR DoS: 99.10% 99.15 DR Probe: 64.72% 80.27% DR U2R: 25.49% 60.78% DR R2L: 6.34% 8.67% DR new attacks: 32.44% 42.12% FP: 0.05% 1.30%
Zhang et al. [495]	Unsupervised Random Forests (Offline)	KDD Cup [257]	40 features labeled by service type	-4 datasets used with % of attack connections: 1%, 2%, 5%, 10% -Platform used: Weka [288]	1% attacks: FP: 1% DR: 95% 10% attacks: FP: 1% DR: 80%
Ahmed et al. [7]	Supervised Kernel Function (Online)	From Abilene backbone network	number of packets, number of individual IP flows	-2 timeseries binned at 5 min intervals -Timeseries dimensions = $F \times T$ - $F = 121$ flows, $T = 2016$ timesteps	T#1 DR: 21/34-30/34 FP: 0-19 T#2 DR: 28/44-39/44 FP: 5-16
Shon et al. [411]	Unsupervised Soft-margin SVM and OCSVM (Offline)	KDD Cup [257] Data collected from Dalhousie U.	selected using GA	-SVM Toolkits [88, 396] -100,000 packets for training -1,000-1,500 packet for testing -GA run for 100 generations 3-cross fold validation	KDD w/ 9 attack types DR: 74.4% Dalhousie Dataset DR: 99.99% KDD w/ 9 attack types FN: 31.3% Dalhousie Dataset FP: 0.01%
Giacinto et al. [165]	Unsupervised Multiple Classifiers (Offline)	KDD Cup [257]	29 features for HTTP 34 features for FTP 16 features for ICMP 31 features for Mail 37 features for Misc 29 features for Private&Other	-494,020 training records -311,029 testing records -1.5% of data records is attacks	v-SVC DR: 67.31%-94.25% v-SVC FP: 0.91%-9.62%
Hu et al. [198]	Supervised Decision stumps with AdaBoost (Offline)	KDD Cup [257]	all 41 features	-494,021 training records -311,029 testing records -Pentium IV with 2.6-GHz CPU and 256-MB RAM -Platform used Matlab 7	DR: 90.04%-90.88% FP: 0.31%-1.79% Mean Training time: 73 sec
Muniyandi et al. [327]	Unsupervised K-Means, C4.5 DT (Offline)	KDD Cup [257]	all 41 features	-15,000 training records -2,500 testing records -Intel Pentium Core 2 Duo CPU 2.20GHz, 2.19GHz, 0.99GB of RAM w/ Microsoft Windows XP (SP2) -Platform: Weka 3.5 [288]	DR: 99.6% FP: 0.1% Precision: 95.6% Accuracy: 95.8% F-measure: 94.0%
Panda et al. [345]	Unsupervised RF, ND, END (Offline)	NSL-KDD [438]	all 41 features	-25,192 training instances -IBM PC of 2.66GHz CPU with 40GB HDD and 512 MB RAM -10-fold cross validation	TP: 99.5 FP: 0.1% F-measure: 99.7% Precision: 99.9% Recall: 99.9% Time to build model: 18.13 sec
Boero et al. [64]	Supervised RBF-SVM (Offline)	Normal: from U. of Genoa Malwares: [126, 292, 348, 351]	7 SDN OpenFlow features	-RBF Complexity par: 20 -RBF kernel par: 2	Normal-TP: 86% Normal-FP: 1.6% Malware-TP: 98.4% Malware-FP: 13.8%

empirical evaluation, the resultant IDS consist of using DT for U2R, SVM for DoS, and and DT-SVM to detect normal traffic. The ensemble of the 3 methods together (with

a voting mechanism) is used to detect probing and R2L attacks. The resultant accuracy for each class is presented in Table 21.

Table 23 Summary of ML for payload-based anomaly detection

Ref.	ML Technique	Dataset	Features	Evaluation	
				Settings	Results
Zanero et al. [493]	Unsupervised A two-tier SOM-based architecture (Offline)	Normal: KDD Cup [257] Attack: Scans from Nessus [44]	Packet headers and payload	-2,000 training packets -2,000 testing packets -10x10 SOM trained for 10,000 epochs -Platform used: SOM toolbox [12]	Improves DR by 75% over 1-tiered S.O.M
Wang et al. [459]	Unsupervised Centroid model (Offline)	KDD Cup [257] & CUCS	Payload of TCP traffic	-2 weeks training data -3 weeks testing data -Inside network TCP data only -Incremental learning	DR w/ payload of a packet: 58.8% DR w/ first 100 bytes of a packet: 56.7% DR w/ last 100 bytes of a packet: 47.4% DR w/ all payloads of a con: 56.7% DR w/ first 1000 bytes of a Con: 52.6% Training time: 4.6-26.2 sec Testing time: 1.6-16.1 sec
Perdisci et al. [356]	Supervised Ensemble of single-class SVM (Offline)	Normal: KDD Cup [257] Normal: GATECH Attack: CLET [117] Attack: PBA [149] Generic [204]	Payload	-50% of dataset for training -50% of dataset for testing -11 OCSVM trained with 2 _v -grams; v=1...10 -5-fold cross validation on KDD cup -7-fold cross validation on GATECH -2 GHz Dual Core AMD Opteron Processor and 8GB RAM	Generic DR w/ FP 10 ⁻⁵ : 60% shell-code DR w/ FP 10 ⁻⁵ : 90% CLET DR w/ FP 10 ⁻⁵ : 90% Detection time KDD Cup: 10.92 ms Detection time GATECH: 17.11 ms
Gornitz et al. [171]	Supervised SVDD (Online)	Normal: from Fraunhofer Inst. Attack: Metasploit	payload	-2,500 training network events -1,250 testing network events -Active Learning -Fraction of Labeled data: 1.5%	DR: 96% FP: 0.0015%

Stein et al. [421] employ DT with GA. The goal of GA is to pick the best feature set out of the 41 features provided in KDD'99 dataset. DT with GA is performed for every category of attacks, rendering a total of 4 DTs. The average error rate achieved by each DT at the end of 20 runs is reported in Table 21. Another interesting ensemble learning approach is the one proposed in [90], where the ensemble is composed of pairs of feature set and classification technique. More specifically, BN and CART classification techniques are evaluated on the KDD'99 dataset with different feature sets. Markov blanket [353] and Gini [76] are adopted as feature selection techniques for BN and CART, respectively. Markov blanket identifies the only knowledge needed to predict the behavior of a particular node; a node here refers to the different categories of attacks. Gini coefficient measures how well the splitting rules in CART separates between the different categories of attacks. This is achieved by pruning away branches with high classification error. For BN, 17 features out of 41 are chosen during the data reduction phase. For CART, 12 variables are selected. CART and BN are trained on the 12 and 17 features set, as well as 19 features set from [326]. They describe the final ensemble method using pairs (#features, classification), which delineates the reduced feature set and the classification technique that

exhibits the highest accuracy for the different categories of attacks and normal traffic. The ensemble model achieves 100% accuracy for normal (12 features set, CART), probe (17 features set, CART), and DoS (17 features set, Ensemble), and 84% accuracy for U2R (19 features set, CART), and 99.47% accuracy for R2L (12 features set, Ensemble).

Miller et al. [314] also devise an ensemble method but based on NB classifiers, denoted as Multi-perspective Machine Learning (MPML). The key idea behind MPML is that an attack can be detected by looking at different network characteristics or "perspective". These characteristics in turn are represented by a subset of network features. Hence, they group the features of a perspective together, and train a classifier using each feature set. The intuition behind this approach is to consider a diverse and rich set of network characteristics (each represented by a classifier), to enhance the overall prediction accuracy. The predictions made by each classifier are then fed to another NB model to reach a consensus.

A limitation of the aforementioned approaches is that they are all employed offline, which inhibits their application in real life. A few related works focused on the training and detection times of their IDS. Most classifiers (e.g., image, text recognition systems) require re-training from time to time. However, for IDSs this retraining may

Table 24 Summary of deep and reinforcement learning for intrusion detection

Ref.	ML Technique	Dataset	Features	Evaluation	
				Settings	Results
Cannady et al. [85]	RL CMAC-NN (Online)	Prototype Appli- cation	Patterns of Ping Flood and UDP Packet Storm attacks	-3 Layers NN -Prototype developed w/ C & Matlab	Learning Error: $2.199 \cdot 10^{-7}$ % New Attack Error: $2.199 \cdot 10^{-14}$ % Recollection Error: $0.038 \cdot 10^{-5}$ % Error after Refinement: 1.24%
Servin et al. [407]	RL Q-Learning (Online)	Generated using NS-2	Congestion, Delay, and Flow-based	-Number of Agents: 7 -DDoS attacks only -Boltzmann's rules for E2	FP: 0-10% Accuracy: ~70%~99% Recall: ~30%~99%
Li et al. [273]	DL DBN w/ Auto-Encoder (Offline)	KDD Cup [257]	all 41 features	-494,021 training records -311,029 testing records -Intel Core Duo CPU 2.10 GHz and 2GB RAM -Platform used: Matlab v.7.11 -3 Layers Encoder: 41,300,150,75,*	TPR: 92.20%-96.79% FPR: 1.58%-15.79% Accuracy: 88.95%-92.10% Training time: [1.147-2.625] sec
Alom et al. [14]	DL DBN (Offline)	NSL-KDD [438]	39 features	-25,000 training & testing records	DR w/ 40% data for training: 97.45% Training time w/ 40% data for training: 0.32 sec
Tang et al. [436]	DL DNN (Offline)	NSL-KDD [438]	6 basic features	-125,975 training records -22,554 testing records -3-Layers DNN: 6,12,6,3,2 -Batch Size: 10 # Epochs: 100 -Best Learning Rate: 0.001	Accuracy: 72.0%-75.75% Precision: 79%-83% Recall: 72%-76% F-measure: 72%-75%
Kim et al. [245]	DL LSTM-RNN (Offline)	KDD Cup [257]	all 41 features	-1,930 training data records -10 test datasets of 5000 records -Intel Core I7 3.60 GHZ, RAM 8GB, OS Ubuntu 14.04 -# Nodes in Input Layer: 41 -# Nodes in Output Layer: 5 -Batch Size: 50 #Epoch: 500 -Best Learning Rate: 0.01	DR: 98.88% FP: 10.04% Accuracy: 96.93%
Javaid et al. [213]	DL Self-taught Learning (Offline)	NSL-KDD [438]	all 41 features	-125,973 training records -22,544 testing records -10-fold cross validation	2-class TP: 88.39% 2-class Precision: 85.44% 2-class Recall: 95.95% 2-class F-measure: 90.4%

Table 25 Summary of ML for Hybrid Intrusion Detection

Ref.	ML Technique	Dataset	Features	Evaluation	
				Settings	Results
Mukkamala et al. [325]	Supervised RBF-SVM (Online)	KDD cup [257]	all 41 features	7,312 training records -6,980 testing records -Platform used: SVMLight [224]	Accuracy: 99.5% Training time: 17.77 sec Testing Time: 1.63 sec
Zhang et al. [494]	Hybrid Hierarchical-RBF (Online)	KDD Cup	all 41 features	-32,000 training records -32,000 testing records	SHIDS Normal DR: ~99.5% SHIDS Normal FP: 1.2% SHIDS Attack DR: [98.2%-99.3%] SHIDS Attack FP: [0%-5.4%] PHIDS level 1 DR: 99.8% PHIDS level 1 DR: 1.2% PHIDS level 2 DR: [98.8%-99.7%] PHIDS level 2 FP: [0%-4%] PHIDS level 3 DR: 86.9% PHIDS level 3 FP: 0% Training time: 5 min
Depren et al. [116]	Hybrid SOM w/ J.48 (Offline)	KDD Cup	6 basic features for SOM all 41 features for J.48	-10-fold cross validation -Two-phases SOM Training -Phase 1 learning rate: 0.6 -Phase 2 learning rate: 0.05 -Confidence Val. for J.48 pruning: 25%	DR: 99.9% Missed Rate: 0.1% FP: 1.25%

be performed daily (or even hourly) due to the fast and ever changing nature of cyber-threats [180]. Hence, fast training times are critical for an adaptable and robust IDS. [198] tackled the challenge of devising an IDS with fast training time using an Adaboost algorithm. The proposed algorithm consists of an ensemble of weak classifiers (decision stumps), where their decisions are then fed to a strong classifier to make the final decision. The fast training time achieved (of 73 s) is attributed to the use of weak classifiers. Another advantage of decision stumps is the ability to combine weak classifiers for categorical features with weak classifiers for continuous features, without any forced conversation as is typically done in most works. During the evaluation, a subset of attack types are omitted from the training set in order to evaluate the algorithm's ability to detect unknown attacks. While the reported accuracy is not significantly high (90%), the training time is promising for real-time deployment. Clearly, there is still a need for a model that can achieve fast training time, without sacrificing the detection accuracy.

Sangkatsanee et al. [402] propose a real-time misuse-based IDS. Information gain is applied to reduce the number of features used (for faster detection), resulting in 12 features. Different ML techniques were assessed, among which DT provided the best empirical results. They developed a tool that runs on traces collected in 2 s time intervals, and shows a detection accuracy of 98%. A post-processing technique is also proposed to reduce FP, which consists of flagging an attack only if 3-out-of-5 consecutive records belonging to the same connection were classified as an attack. While this work is indeed promising, given it is performed in real-time, it suffers from a few limitations: (i) it can only detect two types of attacks (DoS and probe), (ii) it is not compared against other real-time signature-based IDS (e.g. Snort [87]), (iii) it only looks at attacks in windows of 2 s, and (iv) its post-processing approach correlates records between 2 IPs, making it vulnerable to persistent threats and distributed attacks.

A final effort that merits a discussion here is [272]. This work employs Transductive Confidence Machine for k -NN (TCM-KNN), a supervised classification algorithm with a strangeness measure. A high strangeness measure indicates that the given instance is an outlier in a particular class (for which the measurement is being conducted). The strangeness measure is calculated for every instance against each possible classification class. This is achieved by measuring the ratio of the sum of the k -nearest distances from a given class to the sum of the k -nearest distances from all other classes. The strangeness measure is also employed for active learning. Since getting labeled data for attacks is a cumbersome task, active learning can relieve part of this tedious process by indicating the subset of data points that should be labeled to improve the confidence of the classifier. TCM-KNN is evaluated over the

KDD'99 dataset and the results are reported in Table 21. The benefits of active learning is also evaluated. Starting with a training set of just 12 instances, TCM-KNN requires the labeling of an additional 40 actively selected instances to reach a TP of 99.7%. Whereas, random sampling requires the labeling of 2000 instances to attain the same accuracy.

10.2 Anomaly-based intrusion detection

Though misuse-based IDSs are very successful at detecting known attacks, they fail to identify new ones. Network cyber-threats are constantly changing and evolving, making it crucial to identify "zero-day" attacks. This is where anomaly-based intrusion detection comes in. Anomaly-based IDS models normal network behavior, and identify anomalies as a deviation from the expected behavior. A big issue with anomaly-based IDSs is false alarms, since it is difficult to obtain a complete representation of normality. ML for anomaly detection has received significant attention, due to the autonomy and robustness it offers in learning and adapting profiles of normality as they change over time. With ML, the system can learn patterns of normal behavior across environments, applications, group of users, and time. In addition, it offers the ability to find complex correlations in the data that cannot be deduced from mere observation. Though anomaly detection can be broadly divided into flow feature or payload-based detection, recently, deep learning and reinforcement learning are being aptly exploited. Primarily, this is due to their intrinsic ability to extrapolate data from limited knowledge. We delineate and summarize the seminal and state-of-the-art ML-based techniques for anomaly detection in Tables 22, 23 and 24.

10.2.1 Flow feature-based anomaly detection

Flow-based anomaly detection techniques rely on learning the expected (benign) network activities from flow features. The immediate observation in contrast to misuse detection is the application of unsupervised learning and hybrid supervised/unsupervised learning. Some works employed supervised learning for anomaly detection as well. The main difference is instead of teaching the model the expected behavior, in unsupervised learning the model is fed with an unlabeled training set to find a structure, or a hidden pattern, in the data. In anomaly detection, the notion is that benign network behavior is more common and will naturally group together, whereas, anomalous behavior is more sparse and will appear as outliers in the dataset. Hence, the larger and more dense clusters will indicate normal connections, while the smaller more distant data points (or clusters of data points) will indicate malicious behavior. A quick glance at Tables 22, 23, and 24 will reveal that the KDD'99 dataset is the dataset of choice in most anomaly-based intrusion detection

literature, where some have also employed the improved version of the dataset, NSL-KDD [438] released in 2009. In the sequel, we elucidate the most influential work in the application of flow feature-based ML for anomaly detection.

We start-off our discussion by looking at supervised learning techniques. KOAD [7] is an online kernel function-based anomaly detection IDS. The key feature of KOAD is its ability to model normal behavior in face of variable traffic characteristics. It leverages a real-time anomaly detection algorithm that incrementally constructs and maintains a dictionary of input vectors defining the region of normal behavior. This dictionary is built using time series of the number of packets and IP flows. In the evaluation, the authors use a dataset collected by monitoring 11 core routers in the Abilene backbone network for a week. It comprises of two multi-variate time series, the number of packets and the number of individual IP flows. KOAD is evaluated against PCA and One-Class Neighbor Machine (OCNM). In packet time series, OCNM flags 26 out of 34 anomalies but generates 14 FPs, while KOAD gives different TP and FP under different parameters. For instance, it can detect 30 anomaly records with 17 FPs, and 26 anomaly records with 1 FP. However, PCA can detect 25 anomalies with 0 FP. On the other hand, for the flow-count time series, KOAD outperforms PCA and OCNM in terms of detection rate but at the cost of a higher FP.

More recently, Boero et al. [64] leverage a SVM with radial basis function kernel (RBF-SVM) to devise an IDS for SDN-based malware detection. A reduced feature set is evaluated based on features that are collectible via OF and commercial SDN switches. This limits the number of features to 7 consisting of the number of packets, number of bytes, flow duration, byte rate, packet rate, length of the first packet, and average packet length. The dataset used for evaluation consists of normal traffic traces from a university campus and malware traffic traces from [126, 292, 348, 351]. For a dataset with known attacks, both RBF-SVM with limited and all features return a TP above 98% for the malware traces, while TP of RBF-SVM is 86.2% for normal traces.

However, detecting new attacks using the RBF-SVM with limited and full features achieve comparable TP with a high FP of approximately 18% for normal traces. This shows that restricting the features set to those that can be collected via SDN switches slightly impacts the TP rate; however it comes at a cost of a higher FP. Hence there is a need to enlarge the features set that SDN switches monitor and collect. As we will see in the following, the battle between FP and TP will constantly resurface throughout our discussion. This is expected since guaranteeing the ground truth is difficult and requires manual labeling. Furthermore, obtaining a

complete representation of normal behavior is extremely challenging. Thus, any future legitimate behavior that was not part of the trained set might be flagged as an anomaly.

The main application of unsupervised learning for anomaly detection is clustering on the basis that normal data connections will create larger more dense clusters. Jiang et al. [220] challenge this notion by showcasing that the size of the cluster is not sufficient to detect anomalies and has to be coupled with the distance of the cluster from other clusters, to increase accuracy of detection. To this end, the authors propose an Improved Nearest Neighbor (IMM) technique for calculating cluster radius threshold. The KDD dataset is used for evaluation and shows that IMM outperforms three related works [131, 139, 363] in terms of detection rate and FP. A snippet of their reported results is presented in Table 22.

Kayacik et al. [232] leverage unsupervised NN with SOM and investigate their detection capabilities when trained with only 6 of the most basic TCP features, including protocol type, service type, status flag, connection duration, and total bytes sent to destination/source host. They evaluate their work on the KDD dataset, and observe that SOM-based anomaly detection achieves an average DR (ADR) of 89% with FP in the range of [1.7%-4.6%]. Other interesting applications of unsupervised learning for anomaly detection is RF [495] and an ensemble of single-class classifiers [165]. Giacinto et al. [165] train a single-class classifier, based on ν -SVC [405], for each individual protocol and network service; e.g. ICMP, HTTP, FTP, and Mail. This ensures that each classifier is specialized in detecting normal and abnormal characteristics for one these protocols and services. The application of one-class classifier is particularly interesting for cases where there is a skewness in the data. This is in-line with the fact that normal traffic traces are more common than malicious network activities. Thus, the one-class classifier learns the behavior of the dominant class, and dissimilar traffic patterns are then flagged as an anomaly. Results of the evaluation can be found in Table 22.

The majority of works in anomaly-based IDS employed a hybrid of supervised/unsupervised learning techniques. Panda et al. [345] evaluate several hybrid approaches to identify the best combination of supervised and unsupervised data filtering and base classifiers for detecting anomalies. The authors evaluate DT, PCA, stochastic primal estimated sub-gradient solver for SVM (SPegasos), ensembles of balanced nested dichotomies (END), Grading, and RF. They show that RF with nested dichotomies (ND) and END achieve the best results, with a detection rate of 99.5% and a FP of 0.1%. It is also the fastest in terms of performance, requiring 18.13 s to build and provides F-measure, precision, and recall of 99.7%, 99.9, and 99.9%, respectively. Enhanced SVM [411] combines a supervised version of SVM: soft-margin SVM with an unsupervised

version: one-class SVM. The intuition here is that this combination will allow to get the best of both worlds: low FP with ability to detect zero-day attacks. Enhanced SVM consists of 4 phases:

- Create a profile of normal packets using Self-organized Feature Map.
- Packet filtering scheme, using *pOf* [491], based on passive TCP/IP fingerprinting to reject incorrectly formed TPC/IP packets.
- GA to perform feature selection
- Temporal correlation of packets during packet processing

Enhanced-SVM is only trained with normal traffic. The normal to abnormal ratio in the data set consists of 98.5–99 to 1–1.5%. Compared to two commercial IDSs, Bro and Snort, the Enhanced-SVM slightly improves in anomaly detection accuracy on a real dataset with *unknown* traffic traces. However, for known attacks, Snort and Bro significantly outperform Enhanced-SVM. Wagner et al. [456] also leverage a hybrid supervised and unsupervised single-class SVM to detect anomalies in IP NetFlow records. A new kernel function is proposed to measure the similarity between two windows of IP flow records of n seconds. The hybrid SVM is evaluated on a normal dataset obtained from an ISP, with synthetically generated attacks using Flame [74], and with $n = 5$ s. Results show that the hybrid SVM can achieve an ADR of 92%, FP in the range [0–0.033], and TN in the range [0.967–1]. Finally, Muniyandi et al. [327] propose a hybrid anomaly detection mechanism that combines k -Means with C4.5 DT. They build k clusters using k -Means and employ DT for each cluster. DT overcomes the forced assignment problem in k -Means, where k is too small and a class dominates due to skewed dataset. The authors evaluate the hybrid detection on the KDD dataset and show that it outperforms k -Means, ID3, NB, k -NN, SVM, and TCM-KNN, over 6 different metrics, including TP, FP, precision, accuracy, F-measure, and ROC. However, TCM-KNN achieves better results in terms of TPR and FPR.

10.2.2 Payload-based anomaly detection

Payload-based anomaly detection systems learn patterns of normality from packet payload. This provides the ability to detect attacks injected inside the payload that can easily evade flow feature-based IDSs. In this subsection, we discuss ML techniques that have been employed to detect anomalies using packet payload alone or in conjunction with flow features.

PAYL [459] use the *1-gram* method to model packet payloads. n -gram is widely used for text analysis. It consists of a sliding window of size n that scans the payload while counting the occurrence/frequency of each n -gram. In addition to counting the frequency of each byte in the

payload, the mean and the standard deviation is computed. As the payload exhibits different characteristics for different services, PAYL generates a payload model for each service, port, direction of payload, and payload length range. Once the models are generated, Mahalanobis distance is used to measure the deviation between incoming packets and the payload models. The larger the distance, the higher the likelihood that the newly arrived packet is abnormal. The authors leverage incremental learning to keep the model up to date, by updating the Mahalanobis distance to include new information gathered from new packets. PAYL's ability to detect attacks on TCP connections is evaluated using the KDD dataset and data traces collected from Columbia University Computer Science (CUCS) web server. PAYL is able to detect 60% of the attacks on ports 21 and 80 with a FP of 1%. However, it performs poorly when the attacks target applications running on ports 23 and 25. This is due to the fact that attacks on ports 21 and 80 exhibit distinctive patterns in the format of the payload, making them easier to detect than attacks on ports 23 and 25. PAYL can be used as an unsupervised learning technique under the assumption that malicious payloads are a minority, and will have a large distance to the profile than the average normal samples. Hence, by running the learned model on the training set, malicious packets in the set can be detected, omitted, and then the models are retrained on the new training set.

Perdisci et al. [356] design Multiple-Classifer Payload-based Anomaly Detector (McPAD) to infer shell and polymorphic shell code attacks. Shell code attacks inject malicious executable code in the packet payload. As opposed to 1-gram analysis performed by PAYL, McPAD runs a 2_ν -gram analysis technique to model the payload ($\nu = [0 - 10]$). It measures the occurrence of a pair of bytes that are ν positions apart. By varying ν and applying feature reduction, different compact representations of the payload are obtained. Each of these representations is then fed to a 1-class classifier model and majority vote is used to make the final prediction. For evaluation, normal traffic is extracted from two datasets: the 1st week of KDD dataset and 7 weeks of HTTP traffic collected from College of Computing School at the Georgia Tech (GATECH). Attack traffic is collected from a generic dataset in [204], in addition to synthetically generated polymorphic attacks [117] and Polymorphic Blending Attacks (PBAs). In comparison to PAYL, McPAD achieves a DR of 60, 80 and 90% for generic, polymorphic CLET, and shell-code attacks, respectively, with an FP of 10^{-5} for all attacks. While, PAYL reports very low DRs for the same FP. However, the computational overhead of McPAD is much higher than that of PAYL with an average processing time of 10.92 ms over KDD and 17.11 ms over GATECH whereas PAYL runs in 0.039 ms and 0.032 ms, respectively.

Zanero et al. [493] propose a two-tier architecture for anomaly detection. The first tier consists of an unsupervised outliers detection algorithm that classifies each packet. This tier provides a form of feature reduction as the result of the classification “compresses” each packet into a single byte of information. The results from the first tier are fed into the second tier anomaly detection algorithm. In the first tier, both packet header and payload are used for outliers detection. The authors compare three different techniques, including SOM, Principal Direction Divisive Partitioning (PDDP) algorithm and k -Means, with SOM outperforming PDDP and k -Means in terms of classification accuracy with a reasonable computational cost. A preliminary prototype that combines a first tier SOM with a second tier SOM is evaluated over the Nessus [44] vulnerabilities scans. The results show a 75% improvement in DR over an IDS that does not include the first tier.

Gornitz et al. [171] leverage semi-supervised Support Vector Data Description (SVDD) and active learning to build the active SVDD (ActiveSVDD) model for payload-based anomaly detection. It is first trained with unlabeled examples, and subsequently refined by incorporating labeled data that has been queried by active learning rules. The empirical evaluation consists of comparing an unsupervised SVDD with random sampling against ActiveSVDD. The dataset used for the evaluation is HTTP traffic recorded within 10 days at Fraunhofer Institute. Attack data is generated using Metasploit [307] framework. In addition, mimicry attacks are added in the form of cloaked data to evaluate the ability to detect adversarial attacks. The model achieve high accuracy, with random sampling for online applications with cloaked data, 96% DR with a very low FP and 64% DR for ActiveSVDD and SVDD, respectively.

10.3 Deep and reinforcement learning for intrusion detection

As we contemplate the applications of ML for misuse and anomaly detection, we observe that all applications of NN were restricted to networks with at most 2 hidden-layers. DNNs are attractive for the ability to train large NNs with several hidden-layers. As we survey the literature on DL for intrusion detection, we will observe much larger and deeper NNs in terms of number of nodes in each layer, and the number of hidden layers. Conceptually, the results of DNNs get better with more data and larger models.

10.3.1 Deep learning for anomaly detection

Over the past decade, anomaly detection has particularly benefited from self-taught learning (STL) [213], DBN [14, 273], and RNN [245]. Once more, all these works have been evaluated using KDD dataset, and its enhanced

version NSL-KDD [438] dataset. Their results are summarized in Table 24.

In 2007, STL [378] emerged as an improvement over semi-supervised learning. STL uses unlabeled data from other, but relevant, object class to enhance a supervised classification task e.g. using random unlabeled images from the Internet to enhance the accuracy of a supervised classification task for cat images. This is achieved by learning a good feature representation from the unlabeled data and then applying this representation to the supervised classifier. The potential benefit of STL for anomaly detection is clear: intrusion detection suffers from the lack of sufficient amount of labeled data, more specifically for attacks. To this extent, the work in [213] explore the application of STL for anomaly detection. Their proposed model consists of two stages, an Unsupervised Feature Learning (UFL) stage using sparse auto encoder, followed by a classification stage that uses the learned features with soft-max regression (SMR). They evaluate their solution using the NSL-KDD Cup dataset for 2-class and 5-class classifications, and compare against a SMR technique that is not preceded by a UFL stage. The 2-class classification achieves a higher accuracy of 88.39% compared to 78.06% of SMR, and outperforms SMR with respect to recall and F-measure. However, SMR outperforms STL in precision.

Li et al. [273] and Alom et al. [14] explore the use of DBN for anomaly detection. DBN is an interesting class of NN, when trained using unlabeled data it works as a features selector, and when trained with labeled data it acts as a classifier. In [273] DBN is used to perform both of these two tasks. More specifically, an auto-encoder is first used for dimensionality reduction. The proposed DBN is composed of multi-layers of RBM and a layer of BP NN. Unsupervised training is performed on every layer of RBM and the final output is fed to the BP NN for classification. Pre-training and pre-tuning the DBN with auto-encoder over 10 iterations, result in an accuracy of 92.10%, FP of 1.58% and TP of 92.20%. DBN without auto-encoder achieves an accuracy, FP, and TP of 91.4, 9.02, and 95.34%, respectively.

In [14], the authors perform a compare analysis to evaluate the performance of DBN (composed of two-layers RBM) against against SVM, and a hybrid DBN-SVM. This comparative analysis was performed using the NSL-KDD dataset. The results show that DBN runs in 0.32 s, and achieves an accuracy of 97.5% when trained with only 40% of the NSL-KDD dataset, outperforming SVM and DBN-SVM. This exceeds the performance, with respect to training time of similar existing work [400]. In contrast, Tang et al. [436] use DNN for a flow-based anomaly detection in SDNs. They extract six features from the SDN switches and evaluate the accuracy of anomaly detection using the NSL-KDD Cup dataset. As the learning rate is varied, the DNN achieves an accuracy, precision, recall,

and F-measure in the range [72.05-75.75%], [79-83%], [72-76%], and [72-75%], respectively. It is important to note that for the highest learning rate, DNN achieves the highest accuracy on the training dataset. However, its accuracy, recall and F-measure on the test datasets drops. The authors note that such an accuracy drop occurs with a high learning rate since the model becomes trained “too accurately”, i.e. over-fitting. Nevertheless, the accuracy of the DNN is lower than the winner of the KDD Cup, RF, which has an accuracy of 81.59%.

10.3.2 Reinforcement learning for intrusion detection (RL)

MARL [407] is a Multi-Agent Reinforcement Learning system for the detection of DoS and DDoS attacks. MARL is based on Q-learning, and the system consists of a set of heterogeneous sensor agents (SA) and a hierarchy of decision agents (DA). In the proposed setup three SAs and 1 DA are used. Each SA is responsible of collecting either congestion, delay, or flow-based network metrics. These collected metrics represent the local state of every SA. Every SA runs a local RL mechanism to match its local state to a particular communication action-signal. These signals are received by the DA, which given a global view of the state of the network triggers a final action signal that is forwarded to a human in the loop. If the DA (or the higher-layer agent in case of a hierarchy of DAs) makes the appropriate call, all the agents in the system are rewarded. Otherwise, they will all be penalized. MARL is evaluated on NS-2 with 7 nodes, where two nodes generate normal FTP and UDP traffic and one generates the UDP attacks. The remaining four nodes constitute the SA agents and a single DA agent. There is a single baseline run and seven tests are conducted, where each test differs in the normal traffic, attack patterns, or both. The corresponding accuracy, recall, and FPR for each test is presented in Table 24. MARL is also tested on a dataset that contains mimicry attacks, it achieved a recall and accuracy of $\sim 30\%$ and $\sim 70\%$. When little change is inflicted in the traffic pattern, MARL can achieve high 99% accuracy and recall with 0 FP.

A less conventional application of RL is [85], which consists of an online IDS based on adaptive NN with modified RL. Here RL consists of a feedback mechanism. The focus is to detect DoS attacks using Cerebellar Model Articulation Controller (CMAC) NN. The learning algorithm incorporates feedback from the protected system in the form of system state (i.e. response rate, heartbeat). The objective is to leverage the system state to assist in detecting the attacks earlier since the responsiveness of the system reduces under attack. The authors evaluate CMAC NN using a prototype application that simulates ping flooding and UDP packet storm attacks. First, they assess the system's ability to autonomously learn attacks. They find that when the system is trained with gradual

ping flood attack vectors, the error rate is 2.199%, which reduces to $1.94^{-7}\%$ as the training progresses. The authors also evaluate the system's ability to learn new attacks and recognize learned attacks. The error rate results are presented in Table 24. Finally, the benefit of the system's feedback mechanism illustrates that as attacks progress, the system state's responsiveness approaches 0 and the error rate reaches $8.53^{-14}\%$.

10.4 Hybrid intrusion detection

We conclude our survey of ML for intrusion detection by looking at hybrid IDSs that apply both misuse and anomaly-based intrusion detection. Such a hybrid system can make the best of both worlds i.e. high accuracy in detecting patterns of known attacks, along with the ability to detect new attacks. Every time a new attack is detected, it can then be fed to the misuse-detection system to enhance the comprehensiveness of its database. We start off our discussion by looking at the work of Depren et al. [116] that leverages J.48 DT and SOM for misuse and anomaly detection, respectively. Three SOM modules are trained, one for each of the TCP, UDP and ICMP traffic. The output of the misuse and anomaly detection modules are combined using a simple decision support system, that raises an alarm if either one of the modules detect an attack. The authors evaluate their work over the KDD Cup dataset and find that their hybrid IDS achieves a DR of 99.9% with a missed rate of 0.1% and a FP of 1.25%.

Similarly, Mukkamala et al. [325] compare a SVM-based with an NN-based hybrid misuse and anomaly detection models. Their models are trained with normal and attack data and evaluated using the KDD Cup dataset. The SVM-based hybrid model achieves 99.5% accuracy with training and testing times of 17.77 s and 1.63 s, respectively. While, three different NNs are trained and tested, each with a different structure of hidden layers. The three NNs achieve an accuracy of 99.05, 99.25, and 99%, respectively, with a training time of 18 min. Therefore, SVM outperforms NN, slightly in accuracy and significantly in runtime.

Zhang et al. [494] develop a hierarchical IDS framework based on RBF to detect both misuse and anomaly attacks, in real-time. Their hierarchical approach is modular and decreases the complexity of the system. It enables different modules to be retrained separately, instead of retraining the entire system. This is particularly useful in the event of a change that only affects a subset of the modules. Serial hierarchical IDS (SHIDS) is compared against a parallel hierarchical IDS (PHIDS). SHIDS begins by training a classifier with only normal data and as the classifier detects abnormal packets, it logs them in a database. *c*-Means clustering [58] groups the data based on their statistical distributions, and as the number of attack records in the largest group exceeds a pre-defined threshold, a new classifier is trained with that specific

attack data and appended to the end of the SHIDS. PHIDS on the other hand consists of three layers. The anomaly and misuse classifiers are in the first two layers, while the third layer is dedicated to different attack categories. Over time, the data in each attack category is updated as new attacks are identified. The performance of RBF is evaluated using the KDD dataset against a Back-Propagation learning algorithm (BPL). Though, BPL achieves a higher DR for misuse detection, RBF has a smaller training time of 5 min compared to 2 h for BPL. Training time is critical for online IDSs. Further, when training the model with just normal data for anomaly detection, RBF outperforms BPL for each attack category, with respect to DR and FP. Overall, RBF achieves a DR of 99.2% and FP of 1.2%, compared to BPL with a DR of 93.7% and FP of 7.2%. The evaluation of SHIDS and PHIDS are in Table 25.

10.5 Summary

Our survey on the application of ML for network security focused on network-based intrusion detection. We grouped the work into misuse, anomaly, and hybrid network IDSs. In each category, we expose the different ML techniques that were applied, including recent applications of DL and RL. One clear take-away message is the significant benefit that ML has brought to misuse-based intrusion detection. It has really improved on the rule-based systems, and allowed the extraction of more complex patterns of attacks from audit data. It even allowed the ability to detect variants of known attacks. In the field of misuse-detection, a preference is given to white-box models (e.g. DT) as their decision rules can be extracted, as opposed to black-box models (e.g. NN). Ensemble-based methods were also heavily employed by training ML models on different subsets of the dataset or with different feature sets. Ensemble-based methods have been particularly useful in achieving very fast training time.

While the benefits of ML for IDS is clear, there is a lot of speculation on the application of ML for anomaly detection. Despite the extensive literature on ML-based anomaly detection, it has not received the same traction in real deployments [415]. Indeed, the most widely deployed IDS (Snort [45]) is in fact misuse-based [101]. The main culprit for this aversion is not only the susceptibility of anomaly detection to high FPs, but also the high-cost of misclassification in the event of FNs. Compared to the cost of misclassification in an ads recommender system, a missed malicious activity can bring down the system or cause a massive data breach. Another main weakness that we observe in the literature is that most works consist of raising an alarm if an anomaly is detected without giving any hints or leads on the observed malicious behavior (e.g. the attack target). Providing such semantics can be extremely valuable to network analysts [415], and even in reducing FP.

The dataset of choice in the majority of the surveyed literature has been based on KDD'99, an out-dated dataset. On one hand, this has provided the community with the ability to compare and contrast different methods and techniques. On the other hand, it does not reflect the recent more relevant types of attacks. Moreover, even the normal connection traces represent basic applications (e.g. email and file-transfer) without any inclusion to more recent day-to-day applications that swarms the network (e.g. social media and video streaming). This is further aggravated by the several limitations and flaws reported about this dataset [438]. Indeed, there is a dire need for a new dataset for intrusion detection.

To conclude, most works on the application of ML for intrusion detection are offline, and amongst the few real-time IDSs, there is no consideration for early detection (i.e. detecting a threat from the first few packets of a flow). Moreover, there is a gap in the ML for intrusion detection literature with regards to intrusion detection for persistent threats, or correlating among isolated anomaly instances over time. Finally, only a handful of works have actually evaluated the robustness of their algorithm in the event of mimicry attacks, an aspect of critical importance as attackers are constantly looking for ways to evade detection.

11 Lessons learned, insights and research opportunities

We have discussed the existing efforts in employing ML techniques to address various challenges and problems in networking. The success of ML primarily lies in the availability of data, compounded with improved and resilient ML algorithms to solve complex problems. Future networks are envisaged to support an explosive growth in traffic volume and connected devices with unprecedented access to information. In addition, these capabilities will have to be achieved without significantly increasing CAPEX, OPEX or customer tariffs.

In order to be sustainable in a competitive environment, network operators must adopt efficient and affordable deployment, operations and management. Enabling technologies for future networks include SDN, network slicing, NFV, and multi-tenancy, which reduce CAPEX, increase resource utilization and sharing. Similarly, automatic network management frameworks coupled with SDN is envisioned to reduce OPEX. The aforementioned technologies will allow future networks to host a wide variety of applications and services, and a richer set of use cases, including massive broadband, ultra low latency and highly reliable services, machine to machine communications, tactile Internet, industrial applications, autonomous vehicles, real-time monitoring and control.

In this subsection, we describe and delineate prominent challenges and open research opportunities pertaining to

the application of ML in current and future networks, from the network, system and knowledge acquisition perspectives.

11.1 Network perspective

11.1.1 Cost of predictions

The accuracy of network monitoring data comes at the cost of increased monitoring overhead (e.g. consumed network bandwidth and switch memory). This raises the need for network monitoring schemes that are both accurate and cost-effective. Monitoring applications in traditional networks rely on a predefined set of monitoring probes built into the hardware/firmware, which limits their flexibility. With SDN customizable software-based monitoring probes can be deployed on-demand to collect more diverse monitoring data. However, in many instances, e.g. monitoring traffic volume over a given switch interface, these probes need to operate at line rate, which is very expensive over very high speed links and difficult to achieve in software. This makes TSF-based approaches for traffic prediction prohibitive.

Recently, two solutions have been investigated in order to overcome this issue, (i) traffic sampling and interpolation [274], and (ii) leveraging features other than traffic volume for traffic prediction [365]. Indeed, various flow sampling techniques (stochastic/deterministic, spacial/temporal, etc.) to reduce monitoring overhead have been proposed in the literature. Unfortunately, the current ML-based solution proposed in [274], is not conclusive and shows contradicting prediction accuracy results. Instead, Poupert et al. [365] use classifiers to identify elephant flows. Indeed, classifiers operate at a coarser granularity. Therefore, their accuracy can not be compared to the accuracy of regression model operating on the same set of features. Using features other than traffic volumes for accurate traffic prediction remains an open research direction.

11.1.2 Cost of errors and detailed reports

ML for anomaly detection has received significant interest in the literature, without gaining traction in the industry. This is primarily due to the high FPR [27, 415], making them inapplicable in an operational setting. FPRs waste expensive analyst time to investigate the false alarms, and reduce the trust and confidence in the IDS. Another major concern with anomaly detection techniques is the lack of detailed reports on detected anomalies [415]. Typically, a flag is raised and an alarm is triggered whenever there is a deviation from the norm. An efficient IDS is not only responsible for detecting attacks and intrusions in the network, it must provide a detailed log of anomalies for historical data collection and model retraining.

11.1.3 Complexity matters

When performing traffic prediction, classification, routing and congestion control on intermediate nodes in the network, it is crucial that they consume less time and computing resources to avoid degradation in network performance. This requirement is non-trivial, especially, in resource-constrained networks, such as WANETs and IoT. Though, performance metrics for ML evaluation are well-defined, it is difficult to evaluate the complexity of ML-based approaches a priori. Unlike traditional algorithms, the complexity of ML algorithms also rely on the size and quality of data, and the performance objectives. The issue is further exacerbated, if the model is adaptive and relearning is intermittently triggered due to varying network conditions over time. The traditional complexity metrics fail to cover these aspects. Therefore, it is important to identify well-rounded evaluation metrics that will help in assessing the complexity of given ML techniques, to strike a trade-off between performance improvement and computational cost.

11.1.4 ML in the face of the new Web

In an effort to improve security and QoE for end-users, new application protocols (e.g. HTTP/2 [48], SPDY [47], QUIC [211]) have emerged that overcome various limitations of HTTP/1.1. For instance, HTTP/2 offers payload encryption, multiplexing and concurrency, resource prioritization, and server push. Though, the WEB applications over HTTP/2 enjoy the benefits of these enhancements, it further complicates traffic classification by introducing unpredictability in the data used for ML. For example, if we employ flow feature-based traffic classification, the feature statistics can be skewed, as several requests can be initiated over the same TCP connection and responses can be received out of order. Therefore, the challenge lies in exploring the behavior and performance of ML techniques when confronted with such unpredictability in a single TCP connection and even parallel TCP connections [293] in HTTP/2. Similarly, prioritization requested by different WEB clients diminish the applicability of a generic ML-based classification technique for identifying WEB applications.

11.1.5 Rethinking evaluation baseline

Often, proposed ML-based networking solutions are assessed and evaluated against existing non-ML frameworks. These latter act as baseline and are used to demonstrate the benefits, if any, of using ML. Unfortunately, these baseline solutions are often deprecated and outdated. For instance, ML-based congestion control mechanisms are often compared against default TCP implementations, e.g. CTCP, CUBIC, or BIC with typical loss recovery mechanisms, such as Reno, NewReno,

or SACK. However, Yang et al. [486] applied supervised learning techniques to identify the precise TCP protocol used in Web traffic and uncovered that though majority of the servers employ the default, there is a small amount of web traffic that employs non-default TCP implementation for congestion control and loss recovery. Therefore, it is critical to consider TCP variants as comparison baselines that have taken the lead, and are prominently employed for congestion control and loss recovery.

ML-based congestion control mechanisms should be designed and evaluated under the consideration that the standard TCP is no longer the de facto protocol, and current networks implement heterogeneous TCP protocols that are TCP-friendly. Furthermore, it is a good practice to consider TCP variants, particularly enhanced for specific network technologies, such as TCP-Few for WANETs and Hybla for satellite networks. ML-based approaches, such as Learning-TCP [29] and PCC [122], have already taken these considerations into account and provide an enhanced evaluation of their proposed solutions. Therefore, it is imperative to design a standardized set of performance metrics for enabling a fair comparison between various ML-based approaches to different problems in networking.

11.1.6 RL in face of network (in)stability and QoS

There are various challenges in finding the right balance between exploration of and exploitation in RL. When it comes to traffic routing, various routes must be explored before the system can converge to the optimal routing policy. However, exploring new routes can lead to performance instability and fluctuation in network delay, throughput and other parameters that impact QoS. On the other hand, exploiting the same “optimal” route to forward all the traffic may lead to congestion and performance degradation, which would also impact the QoS. Different avenues can be explored to overcome these challenges. For example, increasing the learning rate can help detect early signs of performance degradation. While, load balancing can be achieved with selective routing, which can be implemented by assigning different reward functions to different types of flows (elephant vs. mice, ToS, etc.). Furthermore, instability-awareness at exploration time can be implemented by limiting the scope of the routes to explore those with highest rewards. Indeed, this requires an in-depth study to gauge the impact of such solutions on network performance and their convergence time to optimal routing.

Another direction worth pursuing is to correlate the reward function of an RL-based routing to a desired level of QoS. This involves finding ways to answer questions, such as, which reward function can guarantee that the

delay in the network does not exceed a given threshold? or, given a reward function, what would be the expected delay in the network?

11.1.7 Practicality and applicability of ML

Benchmarks used in the literature for the training and validation of proposed ML-based networking solutions are often far from being realistic. For instance, ML-based admission control mechanisms, are based on simulations that consider traffic from only a small set of applications or services. Furthermore, they disregard diversity of QoS parameters when performing admission control. However, in practice, networks carry traffic from heterogeneous applications and services, each having its own QoS requirements, with respect to throughput, loss rate, latency, jitter, reliability, availability, and so on. Hence, the optimal decision in the context of a simulated admission control mechanism may not be the optimal for a practical network. Furthermore, often synthetic network datasets are used in training and validation. Although, ML models perform well in such settings, their applicability in practical settings remains questionable. Therefore, more research is needed to develop practical ML-based network solutions.

11.1.8 SDN meets ML

Though, there has been a growing interest in leveraging ML to realize autonomic networks, there is little evidence of its application to date. Prohibiting factors include the distributed control and vendor-specific nature of legacy network devices. Several technological advances have been made in the last decade to overcome these limitations. The advent of network softwarization and programmability through SDN and NFV offers centralized control and alleviates vendor lock-in.

SDN can facilitate adaptive and intelligent network probing. Probes are test transactions that are used to monitor network behavior and obtain measurements from network elements. Finding the optimal probe rate will be prohibitively expensive in future networks, due to the large number of devices, the variety of parameters to measure, and the small time intervals to log data. Aggressive probing can exponentially increase the amount of traffic overhead resulting in network performance degradation. In contrast, conservative probing may have the risk of missing some significant anomalies or critical network events. Hence, it is imperative to adapt probing rates that keep traffic overhead within a target value, while minimizing performance degradation. SDN can leverage ML techniques to offer the perfect platform to realize adaptive probing. For example, upon predicting a fault or detecting an anomaly, the SDN controller can probe suspected devices at a faster rate. Similarly, during network overload, the controller may reduce the probing rate and

rely on regression to predict the value of the measured parameters.

11.1.9 Virtualization meets ML

Due to the anticipated rise in the number of devices and expansion in network coverage, future networks will be exposed to a higher number of network faults and security threats. If not promptly addressed, such failures and, or attacks can be detrimental, as a single instance may affect many users and violate the QoS requirements of a number of applications and services. Thus, there is a dire need for an intelligent and responsive fault and security management framework. This framework will have to deal with new faults and attacks across different administrative and technological domains within a single network, introduced by concepts of network slicing, NFV, and multi-tenancy. For instance, any failure in the underlying physical resource can propagate to the hosted virtual resources, though the reverse is not always true. Hence, it will be nearly impossible for traditional approaches to locate the root cause or compromised elements of the fault or an attack, in such a complex network setting.

On the other hand, ML-based approaches on fault and security management focus mostly on single tenant in single layer networks. To develop the fault and security management framework for future networks, existing ML-based approaches need to be extended or re-engineered to take into account the notion of multi-tenancy in multi-layer networks. Due to the versatility of the problem, DNN can be explored to model complex multi-dimensional state spaces.

11.1.10 ML for smart network policies

The unprecedented scale and degree of uncertainty in future networks will amplify the complexity of traffic engineering tasks, such as congestion control, traffic prediction, classification, and routing. Although ML-based solutions have shown promising results to address many traffic engineering challenges, their time complexity needs to be evaluated with the envisioned dynamics, volume of data, number of devices and stringent applications requirements in future networks. To address this, *smart* policy-based traffic engineering approaches can be adopted where operators can efficiently and quickly apply adaptive traffic engineering policies. Policy-based traffic classification using SDN has shown promising results in the treatment of QoS requirements based on operator-engineered policies [334]. Incorporating ML to assist in developing and extracting adaptive policies for policy-based traffic engineering solutions, remains rather unexplored. One possible avenue is to apply RL for generating policies for traffic engineering in future networks.

11.1.11 ML in support of autonomy

Networks are experiencing a massive growth in traffic, and will continue to grow even faster with the advent of IoT devices, tactile Internet, virtual/augmented reality, high definition media delivery, etc. Furthermore, Cisco reports that there is a substantial difference between busy hour and average Internet traffic, such that in 2016, the busy hour Internet traffic increased by 51% in comparison to the 32% growth in average Internet traffic [99]. Such difference is expected to grow further in the next half a decade, where Cisco predicts that the growth rate of busy hour traffic will be almost 1.5 times that of average Internet traffic.

To accommodate such dynamic traffic, network operators can no longer afford the CAPEX for static resource provisioning as per the peak traffic requirements. Therefore, network operators must employ dynamic resource allocation that can scale based on the varying traffic demand. ML is an integral part of dynamic resource allocation that enables demand prediction, facilitates proactive provisioning and release of network resources. In addition, contextual information can be leveraged by ML to anticipate exceptional resource demand and reserve emergency resource in highly volatile environments.

Networks are also experiencing an exponential growth in terms of the number and diversity of supported applications and services. These have stringent and heterogeneous QoS requirements, in terms of latency, jitter, reliability, availability and mobility. It is likely that network operators may not only be unaware of all the devices in their network but also unconscious of all the applications and their QoS requirements. Therefore, it is challenging to devise efficient admission control and resource management mechanisms with limited knowledge. Existing works have demonstrated that both admission control and resource management can be formulated as learning problems, where ML can help improve performance and increase efficiency. A further step would be to explore if admission control and resource management strategies can be learned directly from network operation experience. Considering the intricate relationship between network experience and management strategies, DL can be leveraged to characterize the inherent relationship between inputs and outputs of a network.

11.2 System perspective

11.2.1 Support for adaptive, incremental learning in dynamic network environments

Networks are dynamic in nature. Traffic volume, network topology, and security attack signatures, are some of the many aspects that may change, often in an unexpected and previously unobserved way. Thus, it is fundamental to constantly retrain the ML model to account for these changes. Most ML models are trained offline. Retraining

a model from scratch can be computationally intensive, time consuming, and prohibitive. The ability to retrain the model as new data is generated is fundamental to achieve fast incremental learning, which remains an open research direction. Indeed incremental learning comes with special system needs. In the particular case of RL applied to routing in SDN, a number of simulations are required before the model can converge to the optimal observation-to-action mapping policy. Every time a new flow is injected in the network, the SDN controller is required to find the optimal routing policy for that flow, and a number of simulations are performed as changes are observed in the link status. This calls for a system that fully exploits data and model parallelism to provide millisecond-level training convergence time.

11.2.2 Support for secure learning

ML is prone to adversarial attacks [39], also known as mimicry attacks, that aim to confuse learning. For instance, when employing ML for intrusion detection, an adversarial attack can trick the model into misclassifying malicious events as benign by poisoning the training data. Hence, it is fundamental to train robust ML models that are capable of detecting mimicry attacks. An interesting initiative worth mentioning is Cleverhans [346], a useful library that allows to craft adversarial examples. It provides training datasets that can be used to build robust ML models, capable of distinguishing legitimate datasets from poisoned ones, in the particular area of image recognition. There is indeed an urgent need for a system capable of generating adversarial use cases to be used in training robust models. Secure learning also demands a system that protects the training data from leakage and tampering, enforces privacy, data confidentiality and integrity, and support the secure sharing of data across domains.

11.2.3 Architectures for ML-driven networking

Modern networks generate massive volumes of different types of data (e.g. logs, traffic flow records, network performance metrics, etc.). At 100's of Gbps, even with high sampling rates, a single large network infrastructure element can easily generate hundreds of millions of flow records per day. Recently, the availability of massive data drove rapid advancement in computer hardware and software systems, for storage, processing and analytics. This is evidenced by the emergence of massive-scale datacenters, with tens of thousands of servers and EB storage capacity, the widespread deployment of large-scale software systems like Hadoop MapReduce and Apache Spark, and the increasing number of ML and in particular deep learning libraries built on top of these systems, such as Tensor-Flow, Torch, Caffe, Chainer, Nvidia's CUDA and MXNet. Mostly open-source, these libraries are capable

of scaling out their workloads on CPU clusters enabled by specialized hardware, such as GPUs and TPUs.

GPUs are anticipated to be a key enabler for the next generation SDN [166, 465]. GPU-accelerated SDN routers are reported to have a much improved packet processing capability. Furthermore, the GPUs on SDN controllers may be particularly useful for executing ML and DL algorithms for learning various networking scenarios, and acting according to the acquired knowledge. On the other hand, smaller, resource constrained, smart networked devices, are more likely to benefit from a cloud-edge ML system. A cloud-edge ML system would leverage the large processing and memory resources, robust networks, and massive storage capabilities of the cloud for training computationally intensive models and sharing these with edge devices. Data collection and analytics that require immediate or near-immediate response time would be handled by edge devices. Light-weight ML software systems, such as Caffe2Go and TensorFlowLite, would eventually enable edge devices to by-pass the cloud and build leaner models locally.

11.3 Knowledge perspective

11.3.1 Lack of real-world data

As we surveyed the literature, we observed that numerous works relied on synthetic data, particularly in resource and fault management, network security, and QoE/QoS correlation. Synthetic datasets are usually simplistic and do not truly reflect the complexity of real-world settings. This is not surprising, since obtaining real-world data traces is difficult due to the critical and private nature of network traffic, especially the payload. Furthermore, establishing the ground truth is particularly challenging, given the voluminous amount of traffic making any manual inspection intractable. Although injecting faults and, or attacks in the network can help produce the required data as adopted by [285], it is unrealistic to jeopardize a production network for the sake of generating training data. Such limitations increase the probability of ML techniques being ill-trained and inapplicable in real-world network settings. Thus, it remains unclear how the numerous works in the literature would perform over real data traces. Therefore, a combined effort from both academia and industry is needed, to create public repositories of data traces annotated with ground truth from various real networks.

11.3.2 The need for standard evaluation metrics

As we survey existing works, it became apparent that comparing them within each networking domain is not possible. This is due to the adoption of non-standardized performance metrics, evaluation environments, or datasets [109]. Furthermore, even when the same dataset is adopted, different portions of the data are

used for training and testing, thereby inhibiting any possibility for comparative analysis. Standardization of metrics, data, and environment for evaluating similar approaches is fundamental to provide the ability to contrast and compare the different techniques, and evaluate their suitability for different networking tasks. To fulfill this need, standard bodies such as the Internet Engineering Task Force (IETF), can play a pivotal role by promoting standardization of evaluation procedures, performance metrics, and data formats through Requests for Comments (RFCs).

11.3.3 Theory and ML techniques for networking

As the compute and data storage barriers that thwarted the application of ML in networking are no longer an issue, *what is now preventing an ML-for-networking success story as in games, vision and speech recognition?* Lack of a theoretical model is one obstacle that ML faces in networking. This concern was raised by David Meyer during his talk at IETF97 on machine intelligence and networking [308]. Without a unified theory, each network has to be learned separately. This could truly hinder the speed of adoption of ML in networking. Furthermore, the currently employed ML techniques in networking have been designed with other applications in mind. An open research direction in this realm is to design ML algorithms tailored for networks [306]. Another key issue is the lack of expertise, that is, ML and networking are two different fields, and there is currently a scarcity in the number of people that are experts in both domains. This mandates more cross-domain collaborations involving experts from both networking and ML communities.

12 Conclusion

Over the past two decades, ML has been successfully applied in various areas of networking. This survey provides a comprehensive body of knowledge on the applicability of ML techniques in support of network operation and management, with a focus on traffic engineering, performance optimization and network security. We review representative literature works, explore and discuss the feasibility and practicality of the proposed ML solutions in addressing challenges pertaining to the autonomic operation and management of future networks.

Clearly, future networks will have to support an explosive growth in traffic volume and connected devices, to provide exceptional capabilities for accessing and sharing information. The unprecedented scale and degree of uncertainty will amplify the complexity of traffic engineering tasks, such as congestion control, traffic prediction, classification, and routing, as well as the exposure to faults and security attacks. Although ML-based solutions have shown promising results to address many traffic engineering challenges, their scalability needs to be evaluated with the envisioned volume of data, number

of devices and applications. On the other hand, existing ML-based approaches for fault and security management focus mostly on single-tenant and single-layer networks. To develop the fault and security management framework for future networks, existing ML approaches should be extended or re-architected to take into account the notion of multi tenancy in multi layer networks.

In this survey, we discuss the above issues along with several other challenges and opportunities. Our findings motivate the need for more research to advance the state-of-the-art, and finally realize the long-time vision of autonomic networking.

Acknowledgments

We thank the anonymous reviewers for their insightful comments and suggestions that helped us improve the quality of the paper. This work is supported in part by the Royal Bank of Canada, NSERC Discovery Grants Program, the Quebec FRQNT postdoctoral research fellowship, the ELAP scholarship, and the COLCIENCIAS Scholarship Program No. 647-2014, Colombia.

Authors' contributions

All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details

¹David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada. ²Department of Telematics, University of Cauca, Popayan, Colombia.

Received: 3 January 2018 Accepted: 4 May 2018

Published online: 21 June 2018

References

1. Aben E. NLNR PMA data. 2010. <https://labs.ripe.net/datarepository/data-sets/nlanr-pma-data>. Accessed 27 Dec 2017.
2. Ackley DH, Hinton GE, Sejnowski TJ. A learning algorithm for boltzmann machines. *Cogn Sci*. 1985;9(1):147–69.
3. ACM Special Interest Group on KDD. KDD cup archives. 2016. <http://www.kdd.org/kdd-cup>. Accessed 22 Nov 2017.
4. Adams R. Active queue management: A survey. *IEEE Commun Surv Tutor*. 2013;15(3):1425–76.
5. Adda M, Qader K, Al-Kasassbeh M. Comparative analysis of clustering techniques in network traffic faults classification. *Int J Innov Res Comput Commun Eng*. 2017;5(4):6551–63.
6. Adeel A, Larijani H, Javed A, Ahmadinia A. Critical analysis of learning algorithms in random neural network based cognitive engine for lte systems. In: *Vehicular Technology Conference (VTC Spring)*, 2015 IEEE 81st. IEEE; 2015. p. 1–5.
7. Ahmed T, Coates M, Lakhina A. Multivariate online anomaly detection using kernel recursive least squares. *IEEE*; 2007. pp. 625–33.
8. Ahn CW, Ramakrishna RS. Qos provisioning dynamic connection-admission control for multimedia wireless networks using a hopfield neural network. *IEEE Trans Veh Technol*. 2004;53(1):106–117.
9. Aizenberg IN, Aizenberg NN, Vandewalle JP. *Multi-Valued and Universal Binary Neurons: Theory, Learning and Applications*. Norwell: Kluwer Academic Publishers; 2000.
10. Akan OB, Akyildiz IF. ATL: an adaptive transport layer suite for next-generation wireless internet. *IEEE J Sel Areas Commun*. 2004;22(5):802–17.

11. Albus JS. A new approach to manipulator control: the cerebellar model articulation controller (cmac). *J Dyn Syst Meas. Control*. 1975;97:220–7.
12. Alhoniemi E, Himberg J, Parhankangas J, Vesanto J. S.o.m toolbox; 2000. <http://www.cis.hut.fi/projects/somtoolbox/>. Accessed 28 Dec 2017.
13. Allman M, Paxson V, Blanton E. TCP Congestion Control. RFC 5681, Internet Engineering Task Force. 2009. <https://tools.ietf.org/html/rfc5681>.
14. Alom MZ, Bontupalli V, Taha TM. Intrusion detection using deep belief networks. In: Aerospace and Electronics Conference (NAECON), 2015 National. IEEE; 2015. p. 339–44.
15. Alpaydin E. Introduction to Machine Learning, 2nd ed. Cambridge: MIT Press; 2010.
16. Alpaydin E. Introduction to Machine Learning, 3rd ed. Cambridge: MIT Press; 2014.
17. Alshammari R, Zincir-Heywood AN. Machine learning based encrypted traffic classification: Identifying ssh and skype. In: Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on. IEEE; 2009. p. 1–8.
18. Alsheikh MA, Lin S, Niyato D, Tan HP. Machine learning in wireless sensor networks: Algorithms, strategies, and applications. *IEEE Commun Surv Tutor*. 2014;16(4):1996–2018.
19. Amaral P, Dinis J, Pinto P, Bernardo L, Tavares J, Mamede HS. Machine learning in software defined networks: Data collection and traffic classification. In: Network Protocols (ICNP), 2016 IEEE 24th International Conference on. IEEE; 2016. p. 1–5.
20. Amor NB, Benferhat S, Elouedi Z. Naive bayes vs decision trees in intrusion detection systems. In: Proceedings of the 2004 ACM symposium on Applied computing. ACM; 2004. p. 420–4.
21. Arndt D. HOW TO: Calculating Flow Statistics Using NetMate. 2016. <https://dan.arndt.ca/nims/calculating-flow-statistics-using-netmate/>. Accessed 01 Aug 2017.
22. Arouche Nunes BA, Veenstra K, Ballenthin W, Lukin S, Obraczka K. A machine learning framework for tcp round-trip time estimation. *EURASIP J Wirel Commun Netw*. 2014;2014(1):47.
23. Aroussi S, Bouabana-Tebibel T, Mellouk A. Empirical QoE/QoS correlation model based on multiple parameters for VoD flows. In: Global Communications Conference (GLOBECOM), 2012 IEEE. IEEE; 2012. p. 1963–8.
24. Arroyo-Valles R, Alaiz-Rodriguez R, Guerrero-Curieses A, Cid-Sueiro J. Q-probabilistic routing in wireless sensor networks. In: Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on. IEEE; 2007. p. 1–6.
25. Astrom K. Optimal control of markov processes with incomplete state information. *J Mathl Anal Appl*. 1965;10(1):174–205.
26. Auld T, Moore AW, Gull SF. Bayesian neural networks for internet traffic classification. *IEEE Trans neural networks*. 2007;18(1):223–39.
27. Axelsson S. The base-rate fallacy and the difficulty of intrusion detection. *ACM Trans Inf Syst Secur (TISSEC)*. 2000;3(3):186–205.
28. Ayoubi S, Limam N, Salahuddin MA, Shahriar N, Boutaba R, Estrada-Solano F, Caicedo OM. Machine learning for cognitive network management. *IEEE Commun Mag*. 2018;1(1):1.
29. Badarla V, Murthy CSR. Learning-tcp: A stochastic approach for efficient update in tcp congestion window in ad hoc wireless networks. *J Parallel Distrib Comput*. 2011;71(6):863–78.
30. Badarla V, Siva Ram Murthy C. A novel learning based solution for efficient data transport in heterogeneous wireless networks. *Wirel Netw*. 2010;16(6):1777–98.
31. Bakhshi T, Ghita B. On internet traffic classification: A two-phased machine learning approach. *J Comput Netw Commun*. 2016;2016:2016.
32. Bakre A, Badrinath BR. I-tcp: Indirect tcp for mobile hosts. In: Proceedings of the 15th International Conference on Distributed Computing Systems. ICDCS '95. Washington: IEEE Computer Society; 1995. p. 136.
33. Balakrishnan H, Seshan S, Amir E, Katz RH. Improving tcp/ip performance over wireless networks. In: Proceedings of the 1st Annual International Conference on Mobile Computing and Networking. MobiCom '95. New York: ACM; 1995. p. 2–11.
34. Balakrishnan H, Padmanabhan VN, Seshan S, Katz RH. A comparison of mechanisms for improving tcp performance over wireless links. *IEEE/ACM Trans Netw*. 1997;5(6):756–69.
35. Baldo N, Zorzi M. Learning and adaptation in cognitive radios using neural networks. In: Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE. IEEE; 2008. p. 998–1003.
36. Baldo N, Dini P, Nin-Guerrero J. User-driven call admission control for voip over wlan with a neural network based cognitive engine. In: Cognitive Information Processing (CIP), 2010 2nd International Workshop on. IEEE; 2010. p. 52–6.
37. Baras JS, Ball M, Gupta S, Viswanathan P, Shah P. Automated network fault management. In: MILCOM 97 Proceedings. IEEE; 1997. p. 1244–50.
38. Barman D, Matta I. Model-based loss inference by tcp over heterogeneous networks. In: Proceedings of WiOpt 2004 Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks. Cambridge; 2004. p. 364–73.
39. Barreno M, Nelson B, Sears R, Joseph AD, Tygar JD. Can machine learning be secure? In: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ACM, ASIACCS '06. New York: ACM; 2006. p. 16–25.
40. Barreto GA, Mota JCM, Souza LGM, Frota RA, Aguayo L. Condition monitoring of 3g cellular networks through competitive neural models. *IEEE Trans Neural Netw*. 2005;16(5):1064–75.
41. Baum LE, Petrie T. Statistical inference for probabilistic functions of finite state markov chains. *Ann Math Statist*. 1966;37(6):1554–63.
42. Bay SD, Kibler D, Pazzani MJ, Smyth P. The uci kdd archive of large data sets for data mining research and experimentation. *SIGKDD Explor Newsl*. 2000;2(2):81–5.
43. Bayes M, Price M. An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, f. r. s. communicated by mr. price, in a letter to john canton, a. m. f. r. s. *Philos Trans*. 1763;53(1683-1775): 370–418.
44. Beale J, Deraison R, Meer H, Temmingh R, Walt CVD. Nessus network auditing. Burlington: Syngress Publishing; 2004.
45. Beale J, Baker AR, Esler J. Snort: IDS and IPS toolkit. Burlington: Syngress Publishing; 2007.
46. Bellman R. Dynamic Programming, 1st ed. Princeton: Princeton University Press; 1957.
47. Belshe M, Peon R. SPDY Protocol. Tech. rep., Network Working Group. 2012. <https://tools.ietf.org/pdf/draft-mbelshe-httpbis-spdy-00.pdf>.
48. Belshe M, Peon R, Thomson M. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, IETF. 2015. <http://www.rfc-editor.org/info/rfc7540.txt>.
49. Bengio Y, Lamblin P, Popovici D, Larochelle H. Greedy layer-wise training of deep networks. In: Proceedings of the 19th, International Conference on Neural Information Processing Systems. NIPS'06. Cambridge: MIT Press; 2006. p. 153–60.
50. Benson T. Data Set for IMC 2010 Data Center Measurement. 2010. http://pages.cs.wisc.edu/tbenson/IMC10_Data.html. Accessed 28 Dec 2017.
51. Bergstra JA, Middelburg C. Iu-t recommendation g. 107: The e-model, a computational model for use in transmission planning: ITU; 2003.
52. Bermolen P, Rossi D. Support vector regression for link load prediction. *Comput Netw*. 2009;53(2):191–201.
53. Bermolen P, Mellia M, Meo M, Rossi D, Valenti S. Abacus: Accurate behavioral classification of P2P-tv traffic. *Comput Netw*. 2011;55(6): 1394–411.
54. Bernaille L, Teixeira R. Implementation issues of early application identification. *Lect Notes Compu Sci*. 2007;4866:156.
55. Bernaille L, Teixeira R, Akodkenou I, Soule A, Salamati K. Traffic classification on the fly. *ACM SIGCOMM Comput Commun Rev*. 2006a;36(2):23–6.
56. Bernaille L, Teixeira R, Salamati K. Early application identification. In: Proceedings of the 2006 ACM CoNEXT Conference. ACM; 2006b. p. 61–6:12.
57. Bernstein L, Yuhas CM. How technology shapes network management. *IEEE Netw*. 1989;3(4):16–9.
58. Bezdek JC, Ehrlich R, Full W. Fcm: The fuzzy c-means clustering algorithm. *Comput Geosci*. 1984;10(2-3):191–203.
59. Bhorkar AA, Naghsvar M, Javidi T, Rao BD. Adaptive opportunistic routing for wireless ad hoc networks. *IEEE/ACM Trans Netw*. 2012;20(1): 243–56.
60. Biaz S, Vaidya NH. Distinguishing congestion losses from wireless transmission losses: a negative result. In: Proceedings 7th International Conference on Computer Communications and Networks (Cat. No.98EX226). Piscataway: IEEE; 1998. p. 722–31.
61. Bkassiny M, Li Y, Jayaweera SK. A survey on machine-learning techniques in cognitive radios. *IEEE Commun Surv Tutor*. 2013;15(3): 1136–59.

62. Blanton E, Allman M. On making tcp more robust to packet reordering. *SIGCOMM Comput Commun Rev.* 2002;32(1):20–30.
63. Blenk A, Kalmbach P, van der Smagt P, Kellerer W. Boost online virtual network embedding: Using neural networks for admission control. In: *Network and Service Management (CNSM), 2016 12th International Conference on.* Piscataway: IEEE; 2016. p. 10–8.
64. Boero L, Marchese M, Zappatore S. Support vector machine meets software defined networking in ids domain. In: *Proceedings of the 29th International Teletraffic Congress (ITC), vol. 3.* New York: IEEE; 2017. p. 25–30.
65. Bojovic B, Baldo N, Nin-Guerrero J, Dini P. A supervised learning approach to cognitive access point selection. In: *GLOBECOM Workshops (GC Wkshps), 2011 IEEE.* Piscataway: IEEE; 2011. p. 1100–5.
66. Bojovic B, Baldo N, Dini P. A cognitive scheme for radio admission control in lte systems. In: *Cognitive Information Processing (CIP), 2012 3rd International Workshop on.* Piscataway: IEEE; 2012. p. 1–3.
67. Bojovic B, Quer G, Baldo N, Rao RR. Bayesian and neural network schemes for call admission control in lte systems. In: *Global Communications Conference (GLOBECOM), 2013 IEEE.* Piscataway: IEEE; 2013. p. 1246–52.
68. Bonald T, May M, Bolot JC. Analytic evaluation of red performance. In: *Proceedings IEEE INFOCOM 2000. Conference on, Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064), vol 3.* 2000. p. 1415–24.
69. Bonfiglio D, Mellia M, Meo M, Rossi D, Tofanelli P. Revealing skype traffic: when randomness plays with you. In: *ACM SIGCOMM Computer Communication Review.* ACM; 2007. p. 37–48.
70. Boser BE, Guyon IM, Vapnik VN. A training algorithm for optimal margin classifiers. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, ACM, New York, NY, USA, COLT '92.* New York: ACM; 1992. p. 144–52.
71. Boyan JA, Littman ML. Packet routing in dynamically changing networks: A reinforcement learning approach. In: *Advances in neural information processing systems; 1994.* p. 671–8.
72. Braden B, Clark D, Crowcroft J, Davie B, Deering S, Estrin D, Floyd S, Jacobson V, Minshall G, Partridge C, Peterson L, Ramakrishnan K, Shenker S, Wroclawski J, Zhang L. Recommendations on queue management and congestion avoidance in the internet. RFC 2309, Internet Engineering Task Force. 1998. <https://tools.ietf.org/html/rfc2309>.
73. Brakmo LS, O'Malley SW, Peterson LL. Tcp vegas: New techniques for congestion detection and avoidance. In: *Proceedings of the Conference on Communications Architectures, Protocols and Applications, ACM, New York, NY, USA, SIGCOMM '94.* New York: ACM; 1994. p. 24–35.
74. Brauckhoff D, Wagner A, May M. FLAME: A flow-level anomaly modeling engine. In: *Proceedings of the conference on Cyber security experimentation and test (CSET).* Berkeley: USENIX Association; 2008. p. 1.
75. Breiman L. Bagging predictors. *Mach Learn.* 1996;24(2):123–40.
76. Breiman L, Friedman J, Stone C, Olshen R. Classification and Regression Trees. The Wadsworth and Brooks-Cole statistics-probability series. New York: Taylor & Francis; 1984.
77. Brill E, Lin JJ, Banko M, Dumais ST, Ng AY, et al. Data-intensive question answering. In: *TREC, vol. 56.* 2001. p. 90.
78. Broomhead DS, Lowe D. Radial basis functions, multi-variable functional interpolation and adaptive networks. Memorandum No. 4148. Malvern: Royal Signals and Radar Establishment; 1988.
79. Brownlee J. Practical Machine Learning Problems. 2013. <https://machinelearningmastery.com/practical-machine-learning-problems/>. Accessed 28 Dec 2017.
80. Bryson A, Ho Y. Applied optimal control: optimization, estimation and control. Blaisdell book in the pure and applied sciences. Waltham: Blaisdell Pub. Co; 1969.
81. Bryson AE. A gradient method for optimizing multi-stage allocation processes. In: *Harvard University Symposium on Digital Computers and their Applications.* Boston; 1961. p. 72.
82. Buczak AL, Guven E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun Surv Tutor.* 2016;18(2):1153–76.
83. Caini C, Firrincieli R. Tcp hybla: a tcp enhancement for heterogeneous networks. *Int J Satell Commun Netw.* 2004;22(5):547–66.
84. Cannady J. Artificial neural networks for misuse detection. In: *Proceedings of the 21st National information systems security conference, vol. 26.* Virginia; 1998. p. 368–81.
85. Cannady J. Next generation intrusion detection: Autonomous reinforcement learning of network attacks. In: *Proceedings of the 23rd national information systems security conference.* Baltimore; 2000. p. 1–12.
86. Chabaa S, Zeroual A, Antari J. Identification and prediction of internet traffic using artificial neural networks. *J Int Learn Syst Appl.* 2010;2(03):147.
87. Chakrabarti S, Chakraborty M, Mukhopadhyay I. Study of snort-based ids. In: *Proceedings of the International Conference and Workshop on Emerging Trends in Technology.* New York: ACM; 2010. p. 43–7.
88. Chang CC, Lin CJ. Libsvm: a library for support vector machines. *ACM trans int syst technols (TIST).* 2011;2(3):27.
89. Charonyktakis P, Plakia M, Tsamardinos I, Papadopoulou M. On user-centric modular qoe prediction for voip based on machine-learning algorithms. *IEEE Trans Mob Comput.* 2016;15(6):1443–56.
90. Chebrolu S, Abraham A, Thomas JP. Feature deduction and ensemble design of intrusion detection systems. *Comput secur.* 2005;24(4):295–307.
91. Chen M, Zheng AX, Lloyd J, Jordan MI, Brewer E. Failure diagnosis using decision trees. In: *Autonomic Computing, 2004. Proceedings. International Conference on.* Piscataway: IEEE; 2004. p. 36–43.
92. Chen MY, Kiciman E, Fratkin E, Fox A, Brewer E. Pinpoint: Problem determination in large, dynamic internet services. In: *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on.* Piscataway: IEEE; 2002. p. 595–604.
93. Chen T, Guestrin C. XGBoost: A Scalable Tree Boosting System. In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* New York: ACM; 2016. p. 785–94.
94. Chen Z, Wen J, Geng Y. Predicting future traffic using hidden markov models. In: *Proceedings of 24th IEEE International Conference on Network Protocols (ICNP).* IEEE; 2016. p. 1–6.
95. Cheng RG, Chang CJ. Neural-network connection-admission control for atm networks. *IEE Proc-Commun.* 1997;144(2):93–8.
96. Choi SP, Yeung DY. Predictive q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In: *Advances in Neural Information Processing Systems.* 1996. p. 945–51.
97. Chow CK. An optimum character recognition system using decision functions. *IRE Trans Electron Comput EC.* 1957;6(4):247–54.
98. Chun-Feng W, Kui L, Pei-Ping S. Hybrid artificial bee colony algorithm and particle swarm search for global optimization. *Math Probl Eng.* 2014;2014.
99. Cisco. The Zettabyte Era: Trends and Analysis. 2017. <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html>. Accessed 28 Dec 2017.
100. Cisco Systems. Cisco IOS Netflow. 2012. <http://www.cisco.com/go/netflow>. Accessed 01 Aug 2017.
101. Cisco Systems. Snort: The worlds most widely deployed ips technology. 2014. https://www.cisco.com/c/en/us/products/collateral/security/brief_c17-733286.html. Accessed 25 Apr 2018.
102. Claeys M, Latre S, Famaey J, De Turck F. Design and evaluation of a self-learning http adaptive video streaming client. *IEEE commun lett.* 2014a;18(4):716–9.
103. Claeys M, Latré S, Famaey J, Wu T, Van Leekwijck W, De Turck F. Design and optimisation of a (fa) q-learning-based http adaptive streaming client. *Connect Sci.* 2014b;26(1):25–43.
104. Cortez P, Rio M, Rocha M, Sousa P. Internet traffic forecasting using neural networks. In: *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN).* IEEE; 2006. p. 2635–42.
105. Cox DR. The regression analysis of binary sequences. *J R Stat Soc Ser B (Methodological).* 1958;20(2):215–42.
106. Cybenko G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control. Signals Syst (MCSS).* 1989;2(4):303–14.
107. Dagum P, Galper A, Horvitz EJ. Temporal Probabilistic Reasoning: Dynamic Network Models for Forecasting. Stanford: Knowledge Systems Laboratory, Medical Computer Science, Stanford University; 1991.
108. Dainotti A, Pescapé A, Sansone C. Early classification of network traffic through multi-classification. In: *International Workshop on Traffic Monitoring and Analysis.* Springer; 2011. p. 122–35.
109. Dainotti A, Pescapé A, Claffy KC. Issues and future directions in traffic classification. *IEEE Netw.* 2012;26(1):35–40.

110. Dean T, Kanazawa K. A model for reasoning about persistence and causation. *Comput Intell*. 1989;5(2):142–50.
111. Dechter R. Learning while searching in constraint-satisfaction-problems. In: *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI Press, AAAI'86. Palo Alto: AAAI Press; 1986. p. 178–83.
112. Demirbilek E. The INRS Audiovisual Quality Dataset. 2016. <https://github.com/edipdemirbilek/TheINRSAudiovisualQualityDataset>. Accessed 28 Dec 2017.
113. Demirbilek E, Grégoire JC. INRS audiovisual quality dataset. *ACM*; 2016, pp. 167–71.
114. Demirbilek E, Grégoire JC. Machine learning-based parametric audiovisual quality prediction models for real-time communications. *ACM Transactions on Multimedia Computing. Commun Appl (TOMM)*. 2017;13(2):16.
115. Dempster AP, Laird NM, Rubin DB. Maximum likelihood from incomplete data via the em algorithm. *J R Stat Soc Ser B (Method)*. 1977;39(1):1–38.
116. Depren O, Topallar M, Anarim E, Ciliz MK. An intelligent intrusion detection system (ids) for anomaly and misuse detection in computer networks. *Expert syst Appl*. 2005;29(4):713–22.
117. Detristan T, Ulenspiegel T, Malcom Y, Underduk M. Polymorphic shellcode engine using spectrum analysis. 2003. <http://www.phrack.org/show.php?p=61&a=9>. Accessed 25 May 2018.
118. Ding J, Kramer B, Xu S, Chen H, Bai Y. Predictive fault management in the dynamic environment of ip networks. In: *IP Operations and Management*, 2004. *Proceedings IEEE Workshop on*. Piscataway: IEEE; 2004. p. 233–9.
119. Ding L, Wang X, Xu Y, Zhang W. Improve throughput of tcp-vegas in multihop ad hoc networks. *Comput Commun*. 2008;31(10):2581–8.
120. Domingos P. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Basic Books; 2015.
121. Donato W, Pescapé A, Dainotti A. Traffic identification engine: an open platform for traffic classification. *IEEE Netw*. 2014;28(2):56–64.
122. Dong M, Li Q, Zarchy D, Godfrey PB, Schapira M. Pcc: Re-architecting congestion control for consistent high performance. In: *Proceedings of the 12th, USENIX Conference on Networked Systems Design and Implementation*, USENIX Association, Berkeley, CA, USA, NSDI'15. Berkeley: USENIX Association; 2015. p. 395–408.
123. Dowling J, Cunningham R, Curran E, Cahill V. Collaborative reinforcement learning of autonomic behaviour. In: *Proceedings. 15th International Workshop on Database and Expert Systems Applications*, 2004. 2004. p. 700–4. <https://doi.org/10.1109/DEXA.2004.1333556>.
124. Dowling J, Curran E, Cunningham R, Cahill V. Using feedback in collaborative reinforcement learning to adaptively optimize manet routing. *IEEE Transactions on Systems. Man and Cybern-Part A: Syst Hum*. 2005;35(3):360–72.
125. Dreger H, Feldmann A, Mai M, Paxson V, Sommer R. Dynamic application-layer protocol analysis for network intrusion detection. In: *USENIX Security Symposium*. Berkeley: USENIX Security Symposium; 2006. p. 257–72.
126. Dump CM. Dde command execution malware samples. 2017. <http://contagiodump.blogspot.it>. Accessed 1 Mar 2017.
127. eBay Inc. eBay. 2017. <https://www.ebay.com/>. Accessed 01 Aug 2017.
128. Edalat Y, Ahn JS, Obraczka K. Smart experts for network state estimation. *IEEE Trans Netw Serv Manag*. 2016;13(3):622–35.
129. El Khayat I, Geurts P, Leduc G. Improving TCP in Wireless Networks with an Adaptive Machine-Learnt Classifier of Packet Loss Causes. Berlin, Heidelberg: Springer Berlin Heidelberg; 2005, pp. 549–60.
130. El Khayat I, Geurts P, Leduc G. Enhancement of tcp over wired/wireless networks with packet loss classifiers inferred by supervised learning. *Wirel Netw*. 2010;16(2):273–90.
131. Elkan C. Results of the kdd'99 classifier learning. *ACM SIGKDD Explor Newsl*. 2000;1(2):63–4.
132. Elkotob M, Grandlund D, Andersson K, Ahlund C. Multimedia qoe optimized management using prediction and statistical learning. In: *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*. IEEE; 2010. p. 324–7.
133. Elwhishi A, Ho PH, Naik K, Shihada B. Arbr: Adaptive reinforcement-based routing for dtn. In: *Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2010 IEEE 6th International Conference on. IEEE; 2010. p. 376–85.
134. Erickson BJ, Korfiatis P, Akkus Z, Kline TL. Machine learning for medical imaging. *RadioGraphics*. 2017;37(2):505–15.
135. Erman J, Arlitt M, Mahanti A. Traffic classification using clustering algorithms. In: *Proceedings of the 2006 SIGCOMM workshop on Mining network data*. ACM; 2006a. p. 281–6.
136. Erman J, Mahanti A, Arlitt M. Internet traffic identification using machine learning. In: *Global Telecommunications Conference, 2006. GLOBECOM'06*. IEEE. IEEE; 2006b. p. 1–6.
137. Erman J, Mahanti A, Arlitt M, Cohen I, Williamson C. Offline/realtime traffic classification using semi-supervised learning. *Perform Eval*. 2007a;64(9):1194–213.
138. Erman J, Mahanti A, Arlitt M, Williamson C. Identifying and discriminating between web and peer-to-peer traffic in the network core. In: *Proceedings of the 16th international conference on World Wide Web*. ACM; 2007b. p. 883–92.
139. Eskin E, Arnold A, Prerau M, Portnoy L, Stolfo S. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. *Appl data min comput secur*. 2002;6:77–102.
140. Este A, Gringoli F, Salgarelli L. Support vector machines for tcp traffic classification. *Comput Netw*. 2009;53(14):2476–90.
141. Eswaradass A, Sun XH, Wu M. Network bandwidth predictor (nbp): A system for online network performance forecasting. In: *Proceedings of 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*. IEEE; 2006. p. 4–pp.
142. Fadlullah Z, Tang F, Mao B, Kato N, Akashi O, Inoue T, Mizutani K. State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems. *IEEE Commun Surv Tutor*. 2017;PP(99):1.
143. Farmer D, Venema W. Satan: Security administrator tool for analyzing networks. 1993. <http://www.porcupine.org/satan/>. Accessed 28 Dec 2017.
144. Feng W-C, Shin KG, Kandlur DD, Saha D. The blue active queue management algorithms. *IEEE/ACM Trans Netw*. 2002;10(4):513–28.
145. Fiedler M, Hossfeld T, Tran-Gia P. A generic quantitative relationship between quality of experience and quality of service. *IEEE Netw*. 2010;24(2).
146. Finamore A, Mellia M, Meo M, Rossi D. Kiss: Stochastic packet inspection classifier for udp traffic. *IEEE/ACM Trans Netw*. 2010;18(5):1505–15.
147. Fix E, Hodges JL. Discriminatory analysis-nonparametric discrimination: consistency properties. Report No. 4, Project 21-49-004, USAF School of Aviation Medicine. 1951.
148. Floyd S, Jacobson V. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans Netw*. 1993;1(4):397–413.
149. Fogla P, Sharif MI, Perdisci R, Kolesnikov OM, Lee W. Polymorphic blending attacks. In: *USENIX Security Symposium*. Berkeley: USENIX Association; 2006. p. 241–56.
150. Fonseca N, Crovella M. Bayesian packet loss detection for tcp. In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol 3. 2005. p. 1826–37.
151. Forster A, Murphy AL. Froms: Feedback routing for optimizing multiple sinks in wsn with reinforcement learning. In: *Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International, Conference on*. IEEE; 2007. p. 371–6.
152. Fraleigh C, Diot C, Lyles B, Moon S, Owezarski P, Papagiannaki D, Tobagi F. Design and deployment of a passive monitoring infrastructure. In: *Thyrrhenian International Workshop on Digital Communications*. Springer; 2001. p. 556–75.
153. Freund Y, Schapire RE. Experiments with a new boosting algorithm. In: *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning, ICML'96*. San Francisco: Morgan Kaufmann Publishers Inc.; 1996. p. 148–56.
154. Friedman JH. Greedy function approximation: A gradient boosting machine. *Ann Statist*. 2001;29(5):1189–232.
155. Friedman JH. Stochastic gradient boosting. *Comput Stat Data Anal*. 2002;38(4):367–78.
156. Fu CP, Liew SC. Tcp veno: Tcp enhancement for transmission over wireless access networks. *IEEE J Sel Areas Commun*. 2003;21(2): 216–28.
157. Fukushima K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol Cybern*. 1980;36(4):193–202.

158. Funahashi KI. On the approximate realization of continuous mappings by neural networks. *Neural netw.* 1989;2(3):183–92.
159. Gagniuc P. *Markov Chains: From Theory to Implementation and Experimentation*. Hoboken: Wiley; 2017.
160. Gao Y, He G, Hou JC. On exploiting traffic predictability in active queue management. In: *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3. Piscataway: IEEE; 2002. p. 1630–9.
161. Garcia-Teodoro P, Diaz-Verdejo J, Maciá-Fernández G, Vázquez E. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Comput Secur.* 2009;28(1-2):18–28.
162. Gartner Inc. Gartner Says 8.4 Billion Connected “Things” Will Be in Use in 2017, Up 31 Percent From 2016. 2017. <https://www.gartner.com/newsroom/id/3598917>. Accessed 01 Aug 2017.
163. Geurts P, Khayat IE, Leduc G. A machine learning approach to improve congestion control over wireless computer networks. In: *Data Mining, 2004. ICDM '04. Fourth IEEE International Conference on*; 2004. p. 383–6.
164. Geurts P, Ernst D, Wehenkel L. Extremely randomized trees. *Mach Learn.* 2006;63(1):3–42.
165. Giacinto G, Perdisci R, Del Rio M, Roli F. Intrusion detection in computer networks by a modular ensemble of one-class classifiers. *Inf Fusion.* 2008;9(1):69–82.
166. Go Y, Jamshed MA, Moon Y, Hwang C, Park K. Apunet: Revitalizing gpu as packet processing accelerator. In: *NSDI*. 2017. p. 83–96.
167. Goetz P, Kumar S, Miikkulainen R. On-line adaptation of a signal predistorter through dual reinforcement learning. In: *ICML*. 1996. p. 175–81.
168. Goldberger AS. Econometric computing by hand. *J Econ Soc Meas.* 2004;29(1-3):115–7.
169. Goodfellow I, Bengio Y, Courville A. *Deep Learning*. MIT Press; 2016. <http://www.deeplearningbook.org>.
170. Google. Cloud TPUs - ML accelerators for TensorFlow, Google Cloud Platform. 2017. <https://cloud.google.com/tpu/>. Accessed 01 Aug 2017.
171. Gönitz N, Kloft M, Rieck K, Brefeld U. Active learning for network intrusion detection. In: *Proceedings of the 2nd ACM workshop on Security and artificial intelligence*. New York: ACM; 2009. p. 47–54.
172. Gu Y, Grossman R. Sabul: A transport protocol for grid computing. *J Grid Comput.* 2003;1(4):377–86.
173. Guyon I, Elisseeff A. An introduction to variable and feature selection. *J Mach Learn Res.* 2003;1157–82.
174. Ha S, Rhee I, Xu L. Cubic: A new tcp-friendly high-speed tcp variant. *SIGOPS Oper Syst Rev.* 2008;42(5):64–74.
175. Habib I, Tarraf A, Saadawi T. A neural network controller for congestion control in atm multiplexers. *Comput Netw ISDN Syst.* 1997;29(3):325–34.
176. Haffner P, Sen S, Spatscheck O, Wang D. Acas: automated construction of application signatures. In: *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*. New York: ACM; 2005. p. 197–202.
177. Haining W. An economical broadband network with high added value. In: *Evolving the Access Network*, International Engineering Consortium. Chicago: International Engineering Consortium; 2006. p. 67–70.
178. Hajji H. Statistical analysis of network traffic for adaptive faults detection. *IEEE Trans Neural Netw.* 2005;16(5):1053–63.
179. Hariri B, Sadati N. Nn-red: an aqm mechanism based on neural networks. *Electron Lett.* 2007;43(19):1053–5.
180. Harrison V, Pagliery J. Nearly 1 million new malware threats released every day. 2015. <http://money.cnn.com/2015/04/14/technology/security/cyber-attack-hacks-security/index.html>. Accessed 28 Dec 2017.
181. Hashmi US, Darbandi A, Imran A. Enabling proactive self-healing by data mining network failure logs. In: *Computing, Networking and Communications (ICNC), 2017 International Conference on*. Piscataway: IEEE; 2017. p. 511–7.
182. He L, Xu C, Luo Y. vtc: Machine learning based traffic classification as a virtual network function. In: *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. ACM; 2016. p. 53–56.
183. He Q, Shayman MA. Using reinforcement learning for proactive network fault management. In: *Proceedings of the International Conference on Communication Technologies*. 1999.
184. Hebb D. *The Organization of Behavior: A Neuropsychological Theory*. New York: Wiley; 1949.
185. Henderson T, Floyd S, Gurtov S, Nishida Y. sThe newreno modification to tcp's fast recovery algorithm. RFC 6582, Internet Engineering Task Force. 2012. <https://tools.ietf.org/html/rfc6582>.
186. Hinton GE. Training products of experts by minimizing contrastive divergence. *Training.* 2006;14(8).
187. Hinton GE, McClelland JL, Rumelhart DE. *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1. In: Rumelhart DE, McClelland JL, PDP Research Group C, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA, USA, chap Distributed Representations: MIT Press; 1986. p. 77–109.
188. Hinton GE, Osindero S, Teh YW. A fast learning algorithm for deep belief nets. *Neural Comput.* 2006;18(7):1527–54.
189. Hiramatsu A. Atm communications network control by neural networks. *IEEE Trans Neural Netw.* 1990;1(1):122–30.
190. Hiramatsu A. Integration of atm call admission control and link capacity control by distributed neural networks. *IEEE J Sel Areas Commun.* 1991;9(7):1131–8.
191. Ho TK. Random decision forests. In: *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, IEEE Computer Society, Washington, DC, USA, ICDAR '95. Piscataway: IEEE; 1995. p. 278.
192. Hochreiter S, Schmidhuber J. Long short-term memory. *Neural Comput.* 1997;9(8):1735–80.
193. Hood CS, Ji C. Proactive network-fault detection. *IEEE Trans Reliab.* 1997;46(3):333–41.
194. de Hoon M, Imoto S, Nolan J, Miyano S. Open source clustering software. *Bioinformatics.* 2004;20(9):1453–4.
195. Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. *Proc Natl Acad Sci.* 1982;79(8):2554–8. <http://www.pnas.org/content/79/8/2554.full.pdf>.
196. Hornik K. Approximation capabilities of multilayer feedforward networks. *Neural Netw.* 1991;4(2):251–7.
197. Hu T, Fei Y. Qlar: a machine-learning-based adaptive routing protocol for energy-efficient and lifetime-extended underwater sensor networks. *IEEE Trans Mob Comput.* 2010;9(6):796–809.
198. Hu W, Hu W, Maybank S. Adaboost-based algorithm for network intrusion detection. *IEEE Transactions on Systems, Man, and Cybernetics. Part B (Cybernetics)*. 2008;38(2):577–83.
199. Huang YS, Suen CY. A method of combining multiple experts for the recognition of unconstrained handwritten numerals. *IEEE Trans Pattern Anal Mach Intell.* 1995;17(1):90–4.
200. Hubel DH, Wiesel TN. Receptive fields of single neurones in the cat's striate cortex. *J Physiol.* 1959;148(3):574–91.
201. Hubel DH, Wiesel TN. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J Physiol.* 1962;160(1):106–54.
202. IMPACT Cyber Trust. Information Marketplace for Policy and Analysis of Cyber-risk and Trust. 2017. <https://www.impactcybertrust.org>. Accessed 01 Aug 2017.
203. Information Sciences Institute. The network simulator ns-2. 2014. <http://www.isi.edu/nsnam/ns/>. Accessed 10 Oct 2017.
204. Ingham KL, Inoue H. Comparing anomaly detection techniques for http. In: *International Workshop on Recent Advances in Intrusion Detection (RAID)*. Berlin: Springer; 2007. p. 42–62.
205. Intanagonwiwat C, Govindan R, Estrin D, Heidemann J, Silva F. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans Netw (ToN)*. 2003;11(1):2–16.
206. International Telecommunications Union. G.107 : The E-model: a computational model for use in transmission planning. 2008. <https://www.itu.int/rec/T-REC-G.107>. Accessed 28 Dec 2017.
207. Internet Assigned Numbers Authority. IANA. 2017. <https://www.iana.org/>. Accessed 01 Aug 2017.
208. Internet Engineering TaskForce. n Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. 2002. <https://tools.ietf.org/html/rfc3411>. Accessed 01 Aug 2017.
209. Internet Engineering Task Force. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. 2008. <https://tools.ietf.org/html/rfc5101>. Accessed 01 Aug 2017.
210. Ivakhnenko A, Lapa V, ENGINEERING PULISOE. *Cybernetic Predicting Devices*. Purdue University School of Electrical Engineering; 1965.

211. Iyengar J, Swett I. QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. Tech. rep. Network Working Group; 2015. <https://tools.ietf.org/pdf/draft-tsvwg-quic-protocol-00.pdf>.
212. Jain A, Karandikar A, Verma R. An adaptive prediction based approach for congestion estimation in active queue management (apace). In: Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE, vol. 7. Piscataway: IEEE; 2003. p. 4153–7.
213. Javaid A, Niyaz Q, Sun W, Alam M. A deep learning approach for network intrusion detection system. In: Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS), ICST, (Institute for Computer Sciences Social-Informatics and Telecommunications Engineering). Brussels; 2016. p. 21–6.
214. Jayaraj A, Venkatesh T, Murthy CSR. Loss classification in optical burst switching networks using machine learning techniques: improving the performance of tcp. *IEEE J Sel Areas Commun*. 2008;26(6):45–54.
215. Jaynes ET. Information theory and statistical mechanics. *Phys Rev*. 1957a;106:620–30.
216. Jaynes ET. Information theory and statistical mechanics. ii. *Phys Rev*. 1957b;108:171–90.
217. Jennings A. A learning system for communications network configuration. *Eng Appl Artif Intell*. 1988;1(3):151–60.
218. Jiang H, Moore AW, Ge Z, Jin S, Wang J. Lightweight application classification for network management. In: Proceedings of the 2007 SIGCOMM workshop on Internet network management. ACM; 2007. p. 299–304.
219. Jiang H, Luo Y, Zhang Q, Yin M, Wu C. Tcp-gvegas with prediction and adaptation in multi-hop ad hoc networks. *Wirel Netw*. 2017;23(5):1535–48.
220. Jiang S, Song X, Wang H, Han JJ, Li QH. A clustering-based method for unsupervised intrusion detections. *Pattern Recog Lett*. 2006;27(7):802–10.
221. Jin Y, Duffield N, Haffner P, Sen S, Zhang ZL. Inferring applications at the network layer using collective traffic statistics. In: *Teletraffic Congress (ITC), 2010 22nd International*. IEEE; 2010. p. 1–8.
222. Jin Y, Duffield N, Erman J, Haffner P, Sen S, Zhang ZL. A modular machine learning system for flow-level traffic classification in large networks. *ACM Trans Knowl Discov Data (TKDD)*. 2012;6(1):4.
223. Jing N, Yang M, Cheng S, Dong Q, Xiong H. An efficient svm-based method for multi-class network traffic classification. In: *Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International*. IEEE; 2011. p. 1–8.
224. Joachims T. SVMlight. 1999. <http://svmlight.joachims.org/>. Accessed 28 Dec 2017.
225. Johnsson A, Meirosu C. Towards automatic network fault localization in real time using probabilistic inference. In: *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. Piscataway: IEEE; 2013. p. 1393–8.
226. Jordan MI. Serial order: A parallel distributed processing approach. Tech. Rep. ICS Report 8604. San Diego: University of California; 1986.
227. Juniper Research. Cybercrime will cost businesses over \$2 trillion by 2019. 2015. <https://www.juniperresearch.com/press/press-releases/cybercrime-cost-businesses-over-2trillion>. Accessed 10 Nov 2017.
228. Karagiannis T, Broido A, Brownlee N, Claffy KC, Faloutsos M. Is p2p dying or just hiding? [p2p traffic measurement]. In: *IEEE Global Telecommunications Conference (GLOBECOM), vol. 3*. 2004. p. 1532–8.
229. Karagiannis T, Papagiannaki K, Faloutsos M. BLINC: multilevel traffic classification in the dark. *ACM SIGCOMM Comput Commun Rev*. 2005;35(4):229–40.
230. Karami A. Accpndn: Adaptive congestion control protocol in named data networking by learning capacities using optimized time-lagged feedforward neural network. *J Netw Comput Appl*. 2015;56(Supplement C):1–18.
231. Lab Kaspersky. Damage control: The cost of security breaches. IT security risks special report series Report, Kaspersky. 2015. <https://media.kaspersky.com/pdf/it-risks-survey-report-cost-of-security-breaches.pdf>. Accessed 10 Nov 2017.
232. Kayacik HG, Zincir-Heywood AN, Heywood MI. On the capability of an SOM-based intrusion detection system. In: *Proceedings of the International Joint Conference on Neural Networks*. New York: IEEE; 2003. p. 1808–13.
233. Kelley HJ. Gradient theory of optimal flight paths. *ARS J*. 1960;30(10):947–54.
234. Kennedy J, Eberhart R. Particle swarm optimization. In: *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4. 1995. p. 1942–8.
235. Khan A, Sun L, Ifeachor E. Content-based video quality prediction for mpeg4 video streaming over wireless networks. *J Multimedia*. 2009a;4(4).
236. Khan A, Sun L, Ifeachor E. Content clustering based video quality prediction model for mpeg4 video streaming over wireless networks. *IEEE*; 2009b, pp. 1–5.
237. Khamfer RM, Solana B, Triola J, Barco R, Moltsen L, Altman Z, Lazaro P. Automated diagnosis for umts networks using bayesian network approach. *IEEE Trans veh technol*. 2008;57(4):2451–61.
238. Khayat IE, Geurts P, Leduc G. Machine-learned versus analytical models of tcp throughput. *Comput Netw*. 2007;51(10):2631–44.
239. Khorsandroo S, Noor RM, Khorsandroo S. The role of psychophysics laws in quality of experience assessment: a video streaming case study. In: *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*. ACM; 2012. p. 446–52.
240. Khorsandroo S, Md Noor R, Khorsandroo S. A generic quantitative relationship to assess interdependency of qoe and qos. *KSII Trans Internet Inf Syst*. 2013;7(2).
241. Kiciman E, Fox A. Detecting application-level failures in component-based internet services. *IEEE Trans Neural Netw*. 2005;16(5):1027–41.
242. Kim DS, Nguyen HN, Park JS. Genetic algorithm to improve svm based network intrusion detection system. New York: IEEE; 2005, pp. 155–8.
243. Kim H, Fomenkov M, Claffy KC, Brownlee N, Barman D, Faloutsos M. Comparison of internet traffic classification tools. In: *IMRG Workshop on Application Classification and Identification*; 2007. p. 1–2.
244. Kim H, Claffy KC, Fomenkov M, Barman D, Faloutsos M, Lee K. Internet traffic classification demystified: myths, caveats, and the best practices. *ACM*; 2008, p. 11.
245. Kim J, Kim J, Thu HLT, Kim H. Long short term memory recurrent neural network classifier for intrusion detection. *International Conference on*. IEEE; 2016, pp. 1–5.
246. Klaine PV, Imran MA, Onireti O, Souza RD. A survey of machine learning techniques applied to self organizing cellular networks. *IEEE Commun Surv Tutor*. 2017;PP(99):1.
247. Kogeda OP, Agbinya JI, Omlin CW. A probabilistic approach to faults prediction in cellular networks. *IEEE*; 2006, p. 130.
248. Kogeda P, Agbinya J. Prediction of faults in cellular networks using bayesian network model. UTS ePress; 2006.
249. Kohonen T. Self-organized formation of topologically correct feature maps. *Biol Cybern*. 1982;43(1):59–69.
250. Kohonen T, Hynninen J, Kangas J, Laaksonen J. Som pak: The self-organizing map program package. Report A31, Helsinki University of Technology, Laboratory of Computer and Information Science. 1996.
251. Kotz D, Henderson T, Abyzov I, Yeo J. CRAWDAD dataset dartmouth/campus (v. 2009-09-09). 2009. <https://crawdad.org/dartmouth/campus/20090909>. Accessed 28 Dec 2017.
252. Kruegel C, Toth T. Using decision trees to improve signature-based intrusion detection. In: *Recent Advances in Intrusion Detection*. Springer; 2003. p. 173–91.
253. Kumano Y, Ata S, Nakamura N, Nakahira Y, Oka I. Towards real-time processing for application identification of encrypted traffic. In: *International Conference on Computing, Networking and Communications (ICNC)*; 2014. p. 136–40.
254. Kumar S, Miikkulainen R. Dual reinforcement q-routing: An on-line adaptive routing algorithm. In: *Proceedings of the artificial neural networks in engineering Conference*. 1997. p. 231–8.
255. Kumar Y, Farooq H, Imran A. Fault prediction and reliability analysis in a real cellular network. In: *Wireless Communications and Mobile Computing Conference (IWCMC), 2017 13th International*. IEEE; 2017. p. 1090–1095.
256. Labs ML. Kdd cup 1998 data. 1998. <https://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html>. Accessed 28 Dec 2017.
257. Labs ML. Kdd cup 1999 data. 1999. <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>. Accessed 28 Dec 2017.
258. Lagoudakis MG, Parr R. Model-free least-squares policy iteration. In: *Advances in neural information processing systems*. 2002. p. 1547–54.
259. Lal TN, Chapelle O, Weston J, Elisseeff A. Embedded methods. In: *Feature extraction*. Springer; 2006. p. 137–65.
260. Lapedes AS, Farber RM. How neural nets work. 1987.

261. Laplace PS. *Théorie analytique des probabilités*. Paris: Courcier; 1812.
262. Lawrence Berkeley National Laboratory and ICSI. LBNL/ICSI Enterprise Tracing Project. 2005. <http://www.icir.org/enterprise-tracing/>. Accessed 01 Aug 2017.
263. Le Cun Y. *Learning Process in an Asymmetric Threshold Network*. Berlin, Heidelberg: Springer Berlin Heidelberg; 1986. pp. 233–40.
264. Lee SJ, Hou CL. A neural-fuzzy system for congestion control in atm networks. *IEEE Transactions on Systems, Man, and Cybernetics. Part B (Cybernetics)*. 2000;30(1):2–9.
265. Leela-Amornsri L, Esaki H. Heuristic congestion control for message deletion in delay tolerant network. In: *Proceedings of the Third Conference on Smart Spaces and Next Generation Wired, and 10th International Conference on Wireless Networking*, Springer-Verlag, Berlin, Heidelberg, ruSMART/NEW2AN'10; 2010. p. 287–98.
266. Legendre A. *Nouvelles méthodes pour la détermination des orbites des comètes*. Nineteenth Century Collections Online (NCCO): Science, Technology, and Medicine: 1780–1925, F. Didot. 1805.
267. Lemaître G, Nogueira F, Aridas CK. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *J Mach Learn Res*. 2017;18(17):1–5.
268. Leonardi E, Mellia M, Horvath A, Muscariello L, Niccolini S, Rossi D, Young K. Building a cooperative p2p-tv application over a wise network: the approach of the european fp-7 strep napa-wine. *IEEE Commun Mag*. 2008;46(4):20–2.
269. Li F, Sun J, Zukerman M, Liu Z, Xu Q, Chan S, Chen G, Ko KT. A comparative simulation study of tcp/aqm systems for evaluating the potential of neuron-based aqm schemes. *J Netw Comput Appl*. 2014;41(Supplement C):274–99.
270. Li W, Moore AW. A machine learning approach for efficient traffic classification. In: *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007. MASCOTS'07. 15th International Symposium on*. IEEE; 2007. p. 310–7.
271. Li W, Zhou F, Meleis W, Chowdhury K. Learning-based and data-driven tcp design for memory-constrained iot. In: *2016 International Conference on Distributed Computing in Sensor Systems (DCOSS)*. 2016a. p. 199–205.
272. Li Y, Guo L. An active learning based tcm-knn algorithm for supervised network intrusion detection. *Comput Secur*. 2007;26(7):459–67.
273. Li Y, Ma R, Jiao R. A hybrid malicious code detection method based on deep learning. *Methods*. 2015;9(5).
274. Li Y, Liu H, Yang W, Hu D, Xu W. Inter-data-center network traffic prediction with elephant flows. *IEEE*; 2016b, pp. 206–13.
275. Lin LJ. *Reinforcement learning for robots using neural networks*. PhD thesis. Pittsburgh, PA, USA: Carnegie Mellon University; 1992. uMI Order No. GAX93-22750.
276. Lin SC, Akyildiz IF, Wang P, Luo M. Qos-aware adaptive routing in multi-layer hierarchical software defined networks: a reinforcement learning approach. In: *Services Computing (SCC) 2016, IEEE International Conference on*. IEEE; 2016. p. 25–33.
277. Lin Z, van der Schaar M. Autonomic and distributed joint routing and power control for delay-sensitive applications in multi-hop wireless networks. *IEEE Trans Wirel Commun*. 2011;10(1):102–13.
278. Lincoln Laboratory MIT. DARPA Intrusion Detection Evaluation. 1999. <https://www.ll.mit.edu/ideval/data/1999data.html>. Accessed 01 Aug 2017.
279. Littlestone N, Warmuth MK. The weighted majority algorithm. In: *30th Annual Symposium on Foundations of Computer Science*. 1989. p. 256–61.
280. Littman M, Boyan J. A distributed reinforcement learning scheme for network routing. In: *Proceedings of the international workshop on applications of neural networks to telecommunications*. Psychology Press; 1993. p. 45–51.
281. Liu D, Zhang Y, Zhang H. A self-learning call admission control scheme for cdma cellular networks. *IEEE trans neural netw*. 2005;16(5):1219–28.
282. Liu J, Matta I, Crovella M. End-to-end inference of loss nature in a hybrid wired/wireless environment. 2003.
283. Liu Y, Li W, Li YC. Network traffic classification using k-means clustering. *IEEE*; 2007. pp. 360–5.
284. Liu YC, Douligeris C. Static vs. adaptive feedback congestion controller for atm networks. In: *Global Telecommunications Conference, 1995. GLOBECOM '95*, vol 1. IEEE; 1995. p. 291–5.
285. Lu X, Wang H, Zhou R, Ge B. Using hessian locally linear embedding for autonomic failure prediction. In: *Nature & Biologically Inspired, Computing, 2009. NaBIC 2009. World Congress on*. IEEE; 2009. p. 772–6.
286. Ma J, Levchenko K, Kreibich C, Savage S, Voelker GM. Unexpected means of protocol inference. In: *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. 2006. p. 313–26.
287. Machado VA, Silva CN, Oliveira RS, Melo AM, Silva M, Francês CR, Costa JC, Vijaykumar NL, Hirata CM. A new proposal to provide estimation of qos and qoe over wimax networks: An approach based on computational intelligence and discrete-event simulation. *IEEE*; 2011, pp. 1–6.
288. Machine Learning Group, University of Waikato. WEKA. 2017. <http://www.cs.waikato.ac.nz/ml/weka/>. Accessed 01 Aug 2017.
289. Macleish KJ. Mapping the integration of artificial intelligence into telecommunications. *IEEE J Sel Areas Commun*. 1988;6(5):892–8.
290. MacQueen J. Some methods for classification and analysis of multivariate observations. In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, University of California Press, Berkeley, 1; 1967. p. 281–97.
291. Mahmoud Q. *Cognitive Networks: Towards Self-Aware Networks*. Wiley-Interscience; 2007.
292. Malware-Traffic-Analysis.net. A source for pcap files and malware samples. 2017. <http://www.malware-traffic-analysis.net>. Accessed 15 Dec 2017.
293. Manzoor J, Drago I, Sadre R. The curious case of parallel connections in http/2. In: *12th International Conference on Network and Service Management (CNSM)*. 2016. p. 174–80.
294. Mao H, Alizadeh M, Menache I, Kandula S. Resource management with deep reinforcement learning. In: *HotNets*. 2016. p. 50–6.
295. Marbach P, Mihatsch O, Tsitsiklis JN. Call admission control and routing in integrated services networks using neuro-dynamic programming. *IEEE J Sel areas commun*. 2000;18(2):197–208.
296. Markov AA. An example of statistical investigation in the text of eugene onegin illustrating coupling of tests in chains. In: *Proceedings of the Royal Academy of Sciences of St. Petersburg, St. Petersburg, Rusia*, vol 1. 1913. p. 153.
297. Maron ME. Automatic indexing: An experimental inquiry. *J ACM*. 1961;8(3):404–17.
298. Masoumzadeh SS, Taghizadeh G, Meshgi K, Shiry S. Deep blue: A fuzzy q-learning enhanced active queue management scheme. In: *2009 International Conference on Adaptive and Intelligent Systems*; 2009. p. 43–8.
299. Mathis M, Mahdavi J, Floyd S, Romanow A. Tcp selective acknowledgment options. RFC 2018, Internet Engineering Task Force. 1996. <https://tools.ietf.org/html/rfc2018>.
300. Mathis M, Semke J, Mahdavi J, Ott T. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput Commun Rev*. 1997;27(3):67–82.
301. Maxion RA. Anomaly detection for diagnosis. In: *Fault-Tolerant Computing, 1990. FTCS-20. Digest of, Papers, 20th International Symposium*. IEEE; 1990. p. 20–7.
302. McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. *Bull Math Biophys*. 1943;5(4):115–33.
303. McGregor A, Hall M, Lorier P, Brunskill J. Flow clustering using machine learning techniques. *Passive and Active Netw Meas*. 2004;205–14.
304. McKenney PE. Stochastic fairness queueing. In: *INFOCOM '90, Ninth Annual Joint Conference of the IEEE Computer and Communication Societies. The Multiple Facets of Integration*. Proceedings. IEEE; 1990. p. 733–40. vol.2.
305. Messenger R, Mandell L. A modal search technique for predictive nominal scale multivariate analysis. *J Am Stat Assoc*. 1972;67(340):768–72.
306. Mestres A, Rodriguez-Natal A, Carner J, Barlet-Ros P, Alarcón E, Solé M, Muntés-Mulero V, Meyer D, Barkai S, Hibbett MJ, et al. Knowledge-defined networking. *ACM SIGCOMM Comput Commun Rev*. 2017;47(3):2–10.
307. Metasploit L. The metasploit framework. 2007. <http://www.metasploit.com>. Accessed 28 Dec 2017.
308. Meyer D. Machine Intelligence and Networks. 2016. <https://www.youtube.com/watch?v=XORRw6SqI9Y>. Accessed 28 Dec 2017.

309. Mezzavilla M, Quer G, Zorzi M. On the effects of cognitive mobility prediction in wireless multi-hop ad hoc networks. In: 2014 IEEE International Conference on Communications (ICC); 2014. p. 1638–44.
310. Microsoft Corporation. Skype. 2017. <https://www.skype.com/>. Accessed 01 Aug 2017.
311. Mignanti S, Di Giorgio A, Suraci V. A model based rl admission control algorithm for next generation networks. In: Networks, 2009. ICN'09. Eighth International Conference on. IEEE; 2009. p. 191–6.
312. Mijumbi R, Gorricho JL, Serrat J, Claeys M, De Turck F, Latré S. Design and evaluation of learning algorithms for dynamic resource management in virtual networks. IEEE; 2014, pp. 1–9.
313. Mijumbi R, Hasija S, Davy S, Davy A, Jennings B, Boutaba R. A connectionist approach to dynamic resource management for virtualised network functions. In: Network and Service Management (CNSM) 2016 12th International Conference on. IEEE; 2016. p. 1–9.
314. Miller ST, Busby-Earle C. Multi-perspective machine learning a classifier ensemble method for intrusion detection. In: Proceedings of the 2017 International Conference on Machine Learning and Soft Computing. ACM; 2017. p. 7–12.
315. Minsky M, Papert S. Perceptrons: An Introduction to Computational Geometry. Cambridge: MIT Press; 1972.
316. Mirza M, Sommers J, Barford P, Zhu X. A machine learning approach to tcp throughput prediction. IEEE/ACM Trans Netw. 2010;18(4):1026–39.
317. Mitchell TM. Machine Learning, 1st ed. New York: McGraw-Hill, Inc.; 1997.
318. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D. Human-level control through deep reinforcement learning. Nature. 2015;518(7540):529–33.
319. Montana DJ, Davis L. Training feedforward neural networks using genetic algorithms. In: Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, IJCAI'89. 1989. p. 762–7.
320. Moore AW, Papagiannaki K. Toward the accurate identification of network applications. In: PAM. Springer; 2005. p. 41–54.
321. Moore AW, Zuev D. Internet traffic classification using bayesian analysis techniques. In: ACM SIGMETRICS Performance Evaluation Review, ACM, vol 33. 2005. p. 50–60.
322. Moradi M, Zulkernine M. A neural network based system for intrusion detection and classification of attacks. In: Proceedings of the IEEE International Conference on Advances in Intelligent Systems-Theory and Applications. 2004. p. 15–8.
323. Morgan JN, Sonquist JA. Problems in the analysis of survey data, and a proposal. J Am Stat Assoc. 1963;58(302):415–34.
324. Moustapha AI, Selmic RR. Wireless sensor network modeling using modified recurrent neural networks: Application to fault detection. IEEE Trans Instrum Meas. 2008;57(5):981–8.
325. Mukkamala S, Janoski G, Sung A. Intrusion detection using neural networks and support vector machines. 2002, pp. 1702–7.
326. Mukkamala S, Sung AH, Abraham A. Intrusion detection using ensemble of soft computing paradigms. In: Intelligent systems design and applications. Springer; 2003. p. 239–48.
327. Muniyandi AP, Rajeswari R, Rajaram R. Network anomaly detection by cascading k-means clustering and c4. 5 decision tree algorithm. Procedia Eng. 2012;30:174–82.
328. Mushtaq MS, Augustin B, Mellouk A. Empirical study based on machine learning approach to assess the qos/qoe correlation. In: Networks and Optical Communications (NOC), 2012 17th European Conference on. IEEE; 2012. p. 1–7.
329. Nahm K, Helmy A, Jay Kuo CC. Tcp over multihop 802.11 networks: Issues and performance enhancement. In: Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing, ACM, New York, NY, USA, MobiHoc '05; 2005. p. 277–87.
330. Narendra KS, Thathachar MAL. Learning automata - a survey. IEEE Transactions on Systems. Man Cybern SMC-. 1974;4(4):323–34.
331. Netflix Inc. Netflix. 2017. <https://www.netflix.com/>. Accessed 01 Aug 2017.
332. Networks and Mobile Systems Group. Resilient overlay networks RON. 2017. <http://nms.csail.mit.edu/ron/>. Accessed 27 Dec 2017.
333. Ng AY, Jordan MI. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In: Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, MIT Press, Cambridge, MA, USA, NIPS'01. 2001. p. 841–8. <http://dl.acm.org/citation.cfm?id=2.980539.2980648>.
334. Ng B, Hayes M, Seah WKG. Developing a traffic classification platform for enterprise networks with sdn: Experiences and lessons learned. In: IFIP Networking Conference. 2015. p. 1–9.
335. Nguyen T, Armitage G. Synthetic sub-flow pairs for timely and stable ip traffic identification. In: Proc. Australian Telecommunication Networks and Application Conference. 2006a.
336. Nguyen TT, Armitage G. Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world ip networks. In: Local Computer Networks, Proceedings 2006 31st IEEE Conference on. IEEE; 2006b. p. 369–76.
337. Nguyen TT, Armitage G, Branch P, Zander S. Timely and continuous machine-learning-based classification for interactive ip traffic. IEEE/ACM Trans Netw (TON). 2012;20(6):1880–94.
338. Nguyen TTT, Armitage G. Clustering to assist supervised machine learning for real-time ip traffic classification. In: IEEE International Conference on Communications. 2008a. p. 5857–62.
339. Nguyen TTT, Armitage G. A survey of techniques for internet traffic classification using machine learning. IEEE Commun Surv Tutor. 2008b;10(4):56–76.
340. Nichols K, Jacobson V. Controlling queue delay. Queue. 2012;10(5): 20:20–20:34.
341. NMapWin. Nmap security scanner. 2016. <http://nmapwin.sourceforge.net/>. Accessed 1 Mar 2017.
342. NVIDIA. Graphics Processing Unit (GPU). 2017. <http://www.nvidia.com/object/gpu.html>. Accessed 01 Aug 2017.
343. Padhye J, Firoiu V, Towsley D, Kurose J. Modeling tcp throughput: A simple model and its empirical validation. In: Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM, New York, SIGCOMM '98. 1998. p. 303–14.
344. Pan ZS, Chen SC, Hu GB, Zhang DQ. Hybrid neural network and c4.5 for misuse detection. 2003, pp. 2463–7.
345. Panda M, Abraham A, Patra MR. A hybrid intelligent approach for network intrusion detection. Procedia Eng. 2012;30:1–9.
346. Papernot N, Goodfellow I, Shethsley R, Feinman R, McDaniel P. cleverhans v1.0.0: an adversarial machine learning library. 2016. arXiv preprint arXiv:161000768.
347. Park J, Tyan HR, Kuo CCJ. Internet traffic classification for scalable qos provision. In: Multimedia and Expo, vol 2006 IEEE International Conference on. IEEE; 2006. p. 1221–4.
348. Parkour M. Pcap traffic patterns. 2013. <http://www.mediafire.com/?a49l965nlayad>. Accessed 1 Mar 2017.
349. Parzen E. On estimation of a probability density function and mode. Ann Math Statist. 1962;33(3):1065–76.
350. Paxson V. Bro: A system for detecting network intruders in real-time. Comput Netw. 1999;31(23-24):2435–63.
351. Pcap-Analysis. Malware. 2017. <http://www.pcapanalysis.com>. Accessed 1 Mar 2017.
352. Pearl J. Bayesian networks: A model of self-activated memory for evidential reasoning. Irvine: University of California; 1985, pp. 329–34.
353. Pearl J. Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann; 2014.
354. Peddabachigari S, Abraham A, Grosan C, Thomas J. Modeling intrusion detection system using hybrid intelligent systems. J netw comput appl. 2007;30(1):114–32.
355. Pellegrini A, Di Sanzo P, Avresky DR. A machine learning-based framework for building application failure prediction models. In: Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International. IEEE; 2015. p. 1072–81.
356. Perdisci R, Ariu D, Fogla P, Giacinto G, Lee W. Mcpad: A multiple classifier system for accurate payload-based anomaly detection. Comput netw. 2009;53(6):864–81.
357. Petrangeli S, Claeys M, Latré S, Famaey J, De Turck F. A multi-agent q-learning-based framework for achieving fairness in http adaptive streaming. IEEE; 2014, pp. 1–9.
358. Pfahringer B. Winning the kdd99 classification cup: bagged boosting. ACM SIGKDD Explor Newsl. 2000;1(2):65–6.
359. Piamrat K, Ksentini A, Viho C, Bonnin JM. Qoe-aware admission control for multimedia applications in ieee 802.11 wireless networks. In:

- Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th. IEEE; 2008. p. 1–5.
360. Pietrabissa A. Admission control in umts networks based on approximate dynamic programming. *Eur J control*. 2008;14(1):62–75.
361. Pietrabissa A, Prisco FD, Di Giorgio A, Giuseppe A, Panfil M, Suraci V. An approximate dynamic programming approach to resource management in multi-cloud scenarios. *Int J Control*. 2017;90(3):492–503.
362. Pinson MH, Wolf S. A new standardized method for objectively measuring video quality. *IEEE Trans broadcast*. 2004;50(3):312–22.
363. Portnoy L, Eskin E, Stolfo S. Intrusion detection with unlabeled data using clustering. In: *Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*. Citeseer; 2001.
364. Portoles-Comeras M, Requena-Esteso M, Mangues-Bafalluy J, Cardenete-Suriol M. Extreme: Combining the ease of management of multi-user experimental facilities and the flexibility of proof of concept testbeds. In: *Testbeds and Research Infrastructures for the Development of Networks and Communities, s2006. TRIDENTCOM 2006 2nd International Conference on*. IEEE; 2006. p. 10.
365. Poupart P, Chen Z, Jaini P, Fung F, Susanto H, Geng Y, Chen L, Chen K, Jin H. Online flow size prediction for improved network routing. *IEEE*; 2016, pp. 1–6.
366. Prechelt L. Early stopping-but when? *Neural Netw: Tricks of the trade*. 1998;553.
367. Prevost JJ, Nagothu K, Kelley B, Jamshidi M. Prediction of cloud data center networks loads using stochastic and neural models. *IEEE*; 2011. p. 276–281.
368. Proactcouk. ISS Internet Scanner. 2017. http://www.tech.proact.co.uk/iss/iss_system_scanner.htm. Accessed 28 Dec 2017.
369. Puget JF. What is machine learning? (IT best kept secret is optimization). 2016. https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Is_Machine_Learning. Accessed 01 Aug 2017.
370. Qader K, Adda M. Fault classification system for computer networks using fuzzy probabilistic neural network classifier (fpnnc) *International Conference on Engineering Applications of Neural Networks*. Springer; 2014. p. 217–26.
371. Quer G, Meenakshisundaram H, Tamma B, Manoj BS, Rao R, Zorzi M. Cognitive network inference through bayesian network analysis. 2010, pp. 1–6.
372. Quer G, Baldo N, Zorzi M. Cognitive call admission control for voip over ieee 802.11 using bayesian networks. In: *Global Telecommunications Conference (GLOBECOM 2011)*, 2011 IEEE. IEEE; 2011. p. 1–6.
373. Quinlan J. *Discovering rules form large collections of examples: A case study*. Edinburgh: Expert Systems in the Micro Electronic Age Edinburgh Press; 1979.
374. Quinlan JR. Simplifying decision trees. *Int J Man-Mach Stud*. 1987;27(3):221–34.
375. Quinlan JR. *Learning with continuous classes*. In: *Proceedings of Australian Joint Conference on Artificial Intelligence*, World Scientific. 1992. p. 343–8.
376. Quinlan JR. *C4.5: Programs for Machine Learning*. San Francisco: Morgan Kaufmann Publishers Inc.; 1993.
377. Quinlan JR, et al, Vol. 1. Bagging, boosting, and c4. 5; 1996. 725–30.
378. Raina R, Battle A, Lee H, Packer B, Ng AY. Self-taught learning: transfer learning from unlabeled data. In: *Proceedings of the 24th international conference on Machine learning*. ACM; 2007. p. 759–66.
379. Ramana BV, Murthy CSR. Learning-tcp: A novel learning automata based congestion window updating mechanism for ad hoc wireless networks. In: *Proceedings of the 12th International Conference on High Performance Computing*, Springer-Verlag, Berlin, Heidelberg, HiPC'05. 2005. p. 454–464.
380. Ramana BV, Manoj BS, Murthy CSR. Learning-tcp: a novel learning automata based reliable transport protocol for ad hoc wireless networks. In: *2nd International Conference on Broadband Networks 2005*. 2005. p. 484–493. Vol. 1.
381. Ranzato M, Poultney C, Chopra S, LeCun Y. Efficient learning of sparse representations with an energy-based model. In: *Proceedings of the 19th International Conference on Neural Information Processing Systems*, MIT Press, Cambridge, MA, USA, NIPS'06; 2006. p. 1137–44.
382. Rao S. Operational fault detection in cellular wireless base-stations. *IEEE Trans Netw Serv Manag*. 2006;3(2):1–11.
383. Reichl P, Egger S, Schatz R, D'Alconzo A. The logarithmic nature of qoe and the role of the weber-fechner law in qoe assessment. *IEEE*; 2010, pp. 1–5.
384. Riiser H, Endestad T, Vigmostad P, Griwodz C, Halvorsen P. DATASET: HSDPA-bandwidth logs for mobile HTTP streaming scenarios. 2011. <http://home.ifi.uio.no/paalh/dataset/hsdpa-tcp-logs/>. Accessed 28 Dec 2017.
385. Riiser H, Endestad T, Vigmostad P, Griwodz C, Halvorsen P. Video streaming using a location-based bandwidth-lookup service for bitrate planning. *ACM Trans Multimedia Comput Commun Appl*. 2012;8(3):24:1–24:19. <https://doi.org/10.1145/2.240136.2240137>, <http://doi.acm.org/10.1145/2.240136.2240137>.
386. Rix AW, Beerends JG, Hollier MP, Hekstra AP. Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs. In: *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01)*. 2001 IEEE International Conference on, IEEE, vol 2. 2001. p. 749–52.
387. Rosenblatt F. The perceptron, a perceiving and recognizing automaton. Report No. 85-460-1 Project PARA: Cornell Aeronautical Laboratory; 1957.
388. Rosenblatt M. Remarks on some nonparametric estimates of a density function. *Ann Math Statist*. 1956;27(3):832–837.
389. Ross DA, Lim J, Lin RS, Yang MH. Incremental learning for robust visual tracking. *Int J Comput Vision*. 2008;77(1-3):125–141.
390. Roughan M, Sen S, Spatscheck O, Duffield N. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In: *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM; 2004a. p. 135–148.
391. Roughan M, Zhang Y, Ge Z, Greenberg A. Abilene network; 2004b. <http://www.maths.adelaide.edu.au/matthew.roughan/data/Abilene.tar.gz>. Accessed 28 Dec 2017.
392. Rozhnova N, Fdida S. An effective hop-by-hop interest shaping mechanism for ccn communications. In: *2012 Proceedings IEEE INFOCOM Workshops*; 2012. p. 322–327.
393. Ruiz M, Fresi F, Vela AP, Meloni G, Sambo N, Cugini F, Poti L, Velasco L, Castoldi P. Service-triggered failure identification/localization through monitoring of multiple parameters. In: *ECOC 2016; 42nd European Conference on Optical Communication: Proceedings of VDE*; 2016. p. 1–3.
394. Rumelhart DE, Hinton GE, Williams RJ. *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1. In: Rumelhart DE, McClelland JL, PDP Research Group C, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol 1. Cambridge, MA, USA, chap Learning Internal Representations by Error Propagation: MIT Press; 1986. p. 318–62.
395. Rumery GA, Niranjana M. On-line Q-learning using connectionist systems. CUED/F-INFENG/TR 166, Cambridge University Engineering Department. 1994.
396. Rüping S. mySVM. 2004. <http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/>. Accessed 28 Dec 2017.
397. Russell S, Norvig P. *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River: Prentice Hall Press; 2009.
398. Sabhnani M, Serpen G. Why machine learning algorithms fail in misuse detection on kdd intrusion detection data set. *Intell data anal*. 2004;8(4):403–15.
399. Salakhutdinov R, Hinton G. Deep boltzmann machines. In: *Artificial Intelligence and Statistics*; 2009. p. 448–55.
400. Salama M, Eid H, Ramadan R, Darwish A, Hassanien A. Hybrid intelligent intrusion detection scheme. *Soft comput ind appl*. 2011;293–303.
401. Samuel AL. Some studies in machine learning using the game of checkers. *IBM J Res Dev*. 1959;3(3):210–29.
402. Sangkatsanee P, Wattanapongsakorn N, Charnsripinyo C. Practical real-time intrusion detection using machine learning approaches. *Comput Commun*. 2011;34(18):2227–35.
403. Schapire RE. The strength of weak learnability. *Mach Learn*. 1990;5(2):197–227.
404. Schatzmann D, Mühlbauer W, Spyropoulos T, Dimitropoulos X. Digging into https: Flow-based classification of webmail traffic. In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*; 2010. p. 322–27.
405. Schölkopf B, Platt JC, Shawe-Taylor J, Smola AJ, Williamson RC. Estimating the support of a high-dimensional distribution. *Neural comput*. 2001;13(7):1443–71.

406. Seligman M, Fall K, Mundur P. Alternative custodians for congestion control in delay tolerant networks. In: Proceedings of the 2006 SIGCOMM Workshop on Challenged Networks, CHANTS '06. New York: ACM; 2006. p. 229–36.
407. Servin A, Kudenko D. Multi-agent reinforcement learning for intrusion detection: A case study and evaluation. In: German Conference on Multiagent System Technologies. Springer; 2008. p. 159–70.
408. Shaikh J, Fiedler M, Collange D. Quality of experience from user and network perspectives. *annals of telecommun-annales des telecommun.* 2010;65(1-2):47–57.
409. Shbair WM, Cholez T, Francois J, Chrisment I. A multi-level framework to identify https services. In: IEEE/IFIP Network Operations and Management Symposium (NOMS) 2016. p. 240–8.
410. Shi R, Zhang J, Chu W, Bao Q, Jin X, Gong C, Zhu Q, Yu C, Rosenberg S. Mdp and machine learning-based cost-optimization of dynamic resource allocation for network function virtualization. In: Serv Comput (SCC) 2015 IEEE International Conference on. IEEE; 2015. p. 65–73.
411. Shon T, Moon J. A hybrid machine learning approach to network anomaly detection. *Inf Sci.* 2007;177(18):3799–821.
412. Silva AP, Obraczka K, Burleigh S, Hirata CM. Smart congestion control for delay- and disruption tolerant networks. In: 2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON). 2016. p. 1–9.
413. Smolensky P. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. In: Rumelhart DE, McClelland JL, PDP Research Group C, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*. MIT Press, Cambridge, MA, USA, chap Information Processing in Dynamical Systems: Foundations of Harmony Theory. 1986. p. 194–281.
414. Snow A, Rastogi P, Weckman G. Assessing dependability of wireless networks using neural networks. In: Military Communications Conference, 2005. MILCOM 2005. IEEE. IEEE; 2005. p. 2809–15.
415. Sommer R, Paxson V. Outside the closed world: On using machine learning for network intrusion detection. In: Security and Privacy (SP), 2010 IEEE Symposium on. IEEE; 2010. p. 305–16.
416. Sondik EJ. The optimal control of partially observable markov decision processes. PhD thesis. California: Stanford University; 1971.
417. Soysal M, Schmidt EG. Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison. *Perform Eval.* 2010;67(6):451–67.
418. Sprint. IP network performance. 2017. <https://www.sprint.net/performance/>. Accessed 28 Dec 2017.
419. Srihari SN, Kuebert EJ. Integration of hand-written address interpretation technology into the united states postal service remote computer reader system. In: Proceedings of the 4th International Conference on Document Analysis and Recognition, ICDAR '97. Washington: IEEE Computer Society; 1997. p. 892–6.
420. Stanfill C, Waltz D. Toward memory-based reasoning. *Commun ACM.* 1986;29(12):1213–28.
421. Stein G, Chen B, Wu AS, Hua KA. Decision tree classifier for network intrusion detection with ga-based feature selection. In: Proceedings of the 43rd annual Southeast regional conference-Volume 2. ACM; 2005. p. 136–41.
422. Steinhaus H. Sur la division des corp materiels en parties. *Bull Acad Polon Sci.* 1956;1:801–4.
423. Stigler SM. Gauss and the invention of least squares. *Ann Statist.* 1981;9(3):465–74.
424. Stone P. Tpot-rl applied to network routing. In: ICML. 2000. p. 935–42.
425. Stone P, Veloso M. Team-partitioned, opaque-transition reinforcement learning. In: Proceedings of the third annual conference on Autonomous Agents. ACM; 1999. p. 206–12.
426. Stratonovich RL. Conditional markov processes. *Theory Probab Appl.* 1960;5(2):156–78.
427. Sun J, Zukerman M. An adaptive neuron aqm for a stable internet. In: Proceedings of the 6th International IFIP-TC6 Conference on Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet, Springer-Verlag, Berlin, Heidelberg, NETWORKING'07. 2007. p. 844–54.
428. Sun J, Chan S, Ko Kt, Chen G, Zukerman M. Neuron pid: A robust aqm scheme. In: Proceedings of the Australian Telecommunication Networks and Applications Conference (ATNAC) 2006. 2006. p. 259–62.
429. Sun J, Chan S, Zukerman M. Iapi: An intelligent adaptive pi active queue management scheme. *Comput Commun.* 2012;35(18):2281–93.
430. Sun R, Tatsumi S, Zhao G. Q-map: A novel multicast routing method in wireless ad hoc networks with multiagent reinforcement learning. In: TENCON'02. Proceedings. 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering, vol 1. IEEE; 2002. p. 667–670.
431. Sun R, Yang B, Peng L, Chen Z, Zhang L, Jing S. Traffic classification using probabilistic neural networks. In: Natural computation (ICNC), 2010 sixth international conference on, vol. 4. IEEE; 2010. p. 1914–9.
432. Sun Y, Yin X, Jiang J, Sekar V, Lin F, Wang N, Liu T, Sinopoli B. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In: Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference. ACM; 2016. p. 272–85.
433. Sutton RS. Learning to predict by the methods of temporal differences. *Mach learn.* 1988;3(1):9–44.
434. Sutton RS, Barto AG. A temporal-difference model of classical conditioning. In: Proceedings of the ninth annual conference of the cognitive science society. Seattle, WA; 1987. p. 355–78.
435. Sutton RS, Barto AG. Reinforcement Learning: An Introduction, 2nd ed. MIT Press; 2016.
436. Tang TA, Mhamdi L, McLernon D, Zaidi SAR, Ghogho M. Deep learning approach for network intrusion detection in software defined networking. In: Wireless Networks and Mobile Communications (WINCOM), 2016 International Conference on. IEEE; 2016. p. 258–263.
437. Tarraf AA, Habib IW, Saadawi TN. Congestion control mechanism for atm networks using neural networks. In: Communications 1995. ICC '95 Seattle, 'Gateway to Globalization' 1995 IEEE International Conference on, vol 1. 1995. p. 206–10.
438. Tavallaee M, Bagheri E, Lu W, Ghorbani AA. A detailed analysis of the kdd cup 99 data set. *IEEE; 2009*, pp. 1–6.
439. Telecommunication Networks Group - Politecnico di Torino. Skype testbed traces, TSTAT - TCP Statistic and Analysis Tool. 2008. <http://tstat.tlc.polito.it/traces-skype.shtml>. Accessed 01 Aug 2017.
440. Tesauro G. Practical issues in temporal difference learning. *Mach Learn.* 1992;8(3):257–77.
441. Tesauro G. Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Comput.* 2007;11(1):22–30.
442. Tesauro G, et al. Online resource allocation using decompositional reinforcement learning. 2005. pp. 886–91.
443. Testolin A, Zanforlin M, De Grazia MDF, Munaretto D, Zanella A, Zorzi M. A machine learning approach to qoe-based video admission control and resource allocation in wireless systems. In: Ad Hoc Networking Workshop (MED-HOC-NET), 2014 13th Annual, Mediterranean. IEEE; 2014. p. 31–38.
444. TFreak. Smurf tool. 2003. www.phreak.org/archives/exploits/denial/smurf.c.
445. Tibshirani R. Regression shrinkage and selection via the lasso. *J R Stat Soc Ser B (Methodol).* 1996;58(1):267–288.
446. Tong H, Brown TX. Adaptive call admission control under quality of service constraints: a reinforcement learning solution. *IEEE J sel Areas Commun.* 2000;18(2):209–21.
447. Tsai RF, Hsu YF, Lin CY, Lin WY. Intrusion detection by machine learning: A review. *Expert Systems with Applications.* 2009;36(10):11,994–12,000.
448. Tsetlin M. Automaton Theory and Modeling of Biological Systems. Automaton Theory and Modeling of Biological Systems. Academic Press; 1973.
449. Turing AM. Computing machinery and intelligence. *Mind.* 1950;59(236):433–60.
450. UCI KDD Archive. 2005. <https://kdd.ics.uci.edu/>. Accessed 01 Aug 2017.
451. University of California San Diego Supercomputer Center. CAIDA: Center for Applied Internet Data Analysis. 2017. <http://www.caida.org>. Accessed 01 Aug 2017.
452. Vassis D, Kampouraki A, Belsis P, Skourlas C. Admission control of video sessions over ad hoc networks using neural classifiers. *IEEE; 2014*, pp. 1015–20.
453. Vega MT, Mocanu DC, Liotta A. Unsupervised deep learning for real-time assessment of video streaming services. *Multimedia Tools Appl.* 2017;1–25.
454. Vengerov D. A reinforcement learning approach to dynamic resource allocation. *Eng Appl Artif Intell.* 2007;20(3):383–90.

455. Viterbi A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans Inf Theory*. 1967;13(2): 260–9.
456. Wagner C, François J, Engel T, et al. Machine learning approach for ip-flow record anomaly detection. In: *International Conference on Research in Networking*. Springer; 2011. p. 28–39.
457. WAND Network Research Group. WITS: Waikato Internet Traffic Storage. 2017. <https://wand.net.nz/wits>. Accessed 01 Aug 2017.
458. Wang J, Qiu Y. A new call admission control strategy for lte femtocell networks. In: *2nd international conference on advances in computer science and engineering*. 2013.
459. Wang K, Stolfo SJ. Anomalous payload-based network intrusion detection. In: *RAID*, vol 4. Springer; 2004. p. 203–22.
460. Wang M, Cui Y, Wang X, Xiao S, Jiang J. Machine learning for networking: Workflow, advances and opportunities. *IEEE Netw*. 2018a;32(2):92–9.
461. Wang P, Wang T. Adaptive routing for sensor networks using reinforcement learning. In: *Computer and Information Technology, 2006. CIT'06. The Sixth IEEE International Conference on*. IEEE; 2006. p. 219.
462. Wang P, Lin SC, Luo M. A framework for qos-aware traffic classification using semi-supervised machine learning in sdns. In: *Services Computing (SCC), 2016 IEEE International Conference on*. IEEE; 2016. p. 760–5.
463. Wang R, Valla M, Sanadidi MY, Ng BKF, Gerla M. Efficiency/friendliness tradeoffs in TCP westwood. In: *Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications*. 2002. p. 304–11.
464. Wang R, Liu Y, Yang Y, Zhou X. Solving the app-level classification problem of p2p traffic via optimized support vector machines. In: *Intelligent Systems Design and Applications, 2006. ISDA'06. Sixth International Conference on*, IEEE, vol 2; 2006. p. 534–9.
465. Wang X, Zhang Q, Ren J, Xu S, Wang S, Yu S. Toward efficient parallel routing optimization for large-scale sdn networks using gpgpu. *J Netw Comput Appl*. 2018b.
466. Wang Y, Martonosi M, Peh LS. Predicting link quality using supervised learning in wireless sensor networks. *ACM SIGMOBILE Mob Comput Commun Rev*. 2007;11(3):71–83.
467. Wang Y, Xiang Y, Yu S. Internet traffic classification using machine learning: a token-based approach. *IEEE*; 2011. p. 285–9.
468. Wang Z, Bovik AC, Sheikh HR, Simoncelli EP. Image quality assessment: from error visibility to structural similarity. *IEEE trans image process*. 2004;13(4):600–12.
469. Wang Z, Zhang M, Wang D, Song C, Liu M, Li J, Lou L, Liu Z. Failure prediction using machine learning and time series in optical network. *Optics Express*. 2017;25(16):18553–18565.
470. Watanabe A, Ishibashi K, Toyono T, Kimura T, Watanabe K, Matsuo Y, Shiimoto K. Workflow extraction for service operation using multiple unstructured trouble tickets. *IEEE*; 2016, pp. 652–8.
471. Watkins CJ. Models of delayed reinforcement learning. PhD thesis: Psychology Department, Cambridge University; 1989.
472. Werbos P. Beyond regression: New tools for prediction and analysis in the behavioral sciences. PhD thesis: Harvard University; 1975.
473. Werneck KD. A coupling procedure for the discrimination of mixed data. *Biometrics*. 1992;48(2):497–506.
474. WIDE Project. MAWI Working Group Traffic Archive. 2017. <http://mawi.wide.ad.jp/mawi>. Accessed 01 Aug 2017.
475. Williams M. Net tools 5. 2011. <https://www.techworld.com/download/networking-tools/net-tools-5-3248881/>. Accessed 1 Mar 2017.
476. Williams N, Zander S, Armitage G. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *ACM SIGCOMM Comput Commun Rev*. 2006;36(5): 5–16.
477. Winstein K, Balakrishnan H. Tcp ex machina: Computer-generated congestion control. In: *Proceedings of the ACM SIGCOMM 201 Conference on SIGCOMM, SIGCOMM '13*. New York: ACM; 2013. p. 123–34.
478. Witten IH. An adaptive optimal controller for discrete-time markov environments. *Inf Control*. 1977;34(4):286–95.
479. Wolpert DH, Tumer K, Frank J. Using collective intelligence to route internet traffic. *Adv neural inf process syst*. 1999;952–60.
480. Wolski R. Dynamically forecasting network performance using the network weather service. *Cluster Comput*. 1998;1(1):119–32.
481. Wu C, Meleis W. Fuzzy kanerva-based function approximation for reinforcement learning. In: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2009. p. 1257–8.
482. Xia B, Wahab MH, Yang Y, Fan Z, Sooriyabandara M. Reinforcement learning based spectrum-aware routing in multi-hop cognitive radio networks. In: *Cognitive Radio Oriented Wireless Networks and Communications, 2009. CROWNCOM'09. 4th International Conference on*. IEEE; 2009. p. 1–5.
483. Xu K, Tian Y, Ansari N. Tcp-jersey for wireless ip communications. *IEEE J Sel Areas Commun*. 2004;22(4):747–56.
484. Xu L, Krzyzak A, Suen CY. Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems. Man Cybern*. 1992;22(3):418–35.
485. Yan Q, Lei Q. A new active queue management algorithm based on self-adaptive fuzzy neural-network pid controller. In: *2011 International Conference on, Internet Technology and Applications*; 2011. p. 1–4.
486. Yang P, Luo W, Xu L, Deogun J, Lu Y. Tcp congestion avoidance algorithm identification. In: *2011 31st International Conference on Distributed Computing Systems*. 2011. p. 310–21.
487. Yi C, Afanasyev A, Moiseenko I, Wang L, Zhang B, Zhang L. A case for stateful forwarding plane. *Comput Commun*. 2013;36(7):779–791.
488. YouTube LLC. YouTube. 2017. <https://www.youtube.com/>. Accessed 01 Aug 2017.
489. Yu E, Chen CR. Traffic prediction using neural networks. In: *Proceedings of IEEE GLOBECOM*. IEEE; 1993. p. 991–5.
490. Yu X, Qiao C, Liu Y. Tcp implementations and false time out detection in obs networks. In: *IEEE INFOCOM 2004, vol 2*. 2004. p. 774–84.
491. Zalewski M, Stearns W. p0f. 2014. <http://lcamtuf.coredump.cx/p0f>. Accessed 1 Mar 2017.
492. Zander S, Nguyen T, Armitage G. Automated traffic classification and application identification using machine learning. *IEEE*; 2005, pp. 250–7.
493. Zanero S, Savaresi SM. Unsupervised learning techniques for an intrusion detection system: ACM; 2004, pp. 412–9.
494. Zhang C, Jiang J, Kamel M. Intrusion detection using hierarchical neural networks. *Pattern Recogn Lett*. 2005;26(6):779–91.
495. Zhang J, Zulkernine M. Anomaly based network intrusion detection with unsupervised outlier detection. In: *Communications, 2006. ICC'06. IEEE International Conference on*. IEEE; 2006. p. 2388–93.
496. Zhang J, Chen C, Xiang Y, Zhou W, Xiang Y. Internet traffic classification by aggregating correlated naive bayes predictions. *IEEE Trans Inf Forensic Secur*. 2013;8(1):5–15.
497. Zhang J, Chen X, Xiang Y, Zhou W, Wu J. Robust network traffic classification. *IEEE/ACM Trans Netw (TON)*. 2015;23(4):1257–70.
498. Zhani MF, Elbiaze H, Kamoun F. α -snfqm: an active queue management mechanism using neurofuzzy prediction. In: *2007 12th IEEE Symposium on Computers and Communications*. 2007. p. 381–6.
499. Zhou C, Di D, Chen Q, Guo J. An adaptive aqm algorithm based on neuron reinforcement learning. In: *2009 IEEE International Conference on Control and Automation*; 2009. p. 1342–6.
500. Zhu Y, Zhang G, Qiu J. Network traffic prediction based on particle swarm bp neural network. *JNW*. 2013;8(1):2685–91.
501. Zineb AB, Ayadi M, Tabbane S. Cognitive radio networks management using an anfs approach with qos/qoe mapping scheme. *IEEE*; 2015, pp. 1–6.

An Efficient Mice Flow Routing Algorithm for Data Centers based on Software-Defined Networking

Felipe Amézquita-Suárez*, Felipe Estrada-Solano*, Nelson L. S. da Fonseca†, Oscar Mauricio Caicedo Rendón*

* Department of Telematics, Universidad del Cauca

† Institute of Computing, State University of Campinas

Email: {cfamezquita, omcaicedo, festradasolano}@unicauca.edu.co, nfonseca@ic.unicamp.br

Abstract—A significant problem affecting the overall performance of Data Center Networks (DCNs) based on Software-Defined Networking (SDN) is the delay introduced by controllers to mice flows. In the literature, there exist approaches facing this problem by compiling and installing paths for mice and elephants dynamically. However, such approaches have shortcomings related to the large number of routing rules that the switches must handle, which may also lead to extensive delays to mice flows. In this paper, we propose MiceDCER, an algorithm that efficiently routes mice flows in SDN-based DCNs by assigning internal Pseudo-MAC (PMAC) addresses to the edge switches and hosts. Aiming at reducing the number of routing rules, MiceDCER installs wildcard rules based on the information carried by the Address Resolution Protocol (ARP) packets. Our evaluation reveals MiceDCER significantly reduces the number of rules installed in switches and, therefore, contributes to reducing the delay in SDN-based DCNs.

Index Terms—SDN, DCN, mice flows, flow routing, MAC addressing, wildcard rules

I. INTRODUCTION

In Data Center Networks (DCNs) most of the flows are short-lived and small (*i.e.*, mice), and only very few flows are long-lived and large (*i.e.*, elephants) [1]. Mice flows are usually associated with latency-sensitive and bursty applications, such as Voice over IP and search results [2]. Elephant flows often belong to massive transfers of data, such as in making backups of files [3]. Software-Defined Networking (SDN) has been used for routing mice and elephant flows in DCNs [4]. These flows impact negatively on the performance of SDN-based DCNs since large flows tend to almost fully utilize the switch buffers, introducing delay to mice flows that share these buffers [5]. Moreover, mice flows usually trigger the continuous updating of switching tables, which also generates delay.

Several approaches deal with the routing of mice and elephant flows in SDN-based DCNs by dynamically compiling and installing paths to the elephants [6], [7] and routing the mice by using static rules provided by, for instance, the Equal-Cost Multi-Path (ECMP) protocol. These approaches have some drawbacks related to the complexity of updating the routing rules continuously in case of dynamic changes in the network state, the difficulty of having redundant paths, and the high number of source-destination routing rules. A switch with many routing rules creates scalability issues in DCNs because it increases the time that the switch takes to find the routing rule for a specific flow. Such increase in time may cause the dropping of flows.

Other approaches propose to dynamically compile and install the path for mice and elephants from the SDN controller [8]. These approaches introduce a delay when the switches send the first packet of each flow to the controller. This delay negatively affects the latency-sensitive mice flows [9]. Furthermore, since the controller installs routing rules for each incoming flow, a large number of incoming flows can overload the controller, increasing the processing delay [10]. Moreover, there are also scalability problems faced by the switches with many routing rules. Several approaches [6]–[8], [11]–[13] offer alternatives to re-route mice flows considering the load on links, the limited memory capacity of switches, and the location of network devices in the topology. However, it is still necessary a solution to, first, reduce the delay caused to mice flows by a large number of routing rules in the switches and, second, avoid sending the first flow packet to the controller.

In this paper, we present an OpenFlow-based network algorithm, named Mice Data Center Efficient Routing (MiceDCER), that performs efficient routing of mice flows in SDN-based DCNs. The algorithm relies on the Address Resolution Protocol (ARP) messages to indicate the controller the rules it should install on the switch tables. Unlike several other proposals that depend on addressing the final hosts, MiceDCER relies on the positioning of switches for traffic routing and generating wildcard rules to save space in the switching tables. Instead of receiving the first packet of the flow that is sent by the switch, the controller intercepts ARP messages to interpret the addresses and install the necessary rules in the switch tables. Results reveal that MiceDCER significantly reduces the number of rules installed in the switches.

The structure of this paper is as follows. In Section II, we overview the related work. In Section III, we present MiceDCER and its implementation. In Section IV, we evaluate MiceDCER. Finally, we conclude the paper in Section V.

II. RELATED WORK

Hedera [6] is a flow scheduling system that takes advantage of multiple paths in DCN topologies. This system employs multi-stage switching fabrics to utilize network resources efficiently. Hedera overperforms ECMP in bandwidth utilization for Global First Fit and Simulated Annealing. However, Hedera improves only the forwarding of elephant flows. MiceTrap [7] aims at improving the performance of mice flows by reducing the number of rules in the switch tables. MiceTrap

uses a weighted algorithm to balance the load across multiple links. However, it has inconsistencies regarding the rules to install for mice flows. MiceTrap analytically proved better traffic balancing than ECMP in a non-DCN scenario.

PortLand [8] defines a protocol for discovering the position of switches in the topology and allows the controller to assign internal Pseudo-MAC (PMAC) addresses to end hosts. It supports fault-tolerant PMAC-based routing and seamless VM migration. Portland does not check which destination hosts are up, sending vast amounts of information even when destination hosts are down. Niagara [11] is an algorithm that achieves precise traffic splitting while being extremely efficient in the use of the available rule-table space of the switches. This algorithm generates wildcard rules to handle and split the flows according to different target weights. Niagara is highly scalable and outperforms ECMP.

The approaches mentioned above aim at routing efficiently mice flows in DCNs, dealing with the limited memory capacity of the switches, the position of the devices in the topology, and the available bandwidth in the links. However, these approaches do not relate the number of rules with the delay introduced on the mice flows. Therefore, a solution is necessary to reduce the delay caused to mice by a large number of routing rules in the switches and avoid the sending of the first flow packet to the controller.

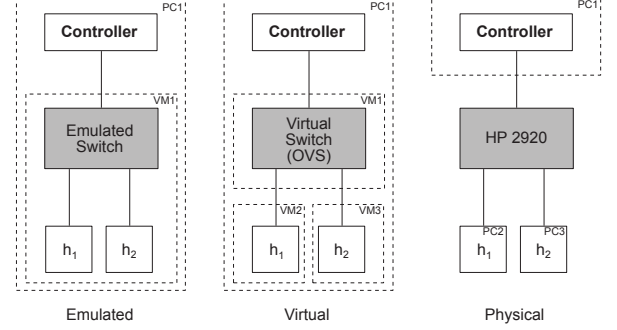
III. MICEDCER

A. Motivation

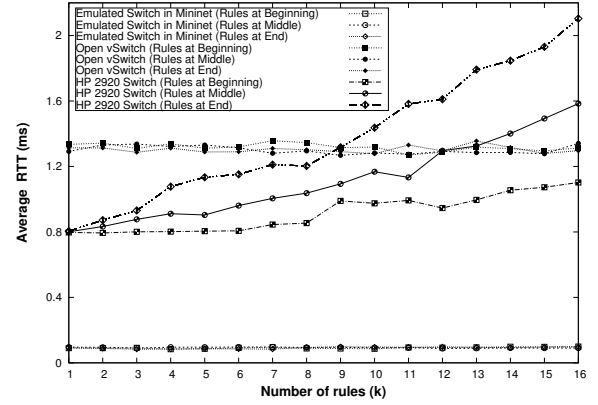
Figure 1(a) shows three experimental deployments that we use to explain how a switch table with a large number of routing rules impacts negatively on the delay of mice flows in DCNs. The first deployment measures the Round Trip Time (RTT), the total time the packet takes to go to the destination and back to the source, in an emulated environment deployed on Mininet 2.2.2 [14], in which a Ryu controller [15] handles an Open vSwitch [16] and two hosts by a single flow-installer algorithm developed in Python [17]. The second deployment measures the RTT in a virtual environment, in which the Ryu controller handles an Open vSwitch 2.5.4 and two virtual hosts (connected through a virtual Ethernet interface) by using the algorithm mentioned above. The third deployment is to measure the RTT in a physical environment, in which the Ryu controller handles (using the same algorithm that the other deployments) an HP 2920 switch that, in turn, handles communication between two physical hosts through an OpenFlow VLAN interface. We used computers with Intel Core i5 2.40GHz (4 cores) processor, 3GB RAM, and Ubuntu 16.04 LTS operative system to run the emulated environment, the Open vSwitches, and the Ryu Controller.

In the three deployments, we measure the average RTT for $r \in R = \{1000, 2000, \dots, 16000\}$, where two rules (*i.e.*, working rules) are intended to communicate the hosts, and the remaining are non-working rules. We installed T times the r flow rules in the switch, setting up the working rules at the beginning, at the middle, and at the end of the switch routing table for each $t \in T$, and took the RTT measurement N times

for each $t \in T$. The experiments were conducted by pinging from a host to another one. In particular, we measure the average RTT in the first host after receiving the reply packets from the switch.



(a) RTT measurement cases.



(b) Average RTT vs Rules installed on the switch. $T = 30$ and $N = 30$.

Fig. 1. Motivation showcase.

For each deployment afore-described, Figure 1(b) depicts the results for $T = 30, N = 30$ when the working rules are at the beginning, the middle, and the end of the switch table. These results reveal that the average RTT for the emulated deployment stays within a range of $[0.08 - 0.1]$ ms regardless of the number of rules installed. Similarly, the average RTT for the virtual deployment stays at ~ 1.3 ms. This higher RTT value is caused by the virtual links used to connect the virtual machines. For the physical deployment, there is a significant change. First, the RTT increases (from ~ 0.8 ms to ~ 2.1 ms) when the number of rules grows. Second, the location of the rules also impacts the RTT. In fact, if the working routing rules are at the beginning of the switch table, the RTT is ~ 1.1 ms. If the rules are the end, the RTT is ~ 2.1 ms.

From the experiments above, we learned that, first, RTT in emulated and virtual deployments remains nearly constant while varying the number of installed rules. Second, in a physical deployment, however, the amount of limited memory of switches implies that the number of installed rules influences the delay of the packets [18] [19] of mice flows. Thus, we can state that the number of rules is an issue that needs to

be addressed to reduce the delay of packets in mice flows in DCNs based on SDN.

B. Overview

We present an OpenFlow-based algorithm named MiceDCER, focused on reducing the delay of mice flows in DCNs. In particular, MiceDCER addresses three issues: (i) the allocation of addresses to the hosts, (ii) the internal identification of these addresses without having to send the first packet of a flow to the controller; and (iii) the need to install an efficient routing path. Moreover, the controller also needs to know the feasible routes for the flows to install the required rules on the switch tables. When a switch is added the controller executes an internal method called *Topology Discovery*. This method checks which switches are connected, how they connect to each other, and their internal position by the Link Layer Discovery Protocol (LLDP).

Figure 2 presents the process to generate and install rules in MiceDCER that aims at tackling the delay of packets in the mice flows. The MiceDCER process involves the following tasks: Generate, Define, Install, and Update. It is noteworthy that these tasks are executed after the SDN controller obtains the topology by Topology Overview.

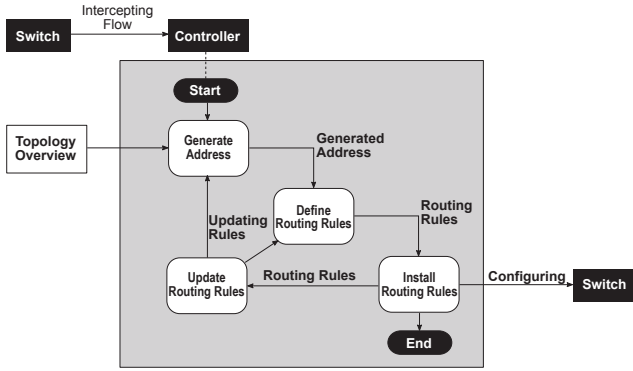


Fig. 2. Process to generate and install rules in MiceDCER.

Generate Address. MiceDCER generates the PMAC of the receiver edge switch (i.e., the switch that received the flow intercepted by the controller) from the Topology Overview. This task is composed of two others. The first one generates the PMAC of the edge switches based on their position in the topology, while the second task stores the new PMACs in a table, associating them with the corresponding actual MAC (AMAC) of each switch. **Define Routing Rules.** Here, the MiceDCER algorithm (see Algorithm 1) provides the generated PMACs of the switches to the controller for defining the set of rules to install R_{ins} . This definition is performed by considering the source MAC and IP addresses of the messages, as well as the input port of the switches. **Install Routing Rules.** MiceDCER instructs the controller to install the rules R_{ins} in the edge switches. **Update Routing Rules.** When a significant change in the network occurs, the controller updates the defined rules R_{ins} . This updating may imply the generation of new PMACs or the definition of new rules.

It is noteworthy that in MiceDCER, we are assuming that we are not going to install ARP routing rules on the core or aggregate switches because the ARP management is done in the edge layer. Each request ARP message is expected to be answered with a reply message, which is determined by the value of its Operation Code field on the ARP packet data. This value is essential when it comes to the management of ARP messages required to install the rules on the switches.

Each core switch will have a rule that will connect to each *pod* or networking group. The aggregate switches will have a group rule that provides connection with the core layer, and a rule for each link they have with the edge layer. The installation of rules in the edge switches is more complicated than it is in the upper layers because the PMAC-AMAC conversion is performed on the edge layer. When the packet comes from the host, the switch rewrites the source MAC field with its associated PMAC, and before sending the packet to its final destination, it rewrites the destination MAC field with the host AMAC before routing it through its respective port.

C. Algorithm

Inputs and outputs. Our algorithm receives as input an intercepted ARP message which contains the primary data $A = \{opcode, eth_src, eth_dst, mac_src, in_port\}$, where *opcode*, *eth_src*, *eth_dst*, *mac_src*, and *in_port* are the Operation Code, Source IP Address, Destination IP Address, Source MAC, and the switch input port, respectively. Although the input port number is not part of the ARP data, our algorithm uses it to generate rules. MiceDCER also receives a set of edge switches $T = \{sw_i | sw_i = (id, pmac)\}$ from the Topology Overview. Here, *id* represents the defined ID of the switch sw_i , while *pmac* is the PMAC that the algorithm assigns on sw_i . It is important to highlight that the defined ID (i.e., *datapath ID*) for each $sw_i \in T$ is a 64-bit defined field, where the 48 least significant bits (LSB) correspond to the switch MAC, while the 16 most significant bits (MSB) depend on the implementation of the switch, which varies with each model. The algorithm also has an ARP stale time threshold Θ which determines how often the connection between the two hosts should be checked.

The outputs of MiceDCER are the PMACs and routing rules to assign to the edge switches. The algorithm provides two tables for storing PMACs: the first one associates the PMACs of the hosts (or virtual machines) according to their IP addresses and AMACs, while the second stores the PMAC headers corresponding to the AMAC of the edge switches. MiceDCER generates and assigns the PMACs using the form $pod.pos.port.vmid$. Here, *pod* reflects the pod number of the edge switch, while *pos* is its position within the pod. *port* is, from its local view, the port number to which the host connects to the switch, and *vmid* is the identifier that corresponds to the virtual machine inside the physical machine (or physical hosts on the other side of the bridge). The PMAC header represented in the switch table will have the form $pod.pos.*.*$.

Procedures. The algorithm procedures are: generation of initial rules for the edge switches, intercepted message man-

Algorithm 1 Rule installation algorithm.

E : Set of edge switches (ToR)
 sw : Message receiver switch $sw \in E$
 A : Intercepted ARP message data
 I : List of known IP addresses
 T_H : Host PMAC addressing table
 T_E : Edge switch PMAC addressing table
 Θ : ARP stale time threshold

```

1: Generate initial rules for edges:
2: for each  $e \in E$  do
3:    $field \leftarrow \{fieldtypes.ARP\}$ ;
4:    $actions \leftarrow [ACTION\_CONTROLLER]$ ;
5:    $installRule(e, field, actions)$ ;
6: end for

7: Intercepted message management:
8: procedure MESSAGEMANAGEMENT( $sw, A$ )
9:   if  $A.dst\_ip \notin I$  then
10:     $actions \leftarrow [ACTION\_FLOOD]$ ;
11:     $out \leftarrow packetOut(A, actions)$ ;
12:    for each  $e \in E$  do
13:      if  $e \neq sw$  then
14:         $generateRequests(e, A)$ ;
15:      end if
16:    end for
17:   else
18:     if  $time(A.src\_ip, A.dst\_ip) > \Theta$  then
19:        $checkHostConnection(A.dst\_ip)$ 
20:     else
21:        $actions \leftarrow [A.in\_port]$ ;
22:        $out \leftarrow packetOut(reply(A), actions)$ ;
23:     end if
24:   end if
25:    $sw.sendMessage(out)$ ;
26: end procedure

27: Generate table entries:
28: procedure GENERATEENTRIES( $sw, A$ )
29:   if  $A.eth\_src \notin T_H$  then
30:      $pmac \leftarrow generatePmac(sw)$ ;
31:      $add\{A.eth\_src, pmac\}$  to  $T_H$ ;
32:   if  $sw.id \notin T_E$  then
33:      $pmacHeader \leftarrow generateHeader(pmac)$ ;
34:      $add\{sw.id, pmacHeader\}$  to  $T_E$ ;
35:   end if
36: end if
37: end procedure
  
```

agement, and generation of table entries. **Generation of initial rules for the edge switches.** MiceDCER initially installs the routing rules for the edge switches with the ARP field type, to allow the controller to intercept the ARP messages that arrive to the switch. The algorithm then performs the following procedure to install the rules on the switch tables. **Intercepted message management.** If the controller does not know the IP destination address of the intercepted ARP request message, the controller indicates the receiver switch (i.e., the switch that received the message intercepted by the controller) to flood (i.e., sends the packet to all ports excluding the input port), and instructs the other edge switches to flood with requests. If the controller knows the IP address, it sends a reply ARP message back to the source host, or it checks if the destination host is still connected after the ARP stale time has passed. **Generation of table entries.** If the source IP address of the intercepted message does not exist in the host PMACs table, MiceDCER proceeds to generate the PMAC and insert the entry into the table, associating it with the source IP address. In this procedure, if the defined ID of the receiving switch is not in the switch PMACs table, the algorithm generates

the PMAC header to add the entry, avoiding to carry out this process again if the same switch receives the flows of several directly connected hosts.

IV. EVALUATION AND ANALYSIS

A. Prototype

MiceDCER relies on information in ARP messages for reducing the number of rules installed by the controller. MiceDCER is similar to PortLand, in that it assigns PMAC addresses to the end hosts according to their position in the topology [8]. However, unlike PortLand, MiceDCER assigns the PMACs before the hosts start communicating, minimizing the delays in the transmission of mice flows. Also, unlike other proposals, MiceDCER is complemented by the *Topology Discovery* method to give the controller an overview of the topology. Once this discovery finishes, our algorithm populates the switching tables with routing rules. Finally, unlike Hedera [6] or Presto [13] that focus on forwarding and splitting flows, respectively, MiceDCER focuses on the establishment of the most viable routes for mice flows.

We implement MiceDCER on the top of a Python-based SDN controller. In particular, we use the libraries offered by the Ryu 4.23 framework to implement the Algorithm 1. This framework supports OpenFlow and ARP that are the fundamental protocols of MiceDCER. The MiceDCER implementation is available in [20].

B. Evaluation Scenario

We evaluate our algorithm analytically to verify if it reduces the delay significantly in the flow packets compared with other routing protocols based on IP or MAC addresses. To carry out the evaluation, first, we check the initialization of the topology with MiceDCER to make sure it installs the correct rules; we use a FatTree topology since it is the most common topology used in DCNs. Second, we calculate the number of switches for each layer in our topology, and the number of rules installed by the three evaluated solutions namely, MiceDCER, MAC routing, and IP routing.

C. Results

Figure 3(a) illustrates the interception of ARP messages that MiceDCER performs. Our algorithm configures the MSB part of the ID of each switch to indicate its internal position in the topology. When the controller (i.e., Ryu) starts running and activates the option of observing the links, MiceDCER automatically checks this MSB value and installs the rules that allow the controller to intercept the ARP packets at the edge switches.

1) *Topology*: After running *Topology Discovery* to establish the switches and their connections, MiceDCER uses LLDP to determine their position in the topology. We assume edge switches are established when not receiving LLDP packets from the half of ports; these ports are connected to hosts. Switches connected to edge switches become aggregates, and the connected to aggregates become cores. After determining the entire topology, MiceDCER assigns the pod and position values of edge switches to encode their position.

2) *Number of Rules*: We first calculate the total number of routing rules that MiceDCER installs in the switches within the topology. The total amount of rules per switch (including group rules) for k -ary FatTree topology can be obtained by using the Equations (1), (2) and (3) for core, aggregate, and edge layers respectively: (1) $C_{DC} = 3 + k$, (2) $A_{DC} = 5 + \frac{k}{2}$, and (3) $E_{DC} = 6 + 2 \sum_{h=1}^{k/2} M_h$.

Each of the $k^2/4$ core switches connects with each of the k pods. Each of the $k^2/2$ aggregate switches has a wildcard rule which routes to the core layer through a group table and a rule for each of the $k/2$ connections with the edge switches. Each of the $k^2/2$ edge switches has two routing tables, where the first table contains the rules for matching the source AMAC, and the second table has the rules for matching the destination PMAC. The number of rules installed for each table is equal to the number of VMs M_h for each host h connected to the rack using a bridged adapter, as the VMs send the packets using their MAC address. All the switches also have extra rules for table-miss action and ARP management.

Figure 3(b) illustrates the PMAC-AMAC association that MiceDCER carries out. The edge switches rewrite the source MAC field with the host PMAC when receiving the packet, and rewrite the destination MAC field back to the host AMAC before sending the packet out.

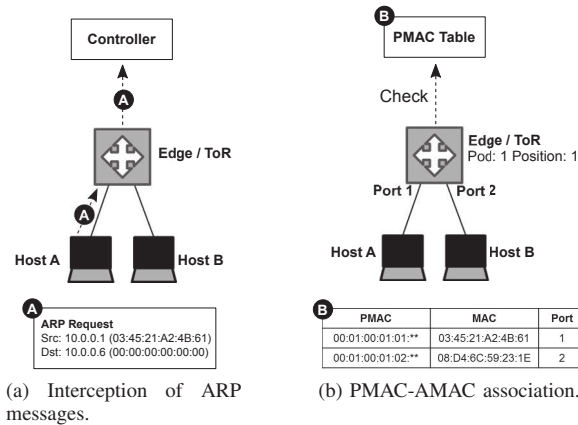


Fig. 3. Evaluation of MiceDCER.

Other approaches install a routing rule depending on the IP address, rather than the MAC of the host or VM [1]. The total amount of rules for IP-based routing, assuming the use of wildcard rules and *Topology Discovery*, is given by the Equations (4), (5) and (6) for each of the three layers: (4) $C_{IP} = 2 + k$, (5) $A_{IP} = 3 + k$, and (6) $E_{IP} = 3 + \frac{k}{2} + \sum_{h=1}^{k/2} M_h$. The total amount of rules for MAC-based routing is given by Equations (7), (8), and (9) for each layer: (7) $C_{MAC} = \frac{k^3}{8}$, (8) $A_{MAC} = \frac{k^3}{8}$, and (9) $E_{MAC} = \frac{k^3}{4}$.

Figure 4 presents the topology elements in a typical Fat-Tree topology when the number of hosts to handle grows. These results reveal that the number of switches grows significantly regarding the number of hosts. Furthermore, 1/5 of switches

in the topology are in the core layer, distributing the remaining switches between the edge and aggregate layers equitably.

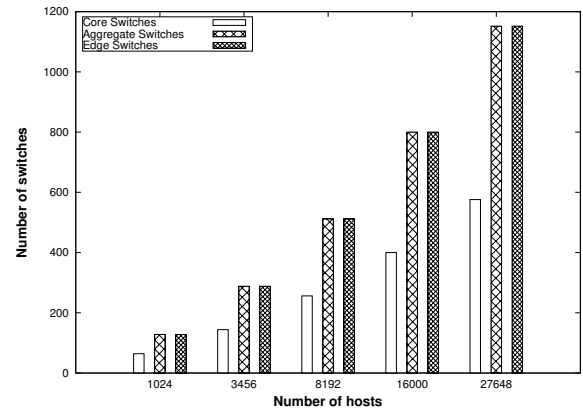


Fig. 4. Fat Tree - Topology elements.

Figure 5(a) presents the number of rules per edge switch generated by MiceDCER, IP-based and MAC-based routing when the number of hosts grows (recall that the edge switches grows too, see Figure 4). We assume we are using 8 non-bridged VMs per host. Results reveal that MAC-based and IP-based routing installs more rules than MiceDCER (e.g., ~1500 rules per edge switch when using 27648 hosts). Considering these results, we can conclude that MiceDCER reduces the number of rules per edge switch significantly when compared with other routing solutions.

Figure 5(b) presents the number of rules per aggregate switch generated by MiceDCER, IP-based and MAC-based routing when the number of hosts grows. These results also reveal that MAC-based routing installs much more rules than MiceDCER. The IP-based routing installs approximately the double of rules than MiceDCER. Thus, we can conclude that MiceDCER reduces the number of rules per aggregate switch significantly when compared with the MAC-based and IP-based routing.

Figure 5(c) presents the number of rules per core switch generated by MiceDCER, IP-based and MAC-based routing when the number of hosts grows. These results reveal again that MAC-based routing installs more rules than MiceDCER. In turn, the IP-based routing installs about the same amount of rules as MiceDCER. We can conclude that MiceDCER reduces or at least generates the same number of routing rules to install in the core switches.

To sum up, regarding the number of rules to install, MiceDCER always outperforms the performance of MAC-routing in SDN-based DCNs that follows a Fat-Tree topology. The IP-based protocol also installs more rules than MiceDCER, in the edge layer. In the core and aggregate layers, MiceDCER and IP-based protocol have a similar behavior. Considering that in total, MiceDCER installs fewer routing rules in switching tables than traditional routing protocols, we can conclude that it contributes to reduce the delay of mice flows.

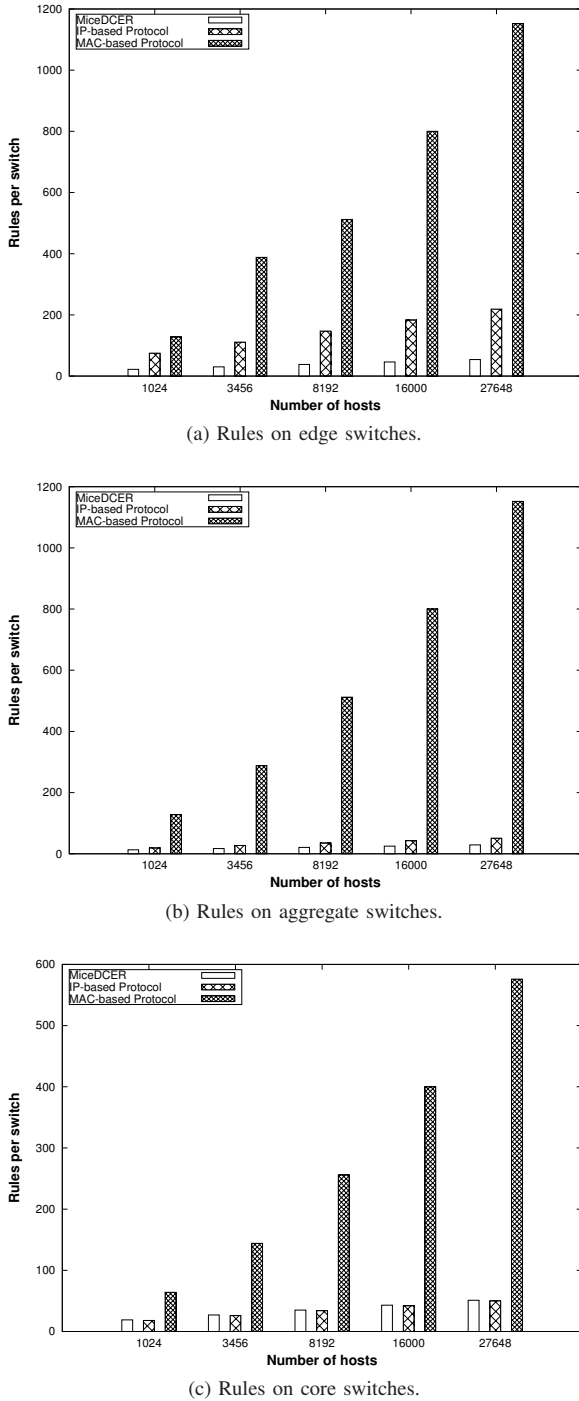


Fig. 5. Rule installation results vs number of hosts for FatTree topology.

V. CONCLUSIONS AND FUTURE WORK

A relevant problem affecting the overall performance of SDN-based DCNs is the delay introduced to mice flows by the logically centralized controllers. Aiming at overcoming this problem, in this paper, we present MiceDCER, an algorithm for routing efficiently mice flows in such DCNs. In particular, our algorithm, first, installs rules relying on the information obtained from ARP messages. Second, it takes advantage of

Topology Discovery to identify the position of the switches and install the appropriate rules for improving routing. Moreover, MiceDCER generates wildcard rules to save memory in switching tables. By experimental results, we demonstrate that MiceDCER significantly reduces the number of rules installed in switches and, therefore, contributes to reducing the delay suffered by mice flows.

As future work, we intend to evaluate MiceDCER in other networks, such as SDWAN and SDWLAN, that follows the SDN paradigm. Also, we want to implement MiceDCER in other controllers.

REFERENCES

- [1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008.
- [2] N. L. S. da Fonseca and R. Boutaba, *Cloud Services, Networking, and Management*, 1st ed. Wiley-IEEE Press, 2015.
- [3] M. Afaq, S. U. Rehman, and W.-C. Song, "A Framework for Classification and Visualization of Elephant Flows in SDN-Based Networks," *Procedia Computer Science*, vol. 65, pp. 672–681, 2015.
- [4] A. M. Abdelmoniem and B. Bensou, "Reconciling mice and elephants in data center networks," in *IEEE CloudNet*, ON, Canada, 2015, pp. 119–124.
- [5] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data Center Network Virtualization: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 909–928, 2013.
- [6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *NSDI*, CA, USA, 2010, pp. 1–15.
- [7] R. Trestian, G.-M. Muntean, and K. Katrinis, "MiceTrap: Scalable traffic engineering of datacenter mice flows using OpenFlow," in *IFIP/IEEE IM*, Ghent, Belgium, 2013, pp. 904–907.
- [8] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric," *ACM SIGCOMM*, pp. 39–50, 2009.
- [9] B. Nunes Astuto, M. Mendonça, X. Nam Nguyen, K. Obraczka, and T. Turetti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *HAL Archive*, vol. 16, no. 3, pp. 1617–1634, 2014.
- [10] P. Song, Y. Liu, T. Liu, and D. Qian, "Controller-proxy: Scaling network management for large-scale SDN networks," *Elsevier Computer Communications*, vol. 108, pp. 52–63, 2017.
- [11] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, "Efficient traffic splitting on commodity switches," *CoNEXT*, pp. 1–13, 2015.
- [12] L. A. D. Knob, R. P. Esteves, L. Z. Granville, and L. M. R. Tarouco, "Mitigating elephant flows in SDN-based IXP networks," *IEEE ISCC*, pp. 1352–1359, jul 2017.
- [13] K. He, E. Rozner, K. Agarwal, W. Felter, and J. Carter, "Presto: Edge-based Load Balancing for Fast Datacenter Networks," *ACM SIGCOMM*, vol. 45, no. 4, pp. 465–478, 2015.
- [14] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *ACM SIGCOMM*, ser. Hotnets-IX, New York, NY, USA, 2010, pp. 19:1–19:6.
- [15] "Ryu Software-Defined Networking Framework." [Online]. Available: https://ryu.readthedocs.io/en/latest/getting_started.html
- [16] "Open vSwitch." [Online]. Available: <https://www.openvswitch.org/>
- [17] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed SDN controller system: A survey on design choice," *Elsevier Computer Networks*, vol. 121, pp. 100–111, 2017.
- [18] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Elsevier Computer Networks*, vol. 71, pp. 1–30, 2014.
- [19] J. Jung, K. Kim, and H. Kim, "On the Necessity of VM Migration: Simulation on Datacenter Network Resources," *Wireless Personal Communications*, vol. 86, no. 4, pp. 1797–1812, feb 2016.
- [20] "MiceDCER Implementation Code." [Online]. Available: <https://github.com/cfamezquita/MiceDCER>

NELLY: Flow Detection Using Incremental Learning at the Server Side of SDN-Based Data Centers

Felipe Estrada-Solano , Graduate Student Member, IEEE, Oscar M. Caicedo , Member, IEEE, and Nelson L. S. Da Fonseca , Senior Member, IEEE

I. INTRODUCTION

Abstract—The processing of big data generated by the Industrial Internet of Things (IIoT) calls for the support of processing at the edge of the network, as well as at the cloud data centers. The equal-cost multipath, which is the default routing technique in the cloud data centers, can degrade the network performance when handling mouse and elephant flows. Such degradation of performance can compromise the support of the strict quality of service requirements of the IIoT over 5G networks. Novel techniques for scheduling the elephant flows can alleviate this problem. Recently, several approaches have incorporated machine learning techniques at the controller-side in software-defined data center networks (SDDCNs) to detect elephant flows. However, these approaches can produce heavy traffic overhead, low scalability, low accuracy, and high detection time. This article introduces the Network Elephants Learner and analyzer (NELLY), a novel and efficient method for applying incremental learning at the server side of SDDCNs to accurately and timely identify elephant flows with low traffic overhead. Incremental learning enables NELLY to adapt to varying network traffic conditions and perform continuous learning with limited memory resources. NELLY has been extensively evaluated using real traces and various incremental learning algorithms. Results show that NELLY is accurate and supports low classification time when using adaptive decision trees algorithms.

Index Terms—Data center networks (DCNs), flow classification, machine learning, software-defined networking (SDN).

THE Industrial Internet of Things (IIoT) aims at automating industrial processes that can be supported by the analysis of big data generated by a large number of interconnected devices [1]. The real-time requirements of industrial applications and the access demand of massive machine-type communications call for the employment of the 5G technology in the foreseen IIoT. In industrial plants, edge devices (fog nodes) will be used for the processing of delay-sensitive data, while cloud servers will be employed for the processing of the huge amount of data generated by sensors. Extracting a value from such big data will be fundamental for enabling customized and flexible mass production of goods [2], [3].

To support the big data demand in the IIoT, cloud data centers provide significant bandwidth capacity for a large number of servers interconnected by an especially designed network, called data center network (DCN) [4]. This bandwidth capacity can be optimized by using *multipath routing*, which distributes traffic over multiple concurrent paths [5]. Nowadays, the equal-cost multipath (ECMP) is the default multipath routing mechanism for DCNs [6]. However, the ECMP can degrade the performance of DCNs due to the coexistence of many small short-lived flows (i.e., mice) and few large long-lived flows (i.e., elephants), since the ECMP can assign more elephant flows to the same path, generating *hotspots* (i.e., some links overused, while others underused). Flows traversing hotspots suffer from low throughput and high latency. Mice and elephants are characteristic in cloud data centers running big data analytics technologies [7]–[9], such as MapReduce and Hadoop, which are pivotal in IIoT systems for delivering a value from the big data and making business decisions [10]. Then, similar flows will be present in the cloud data centers of IIoT systems. As a consequence, DCNs that use only the ECMP for managing the bandwidth demanded by the big data in the IIoT likely will not meet the target efficiency specified for 5G networks.

Recent multipath routing mechanisms have leveraged software-defined networking (SDN) to face the ECMP limitations; DCNs using SDN are referred to as software-defined data center networks (SDDCNs). SDN allows a logically centralized controller to dynamically make and install routing decisions on the basis of a global view of the network [11], [12]. SDN-based multipath routing dynamically reschedules elephant flows, while

Manuscript received May 30, 2019; revised September 9, 2019; accepted October 7, 2019. Date of publication October 14, 2019; date of current version January 14, 2020. This work was supported in part by the Administrative Department of Science, Technology and Innovation of Colombia COLCIENCIAS under Grant 647-2014 and in part by the São Paulo Research Foundation FAPESP, Brazil, under Grant #2015/24494-8 and Grant #2019/04914-3. Paper no. TII-19-2157. (Corresponding author: Nelson L. S. da Fonseca.)

F. Estrada-Solano is with the Department of Telematics, University of Cauca, Popayán 190003, Colombia, and also with the Institute of Computing, State University of Campinas, Campinas 13083-852, Brazil (e-mail: festradasolano@unicauca.edu.co).

O. M. Caicedo is with the Department of Telematics, University of Cauca, Popayán 190003, Colombia (e-mail: omcaicedo@unicauca.edu.co).

N. L. S. Da Fonseca is with the Institute of Computing, State University of Campinas, Campinas 13083-852, Brazil (e-mail: nfonseca@ic.unicamp.br).

Digital Object Identifier 10.1109/TII.2019.2947291

1551-3203 © 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

handling mouse flows by employing default routing such as ECMP [6] and MiceDCER [13]. Reactive *flow detection* methods, which are at the heart of SDN-based mechanisms, discriminate elephants from mice by using static thresholds [14]–[16]. However, reactive methods are not suitable for SDDCNs, since hotspots may occur before the elephant flows are detected.

Novel SDN-based flow detection methods incorporate machine learning (ML) for proactively identifying elephant flows [17]. However, ML-based methods operate at the *controller side* of SDDCNs, requiring the central collection of either *per-flow* data [18] or *sampling-based* data [19], [20]. The central collection of per-flow data, however, causes problems, such as heavy traffic overhead and poor scalability. Sampling-based data, on the other hand, tend to provide delayed and inaccurate flow information. Moreover, sampling techniques that mitigate the problem rely on nonstandard SDN specifications. Using ML on either the *switch-side* or *server-side* represents a potential solution to the controller-side problems, since these locations enable prompt and per-flow data with low traffic overhead. Switch-side flow detection methods based on ML are impractical because they require specialized hardware and put a heavy processing load on the switches. Conversely, ML-based flow detection methods at the server side require only software modifications in the servers; nonetheless, these methods have not been fully explored.

In this article, we propose a novel flow detection method denominated Network Elephants Learner and anaLYzer (NELLY), which applies incremental learning at the server side of SDDCNs for accurately and timely identifying elephant flows while generating low control overhead. Incremental learning allows NELLY to constantly train a flow size classification model from continuous and dynamic data streams (i.e., flows) [17], [21], providing a constantly updated model and reducing time and memory requirements. Thus, NELLY adapts to the variations in traffic characteristics and performs endless learning with limited memory resources. We extensively evaluate NELLY using datasets extracted from real packet traces and incremental learning algorithms. Quantitative evaluation demonstrates that NELLY is efficient in relation to accuracy and classification time when adaptive decision trees algorithms are used. Analytic evaluation corroborates that NELLY is scalable, causes low traffic overhead, and reduces detection time, yet it is in conformance with SDN standards.

The remainder of this article is organized as follows. Section II introduces NELLY. Section III presents a quantitative evaluation of NELLY using incremental learning algorithms and real packet traces. Section IV compares NELLY to other related work. Section V concludes this article.

II. NELLY

Fig. 1 introduces NELLY, a flow detection method that applies incremental learning at the server side of SDDCNs to identify elephant flows accurately in a reasonable time while generating low control overhead. NELLY operates as a software component either in the kernel of the host operating system (OS) or in the hypervisor of servers in the SDDCN with the aim of monitoring

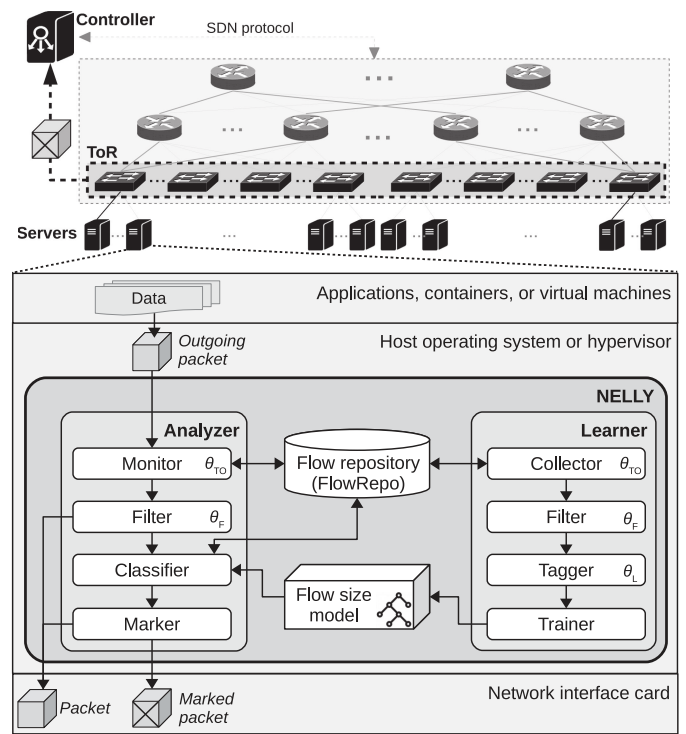


Fig. 1. NELLY Architecture.

all packets sent by the applications, containers, and virtual machines. Since NELLY detects elephant flows at their origin, a small overhead is demanded.

The architecture of NELLY (see Fig. 1) has two subsystems: *Analyzer* and *Learner*. The Analyzer applies a flow size classification model for detecting and marking elephant flows on the fly. The Learner then applies an incremental learning algorithm for building and updating the flow size classification model. This model maps online features (i.e., features extracted from the first few packets of a flow) onto the corresponding class of flows (i.e., mice or elephants). The processes of the Analyzer and the Learner run concurrently, as depicted in Algorithms 1 and 2, respectively.

NELLY is conceived for recognizing and handling elephant flows in real SDN implementations. NELLY can run on any host OS or hypervisor. In the control plane, any OpenFlow-compliant controller (e.g., OpenDaylight) can be used, since NELLY operates at the server side. In the data plane, OpenFlow-compliant switches (e.g., Open vSwitch) can be employed, since NELLY requires only that the top-of-rack (ToR) switches include a preconfigured routing rule to forward elephant flows to the controller.

A. Analyzer

As illustrated in Fig. 1, the Analyzer consists of four modules: *Monitor*, *Filter*, *Classifier*, and *Marker*. The process of each module is detailed in Algorithm 1. As shown in lines 1 and 2, the Monitor keeps track of flows by extracting the header, size, and timestamp of each outgoing packet. A flow consists

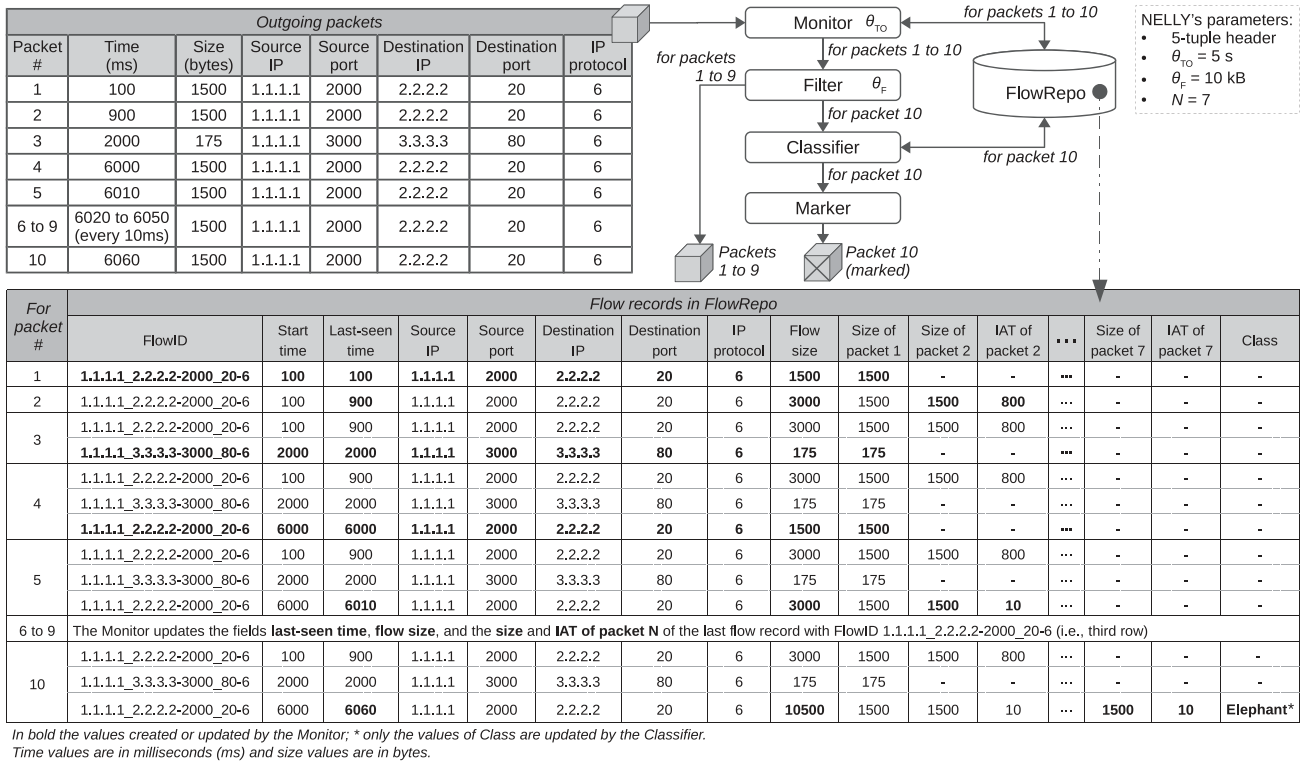


Fig. 2. Example of how flow records are created and updated in the FlowRepo.

of subsequent packets sharing the same value for certain header fields and separated by a time space shorter than a threshold timeout (θ_{TO}). NELLY enables a flexible configuration of these flow parameters, namely, flow header fields and θ_{TO} . For example, the flow header fields can be set as the well-known 5-tuple: source IP, source port, destination IP, destination port, and IP protocol. These flow header fields can also include MAC addresses and VLAN ID. On the other hand, the configuration of θ_{TO} is discussed in Section II-B.

The Analyzer manages a flow record in the Flow Repository (FlowRepo) for each observed flow. As illustrated in Fig. 2, the flow record includes the flow identifier (FlowID), start time, last-seen time, packet header (e.g., 5-tuple), flow size, the size and interarrival time (IAT) of the first N packets, as well as the identified class (i.e., mice or elephants). Note that the IAT of the first packet is not included because it does not provide distinctive flow information (i.e., the IAT is always zero for the first packet of every flow).

As depicted in lines 3–13 in Algorithm 1, the Monitor then generates a FlowID from the flow header fields of each packet and checks to see if it exists in the FlowRepo. If this FlowID is missing (e.g., for packets 1 and 3 in Fig. 2), or if the time since the last update of an existing record with this FlowID is longer than θ_{TO} (e.g., for packet 4), the Monitor creates a new record in the FlowRepo (Algorithm 1, lines 25–32). Otherwise, the Monitor fetches and updates the flow record (Algorithm 1, lines 33–43) using the FlowID stored in the FlowRepo (e.g., for packets 2 and 5–10 in Fig. 2). When multiple flow records sharing the same FlowID exist in the FlowRepo,

the Monitor always works with the most recent one (e.g., for packets 5–10).

Using the updated flow record, the Filter (Algorithm 1, line 14) avoids the introduction of a delay in the classification of a large number of mouse flows (usually latency-sensitive [14], [15]) by sending the packets of flows with a size below a certain threshold (θ_F) directly to the SDDCN without further processing (e.g., for packets 1–9 in Fig. 2). The Filter also ensures that the Classifier receives all the required online features for making the classification. The online features refer to flow data extracted from the first N packets of a flow. The Filter then guarantees the size and IAT of the first N packets of a flow since the maximum value of N depends on θ_F . For example, $\theta_F = 10$ kB would require an $N \leq 7$ over Ethernet; otherwise, data from some packets would be missed. Consequently, the Classifier operates once the Monitor has processed packets that increment the size of flows over θ_F (e.g., for packet 10).

The Classifier (Algorithm 1, lines 15–18) applies the flow size classification model to the online features to identify flows as either mice or elephants. This model results from an incremental learning algorithm, which maps the online features to the corresponding class of flows used as training data. After applying the flow size classification model, the Classifier stores the identified class in the FlowRepo for each flow record with flow size greater than θ_F (e.g., elephant for flow of packet 10 in Fig. 2). Therefore, when processing a packet of a previously identified flow, the Classifier checks the fetched class from the FlowRepo to avoid any delay from the classification. The Classifier then reports to the Marker the class of the flow for each packet. We discuss

Algorithm 1: Analyzer.

```

input : outgoing packet  $p$  with header  $h_p$ , size  $s_p$ , and timestamp  $t_p$ ,
        and flow size classification model  $m$ 
output: either packet  $p$  or packet marked  $p^*$ 
data : flow timeout threshold  $\theta_{TO}$ , filtering flow size threshold  $\theta_F$ ,
        and number of first packets  $N$ 

1 begin on receiving  $p$ 
  // Monitor
2   get  $h_p$ ,  $s_p$ , and  $t_p$  from  $p$ ;
3    $fid \leftarrow$  compute FlowID using the flow header fields from  $h_p$ ;
4   if  $fid \notin FlowRepo$  then
5      $f \leftarrow$  call CREATE_FLOW( $fid$ ,  $h_p$ ,  $s_p$ ,  $t_p$ )
6   else
7      $F \leftarrow$  fetch the last flow  $f \in FlowRepo$  such that  $f.id = fid$ ;
8     if  $(currentTime - f.lastSeenTime) > \theta_{TO}$  then
9        $f \leftarrow$  call CREATE_FLOW( $fid$ ,  $h_p$ ,  $s_p$ ,  $t_p$ )
10    else
11       $f \leftarrow$  call UPDATE_FLOW( $f$ ,  $s_p$ ,  $t_p$ )
12    end
13  end
  // Filter
14  if  $f.size < \theta_F$  then return  $p$ ;
  // Classifier
15  if  $f.class$  then
16     $f.class \leftarrow m.CLASSIFY(f)$ ;
17    update  $f \rightarrow FlowRepo$ ;
18  end
  // Marker
19  if  $f.class = \text{"Elephant"}$  then
20     $p^* \leftarrow$  mark  $p$ ;
21    return  $p^*$ ;
22  end
23  return  $p$ ;
24 end

25 function CREATE_FLOW( $fid$ ,  $h_p$ ,  $s_p$ ,  $t_p$ ):
26    $f \leftarrow$  initialize a new flow with FlowID  $fid$ ;
27    $f.headerFields[] \leftarrow$  array of flow header fields from  $h_p$ ;
28    $f.startTime \leftarrow f.lastSeenTime \leftarrow t_p$ ;
29    $f.size \leftarrow f.sizePackets[0] \leftarrow s_p$ ;
30   create  $f \rightarrow FlowRepo$ ;
31   return  $f$ 
32 end

33 function UPDATE_FLOW( $f$ ,  $s_p$ ,  $t_p$ ):
34    $n \leftarrow$  current number of packets of  $f$ ;
35   if  $n \leq N$  then
36      $f.sizePackets[n] \leftarrow s_p$ ;
37      $f.iatPackets[n] \leftarrow t_p - f.lastSeenTime$ ;
38   end
39    $f.size \leftarrow f.size + s_p$ ;
40    $f.lastSeenTime \leftarrow t_p$ ;
41   update  $f \rightarrow FlowRepo$ ;
42   return  $f$ 
43 end

```

in Section II-B how the Learner collects the training data for building and updating the flow size classification model.

The Marker (Algorithm 1, lines 19–23) forwards the packets of flows classified as mice without changes but marks those classified as elephants (e.g., packet 10 in Fig. 2). To mark a packet, the Marker sets a predefined value in a code point header field supported by SDN switches. For example, OpenFlow switches support matching in two code point header fields. The first of these is the 6-bit differentiated service code point (DSCP) field of the IPv4 header. This DSCP reserves a code point space for experimental and local usage (i.e., $****11$, where $*$ is 0 or 1). The second is the 3-bit 802.1Q priority code point (PCP) field of the Ethernet header. In practice, NELLY can rely on either

Algorithm 2: Learner.

```

input : flow size classification model  $m$ 
output: either actual  $m$  or updated  $m$ 
data : learning time interval  $T$ , flow timeout threshold  $\theta_{TO}$ , filtering
        flow size threshold  $\theta_F$ , and labeling flow size threshold  $\theta_L$ 

1 begin every  $T$ 
  // Collector
2    $F \leftarrow$  fetch flows  $f \in FlowRepo$ ;
3   for  $f \in F$  do
4     if  $currentTime - f.lastSeenTime > \theta_{TO}$  then
5       delete  $f \rightarrow FlowRepo$ ;
6       // Filter
7       if  $f.size \geq \theta_F$  then
8         // Tagger
9         if  $f.size \geq \theta_L$  then  $f.class \leftarrow \text{"Elephant"}$ ;
10        else  $f.class \leftarrow \text{"Mouse"}$ ;
11        // Trainer
12         $m \leftarrow m.UPDATE(f.headerFields[], f.sizePackets[],$ 
13           $f.iatPackets[], f.class)$ ;
14      end
15    end
16  end
17  return  $m$ ;
18 end

```

one of these fields, since it is improbable that a data center uses both DSCP and PCP simultaneously [15].

The Marker can be extended by enabling a flexible configuration of the number of subsequent packets in an elephant flow to be marked (M), thus enabling a tradeoff between reliability and latency. For instance, as M increases, the lesser the probability that the controller will miss elephant flows due to losses of marked packets in the SDDCN. However, a higher M introduces a delay in the Marker for a higher number of packets of elephant flows. Once the controller has installed a higher priority routing rule for handling a specific elephant flow across the SDDCN, the subsequent marked packets of this flow are not forwarded to the controller.

B. Learner

As depicted in Fig. 1, the Learner consists of four modules: *Collector*, *Filter*, *Tagger*, and *Trainer*. The process of each module is detailed in Algorithm 2. As shown in lines 1–4, the Collector fetches terminated flows from the FlowRepo at every interval T . A flow is considered terminated if it remains idle for longer than θ_{TO} . Therefore, the Collector recognizes terminated flows by checking that a time longer than θ_{TO} has passed, since the last-seen time of the FlowID records in the FlowRepo. Note that the Collector relies on the FlowID records updated by the Monitor for the recognition of the terminated flows, so their actual size can be obtained.

The Collector avoids increasing memory consumption in NELLY by removing terminated flows from the FlowRepo (Algorithm 2, line 5). The actual size of terminated flows can also be further used to provide fixed-memory probability distributions that support autonomous configuration of flow size thresholds. Memory requirements in the FlowRepo thus depend on both T and θ_{TO} . T provides a tradeoff between memory and processing. As T decreases, the Collector removes the terminated flows from the FlowRepo more quickly, consuming

TABLE I
DETAILS OF PACKET TRACES AND IPV4 FLOWS OBTAINED USING THE 5-TUPLE HEADER AND $\theta_{TO} = 5$ s

Packet traces [23]		UNI1		UNI2	
Duration		65 min		158 min	
Packets		19.85 M		100 M	
IPv4 Percentage of total traffic		98.98% (mostly TCP)		99.9% (mostly UDP)	
IPv4 flows		1.02 M (TCP and UDP)		1.04 M (mostly UDP)	
Details of IPv4 flows	Flow size	Percentage of IPv4 flows	Percentage of IPv4 traffic	Percentage of IPv4 flows	Percentage of IPv4 traffic
	≥ 10 kB	7.16%	95.06%	5.91%	98.81%
	≥ 100 kB	0.83%	83.71%	1.93%	96.86%
	≥ 500 kB	0.14%	73.14%	0.76%	93.52%
	≥ 1 MB	0.07%	69.52%	0.48%	90.83%
	≥ 5 MB	0.01%	60.33%	0.17%	81.34%

less memory, but leading to more processing. In turn, θ_{TO} directly affects the number of FlowID records stored in memory. As θ_{TO} increases, the FlowRepo retains FlowID records for a longer time. θ_{TO} is related to the *inactive timeout* configuration of flow rules in SDN-enabled switches, which provides a tradeoff between flow table occupancy and miss-rate (i.e., when the packet IAT is greater than the timeout) [22].

The Filter of the Learner (Algorithm 2, line 6) receives the terminated flows from the Collector and reports to the Tagger only those with size greater than θ_F . The terminated flows are then used by the Trainer to build the flow size classification model. Since the Classifier operates only with flows of a size greater than θ_F , the Filter of the Learner prevents the introduction of noise to the model.

The Tagger (Algorithm 2, lines 7–8) compares the actual size of the filtered flows to a labeling threshold (θ_L) so that they can be tagged as either mice or elephants. θ_L will vary (e.g., 100 kB or 1 MB) as a function of the traffic characteristics and performance requirements of SDDCNs. Labeled flows provide the Trainer (Algorithm 2, line 9) with the *ground truth* for building a supervised learning model for flow size classification [17]. This classification model maps online features (i.e., packet header, size, and IAT of the first N packets) onto the corresponding class (i.e., mice or elephants). Recall that the Classifier relies on the flow size classification model to identify elephant flows.

Since flows represent continuous and dynamic data streams, the Trainer uses an incremental learning algorithm (e.g., Hoeffding tree and online ensembles) for building the flow size classification model. Incremental learning enables updating the flow size classification model as the Trainer receives labeled flows over time, rather than retraining from the beginning [17], [21]. Therefore, NELLY adapts to varying traffic characteristics and performs continuous learning with limited memory resources. There is no need for the Trainer to maintain labeled flows in memory. This is an important characteristic of NELLY, since it helps to reduce the consumption of resources in all the servers of the SDDCN.

III. EVALUATION

This section presents the evaluation of NELLY in relation to accuracy and classification time by using real packet traces and incremental learning algorithms. A generic approach for

designing ML-based solutions in networking [17] is used to describe and conduct the evaluation of NELLY.

A. Datasets

Two real packet traces, UNI1 and UNI2, captured in university data centers [23], were employed to evaluate NELLY (Table I summarizes their characteristics). These two traces are shorter than 3 h long, but their mice and elephant distributions are similar to those found in nonpublic traces collected at different periods along the day [8], [9]. On the other hand, to the best of our knowledge, neither traces nor datasets of IPv6 traffic in DCNs are publicly available. In line with that, NELLY was evaluated using IPv4 traffic only, which represents over 99% of the packets in UNI1 and UNI2.

Only the following parameters needed to be defined to generate the datasets: the flow header fields, θ_{TO} , and N . First, the 5-tuple header (i.e., source IP, destination IP, source port, destination port, and IP protocol) as the flow header fields, since it sufficiently characterizes IPv4 flows; hereinafter, they are referred just as flows. Second, $\theta_{TO} = 5$ s was established on the basis of the break-even point analysis between the flow table occupancy and the miss-rate in OpenFlow switches for DCNs considered by [22]. Then, since the maximum value of N depends on θ_F , $N = 7$ was set as the maximum for $\theta_F = 10$ kB. As shown in Table I, the selected θ_F encompasses all the potential elephants (i.e., flows carrying more than 95% of the traffic) and avoids the introduction of the classification delay to mice (for more than 93% of the flows). Using these parameters, the UNI1 and UNI2 data traces were processed to generate the corresponding flow size datasets, each containing somewhat more than a million flows (see Table I). Since NELLY only classifies flows greater than θ_F , those smaller than $\theta_F = 10$ kB were removed from both datasets. Therefore, the UNI1 and UNI2 datasets consisted of approximately 70 000 and 60 000 flows, respectively.

The datasets [24] included the following flow information: start time, end time, 5-tuple header, size, and IAT of the first seven packets, as well as flow size. The start and end times enabled a more realistic evaluation (see Section III-C). The 5-tuple header and the size and IAT of the first seven packets represented the online features for the flow size classification model. The flow size is compared to different θ_L (e.g., 50, 100, and 500 kB) to label the flows as mice or elephants (i.e., classes of interest). Unless otherwise stated, the datasets with

$\theta_L = 100$ kB were used. Labeled flows represented the ground truth for learning and validating the flow size classification model.

For feature engineering [17], various different data types were considered for the online features, particularly for the 5-tuple header. Certainly, the size and IAT of the first seven packets (13 features, since the IAT of the first packet is not included) indicate a measurement, hence, numeric data, whereas the 5-tuple header contains two IP addresses in dotted-decimal notation (i.e., categorical data) and three numeric codes (i.e., nominal data). However, the huge set of possible categories for IP addresses (i.e., 2^{32}) hinders a real implementation. To address this problem, the IP addresses were divided into four octets, resulting in a total of 11 nominal features for the 5-tuple header. To handle these 11 nominal features as numeric data, a *Numeric* (Num) header type was defined. These features were then transformed into binary digits (bits), generating 104 features for the 5-tuple header. Considering these binary features, two more header types were defined: *Binary-Numeric* (BinNum) to treat binary features as numeric data (i.e., a value between zero and one) and *Binary-Nominal* (BinNom) to handle binary features as nominal data (i.e., zero or one). Unless otherwise stated, the datasets with BinNom header were used.

B. Accuracy Metrics

Metrics derived from the confusion matrix were used, including the true positive rate (TPR) and the false positive rate (FPR), thus avoiding the overoptimism of the conventional accuracy metric caused by an imbalance of classes [17]. In the datasets, the imbalance between mice and elephants depends on θ_L . For example, assuming $\theta_L = 100$ kB, only 12% of flows above 10 kB in the UNI1 dataset represent the elephant class (see Table I). The imbalance grows as θ_L increases.

Recall that flows classified as elephants are forwarded to the controller for further processing, thus introducing transmission and processing delays. Therefore, NELLY aims at detecting as many elephants, while negatively affecting as few latency-sensitive mice as possible. Considering elephants as the positive condition, the TPR describes the proportion of detected elephants, whereas the FPR provides the ratio of negatively affected mice. Both TPR and FPR range between 0 and 1. Furthermore, the Matthews correlation coefficient (MCC) was used to analyze the balance between the TPR and the FPR. The MCC takes all values from the confusion matrix to provide a measure between 1 and -1 . As the MCC gets closer to 1, the difference of the TPR over the FPR increases, leading to a more accurate classifier. An MCC between 0 and -1 means that $\text{TPR} \leq \text{FPR}$, which would be less accurate than a random classifier. In our experiment, the MCC values were always greater than 0; hence, we use a range between 0 and 1 to plot TPR, FPR, and MCC in Figs. 3 and 4.

The MCC metric is employed in the performance analysis because it is recommended for imbalanced datasets (like UNI1 and UNI2) [25]. The MCC score is only high when the classification algorithms are doing well in both the positive and negative elements (i.e., elephants and mice, respectively). The receiver operating characteristic curve has also proven to be useful

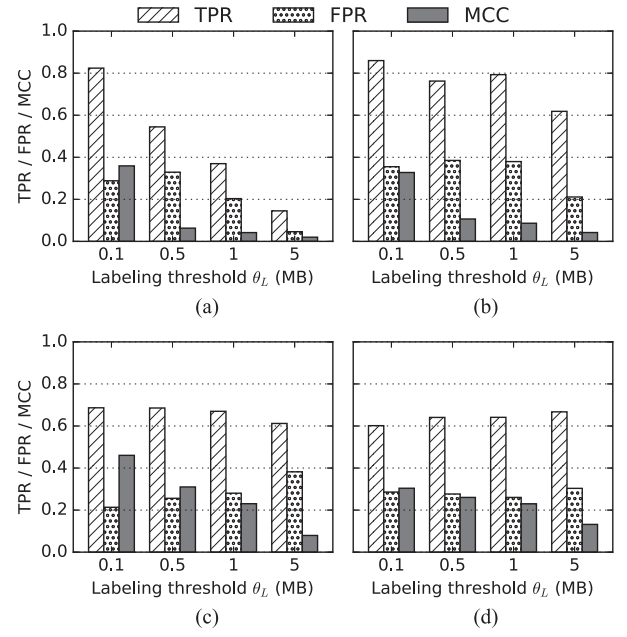


Fig. 3. Accuracy of NELLY with the ARF (left) and AHOT (right) algorithms when varying the labeling threshold (θ_L) for the UNI1 (top) and UNI2 (bottom) datasets. (a) ARF for UNI1. (b) AHOT for UNI1. (c) ARF for UNI2. (d) AHOT for UNI2.

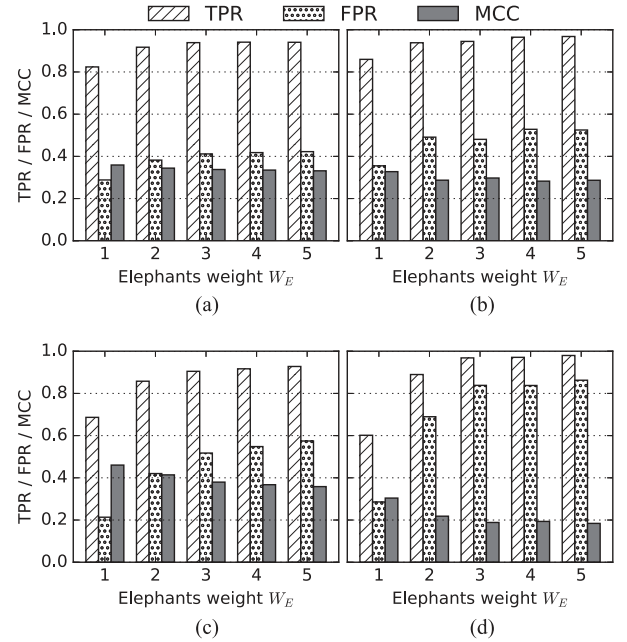


Fig. 4. Accuracy of NELLY with the ARF (left) and AHOT (right) algorithms when varying the range for the inverse weights of elephant flows (W_E) for the UNI1 (top) and UNI2 (bottom) datasets. (a) ARF for UNI1. (b) AHOT for UNI1. (c) ARF for UNI2. (d) AHOT for UNI2.

for imbalanced datasets, but it is more appropriate to analyze classification algorithms that output a real value [26]. Thus, we preferred the MCC because the output of the incremental learning classification algorithms employed in this article is a single class value (either mouse or elephant) rather than the real value.

TABLE II
PERFORMANCE OF NELLY WITH DIFFERENT INCREMENTAL LEARNING ALGORITHMS FOR CLASSIFYING FLOWS AS MICE AND ELEPHANTS

Algorithms	UNI1					UNI2				
	TPR(%)	FPR(%)	MCC	$T_C(\mu s)$	Header type	TPR(%)	FPR(%)	MCC	$T_C(\mu s)$	Header type
AHOT	85.97	35.52	0.327	4.07	BinNom	60.16	28.58	0.304	10.17	BinNom
ARF	82.39	28.82	0.359	12.01	BinNom	68.65	21.33	0.460	17.39	BinNom
Hoeffding tree	86.79	36.38	0.326	3.18	BinNom	57.92	28.46	0.284	4.64	BinNom
kNN-PAW	25.30	2.99	0.311	473.1	Num	40.29	10.22	0.302	454.1	Num
NB	74.76	35.69	0.254	4.76	BinNom	49.74	23.18	0.267	4.82	BinNom
OAUE	86.79	33.63	0.347	25.58	BinNom	63.28	28.65	0.332	33.06	BinNom
OzaBag	87.78	37.11	0.327	23.98	BinNom	64.17	31.13	0.314	36.61	BinNom
OzaBoost	75.88	29.93	0.307	11.56	BinNom	64.62	32.22	0.307	16.82	BinNom
Rule-NB	74.44	33.77	0.267	18.47	BinNom	54.83	29.15	0.248	18.59	BinNom
SGD-SVM	16.76	10.21	0.067	0.81	Num	38.69	30.99	0.076	0.8	Num

The top five results of TPR and MCC are in bold, and the T_C results shorter than 17.5 μs , for both UNI1 and UNI2.

C. Experiment Setup

Incremental learning algorithms are commonly evaluated using the interleaved test-then-train approach [27]. This approach refers to going through each flow to classify it first by working only with the online features and then use its actual class for training the flow size classification model. However, since flows start and end over time, some order of the flows must be established. Moreover, under real conditions, some flows start before a classified flow ends, whereas others end before a new flow starts. Therefore, the flows are classified at the start time, and the model is trained at the end time, so the performance evaluation will be based on more realistic conditions.

The imbalance of classes in the UNI1 and UNI2 datasets was addressed by training the flow size classification model using *inverse weights*, as in [18], i.e., weights (between 0 and 1) inversely proportional to the ratio of training instances previously encountered by the model for each class. If the model is trained with a *single weight* (i.e., 1 by default in the massive online analysis (MOA) tool [27]), it would tend to classify all flows as mice due to the imbalance of classes.

D. Performance Analysis

To determine the consideration for the best performance of NELLY, the UNI1 and UNI2 datasets were used with different header types (i.e., Num, BinNum, and BinNom), as well as 50 incremental learning classification algorithms available in MOA. The performance evaluation included the accuracy metrics (i.e., TPR, FPR, and MCC) and the classification time per flow (T_C). The algorithms were executed with their default settings (except for the training weights) and without previous model initialization.

For the sake of brevity, Table II presents ten algorithms, namely, adaptive Hoeffding option tree (AHOT), adaptive random forest (ARF), Hoeffding tree, k-nearest neighbors (kNN) with probabilistic adaptive windowing (PAW), Naive Bayes (NB), online accuracy updated ensemble (OAUE), OzaBag, Oza and Russell's Bagging (OzaBag) and Boosting (OzaBoost), rule classifier with NB (Rule-NB), and stochastic gradient descent (SGD) for support vector machines (SVMs). These algorithms were selected on the basis of the best performance results between algorithms with a similar learning approach. Furthermore,

Table II includes only the best results of each algorithm, taking into account both accuracy and classification time for a specific header type. The BinNom headers were found to enable the best performance of the majority of the algorithms for the UNI1 and UNI2 datasets. This was due to the fact that most algorithms achieved greater accuracy using the BinNom headers than the Num headers for a comparable classification time. The use of the BinNum headers is strongly discouraged; although similar or slightly better accuracy results were obtained, there was a significant increase in the classification time (up to $4\times$).

The accuracy results show that no single algorithm achieves the best values of the TPR and MCC for the UNI1 and UNI2 datasets. This is due to the fact that the flow size distribution and the features of the elephant and mouse flows were specific for each dataset. Therefore, the top five results were used to analyze the accuracy performance. Regarding the T_C , most algorithms introduced a classification delay per flow shorter than 17.5 μs , but this represents only a small percentage (7%) of the round-trip time (RTT) in DCNs (i.e., 250 μs in the absence of queuing [28]).

Both Hoeffding tree and NB represent the state of the art in incremental learning algorithms. Their simplicity and low computational cost enabled a very short delay ($T_C < 5 \mu s$) that accounts for only 2% of the RTT in DCNs. However, only the Hoeffding tree represents a valid alternative for the traffic similar to that of UNI1 because its TPR and MCC were among the top five results for the UNI1 dataset. The Hoeffding tree in MOA uses NB classifiers on the leaves (nodes), which improves the accuracy without compromising the computational cost.

The ARF, OAUE, OzaBag, and OzaBoost are ensemble-based algorithms that combine multiple Hoeffding trees (ten in our evaluation) for improving the accuracy at the expense of increasing the computational cost. The ARF and OzaBoost algorithms introduced a T_C shorter than 7% of the RTT in DCNs. ARF provided the best MCC and a TPR among the top five for the UNI1 and UNI2 datasets. OzaBoost can be seen as an option for the traffic similar to that of UNI2, since it was in the top five accuracy results only for the UNI2 dataset. In contrast, although the OAUE and OzaBag algorithms also provided good accuracy results (particularly for the UNI1 dataset), they introduced a T_C twice longer than the T_C of ARF and OzaBoost. This long T_C is because OAUE and OzaBag rely on ensemble methods (block-weighting and bagging, respectively) that demand more

computation than those used by ARF and OzaBoost (random forests and boosting, respectively).

Similar to ARF, the AHOT algorithm figured in the top five accuracy results for both datasets. Moreover, AHOT only introduced a T_C shorter than 2% (5 μ s) and 4% (10 μ s) of the RTT in DCNs for the UNI1 and UNI2 datasets, respectively. AHOT is capable of improving the accuracy of the Hoeffding tree algorithm without demanding too much computation by providing an intermediate solution between a single Hoeffding tree and an ensemble of Hoeffding trees. AHOT uses additional option paths (five maximum in our evaluation) to build a single structure that efficiently represents multiple Hoeffding trees.

The implementations in MOA of the Rule-NB, SGD-SVM, and kNN-PAW algorithms are strongly discouraged. The Rule-NB presented accuracy results outside the top five for both datasets and a T_C slightly longer than 7% of the RTT in DCNs. This is because rule-based algorithms focus on building more interpretable models than does the Hoeffding tree algorithm, which increases the computational cost but not necessarily improves the accuracy. The SGD-SVM algorithm introduced the shortest classification delay ($T_C < 1 \mu$ s), but it produced the worst values in the TPR and MCC metrics. The reason for these values is that MOA implements a very simple SGD-SVM algorithm that uses a linear kernel, which is not sufficient to model different patterns in flows of packets. The kNN-PAW provided the second-worst TPR for both datasets and a very long classification delay ($T_C > 450 \mu$ s), which increased up to 3000 μ s with the BinNum and BinNom headers (i.e., 12 $\times$ the RTT in DCNs). This long T_C value is a consequence of the computation of a distance metric by the algorithms based on kNN every time the classification is performed.

In conclusion, NELLY achieves the best performance by using the BinNom headers along with the following incremental learning algorithms.

- 1) The ARF is good for any type of traffic and if the RTT is flexible. It achieved the best MCC for the UNI1 and UNI2 datasets, and it was also the fifth- and second-best for the TPR while introduced a T_C lesser than 7.5% of the RTT in DCNs.
- 2) The AHOT is good for any type of traffic and a strict RTT. The TPR and MCC ranked among the top five for both datasets, while the T_C was shorter than that of the ARF, especially for the UNI1 dataset.
- 3) The Hoeffding tree is good for traffic similar to that of UNI1 and if the RTT is very strict. The TPR was the second-best and the MCC was the fifth (quite close to the AHOT) for the UNI1 dataset while introduced a very short T_C . When the RTT constraint takes precedence over the accuracy, this would be a good option for traffic similar to that of UNI2 because a very short T_C was maintained while provided the sixth-best TPR and MCC for such traffic.

The accuracy of NELLY was also evaluated with the ARF and AHOT algorithms for different values of θ_L , since this threshold may vary as a function of traffic and routing requirements. Both ARF and AHOT ranked among the top five in accuracy for both datasets with a T_C shorter than 7.5% of the RTT in DCNs.

As shown in Fig. 3, the MCC results of both algorithms were degraded as θ_L increased, especially for the UNI1 dataset. This is because the difference between the features of the elephants and mice becomes less significant as θ_L increases. In contrast, the TPR remained very similar as θ_L increased, except that the ARF suffered from a significant reduction in the TPR for the UNI1 dataset. Therefore, the AHOT was more robust to variations in θ_L for traffic similar to that of UNI1, although the performance of both algorithms was similar for the UNI2 dataset. Based on this summary, NELLY with the AHOT algorithm enables a flexible configuration of θ_L , while providing great elephant flow detection in data centers regardless of the type of traffic. For traffic similar to that of UNI2, both ARF and AHOT represent valid alternatives for the use of NELLY and the flexible configuration of θ_L is possible, since they perform similarly in terms of elephant detection.

Finally, the effect of the handling of different ranges of the inverse weights in the two classes on the accuracy of NELLY with the two algorithms (ARF and AHOT) was analyzed. The weights of the mice were maintained between 0 and 1, whereas the weights of the elephants ranged from 0 to W_E , where W_E varied from 1 to 5. Fig. 4 shows that both the ARF and AHOT algorithms achieved a higher TPR for both datasets as W_E increased (up to 94% and 98% of elephant detection, respectively). These results were expected, since establishing greater weights for the elephant class makes the learning algorithms increment the influence of the features of the elephant flows in the classification model. Moreover, the tradeoff between the TPR and FPR (i.e., MCC) remained quite similar for UNI1-type traffic, whereas that of UNI2 was degraded as W_E increased. This is due to the greater differences between the elephants and mice for the UNI1 dataset than for UNI2 when $\theta_L = 100$ kB. Therefore, as W_E increased for the UNI2 dataset, the increment of mouse flows wrongly classified as elephants (i.e., FPR) was greater than that of elephant flows correctly classified (i.e., TPR). In conclusion, NELLY supports a flexible configuration of inverse weights for meeting different accuracy requirements.

IV. COMPARATIVE ANALYSIS

NELLY was compared with the online flow size prediction (OFSP) [18], the efficient sampling and classification approach (ESCA) [19], FlowSeer [20], and Mahout [15]. OFSP, ESCA, and FlowSeer incorporate ML at the controller side of SDDCNS for proactively detecting elephant flows, whereas Mahout performs reactive detection at the server side. The results reported by each work for the UNI1 dataset were used to compare them in relation to learning approach, elephant detection, false elephants, table occupancy, control overhead, detection time, network modifications, and performance factors. The works involving Hedera [16] and DevoFlow [14] were not considered. These approaches perform reactive flow detection and their limitations hinder real implementation. Hedera causes large control traffic overhead and has poor scalability, whereas DevoFlow requires custom-made switch hardware and imposes a heavy burden on switches.

A. Learning Approach

ML algorithms used for detecting elephant flows can involve batch or incremental learning. Batch learning refers to the use of training models based on static datasets (i.e., all training data are simultaneously available). However, batch learning requires the storage of unprocessed data to cope with traffic variations in DCNs, so the models must repeatedly work from scratch. This is time consuming and prone to outdated models. Conversely, incremental learning continuously adapts the ML models on the basis of streams of training data, enabling constantly updated models and reducing time and memory requirements [17], [21]. The ESCA relies on batch learning, whereas NELLY and OFSP rely on incremental learning for detecting elephant flows. FlowSeer is a mixed approach using batch learning for the identification of potential elephants and incremental learning for the classification of the potential ones. Mahout has no learning approach, since it performs reactive elephant detection.

B. Elephant Detection

The main goal of flow detection methods is to identify elephant flows (i.e., TPR). NELLY, OFSP, and FlowSeer all proactively detected more than 95% of elephant flows, whereas the ESCA detected a maximum of 88.3%. Mahout provides perfect detection, although this is reactive.

C. False Elephants

Mouse flows mistakenly identified as elephants (i.e., FPR) are needlessly forwarded to and processed by the controller. For achieving the highest elephant detection rate, FlowSeer informed the controller of 29% of mice as potential elephants, whereas OFSP and ESCA only reported around 2%. NELLY yielded an FPR of 40%, but this was computed using only 7% of the flows (i.e., $\theta_L \geq 10$ kB). NELLY thus forwards only 2.5% of mice to the controller. No mouse flow is reported to the controller by Mahout, since detection is reactive.

D. Table Occupancy

Controller-side flow detection methods install flow table entries in ToR switches for centrally collecting flow data. The smaller the number of flow table entries, the more efficient is the resource utilization. OFSP requires one entry per flow, thus constraining its scalability because of the limited memory in SDN switches. ESCA and FlowSeer install wildcard entries for sampling packets of flows. They reported 236 and 50 flow table entries, respectively, for achieving their highest detection rate in the UNI1 dataset. Conversely, NELLY and Mahout do not require flow table entries for collecting data, since they operate at the server side.

E. Control Overhead

Flow detection methods require ToR switches to send control packets to the controller, either for the collection of flow data or for the reporting of detected elephant flows. The smaller the control overhead, the lower are the link utilization and the impact

on the controller performance (since it has to process fewer control packets). The overhead of this control was computed by assuming no loss in the network and a control packet of 64 bytes. OFSP collects information from the first three packets of each flow, generating a control overhead of 402 kb/s. FlowSeer collects information from the first five packets of sampled flows (i.e., 30% of the flow data) and potential elephants, yielding a control overhead of 288 kb/s. The ESCA reduces the control overhead to 215 kb/s by using a sampling method that only reports information from the first packet. In contrast, NELLY and Mahout merely require that ToR switches send information of flows marked as elephants, greatly reducing the control message overhead to 4.4 and 1.1 kb/s, respectively.

F. Detection Time

Timely detection of elephant flows enables the controller to make early decisions to improve routing. OFSP, ESCA, FlowSeer, and NELLY enable a short detection time by proactively detecting elephant flows. The ESCA reported a detection time of 1.98 s for achieving the highest detection rate. OFSP and NELLY detect elephants in a shorter time, since they rely on the first N packets. On average, the detection time was 0.5 s for OFSP ($N = 3$) and 0.8 s for NELLY ($N = 7$). Further experimentation is needed to evaluate the detection time of FlowSeer. Nevertheless, the detection time of the latter would be slightly greater than for ESCA, since it is also based on sampling and considers the first five packets (ESCA considers only one packet). In contrast, Mahout relies on a reactive mechanism that detects elephant flows after their corresponding socket buffer in a server surpasses a certain threshold. Assuming a small threshold of 100 kB, the average detection time of Mahout is 3.8 s. However, unlike ML-based flow detection methods, the detection time of Mahout becomes longer as the threshold increases, which may cause hotspots before the traffic carried by elephant flows reaches the threshold.

G. Network Modifications

ESCA proposes a sampling method that depends on nonexisting SDN specifications, hence requiring custom-made switch hardware. In contrast, OFSP, FlowSeer, NELLY, and Mahout rely on OpenFlow [29], therefore enabling the use of commercial switches. Essentially, NELLY and Mahout require the installation of additional software in the servers, which need only to be done once with further configuration possible on the basis of a policy manager or autonomously. This installation can be carried out by using DevOps automation tools, such as Puppet and Chef, that support the distribution of software components to the OSs of servers [30]. Moreover, virtualization platforms, such as VMWare and Xen, support software distribution to the servers as updates to the hypervisor without interrupting running virtual machines (either by live-migration or live-updating) [31].

H. Performance Factors

Depending on the location of the flow detection method, different factors may affect its performance. Controller-side

methods (i.e., OFSP, ESCA, and FlowSeer) rely on the resources available at the controller and ToR switches. The controller should be powerful enough for detecting all the elephants and processing the control packets sent by the ToR switches in the DCN. Similarly, the ToR switches should have enough memory for installing the required flow table entries. Moreover, the accuracy of the controller-side methods can be negatively affected if the ToR switches drop some of the first packets of the elephant flows. On the other hand, NELLY and Mahout operate at the server side, so they depend on servers resources. As NELLY is based on ML, it requires more resources than does Mahout. Both server-side methods detect the elephants generated by each server (i.e., distributed operation). Note that servers should be able to monitor the first packets of the elephant flows for avoiding a decrease in accuracy.

V. CONCLUSION

In this article, we introduced NELLY to deal with the inaccuracy, large overhead, and poor scalability of current flow detection methods utilized in SDDCNs. NELLY is a novel flow detection method based on incremental learning that operates as a software component installed in every server of SDDCNs. An extensive evaluation demonstrated the accuracy and speed of NELLY, as well as its generation of low traffic overhead and adaptation to varying traffic characteristics. NELLY performs continuous learning and requires limited memory resources when used with the ARF and AHOT algorithms. The evaluation also corroborated the scalability of NELLY and the fact that no modifications in SDN standards are required.

As future work, we intend to implement NELLY as an in-kernel software component for evaluating its impact cost to server resources, including processing and memory consumption. Furthermore, we plan to evaluate NELLY in an emulated SDDCN by installing the software component into micro virtual machines connected to Open vSwitch instances.

Finally, although this article proved that incremental learning algorithms were efficient to detect elephant flows in DCNs, there are still research challenges to be addressed. First, there is no consistent and accepted method for defining the threshold value that discriminates between mice and elephants in DCNs. In this article, we evaluated different thresholds but did not specify how to select the appropriate threshold value for the traffic and routing requirements. Second, there was a need to create publicly available IPv6 dataset to allow the performance of ML-based elephant detection methods on such a dataset.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their insightful comments and suggestions that helped to improve the quality of this paper.

REFERENCES

- [1] Z. Lv, H. Song, P. Basanta-Val, A. Steed, and M. Jo, "Next-generation big data analytics: State of the art, challenges, and future research topics," *IEEE Trans. Ind. Informat.*, vol. 13, no. 4, pp. 1891–1899, Aug. 2017.
- [2] M. Aazam, S. Zeadally, and K. A. Harras, "Deploying fog computing in Industrial Internet of Things and Industry 4.0," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4674–4682, Oct. 2018.
- [3] D. A. Cheklsed, L. Khoukhi, and H. T. Mouftah, "Industrial IoT data scheduling based on hierarchical fog computing: A key for enabling smart factory," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4590–4602, Oct. 2018.
- [4] N. L. S. Da Fonseca and R. Boutaba, *Cloud Services, Networking, and Management*, 1st ed. Hoboken, NJ, USA: Wiley, 2015.
- [5] S. K. Singh, T. Das, and A. Jukan, "A survey on Internet multipath routing and provisioning," *IEEE Commun. Surv. Tuts.*, vol. 17, no. 4, pp. 2157–2175, Fourth Quarter 2015.
- [6] M. Chiesa, G. Kindler, and M. Schapira, "Traffic engineering with equal-cost-multipath: An algorithmic perspective," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 779–792, Apr. 2017.
- [7] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 123–137, Aug. 2015.
- [8] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM SIGCOMM Conf. Internet Meas. Conf.*, Melbourne, VIC, Australia, Nov. 2010, pp. 267–280.
- [9] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. ACM SIGCOMM Conf. Internet Meas. Conf.*, Chicago, IL, USA, Nov. 2009, pp. 202–208.
- [10] M. H. ur Rehman, I. Yaqoob, K. Salah, M. Imran, P. P. Jayaraman, and C. Perera, "The role of big data analytics in Industrial Internet of Things," *Future Gener. Comput. Syst.*, vol. 99, pp. 247–259, Oct. 2019.
- [11] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, Feb. 2013.
- [12] F. Estrada-Solano, A. Ordonez, L. Z. Granville, and O. M. C. Rendon, "A framework for SDN integrated management based on a CIM model and a vertical management plane," *Comput. Commun.*, vol. 102, pp. 150–164, Apr. 2017.
- [13] F. Amezcua-Suarez, F. Estrada-Solano, N. L. S. da Fonseca, and O. M. C. Rendon, "An efficient mice flow routing algorithm for data centers based on software-defined networking," in *Proc. IEEE Int. Conf. Commun.*, Shanghai, China, May 2019, pp. 1–6.
- [14] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.
- [15] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead data-center traffic management using end-host-based elephant detection," in *Proc. IEEE INFOCOM*, Shanghai, China, Apr. 2011, pp. 1629–1637.
- [16] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, San Jose, CA, USA, Apr. 2010, p. 19.
- [17] R. Boutaba *et al.*, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *J. Internet Services Appl.*, vol. 9, no. 1, p. 16, Jun. 2018.
- [18] P. Poupart *et al.*, "Online flow size prediction for improved network routing," in *Proc. IEEE 24th Int. Conf. Netw. Protocols*, Singapore, Nov. 2016, pp. 1–6.
- [19] F. Tang, L. Li, L. Barolli, and C. Tang, "An efficient sampling and classification approach for flow detection in SDN-based big data centers," in *Proc. IEEE 31st Int. Conf. Adv. Inf. Netw. Appl.*, Taipei, Taiwan, Mar. 2017, pp. 1106–1115.
- [20] S. C. Chao, K. C. J. Lin, and M. S. Chen, "Flow classification for software-defined data centers using stream mining," *IEEE Trans. Services Comput.*, vol. 12, no. 1, pp. 105–116, Jan./Feb. 2019.
- [21] S. Ayoubi *et al.*, "Machine learning for cognitive network management," *IEEE Commun. Mag.*, vol. 56, no. 1, pp. 158–165, Jan. 2018.
- [22] A. Zarek, "Openflow timeouts demystified," master's thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2012.
- [23] T. Benson, "Data set for IMC 2010 data center measurement," University of Wisconsin-Madison, Oct. 1, 2018. [Online]. Available: http://pages.cs.wisc.edu/tbenson/IMC10_Data.html
- [24] F. Estrada-Solano, "NELLY datasets," GitHub, May 29, 2019. [Online]. Available: <https://github.com/festradasolano/nelly/tree/master/nelly-ml/analysis/datasets>
- [25] S. Boughorbel, F. Jarray, and M. El-Anbari, "Optimal classifier for imbalanced data using matthews correlation coefficient metric," *PLOS One*, vol. 12, no. 6, pp. 1–17, Jun. 2017.

- [26] D. Chicco, "Ten quick tips for machine learning in computational biology," *BioData Mining*, vol. 10, no. 35, pp. 1–17, Dec. 2017.
- [27] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive online analysis," *J. Mach. Learn. Res.*, vol. 11, pp. 1601–1604, Aug. 2010.
- [28] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 63–74, Aug. 2010.
- [29] N. McKeown *et al.*, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.
- [30] M. Migliarina, "Application deployment and management in the cloud," in *Proc. 16th Int. Symp. Symbolic Numeric Algorithms Sci. Comput.*, Timișoara, Romania, Sep. 2014, pp. 422–428.
- [31] F. F. Brasser, M. Bucicoiu, and A.-R. Sadeghi, "Swap and play: Live updating hypervisors and its application to Xen," in *Proc. ACM Workshop Cloud Comput. Secur.*, Scottsdale, AZ, USA, Nov. 2014, pp. 33–44.



Felipe Estrada-Solano (GS'13) received the bachelor's degree in electronics and telecommunications engineering and the M.Sc. degree in telematics engineering from the University of Cauca, Popayán, Colombia, in 2016 and 2010, respectively. He is currently working toward the Ph.D. degree in telematics engineering with the University of Cauca and the Ph.D. degree in computer science with the State University of Campinas, Campinas, Brazil.

His research interests include network and service management, cloud computing, network virtualization, network softwarization, and machine learning.



Oscar M. Caicedo (GS'11–M'15) received the bachelor's degree in electronics and telecommunications engineering and the M.Sc. degree in telematics engineering from the University of Cauca, Popayán, Colombia, in 2001 and 2006, respectively, and the Ph.D. degree in computer science from the Federal University of Rio Grande do Sul, Porto Alegre, Brazil, in 2015.

He is currently a Full Professor with the Department of Telematics, University of Cauca, where he is a member of the Telematics Engineering Group. His research interests include network and service management, network virtualization, and software-defined networking.



Nelson L. S. da Fonseca (M'88–SM'01) received the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 1994.

He is currently a Full Professor with the Department of Computing Systems, Institute of Computing, State University of Campinas, Campinas, Brazil. He has authored or coauthored more than 400 papers and supervised more than 70 graduate students.

Dr. da Fonseca is currently the Vice-President of Technical and Educational Activities of the IEEE Communications Society (ComSoc). He served as the ComSoc Vice President of Publications and of Member Relations, and the Director of Conference Development, of Latin America Region, and of On-Line Services. He is the former Editor-in-Chief of the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS. He was a recipient of the 2012 IEEE ComSoc Joseph LoCicero Award for Exemplary Service to Publications, the Medal of the Chancellor of the University of Pisa in 2007, and the Elsevier *Computer Networks* Journal Editor of Year 2001 Award.