UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

Francisco Germano Vogt

In-band telemetry using inter-packet gap metrics for network and service management

Telemetria em banda usando métricas de intervalo entre pacotes para gerenciamento de rede e serviço

Campinas

2023

Francisco Germano Vogt

In-band telemetry using inter-packet gap metrics for network and service management

Telemetria em banda usando métricas de intervalo entre pacotes para gerenciamento de rede e serviço

Dissertation presented to the Faculty of Electrical and Computer Engineering of the University of Campinas in partial fulfillment of the requirements for the degree of Master, in the area of Computer Engineering.

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica, na Área de Engenharia de Computação.

Supervisor: Prof. Dr. Christian Rodolfo Esteve Rothenberg

Este exemplar corresponde à versão final da tese defendida pelo aluno Francisco Germano Vogt, e orientada pelo Prof. Dr. Christian Rodolfo Esteve Rothenberg

Campinas

2023

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

V868i

Vogt, Francisco Germano, 1999-
In-band telemetry using inter-packet gap metrics for network and service management / Francisco Germano Vogt. – Campinas, SP : [s.n.], 2023.

Orientador: Christian Rodolfo Esteve Rothenberg.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes definidas por software (tecnologia de rede de computador). 2. Software - Desempenho. 3. Redes locais de computação. I. Esteve Rothenberg, Christian Rodolfo, 1982-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações Complementares

Título em outro idioma: Telemetria em banda usando métricas de intervalo entre pacotes para gerenciamento de rede e serviço
Palavras-chave em inglês:
Software defined networks
Software - Performance
Local computer networks
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora:
Christian Rodolfo Esteve Rothenberg [Orientador]
Marco Aurélio Amaral Henriques
Rodolfo da Silva Villaca
Data de defesa: 12-06-2023
Programa de Pós-Graduação: Engenharia Elétrica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0003-1467-7146
- Currículo Lattes do autor: http://lattes.cnpq.br/5626844343227840

COMISSÃO JULGADORA - DISSERTAÇÃO DE MESTRADO

Candidato: Francisco Germano Vogt RA: 234632

Data da Defesa: 12 de junho de 2023

Título da Tese: "In-band telemetry using inter-packet gap metrics for network and service management"

Prof. Dr. Christian Rodolfo Esteve Rothenberg (FEEC/UNICAMP)(Presidente)

Prof. Dr. Marco Aurélio Amaral Henriques (FEEC/UNICAMP)

Prof. Dr. Rodolfo da Silva Villaca (UFES)

A ata de defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no SIGA (Sistema de Fluxo de Dissertação/Tese) e na Secretaria de Pós Graduação da Faculdade de Engenharia Elétrica e de Computação.

This work is dedicated to my family, friends, and everyone who somehow participated in my academic trajectory, especially my mother, Noemi, who raised me alone and went to great lengths to get me here.

Acknowledgements

First, I would like to thank my family, especially my mother, for the love, affection, understanding, and help they have given me so far. Indeed, without your support, none of this would have been possible. In addition to the family, I especially thank two professors, my advisor Prof. Christian Esteve Rothenberg, and former advisor Prof. Marcelo Caggiani Luizelli, whom I am very proud to have been guided. So, thank you for all the motivation, teachings, opportunities, and friendship offered throughout my trajectory. Also, I'm very grateful for all the friends I've made in Campinas, especially my housemates. So, to all my friends from Republica Marimbondos, thank you very much for the reception, hospitality, support, and principally true friendship.

This work was supported by the Innovation Center, Ericsson S.A. and by the Sao Paulo Research Foundation (FAPESP), grant 2021/00199-8, CPE SMARTNESS.

Abstract

With the rapid expansion of network traffic and the growing demand for high-quality multimedia applications, the need to ensure optimal network performance and user satisfaction has become increasingly crucial. Real-time monitoring and analysis of network behavior have emerged as essential practices in this evolving landscape. In-band Network Telemetry (INT) has quickly gained recognition as a powerful technique for capturing fine-grained network measurements, enabling network management to make informed and intelligent decisions.

However, the effectiveness of INT data collection is influenced by various factors, such as packet size limitations and processing overhead. Therefore, it becomes crucial to carefully select the data to be collected and determine the most appropriate timing for data collection. In this research, we propose a novel approach to network monitoring by leveraging the Inter Packet Gap (IPG) metric, collected using INT, as the primary metric of interest. By monitoring the IPG hop-by-hop, we can harness this valuable information to identify and detect network anomalies more accurately.

This study focuses on two critical use cases where the application of INT and IPG metrics proves to be particularly beneficial: microburst detection and Quality of Experience (QoE) inference in Virtual Reality (VR) video streaming applications. Through extensive research and development, we have devised autonomous strategies that operate exclusively on the data plane, eliminating the need for any intervention from the control plane. These strategies have been designed to efficiently detect microbursts, which are rapid bursts of network traffic that can lead to performance degradation, as well as estimate the QoE for VR video streaming applications.

By integrating INT and IPG metric analysis into the data plane, our strategies offer significant advantages in terms of real-time monitoring and decision-making. The autonomous nature of our approach enables timely and accurate detection of network anomalies, allowing for proactive management and mitigation of potential issues. Furthermore, by estimating QoE in the data plane, we can gain valuable insights into the user experience during VR video streaming, enabling network operators to optimize their services and enhance user satisfaction.

Keywords: SDN; P4; Network monitoring.

Resumo

Com a rápida expansão do tráfego de rede e a crescente demanda por aplicativos multimídia de alta qualidade, a necessidade de garantir o desempenho ideal da rede e a satisfação do usuário tornou-se cada vez mais crucial. O monitoramento e a análise em tempo real do comportamento da rede surgiram como práticas essenciais nesse cenário em evolução. A Telemetria de Rede In-Band (INT) ganhou rapidamente reconhecimento como uma técnica poderosa para capturar medições de rede refinadas, permitindo que o gerenciamento de rede tome decisões informadas e inteligentes.

No entanto, a eficácia da coleta de dados INT é influenciada por vários fatores, como limitações de tamanho de pacote e sobrecarga de processamento. Portanto, torna-se crucial selecionar cuidadosamente os dados a serem coletados e determinar o momento mais adequado para a coleta de dados. Nesta pesquisa, propomos uma nova abordagem para monitoramento de rede, aproveitando a métrica Inter Packet Gap (IPG), coletada usando INT, como a principal métrica de interesse. Ao monitorar o IPG salto a salto, podemos aproveitar essas informações valiosas para identificar e detectar anomalias de rede com mais precisão.

Este estudo se concentra em dois casos de uso críticos em que a aplicação da métrica INT e IPG se mostra particularmente benéfica: detecção de microburst e inferência de Qualidade de Experiência (QoE) em aplicativos de streaming de vídeo de Realidade Virtual (VR). Através de extensa pesquisa e desenvolvimento, criamos estratégias autônomas que operam exclusivamente no plano de dados, eliminando a necessidade de qualquer intervenção do plano de controle. Essas estratégias foram projetadas para detectar microbursts com eficiência, que são rajadas rápidas de tráfego de rede que podem levar à degradação do desempenho, bem como estimar a QoE para aplicativos de streaming de vídeo VR.

Ao integrar a análise métrica INT e IPG no plano de dados, nossas estratégias oferecem vantagens significativas em termos de monitoramento em tempo real e tomada de decisões. A natureza autônoma de nossa abordagem permite a detecção oportuna e precisa de anomalias de rede, permitindo o gerenciamento proativo e a mitigação de possíveis problemas. Além disso, ao estimar a QoE no plano de dados, podemos obter informações valiosas sobre a experiência do usuário durante o streaming de vídeo VR, permitindo que as operadoras de rede otimizem seus serviços e melhorem a satisfação do usuário.

Palavras-chaves: Redes definidas por software; P4; Monitoramento de rede.

"Sempre fui sonhador, e é isso que me mantém vivo" (Racionais MC's)

List of Figures

List of Tables

Acronyms

ABR Adaptative Bit Rate.

ANN Artificial Neural Networks.

ASIC Application-specific Integrated Circuit.

CLI Command-Line Interface.

CPU Central Processing Unit.

CUSUM Cumulative Sums.

DAC Direct Attach Copper.

DDoS Distributed Denial-of-Service.

DEMA Double Exponentially Weighted Moving Average.

DT Decision Trees.

EITF Internet Engineering Task Force.

EWMA Exponentially Weighting Moving Average.

FOV Field Of View.

FPGA Field Programmable Gate Array.

ICMP Internet Control Message Protocol.

INT In-band Network Telemetry.

INT-MD INT eMbed Data.

INT-MX INT instruct(X)ions Data.

INT-XD INT eXport Data.

IPG Inter-Packet Gap.

IPGNET in-band Inter Packet Gap NEtwork Telemetry.

KNN K-Nearest Neighbors.

MAU	Match-Actions Units.
MOS	Mean Opinion Score.
NIC	Network Interface Card.
NPU	Network Processing Unit.
NWDAF	Network Data and Applica- tion Function.
P4	Programming Protocol-Independent Packet Processors.
P7	P4 Programmable Patch Panel.
QoE	Quality of Experience.
QoS	Quality of Service.
RF	Random Forests.
RTT	Round Trip Time.
SDE	Software Developer Environment.
SDN	Software Defined Networking.
SFP	Small Form Pluggable.
SNMP	Simple Network Management Protocol.
TNA	Tofino Native Architecture.
VR	Virtual Reality.

Contents

1 Introduction

1.1 Context and Motivation

The rapid growth of network traffic in recent years has presented significant challenges for network operators and service providers. Bandwidth-intensive applications, such as video streaming, Virtual Reality (VR), and cloud-based services are already a reality. VR head-mounted displays are expected to grow to nearly 34 million by the end of 2023, while the associated network traffic is expected to increase at least 12-fold (CISCO, 2019). Consequently, the networks face unprecedented demands in terms of capacity, latency, and quality of service. As a result, monitoring and analyzing network performance have become crucial for ensuring efficient operations and delivering a satisfactory user experience.

In-band Network Telemetry (INT) (TAN *et al.*, 2021) is an emerging technique for network monitoring in the Software Defined Networking (SDN) context (KREUTZ *et al.*, 2014). Different from the traditional network monitoring techniques like Internet Control Message Protocol (ICMP) (GUO *et al.*, 2015), Traceroute (AGARWAL *et al.*, 2014), NetFlow (SOMMER; FELDMANN, 2002), and Simple Network Management Protocol (SNMP) (HARRINGTON *et al.*, 2002), INT can provide finer-grained network monitoring with better network visibility and coverage (LIU *et al.*, 2018). Through programmable devices and high-level languages like Programming Protocol-Independent Packet Processors (P4) (BOSSHART *et al.*, 2014), INT can collect a large and flexible set of information (e.g., ingress timestamp, queue length) by the network. This information can be used to detect and mitigate several network problems, like network congestion, packet loss, and delay. Each application used to detect and mitigate these problems requires a specific set of network information. However, INT data collection is limited by some network factors (e.g., bandwidth and latency of INT processing (KIM *et al.*, 2018)), making it too costly to collect all network information every time due to the associated overheads.

Due to concerns regarding scalability and overhead in running INT, it is crucial to carefully select the collected information. Recent research efforts have focused on INT orchestration, maximizing, and optimizing information collection (HOHEMBERGER *et al.*, 2019) (CANOFRE *et al.*, 2022). In this work, we propose a different approach by arguing that instead of collecting more network information, it is more beneficial to focus on gathering information that can be utilized for multiple use cases. In this context, the

Inter-Packet Gap (IPG) emerges as a promising network measurement and management metric. The IPG metric offers several advantages: it enables the identification of a broader range of events with high accuracy compared to traditional metrics, and it requires minimal storage capacity (e.g., only 16 bits for IPG size in heavy hitters detection (SINGH *et al.*, 2022). Previous studies have demonstrated the effectiveness of IPG for various network problems, including packet loss (SA-INGTHONG *et al.*, 2021), delay measurement (PIRATLA *et al.*, 2004), and heavy hitters detection (SINGH *et al.*, 2022).

1.2 Research Goals

We present in-band Inter Packet Gap NEtwork Telemetry (IPGNET), a system to detect network anomalies based on the analysis and collection of the IPG. The main idea is to develop a low-cost autonomous data plane solution to detect and report these anomalies. In our context, we focus specifically on two use cases: (i) The microbursts detection report and (ii) The QoE estimation of VR video streaming.

Microbursts, short-duration bursts of high packet intensity, often remain unnoticed by traditional network monitoring methods due to their brief duration and high data rates. These microbursts can cause network congestion, packet loss, and ultimately degrade application performance. Additionally, providing an accurate Quality of Experience (QoE) inference for VR video streaming allows for targeted optimizations, reducing latency, improving video quality, and delivering an immersive user experience. The Figure 1.1 shows the proposed architecture with the microburst detector and the QoE estimator running autonomously on the data plane, and reporting the data just when necessary, using INT.

Furthermore, we seek to evaluate our strategies using high-fidelity experiments. For this, we focus on implementing our solution and running the experiments using physical hardware (i.e., Tofino switch). However, as we have a limited physical network infrastructure available, we look for alternatives to perform these high-fidelity experiments. Nevertheless, the existing software-based solutions like Mininet (LANTZ *et al.*, 2010) can suffer from performance and scalability limitations, while hardware-based solutions like (KANNAN *et al.*, 2018) are limited by factors like, not providing support for emulating all link characteristics and the support of a custom P4 code.

Then, to address these limitations, we propose the development of an emulation tool capable of simulating different events with high fidelity. So, we proposed P4 Programmable Patch Panel (P7). With P7, we can include in our experiments limitations like bandwidth, latency, jitter, and packet loss. Thus, we can evaluate how the proposed strategies behave in different network scenarios, even using a physical device like Tofino.

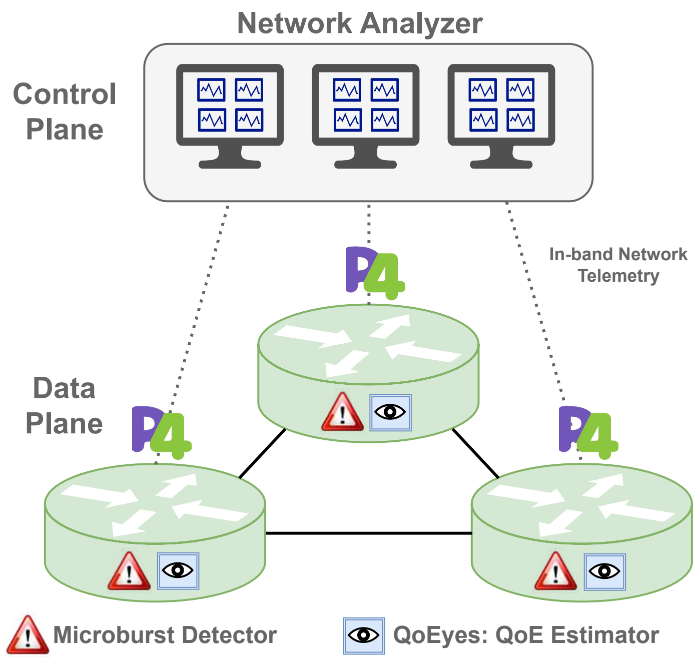

Figure 1.1 – IPGNET architecture.

So, the objectives of this work can be summarized as:

- Explore the capabilities of IPG as a network metric for capturing fine-grained packet transmission characteristics.
- Develop algorithms and methodologies for microburst detection using IPG monitoring within the INT framework.
- Investigate the correlation between IPG metrics and the estimated QoE in VR video streaming.
- Evaluate the performance and effectiveness of the proposed techniques through extensive experimental simulations and real-world network deployments.

1.3 Main Contributions

Considering the proposed objectives, the contributions of this dissertation can be summarized as follows:

- **Data plane autonomous solution**. We have developed a low-cost solution to two network problems that are able to operate autonomously in the data plane without the need for intervention from the control plane;
- **Microburst detection strategy**. We implemented a microburst detection strategy able to report the contributing flows;

- **QoE estimation in the data plane**. We designed an in-network VR video streaming QoE estimation directly in the data plane;
- **Realistic implementation and evaluation**. We developed our prototype implementation using Barefoot Tofino hardware;
- **Advances to P4-based network emulation.** We contributed to the development of a P4-based network emulation platform that offers high-fidelity 100G traffic network emulation;
- **Open-source software artifacts.** Both our use cases and our P4-based emulator are open-source projects.

This work has benefited from previously published demos and papers, taking advantage of the feedback received from the network community to improve the work quality. Below we summarize the publications made during the development of this work, as well as the awards received.

1.3.1 Publications

1. Fabricio Rodriguez, **Francisco Germano Vogt** Ariel Góes De Castro, Marcos Felipe Schwarz, and Christian Rothenberg. "P4 Programmable Patch Panel (P7): An Instant 100G Emulated Network on Your Tofino-based Pizza Box". In: *ACM SIGCOMM 2022 Demos and Posters* 2022.
2. **Francisco Germano Vogt** Fabricio Rodriguez, Christian Rothenberg, and Gergely Pongrácz. "In-band Inter Paket Gap Telemetry (IPGNET): Unlocking Novel Network Monitoring Methods". In: *2022 IEEE Global Communications Conference (GLOBECOM) Industry Demo Sessions*. IEEE. 2022.
3. **Francisco Germano Vogt** Fabricio Rodriguez, Christian Rothenberg, and Gergely Pongrácz. Innovative network monitoring techniques through In-band Inter Packet Gap Telemetry (IPGNET). In *P4 Workshop in Europe (EuroP4 '22), December 9, 2022, Roma, Italy.* ACM, New York, NY, USA, 4 pages.
4. Fabricio Rodriguez, **Francisco Germano Vogt** Ariel Góes De Castro, Marcos Felipe Schwarz, and Christian Rothenberg. "Network Emulation with P7: A P4 Programmable Patch Panel on Tofino-based Hardware". In: *Salão de Ferramentas do Simpósio Brasileiro de Redes de Computadores 2023.*
5. **Francisco Germano Vogt**, Fabricio Rodriguez, Ariel Góes De Castro, Marcelo Luizelli, Christian Rothenberg, and Gergely Pongrácz. "QoEyes: Towards Virtual

Reality Streaming QoE Estimation Entirely in the Data Plane". In: *Manuscript Accepted in IEEE International Conference on Network Softwarization. 2023.*

6. **Francisco Germano Vogt**, Fabricio Rodriguez, Ariel Góes De Castro, Marcelo Luizelli, Christian Rothenberg, and Gergely Pongrácz. "Demo of QoEyes: Towards Virtual Reality Streaming QoE Estimation Entirely in the Data Plane". In: *Demo Accepted in IEEE International Conference on Network Softwarization. 2023.*

7. Fabricio Rodriguez, **Francisco Germano Vogt** Ariel Góes De Castro and Christian Rothenberg. "Towards Multiple Pipelines Network Emulation with P7". In: *Demo Accepted in IEEE International Conference on Network Softwarization. 2023.*

1.3.2 Awards

- ★ **2º Best Paper** of the Student Research Competition at ACM SIGCOMM 2022. Title: "P4 Programmable Patch Panel (P7): An Instant 100G Emulated Network on Your Tofino-based Pizza Box", Aug, 2022.
- ★ **Best Paper** presented at Salão de Ferramentas do Simpósio Brasileiro de Redes de Computadores. Title: "Network Emulation with P7: A P4 Programmable Patch Panel on Tofino-based Hardware", May, 2023.

1.3.3 Open Source Artefacts

- **P7 Emulator.** <https://github.com/intrig-unicamp/p7>
- **IPGNET Microburst Detector.** <https://github.com/intrig-unicamp/IPGNET>
- **QoEyes.** <https://github.com/intrig-unicamp/QoEyes>

1.4 Outline

The dissertation is organized as follows. Chapter 2 presents a literature review by introducing the key background topics and related work. Chapter 3 presents the design an implementation of our proposed system (IPGNET) along with the problem definition and a discussion on challenges and limitations. Chapter 4 introduces the P7 network emulator used in our experimental evaluation, which is the focus of Chapter 5 divided into the two main use cases explored. Finally, Chapter 6 concludes the work with final remarks and future work perspectives.

2 Literature Review

In this chapter, we review relevant topics in the literature for our work. For this, we start reviewing the background concepts in section 2.1, and next, we review recent research efforts related to our work in section 2.2, emphasizing our contributions.

2.1 Background

In this section, we discuss the background topics related to our proposed system. These topics include an overview of SDN, P4 and Tofino Native Architecture (TNA), INT, IPG metric, QoE, and VR video streaming. Each of these sections plays a fundamental role in the context of the work carried out, and some of them are directly linked to the development of the proposed method

2.1.1 Software-Defined Networks (SDN)

SDN is an architectural approach to network design and management that separates the control plane from the data plane. It introduces programmability, centralized control, and network virtualization, offering greater flexibility, scalability, and agility in network operations. These benefits are included because with the control plane decoupled from the data plane and centralized in a software-based controller, the controller acts as a single point of control for the entire network, providing a global view and making intelligent decisions on how to forward traffic. In the next sessions, we discuss how SDN is implemented, highlighting its differences with traditional network architecture, and its key benefits such as network programmability.

2.1.1.1 Traditional vs Software-Defined Networks

Since the adoption of the traditional IP networks, tasks like network management and reconfiguration are complex due to the complexity of managing this kind of networks (BENSON *et al.*, 2009). One example of these difficulties is the migration from IPv4 to IPv6 (WU *et al.*, 2012), which has not been finished yet. Furthermore, the configuration of current networks based on predefined policies presents significant complexities, rendering the network inflexible and inefficient in responding to faults and changes in workload. When network operators need to establish new high-level policies, they must individually configure each network device, further adding to the intricacy. Moreover, implementing these policies often involves using device vendor-specific Command-Line

Interface (CLI) and low-level commands (Kim; Feamster, 2013). Additionally, the tight coupling of the management, forwarding, and control planes in network devices exacerbates the configuration complexity. This inherent complexity makes it challenging, and in some cases even unfeasible, for the network to dynamically adapt and respond to faults and network alterations.

These difficulties occur mainly because the control and data planes are tightly coupled in traditional network architectures, which makes network management and operation ineffective. So, to address these difficulties, Software-Defined Networks (SDN) have emerged as a new network paradigm to improve network operation and management; One of the main contributions of the SDN paradigm is decoupling the control and data planes as two different entities, one logically centralized (control plane), and other distributed in all network devices (data plane).

In the network context, the control plane is the layer responsible for managing the network, creating and defining the network policies (e.g., using routing algorithms), while the data plane is responsible for forwarding the packets according to these policies defined by the control plane. Decoupling the control and data plane, we get some benefits like (i) The network management is eased because it is possible to define all network policies independently and in a centralized way, with a global vision of the network;(ii) The network applications (network functions) now run on the control plane and can be developed with architectural-independence. (iii) The cost of network devices (routers) is reduced, because they need forward the packets. Figure 2.1 compares the SDN and traditional networks, emphasizing their main differences. Note that in traditional networks, the control and data plane are executed in the network devices, and the network functions are physical middle-boxes. On the other hand, in the SDN network, the control plane runs as an external and centralized entity, executing all network functions and communicating with the data plane (routers) using protocols like OpenFlow(MCKEOWN *et al.*, 2008).

2.1.1.2 Data Plane Programmability

Another advantage of the decoupling between the control plane and the data plane introduced by the SDN architecture is the possibility of programming the data plane with high-level languages. It provides the ability to customize and manipulate the behavior of the network's data forwarding plane. This ability allows network operators to define and implement specific forwarding rules and actions, tailoring the network's behavior to meet their unique requirements.

With dataplane programmability, network devices such as switches and routers can be programmed to process and forward packets based on specific criteria, such as packet headers, protocols, or even application-specific information. This level of pro-

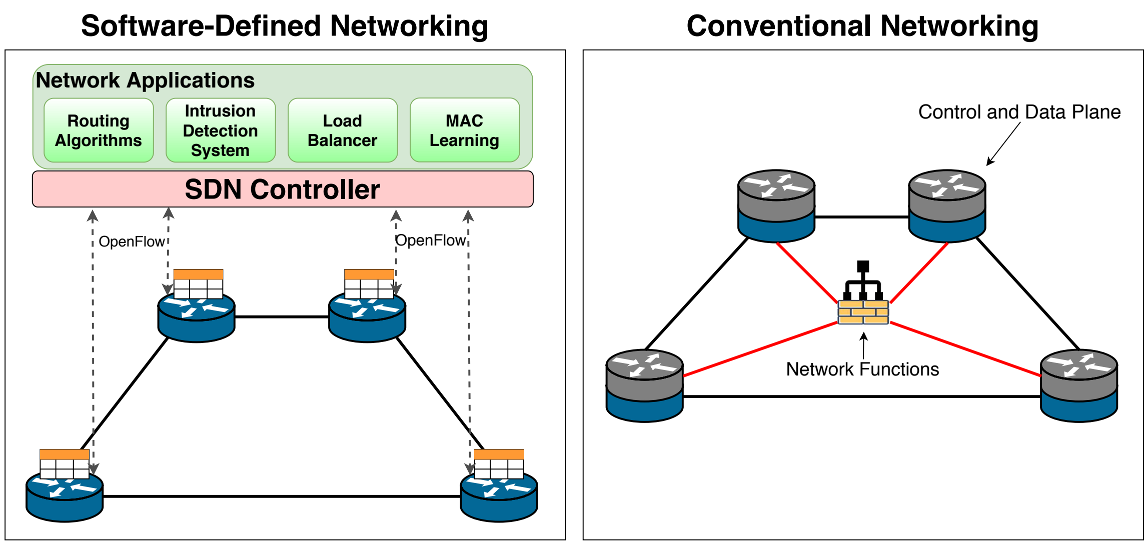

Figure 2.1 – Comparing traditional networks with SDN. Figure adapted from (KREUTZ *et al.*, 2015).

grammability empowers operators to adapt and optimize their networks dynamically, enabling the implementation of advanced features, network slicing, traffic engineering, and the efficient allocation of network resources. By decoupling the control plane from the data plane and providing programmable interfaces, dataplane programmability enhances network flexibility, scalability, and innovation, enabling the development of more intelligent and responsive networks.

Data plane programmability is marked by advancements in programming languages, hardware support, and frameworks tailored for network devices. These developments have significantly expanded the capabilities and flexibility of programmable data planes. The principal programming language in this domain is P4, which will be discussed in more detail in section 2.1.2.

2.1.2 Programming Protocol-Independent Packet Processors (P4)

The P4 language (BOSSHART *et al.*, 2014), is an open, domain-specific programming language designed for network switches and routers. It was developed to address the need for flexible and programmable data plane processing in modern networking devices. Traditionally, network devices have fixed-function data planes that handle packet processing tasks such as parsing, matching, and forwarding. However, with the increasing complexity and diversity of network protocols and applications, there is a demand for more programmable and customizable data planes.

P4 allows network engineers and researchers to define the behavior of the data plane in a high-level language. It provides a way to describe how packets should be processed, parsed, matched, and transformed as they traverse through the network device. P4 programs specify how packets are processed protocol-independently, enabling network devices to support new protocols and adapt to changing requirements without requiring hardware modifications.

One of the key principles of P4 is protocol independence. P4 programs are not tied to any specific network protocol, allowing network engineers to define their own protocols or customize existing ones. This flexibility enables innovation and experimentation in network design and protocol development. P4 programs are typically compiled into target-specific configurations, which can be deployed on programmable network devices that support the P4 language. These devices use programmable forwarding chips or SDN frameworks to execute the P4 program and process packets according to the defined data plane logic.

2.1.2.1 Traditional P4 Architecture

The traditional P4 language architecture refers to the original design and principles of the P4 language. These definitions consist of several key components:

- **Parser:** The parser is responsible for converting raw packet data into a structured format that can be processed by subsequent stages. It extracts header fields from the packet and populates metadata associated with each field.
- **Control Plane:** The control plane defines the high-level behavior of the packet processing pipeline. It typically includes tables, Match-Actions Units (MAU), and control flow constructs such as conditionals and loops. The control plane decides how packets are processed and forwarded based on their header fields and metadata.
- **Tables:** Tables define the forwarding behavior of the network device. They contain match-action entries that specify how to handle packets that match certain criteria. The control plane determines which entry applies to a given packet and invokes the associated actions.
- **Actions:** Actions define the operations to be performed on packets that match specific table entries. Examples of actions include modifying header fields, forwarding packets to specific ports, dropping packets, or sending packets to the Central Processing Unit (CPU) for further processing.
- **Deparser:** The deparser is responsible for converting the processed packet back into its raw format. It takes the modified headers and metadata and assembles them into the final packet that will be transmitted over the network.

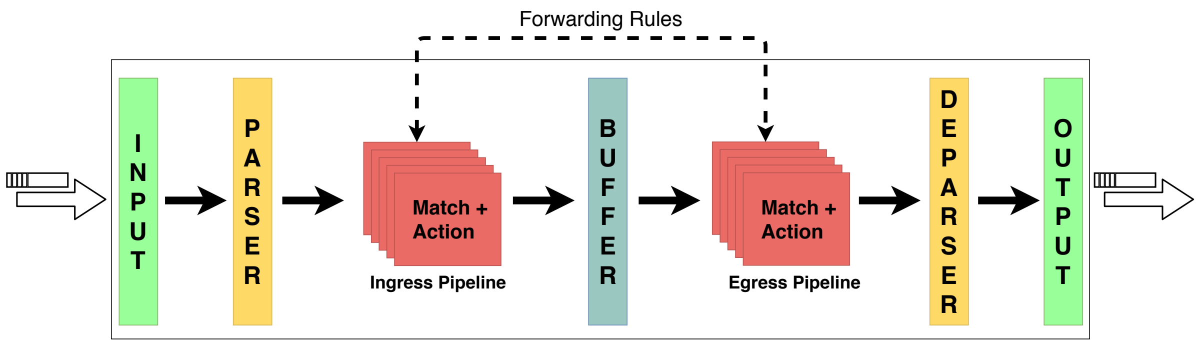

Figure 2.2 – P4 Pipeline

- **Metadata:** Metadata holds additional information associated with packets beyond the header fields. It can be used by the control plane to make forwarding decisions or pass information between different stages of the pipeline.

Figure 2.2 illustrates an abstraction of the P4 traditional forwarding model; When a packet is received on the switch, the first stage is the parser. In the parser, the packet headers and payload are extracted and stored in internal memory structures. The packet payload is buffered (unavailable for matching), and the header fields are used to match with the protocols supported by the switch and define the sequence of actions to be performed. Then, the extracted headers are forwarded to match/action tables. The match/action tables are divided into ingress and egress, although both can modify the headers fields, they do different tasks. Ingress match/action tables are responsible for determining the egress port/queue into which the packet is placed. Then, based on this decision, the packet is forwarded, replicated, or dropped. In the egress, match/action tables are performed only per-instance modifications as an example by adding or removing monitoring information (e.g., timestamp). Last, the packet is remounted on the departure, finishing processing.

2.1.2.2 Tofino Native Architecture (TNA)

Despite being based on the P4 language, the TNA refers to a programming model and hardware design specifically tailored for the Tofino family of programmable networking Application-specific Integrated Circuit (ASIC) developed by Barefoot Networks. This programming model, and consequently the Tofino Switch, were used in our implementations and evaluations, so their details and differences are discussed below.

The TNA provides a set of programming abstractions and tools that allow network operators to directly control and customize the behavior of Tofino ASICs. It enables the creation of highly efficient and flexible data plane pipelines that can be customized to meet specific network requirements. Additionally, TNA also includes a compiler and run-

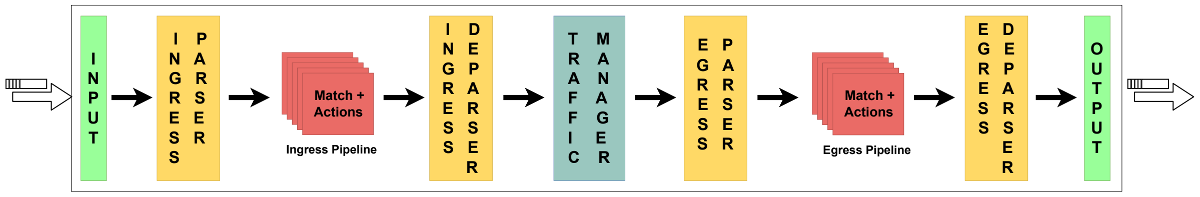

Figure 2.3 – TNA P4 Pipeline

time system that translates P4 programs into low-level instructions suitable for execution on Tofino ASICs. This compilation process optimizes the program for performance and efficiency, taking advantage of the capabilities of the underlying hardware.

Furthermore, TNA incorporates a highly parallel and programmable pipeline design, supporting the execution of 2 or even 4 P4 codes running in parallel pipelines. The pipeline consists of multiple stages that can be customized to perform specific packet processing tasks efficiently. This design allows for fine-grained control over packet processing and enables the execution of complex forwarding behaviors at high speeds.

The TNA P4 pipelines, although similar to the traditional ones, include additional components (see Figure 2.3). These components, presented in order in Figure 2.3, are responsible for:

- **Traffic Manager**: The TNA P4 pipeline includes a built-in traffic manager as a core component. The traffic manager handles various functions related to traffic scheduling, buffering, and congestion management. It helps optimize the packet flow and ensure efficient utilization of network resources within the Tofino switch.

- **Ingress Parser/Deparser**: In addition to the standard parser and deparser components found in traditional P4 pipelines, the TNA P4 pipeline includes a specialized ingress parser and deparser stages. The ingress parser processes incoming packets, extracting the packet headers and relevant information for subsequent processing. The ingress deparser is responsible for reconstructing packets with modified headers or other modifications before forwarding them to the appropriate egress processing stages.

- **Egress Parser/Deparser:** Similar to the ingress side, the TNA P4 pipeline includes a dedicated egress parser and deparser stages. The egress parser is responsible for processing packets before they are transmitted, handling any modifications or encapsulation required for the outbound traffic. The egress deparser handles the final packet reconstruction and prepares the packet for transmission.

These additional components in the TNA P4 pipeline, such as the traffic manager, ingress parser/deparser, and egress parser/deparser, are specifically designed to enhance the functionality and performance of the Tofino ASICs. They contribute to the efficient processing and management of packets within the Tofino-based switches, ensuring optimized traffic handling and transmission.

2.1.2.3 Differences Between Traditional P4 and TNA

In addition to the differences in its components, and despite both the traditional P4 pipelines and the TNA are based on the P4 language and share some similarities, but there are differences between them. Here are the key distinctions:

- **Hardware Target:** Traditional P4 Pipeline: The traditional P4 pipeline is designed to be hardware-agnostic, allowing P4 programs to be compiled and executed on various programmable forwarding devices, including Field Programmable Gate Array (FPGA)s, Network Processing Unit (NPU)s, and ASICs from different vendors. TNA P4 Pipeline: The TNA P4 pipeline is specifically tailored for the Barefoot Tofino family of programmable Ethernet switches, which are ASICs optimized for P4-based packet processing.
- **Pipeline Structure:** Traditional P4 Pipeline: The traditional P4 pipeline provides a flexible and customizable pipeline structure. It allows users to define the number and order of pipeline stages, including parsing, matching, and actions, providing fine-grained control over packet processing. TNA P4 Pipeline: The TNA P4 pipeline follows a fixed pipeline structure designed by Barefoot Networks. It consists of several predefined stages, including parser, deparser, match-action units (MAUs), and egress processing. While it offers less flexibility than the traditional P4 pipeline, it is optimized for efficient packet processing on Tofino ASICs.
- **Architecture Features:** Traditional P4 Pipeline: The traditional P4 pipeline allows users to define their own packet header formats, parsers, match-action tables, and actions using the P4 language designed for the v1 model architecture. It provides a high level of programmability and extensibility, enabling the creation of custom forwarding behaviors. TNA P4 Pipeline: The TNA P4 pipeline supports the P4 language but is specifically designed to leverage the unique capabilities of Tofino ASICs. It provides enhanced features and optimizations that take advantage of Tofino's hardware capabilities, such as large match-action tables, efficient table lookups, and high throughput packet processing. However, it also includes a series of programming limitations, such as a limited number of stages and restrictions on operations.

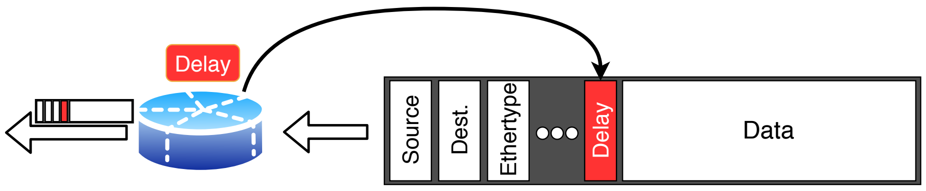

Figure 2.4 – Embedding telemetry information in the packet

- **Performance and Efficiency:** Traditional P4 Pipeline: The performance and efficiency of the traditional P4 pipeline depend on the target hardware platform and its capabilities. It allows users to optimize their P4 programs for specific hardware architectures and performance requirements. TNA P4 Pipeline: The TNA P4 pipeline is optimized for the Tofino ASICs and takes advantage of their advanced features, such as the ability to process packets at a line rate with high throughput and low latency. The TNA pipeline aims to deliver high-performance packet processing specifically for Tofino-based switches.

In summary, while both the traditional P4 pipeline and the TNA P4 pipeline are based on the P4 language, the TNA P4 pipeline is specifically designed for the Barefoot Tofino ASICs, providing optimizations and features that leverage the capabilities of these switches. The traditional P4 pipeline, on the other hand, is hardware-agnostic and offers more flexibility and programmability to target various programmable forwarding devices.

2.1.3 In-band Network Telemetry (INT)

As previously discussed, the network programmability and the P4 language enable several benefits related to the management and operation of network infrastructures. One of them is the INT monitoring, which has emerged as a promising network monitoring mechanism that provides higher network-wide visibility to network operators (LIU *et al.*, 2018).

INT offers several advantages over traditional monitoring methods. By embedding telemetry data in the packets themselves (see Figure 2.4), it provides fine-grained visibility into the network without requiring additional hardware or out-of-band probes. This approach allows for real-time monitoring, reduces network monitoring complexity, and enables more accurate and detailed analysis of network behavior.

Since its initial introduction at P4.org in 2015, INT has undergone significant evolution and discussion within the Internet Engineering Task Force (EITF) and industry communities. Furthermore, the term 'INT' has been utilized to encompass a broader concept of data plane telemetry, extending beyond the conventional classic INT, where both instructions and metadata are embedded in data packets. As a result, different modes of INT operation have been established, considering the extent of packet modifications and what specific content is to be embedded within the packets. Figure 2.5 summarizes the INT operation modes as described in (P4.ORG APPLICATIONS WORKING GROUP, 2020).In addition, these modes are described below:

- **INT eXport Data (INT-XD):** In INT-XD, metadata is exported directly from INT nodes in the dataplane to the monitoring system based on the configured INT instructions in their Flow Watchlists. This export of metadata occurs without the need for packet modification.
- **INT instruct(X)ions Data (INT-MX):** In INT-MX, the INT Source node embeds INT instructions in the packet header. Then, each INT Transit node and the INT Sink node follow these embedded instructions to directly send the metadata to the monitoring system. The INT Sink node removes the instruction header before forwarding the packet to the receiver. Packet modification is limited to the instruction header, ensuring that the packet size doesn't increase as it traverses more Transit nodes.
- **INT eMbed Data (INT-MD):** In INT-MD, both INT instructions and metadata are written into the packets. This mode represents the classic hop-by-hop INT. The process involves the INT Source embedding instructions, the INT Source and Transit nodes embedding metadata, and the INT Sink node stripping the instructions and aggregated metadata from the packet. The INT Sink node selectively sends this data to the monitoring system. This mode modifies the packet the most but reduces overhead at the monitoring system by collating reports from multiple INT nodes.

In general, INT has gained traction in modern networking environments, especially in data centers and cloud-based networks, where visibility and performance monitoring are critical for efficient operations and troubleshooting. It complements other monitoring techniques and contributes to the advancement of network monitoring and analytics capabilities. In this work, we use the INT-XD strategy to export INT information related to the bursts and QoE estimation (More details of this will be discussed in Chapter 3).

2.1.4 Inter-Packet Gap (IPG)

The IPG is a promising network metric that has proven recently to be efficient in solving several network problems such as microbursts (VOGT *et al.*, 2022), heavy hitters detection (SINGH *et al.*, 2022) and QoE inference (MUSTAFA; ROTHENBERG, 2022). Traditionally, the IPG refers to the arrival time difference between two consecutive

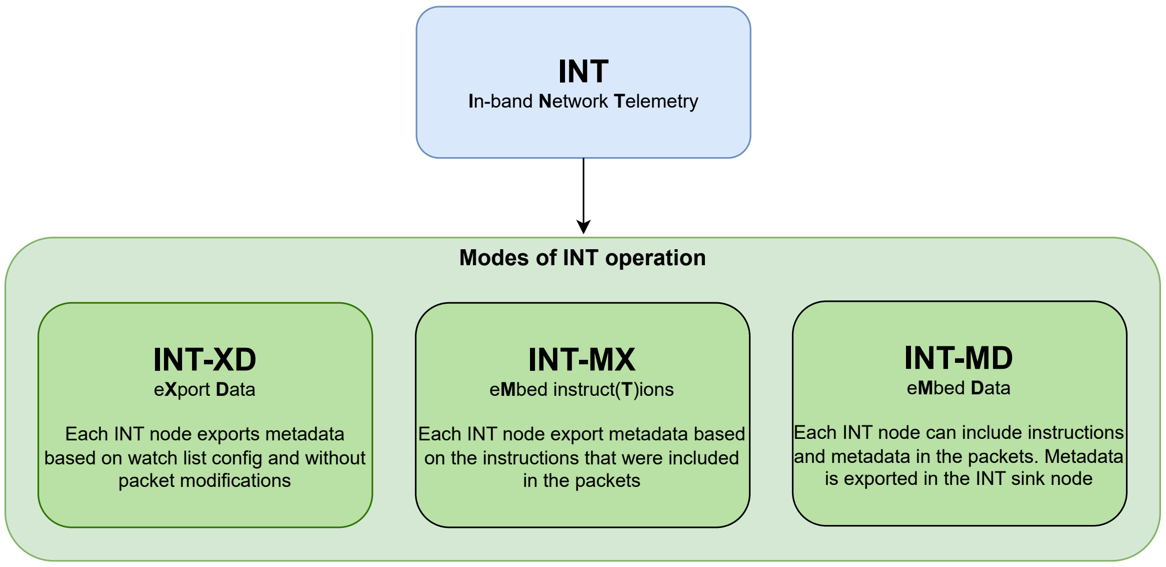
INT
In-band Network Telemetry
Modes of INT operation
INT-XD
eXport Data
Each INT node exports metadata based on watch list config and without packet modifications
INT-MX
eMbed instruct(T)ions
Each INT node export metadata based on the instructions that were included in the packets
INT-MD
eMbed Data
Each INT node can include instructions and metadata in the packets. Metadata is exported in the INT sink node

Figure 2.5 – INT monitoring operation types

network packets. and can be calculated according to Equation 2.1, where TS_l and TS_c are the arrival time of the last and penultimate packets, respectively.

$$IPG_c = TS_c - TS_l \tag{2.1}$$

Despite being a simple metric to be calculated, the IPG metric provides valuable information about the timing characteristics of packets in a network stream. It can be used to analyze network behavior, identify congestion, assess QoS parameters, and troubleshoot network issues. By monitoring and analyzing IPG values, we can gain a better understanding of network performance and make informed decisions to optimize and improve the network infrastructure.

So, in this work, we argue that the IPG monitoring, combined with other mathematical methods such as Exponentially Weighting Moving Average (EWMA), can be used for several proposes. More details about the use cases explored and the IPG implementation details can be found in Chapter 3.

2.1.5 Quality of Experience (QoE)

QoE is a measure that assesses users' overall satisfaction and subjective perception when interacting with a particular application, service, or system. It takes into account various factors that influence the user's perception of quality, including technical performance, usability, and user expectations.

QoE has significant implications for businesses and service providers. A positive user experience and high QoE can lead to increased user engagement, customer loyalty, and business success. On the other hand, poor QoE can result in user dissatisfaction, churn, and negative impacts on brand reputation and revenue.

In general, the QoE measurement is subdivided into subjective and objective approaches.

- **Objective Approach:** The objective approach focuses on measuring specific technical parameters and performance metrics that can impact the user's experience. These metrics are typically obtained through automated monitoring and measurements. Examples of objective metrics include network latency, packet loss rate, video bitrate, audio distortion, or application response time. By quantifying these metrics, service providers can assess the technical performance of their systems and identify potential issues that may degrade the user experience. However, objective metrics alone may not capture the full user experience and may not align with the user's subjective perception.
- **Subjective Approach:** The subjective approach involves collecting user feedback through surveys, interviews, rating scales, or other qualitative methods. Users are asked to provide their subjective assessment of the quality of their experience, considering factors such as audiovisual quality, responsiveness, usability, and overall satisfaction. This approach captures the user's emotional and cognitive responses and provides insights into their preferences, expectations, and overall perception of quality. Subjective assessment is often quantified using metrics such as Mean Opinion Score (MOS), which represents the average user rating of the quality on a numerical scale.

The MOS is a widely used subjective metric for assessing QoE. MOS represents the average rating provided by a group of users on a predefined scale. The scale typically ranges from 1 to 5, with 1 representing the lowest perceived quality and 5 representing the highest. Users are asked to rate the quality of their experience based on specific criteria, such as audio quality, video quality, or overall satisfaction. MOS provides a standardized way to quantify the user's subjective perception of quality and allows for easy comparison and benchmarking across different services or systems.

To better capture the user's experience, subjective and objective metrics are often best. It's important to note that subjective and objective approaches complement each other in assessing QoE. While subjective assessment captures the user's perception and satisfaction, objective metrics provide insights into the technical performance and potential issues. By combining both approaches, service providers can gain a comprehensive understanding of the user experience and take action to improve the quality and meet user expectations.

So, in this work, we have focused in develop an objective approach that estimates the QoE entirely in the data plane. To validate our technique, we compare our estimation results with a subjective approach, calculating the real MOS. More detail about this will be discussed in the Chapters 3 and 5

2.1.6 Virtual Reality (VR) Video Streaming

A real scenario where QoE methods can be applied is VR video streaming. VR video streaming refers to delivering 360-degree video content that users can interact with and explore as if they were in the environment. Unlike traditional 2D video, it provides a fully immersive experience that allows users to look around and explore the virtual world from different perspectives.

However, VR video streaming faces several technical challenges, principally from a network perspective, such as high bandwidth requirements, high image quality, and low latency. These challenges are summarized below:

- **Bandwidth and Quality:** VR video streaming requires higher bandwidth compared to traditional video streaming due to the larger data size and the need to transmit multiple perspectives to support the 360-degree view. To ensure a smooth and high-quality VR experience, a reliable and high-speed network connection is crucial to avoid buffering or latency issues that can disrupt the immersion.
- **Latency and Synchronization:** In VR streaming, minimizing latency is crucial to maintain the synchronization between head movements and the corresponding video playback. High latency can cause a noticeable lag between the user's actions and the visual response, leading to a less immersive experience. Efficient encoding, streaming protocols, and network optimization techniques are employed to minimize latency and ensure smooth synchronization.

These difficulties stem from the need to project/decompose the video into tiles for the user. Figure 2.6 summarizes an 8x5 tiling scheme. In summary, we break down the VR video into smaller tiles or fragments, encoding each tile separately, and then, the tiles are transmitted over the network. The goal of the tiling is to provide the viewer with the ability to view a high-resolution VR video without having to download the entire video beforehand. To that end, the tiling scheme also allows for the use of Adaptative Bit Rate (ABR) algorithms, which adjust the bitrate of each tile based on the network conditions to ensure smooth playback. For example, Zone $z1$ is defined as containing only the viewport's central tile, Zone $z2$ encompasses the viewport border tiles (8 tiles), and Zone $z3$ has the 31 remaining tiles.

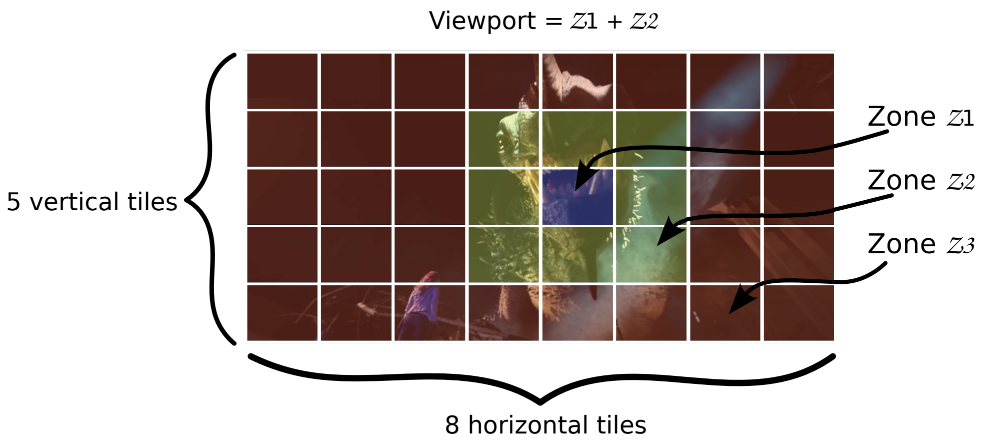

Figure 2.6 – Three zones 8×5 tiling scheme.

To overcome these challenges, developers are improving VR video compression algorithms, hardware, and software solutions. To reduce the high bandwidth demands, the user viewport is limited to the portion of the virtual environment that the user can see at a given time and is defined by the Field Of View (FOV) of the VR headset or device, which determines the size and shape of the virtual environment that is visible to the user. With that in mind, a key aspect in the future of VR is selecting a set of objective metrics (e.g., Quality of Service (QoS) metrics). However, little has been done (YU *et al.*, 2015; ZAKHARCHENKO *et al.*, 2016) in a network perspective (such as data plane solutions) towards which metrics can be better used to guarantee a better experience for the user in real-time.

2.2 Related Work

This section provides an overview of related works that share similarities with our proposal. Firstly, we discuss studies that employ the IPG metric to identify and address network issues. Next, we present studies that focus on detecting microbursts, which is one of the use cases addressed in our proposal. Subsequently, we delve into works that are relevant to our second use case, which involves measuring Quality of Experience (QoE). Additionally, we examine alternative monitoring strategies that aim to identify and mitigate various network problems, particularly those based on INT monitoring. Finally, we summarize all the presented works, highlighting their key contributions and points of differentiation from our proposal (refer to Table 2.1 for a comprehensive comparison).

2.2.1 Inter-Packet Gap Based Solutions

In the past, the IPG metric demonstrated its potential when used for some network applications like packet loss detection (SA-INGTHONG *et al.*, 2021), delay measurement (PIRATLA *et al.*, 2004), traffic classification (HAO; LAKSHMAN, 2011) and network performance measurement (SUN V. XU; DAFTARY, 2015).

Most recently, with the network programmability enabled by SDN, (SINGH *et al.*, 2022) have used the IPG as an alternative to packet counters and slide windows for heavy hitter detection. Similarly to the strategy used in this work, the authors calculate the IPG as an EWMA in the data plane and use a tau threshold to decide the heavy hitters. Their results suggest that IPG metrics are a strong candidate for HH detection and other traffic management applications.

Similarly, and related to QoE inference, (MUSTAFA; ROTHENBERG, 2022) reveals a high correlation between the QoS features derived from IPG and the QoE of encrypted DASH traffic. In this case, the IPG is derived in three sub-features: Cumulative Sums (CUSUM), EWMA, and Double Exponentially Weighted Moving Average (DEMA). To further investigate, various machine learning classifiers, including Artificial Neural Networks (ANN), K-Nearest Neighbors (KNN), Decision Trees (DT), and Random Forests (RF), were employed. These classifiers were used to predict the QoE class based on ITU-T Rec. P.1203 QoE standard, specifically mode 0. The prediction was achieved by considering metadata only, such as bitrate, frame rate, and resolution. The results demonstrate high accuracy in predicting the QoE class using these classifiers.

However, we argue that none of the cited works uses the full potential of the IPG. First, none of them explore problems like microburst detection data plane QoE estimation, and second, they do not explore the possibility of detecting multiple network events using the same IPG calculation and implementation.

2.2.2 Microburst Detection Strategies

Following, related to the microburst detection use case, recent research efforts have made improvements to detect microbursts on programmable dataplanes. In this context, BurstRadar (JOSHI *et al.*, 2018) is the state-of-the-art for full dataplane microburst detection strategies. The authors proposed an approach to detect the microbursts based on a queue depth threshold (specified as an Round Trip Time (RTT) increase tolerance) and report all the packets in the queue with a snapshot algorithm. As a result, they can detect the microbursts with ten times less data collection than the existing strategies. However, BurstRadar can not detect microbursts in multiple queues and report all packets on the queue, while our strategy reports just the contributing flows (we will discuss more in Chapter 3).

As a breakthrough to BurstRadar (JOSHI *et al.*, 2018) Gao et al. (GAO *et al.*, 2022) proposed BurstScope: a hybrid control and data plane microburst detection strategy. Burst scope emerged as an alternative to the limitations present in BurstRadar, like it can not detect the bursts on multiple queues and always report all flows in a burst case. So BurstScope can detect the bursts in multiple queues and seeks to report just the contributing flows through a strategy with an invertible sketch. Their results show that BurstScope can detect microbursts with high accuracy (95%) and can reduce the bandwidth overhead by 60x when compared with burst radar. However, BurstScope uses a hybrid strategy (combining control and data plane) to detect the contributing flows, implying a longer response time in detecting microbursts. It occurs due to the necessary communication between the control and data plane and the time it takes, which can be significant for this kind of problem.

Alternatively, (CHEN *et al.*, 2018b) proposed Snappy, a probabilistic strategy to detect the microbursts and report the contributing flows. Snappy estimates the microburst queue and identifies the contributing flows through a probabilistic algorithm (Recall). However, to determine the contributing flows in practice with high accuracy, it is expected that Snappy to require more than 128 pipeline stages (JOSHI *et al.*, 2018), which is infeasible in today's programmable switching ASICs. So, using Snappy to detect the microbursts in the control plane is infeasible, considering the resource utilization and restrictions.

2.2.3 QoE Measurement and Prediction

Related to QoE estimation or prediction, we have works like VR EXP (FILHO *et al.*, 2019), which is a platform that enables a set of adaptive tile-based schemes for various network conditions. PREDICTIVE (FILHO *et al.*, 2018) is a two-stage ML-assisted approach that infers how the user perceives the resulting VR video playout performance. In summary, a set of predictors have the network QoS (i.e., delay, packet loss, and TCP throughput) and the tiling scheme as input. In contrast, Vidhya et al. (VIDHYA *et al.*, 2020) introduced a fuzzy logic mechanism within the Network Data and Application Function (NWDAF) entity in 5G to perform QoE evaluation. Similarly, Schwarzmann et al. (SCHWARZMANN *et al.*, 2022) leverages NWDAF standardized interface capabilities in 5G networks to estimate the accuracy of different state-of-the-art regression techniques. Chen et al. (CHEN *et al.*, 2018a) propose an SSIM-based approach for 360-degree video quality assessment. The algorithm explores a correlation between 2D and spherical projection. Also, it is verified on a subjective 360-degree video quality assessment database. On the other hand, FastInter360 (STORCH *et al.*, 2021) exploits a set of texture features to reduce the encoding time of 360-degree videos with equirectangular projection (ERP) while Upenik et al. (UPENIK *et al.*, 2017) extends Yu et al. (YU *et al.*, 2015) and benchmarks PSNR-based approaches against ground-truth subjective quality data.

Iurian et al. (IURIAN *et al.*, 2022) analyzed the impact of priority queues in a video streaming scenario. However, the work is preliminary and does not provide QoE inference. Conversely, Bhat et al. (BHAT *et al.*, 2019) leverage Q-in-Q tunneling and translates the application- into link-layer header information at the edge to infer objective QoE metrics and the model QoE value score of VR streaming sessions.

Despite existing research efforts to achieve QoE closer to the edge, to the best of our knowledge, our proposal is the first approach toward designing an in-network VR video streaming QoE estimation directly in the data plane.

2.2.4 Monitoring Systems

Concerning monitoring systems similar to our proposal, looking to solve multiple problems or use INT-based network monitoring, (ZHAO *et al.*, 2021) proposed LightGuardian, a monitoring system based on in-band telemetry and a new sketch strategy to provide complete network visibility with low overhead. LightGuardian combines the traditional INT with per-flow sketches called "sketchlets" to include fewer telemetry data in the packets but maintain accuracy. Their results show that LightGuardian can acquire per-flow per-hop flow-level information within 1.0 - 1.5 seconds, using only 0.07% of the total network bandwidth. However, LightGuardian seeks to collect all network information, unlike our work which detects multiple events using a single network metric. Consequently, LightGuardian needs more control and data plane processing and data to maintain and reconstruct the sketches, increasing the overhead.

Similarly, (ZHOU *et al.*, 2020) proposed NetSeer, a monitoring system to detect network performance anomalies such as packet drops, congestion, path change, and packet pause principally for the cloud network context. NetSeer is based on the idea of detecting the flow events in the data plane and reporting it using compressed data for the management plane. Their implementation for Barefoot Tofino can reduce the cause location time with a low traffic overhead (0.01%).

Focusing on performing the root cause analysis, (TAN *et al.*, 2019) proposed NetBouncer, and an active probing system to detect link and device failures, especially gray failures. This kind of failure drops packets probabilistically, making them not so easily detectable by traditional monitoring applications or by a simple connectivity test. Then, the authors developed a packet bouncing technique to probe the designed paths and distinguish device failures. However, despite not presenting false positives in practice, NetBouncer has some limitations, such as including extra packets in the network (the probe packets) and suffering from transient failures and packet loss. Also, for gray failure detection, (MOLERO *et al.*, 2022) proposed a gray failure detector (FANcY) for the ISP's context. FANcY uses a solution based on in-switch packet counters that perform packet counts and compare the results to detect packet losses. Despite its success in detecting gray failures in ISP configurations, FANcY cannot detect other network problems (such as microbursts) and uses more hardware resources.

Recently, several works (JIA *et al.*, 2020) (BASAT *et al.*, 2020) (CANOFRE *et al.*, 2022) (HUANG *et al.*, 2020) have focused on improving network management and problem identification by optimizing the collection of telemetry data acquired using INT. So, all of these works seek to collect several network information but with lower overhead (in terms of reports in comparison with the traditional INT) through the use of techniques like aggregations, probabilistic methods, and moving averages. In contrast, our work seeks to collect only one network information and optimize its use, using it for several use cases. However, this does not prevent our IPG method from using any of these INT collection methods, which could further reduce its overhead.

2.2.5 Comparison

In this section, we summarize the related work presented in the previous sections and classify them into four general categories for comparison purposes:

- **IPG-based solutions:** In this category, we include all works that also use IPG-based solutions. With this, we can observe that despite using the IPG, these works do not explore its full potential.
- **Multiple use cases:** In this category, we include all works proposing solutions for multiple use cases or minimally collecting the necessary information for this, as in the cases of monitoring systems based on INT.
- **Data plane solutions:** In this classification, we include all works that implement your solution at least partially in the data plane. This is important because this type of implementation brings benefits such as reduced network control overhead, better management, and low response latency.
- **Depends on the control plane:** Here, we highlight the developed solutions that are dependent or fully implemented in the control plane. This, even though it is often essential, includes a time overhead (when compared to full-data plane solutions) in detecting and responding to network problems

Table 2.1 presents this classification for all previously described related works. As we can see in the table, although there are solutions for multiple use cases implemented in the data plane (especially INT-based monitoring solutions), all these solutions also depend on the control plane to work. In addition, they are usually based on the collection/analysis of multiple network metrics, while our solution manages to perform using only one, the IPG. In addition, we can highlight other solutions like (SINGH *et al.*, 2022), which is capable of using only a single metric (IPG) and implementing a solution in the data plane that works autonomously (without dependence on the control plane); however, it solves only one problem, heavy hitters detection. Altogether, our IPGNET proposal is the first low-cost solution based on a single network metric (IPG) capable of assisting with multiple network problems entirely in the data plane.

Proposal	IPG-based solution	Multiple use case	Data plane	Depends on Control plane
(SA-INGTHONG *et al.*, 2021)	✓	✗	✗	✓
(PIRATLA *et al.*, 2004)	✓	✗	✗	✓
(HAO; LAKSHMAN, 2011)	✓	✗	✗	✓
(SUN V. XU; DAFTARY, 2015)	✓	✗	✗	✓
(SINGH *et al.*, 2022)	✓	✗	✓	✗
(MUSTAFA; ROTHENBERG, 2022)	✓	✗	✗	✓
(JOSHI *et al.*, 2018)	✗	✗	✓	✗
(GAO *et al.*, 2022)	✗	✗	✓	✓
(CHEN *et al.*, 2018b)	✗	✗	✓	✗
(FILHO *et al.*, 2019)	✗	✗	✗	✓
(FILHO *et al.*, 2018)	✗	✗	✗	✓
(VIDHYA *et al.*, 2020)	✗	✗	✗	✓
(SCHWARZMANN *et al.*, 2022)	✗	✗	✗	✓
(CHEN *et al.*, 2018a)	✗	✗	✗	✓
(STORCH *et al.*, 2021)	✗	✗	✗	✓
(UPENIK *et al.*, 2017)	✗	✗	✗	✓
(YU *et al.*, 2015)	✗	✗	✗	✓
(IURIAN *et al.*, 2022)	✗	✗	✗	✓
(BHAT *et al.*, 2019)	✗	✗	✗	✓
(ZHAO *et al.*, 2021)	✗	✓	✓	✓
(ZHOU *et al.*, 2020)	✗	✓	✓	✓
(TAN *et al.*, 2019)	✗	✗	✓	✓
(MOLERO *et al.*, 2022)	✗	✗	✓	✓
(JIA *et al.*, 2020)	✗	✓	✓	✓
(BASAT *et al.*, 2020)	✗	✓	✓	✓
(CANOFRE *et al.*, 2022)	✗	✓	✓	✓
(HUANG *et al.*, 2020)	✗	✓	✓	✓
IPGNET	✓	✓	✓	✗

Table 2.1 – Review of related works

3 IPGNET Design & Implementation

In this chapter, we discuss the aspects of our proposed monitoring system, focusing on how it works to identify and react autonomously to network problems. We visit the strategies used for the proposed use cases, which include the microburst detection and contributing flows reporting, and the QoE estimation entirely in the data plane. In each use case, we explain the problem; the solution developed, and its challenges and limitations.

3.1 Microburst Detection and Report

In this section, we detail our microburst detection use case. Initially, we give a general description of the problem, and then we set out to formalize it. Then, we proceed to an overview of the proposed strategy, following the data plane design, implementation, challenges, and limitations.

3.1.1 Problem Overview

In computer networks, microbursts are brief and intense bursts of network traffic transmitted in a short period of time. They are characterized by a sudden surge in packet transmission at extremely high data rates, followed by a period of low or normal traffic. Usually, they are often caused by bursty traffic patterns, where packets arrive in rapid succession instead of being evenly distributed. This can happen due to various factors, such as sudden data transfers or network protocols with variable packet arrival rates.

Network devices typically use buffers to temporarily store incoming packets before forwarding them. Buffers help smooth out fluctuations in traffic and prevent traditional congestion. However, during microbursts, the burst of incoming packets can exceed the buffer capacity, leading to congestion and packet loss. The impact of these events on network performance can be significant; For example, when packets are dropped due to congestion, retransmissions are required, which reduces overall throughput. Additionally, the dropped packets can also cause delays and increased latency, affecting the quality of service experienced by network users.

Despite their impact detecting microbursts can be challenging, principally using traditional monitoring strategies, because they are transient events that occur at a small timescale. Below we discuss the main challenges encountered in detecting microbursts:

- **Information granularity:** Microbursts occur over very short durations, often in the range of microseconds to milliseconds. Traditional monitoring tools and strategies (e.g., SNMP) may not capture or analyze traffic patterns with such fine granularity. They are designed to operate at longer timescales, such as seconds or minutes, making it difficult to identify and measure these brief bursts accurately.
- **Limited information:** In addition to the low granularity of the information collected, traditional monitoring tools may not have visibility into the buffer occupancy levels or queue lengths at fine timescales. As a result, they may not directly observe the congestion caused by microbursts, making their detection challenging.
- **Packet/flow level information:** Traditional monitoring techniques often focus on aggregated traffic statistics, such as average bandwidth or utilization over longer time intervals. They may not provide detailed packet-level information required to identify microbursts. Since microbursts involve rapid packet arrivals within a short period, analyzing individual packets becomes crucial for detection.

To overcome these challenges, specialized monitoring tools and techniques should be developed (see Section 2.2.2) specifically for detecting microbursts and their contributing flows (discussed in detail in Section 3.1.1.1). These tools often employ high-speed packet capture, fine-grained analysis, and real-time monitoring to capture and analyze traffic at the required timescales.

3.1.1.1 Contributing Flows

In a microburst scenario, the concept of contributing flows, also covered in (GAO *et al.*, 2022), refers to the individual flows or streams of network traffic that collectively contribute to the occurrence and intensity of a microburst event. A microburst can be composed of multiple contributing flows, where each flow represents a specific communication session or data stream between network devices. These flows may be generated by different applications, services, or users, and they collectively contribute to the overall burst of traffic.

Contributing flows are crucial in microbursts because their combined transmission patterns can lead to congestion and overload network buffers. When multiple flows coincide and transmit a significant amount of data simultaneously, the resulting packet burst can exceed the network devices' capacity, causing congestion and potential packet loss.

Identifying the contributing flows within a microburst is important for understanding the root causes and characteristics of the event. By analyzing the properties of individual flows, such as their packet arrival rates, data volume, or source-destination pairs, network administrators can gain insights into the specific flows that contribute to the burst traffic. Also, distinguishing the contributing flows within a microburst can aid traffic engineering and capacity planning. By understanding which flows are most responsible for the bursty behavior, network administrators can allocate resources, adjust the quality of service settings, or apply traffic shaping techniques to prioritize or regulate specific flows and minimize the occurrence of microbursts. So, understanding and managing these contributing flows are essential for effectively addressing microburst-related challenges in network management.

3.1.2 Problem Definition

Understanding the importance of detecting microbursts, and identifying the contributing flows, we can then formalize the problem as follows; Considering a physical network infrastructure $G = (D, L)$ and a set of active network flows F. The set D in the network G represents all programmable forwarding devices, then $D = \{1, ..., |D|\}$, while set L represents the links interconnecting pair of devices $(d_1, d_2) \in (D \times D)$. Each flow $f \in F$ has two endpoints (i.e., ingress and egress forwarding devices) and is routed through the network infrastructure G using a simple path. Furthermore, each flow f transmits packets at a rate $Tx(f)$.

A device d is considered in burst state $b \in B$ (set of all microbursts) when its queue occupancy $Qocup(d)$ is greater than a threshold th. Then the duration time of a burst b_x can be defined as $T(b_x)$, which corresponds to the interval $[T_s, T_e]$ representing the time that the burst started and ended ($Qocup(d)$ is again smaller than th), respectively. Additionally, between each time interval $[T_s, T_e]$, there is a subset $CB \subseteq F$ that corresponds to the contributing flows. These flows are defined when the rate $Tx(f)$ of a flow f is greater than a threshold th_f.

The problem is to detect all existing microbursts and their contributing flows. Therefore, we first need to detect all bursts $b \in B$ that occur in the devices $d \in D$, setting an appropriate th threshold. In the sequence, we need to find the set CB corresponding to each burst $b \in B$ corresponding to the list of contributing flows of each burst. Note that in these cases, the biggest challenges are how to define the th and th_f variables, which will define when a burst will be detected, and which flows will be chosen as contributing flows. These decisions will directly impact the method's effectiveness and, consequently, the network management.

3.1.3 Proposed Approach

To perform the microburst detection and report the contributing flows, unlike (GAO *et al.*, 2022) we use a full-data plane approach. The main idea of our strategy is to detect the microbursts based on the queue depth and then report the contributing flows for the control plane based on their IPGs. To do this, we use the basic IPG calculation described in section 2.1.4 combined with an EWMA function to consider recently measured IPGs in the calculation.

So, we perform the per-flow IPG_w calculation according to Equation 3.1, where IPG_c is calculated using Equation 2.1, and α is a parameter between $[0, 1]$;

$$IPG_w = \alpha \cdot IPG_{w-1} + (1 - \alpha) \cdot IPG_c \tag{3.1}$$

We argue that differently from what is proposed in (GAO *et al.*, 2022), it is possible to detect the contributing flows in a burst scenario entirely in the data plane, just by calculating the proposed IPG. Furthermore, with the proposed method, it is not necessary to use as many hardware resources as in the probabilistic technique proposed by (CHEN *et al.*, 2018b), which needs a large number of pipeline stages to get a decent recall.

This is possible because the use of the IPG has already proven efficient in detecting heavy flows in the data plane (SINGH *et al.*, 2022), making a precise detection with low overhead. Then, as they are similar events (flow bursts and heavy hitters), with some modifications in the detection strategy, it is possible to use the same IPG calculation for both problems. Below, we discuss in more detail how our approach works.

3.1.3.1 Workflow

Figure 3.1 presents the workflow of our microburst detection strategy, demonstrating the process performed from the moment the device receives the packet until it is forwarded. Note that although IPG monitoring is performed per flow, this process is performed packet by packet, which provides a very fine monitoring granularity, and, consequently, a very fast burst detection. Below, we discuss each of the steps presented in the figure.

- **Packet arrives:** The packet arrives at the network device, then the packet is preprocessed, identifying, for example, which flow it belongs to.
- **IPGw calculation:** The flow IPGw is calculated using Equations 2.1 and 3.1, and the result is stored in a register for later use.

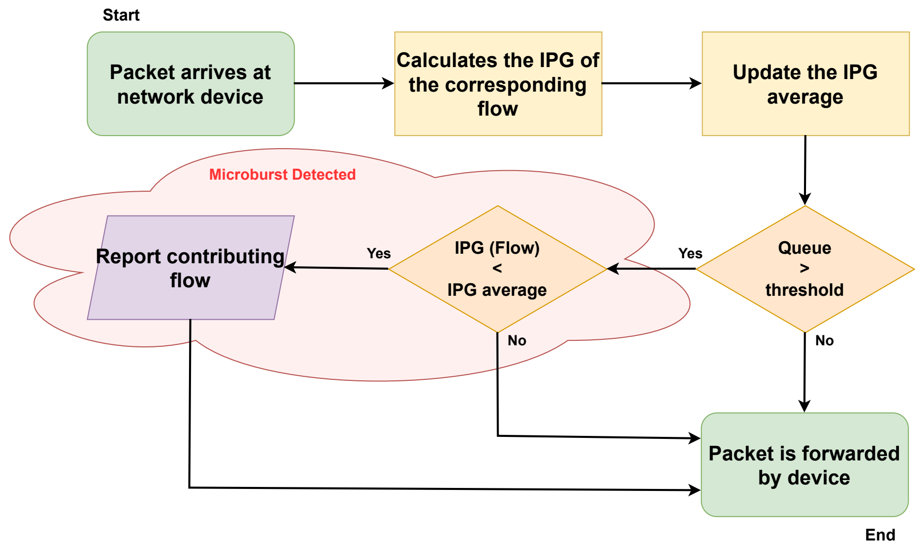

Figure 3.1 – Microburst detection workflow

- **IPGw average:** With the IPGw calculation of the flow already performed, now the average IPGw is calculated, which corresponds to the average of all active flows. The result is also stored for later use.
- **Burst detection:** Performs the detection of microbursts based on the queue occupancy (metadata provided by Tofino), which is compared with a previously defined threshold.
- **Contributing flow detection:** If a burst is detected, it uses the pre-calculated IPGw and IPGw average values to decide whether the flow is a contributor or not. If the IPGw is less than the average IPGw, the flow is considered a contributor.
- **Contributing flow reporting:** If the flow is marked as a contributor, report the flow to the control plane using the INT-XD concept, discussed in Section 2.1.3.
- **Packet forwarding:** Performs post-processing of the packet, and normally forwards it to the network.

3.1.3.2 Design on Programmable Data Planes

Our algorithm is implemented using P4 language for the TNA (discussed in Section 2.1.2.2). As we discussed in our background, the P4 language naturally presents implementation challenges, especially when used with the TNA architecture. In the following section, we will describe the main challenges encountered, followed by details of the implemented solution and its limitations.

3.1.3.2.1 Challenges

The utilization of P4 introduces an exciting opportunity to execute algorithms directly on programmable switch ASICs. However, this implementation poses certain challenges. In this discussion, we delve into the engineering obstacles associated with integrating algorithms on switch ASICs and highlight the adopted solutions. These solutions were employed to successfully implement a P4 pipeline for the proposed IPG-based method, which underwent validation on a programmable switch ASIC.

Below we summarize the main challenges encountered and the strategies used to deal with them.

- **Access Limitations to Registers:** In per-flow monitoring algorithms, the switch typically performs an initial check of the flow ID for incoming packets against the entries in the register. Subsequently, the corresponding parameters, such as IPGs or packet counters, are updated accordingly. For instance, in a packet counting-based solution, the counter is decremented, and the flow is evicted from the data structure once the counter reaches zero. However, since the flow ID register has already been accessed to confirm the match, it cannot be accessed again to replace the entry. In the Tofino switch ASIC, the register can only be accessed once per packet lifetime. To overcome this challenge, the solution implemented is rewriting the IPG registers every time there is a flow ID match. This approach eliminates the need for recirculations (e.g., for a second update) and ensures efficient operation.

- **Arithmetic and Comparison Operations.** Conventional switch hardware lacks support for multiplication and division operations in register actions. Additionally, comparison operations are limited to a fixed number of bits and can only be performed between constant and variable values, preventing the comparison of two variables. However, the Tofino Switch offers bit operations that can be utilized to perform basic multiplications and divisions. Implementing an exponential weighted moving average (EWMA) calculation in an HW pipeline, as per Equation 3.1, can be challenging. To address this challenge, our solution is based on an approximate EWMA calculation that utilizes the available arithmetic and comparison bit operations within the limited hardware resources.

- **Limited Number of Stages:** In order to achieve low and predictable per-packet latency, the Tofino hardware architecture relies on a fixed number of pipeline stages. To accommodate our program within the available stages, it is crucial to minimize unnecessary table dependencies in implementing our proposed algorithm. Currently, our algorithm requires ten stages to complete all the necessary steps. Running our algorithm on top of a baseline switch.p4 (P4.ORG, 2015), compiled on Tofino, demonstrates that it can operate alongside other switching functions without imposing significant additional resource and stage constraints. A more detailed analysis of the resource utilization of our strategy will be presented in Section 5.1.2.

3.1.3.2.2 Implementation

So, in the P4 code, to calculate and store the per-flow IPGw, we use registers and actions. For the IPG_f^w, we use 16-bit registers, and for TS_f^l 32-bit registers, totaling 48-bit of memory for each flow entry. We assume a capacity for 2048 flow entries, resulting in a total of 98k bits of storage used. In addition, we use the value of $\alpha = 0.99$, as it proved to be the most suitable value for the detection of heavy flows in (SINGH *et al.*, 2022).

```
Control IngressPipeline():
    RegisterAction ComputeIPG($IPG_c$):
        $IPG_w = \alpha * IPG_{w-1} + (1 - \alpha) * IPG_c$
        return IPGw
    Register<bit<16>> $IPG_{avg}$
    Register<bit<16>> $IPG_w$
    Register<bit<32>> $TS_l$
    Apply:
        flowIndex = computeFlowIndex()
        $TS_l$ = readAndUpdateTimestamp(flowIndex)
        $IPG_c = TS_c - TS_l$
        $IPG_w$ = ComputeIPG($IPG_c$, flowIndex)
        $IPG_{avg}$ = ComputeIPG($IPG_w$)
        InsertIPGHeader()

Control EgressPipeline():
    if $deqDepth > threshold$ then
        slotsRemain[queueID] = deqDepth
        if $IPG_w < IPG_{avg}$ then
            reportContributingFlow(pkt)
        end
    end
    else if $slotsRemain[queueID] > 0$ then
        slotsRemain[queueID] = slotsRemain[queueID] - pktLen/slotSize
        if $IPG_w < IPG_{avg}$ then
            reportContributingFlow(pkt)
        end
    end
```

Algorithm 1: Microburst Detection Algorithm

The Algorithm 1 illustrates how our approach implemented in P4 works. For simplicity, we present only the ingress pipeline and egress pipeline blocks, as our parser/deparser blocks are only used to extract/include the standard headers (ethernet, IP, and transport protocol). So, the first part performed when the packet enters the ingress pipeline is to compute its flow index using a crc32 hash function. This hash function uses as input the source and destination IP, transport protocol, and source and destination port. The result is used to identify the flow and indicate the correct position of the registers in which its IPG is stored and must be updated.

In the sequence, we start to calculate the flow IPGw, and IPGw overall average. For this, we use the Equations 2.1 and 3.1 with the help of a register action and a math unit. After calculating and storing the results, in addition to updating the timestamp of the last package to the current timestamp, we include these results in the packet to be used in the egress pipeline. Note that this inclusion is necessary because the queue occupancy metrics are only available in the egress pipeline. Therefore the detection of bursts and the decision of the contributing flows has to be done in the egress pipeline.

So, after finishing the ingress pipeline and after the packet passes through the traffic manager, the packet enters the egress pipeline. Now, the algorithm checks if the occupation of the queue that the packet passed through the traffic manager is greater than the threshold predefined by the network operator. The network operator can define a threshold for an increase in latency (expressed as a percentage). As an illustration, if the network's round-trip time (RTT) with no queuing is 50 μs, the operator can set a 30% threshold, corresponding to a minimum latency increase of 15 μs. Consequently, the algorithm would disregard any microbursts resulting in a delay of less than 15 μs delay. In practice, this threshold means a queue of 18750 bytes in a queue with a throughput of 10Gbps and must be divided by the size of the queue slots to find the number of slots corresponding to the desired threshold (e.g., 293 for 64B slots).

So, when the queue depth is greater than the threshold, we have detected the burst and must decide whether the packet belongs to a contributing flow or not. Then, we check if the IPG of the flow to which the packet belongs is less than the overall IPG average, and if so, we report the flow as a contributor to the control plane, using the INT concept as discussed in Sec. 2.1.3.

However, note that there is another case, where the queue depth is smaller than the threshold, but we still check whether the flow is contributing. This situation arises when the current occupancy of the queue does not surpass the threshold, yet there are still packets being released from the queue that were present during a previous period when the occupancy exceeded the threshold. In other words, these packets were part of an earlier burst. In this case, as before, we check if the packet belongs to a contributing flow, and report to the control plane if necessary. Note that to maintain this control, we have a variable called $slotsRemain$, which stores the total occupation of the queue each time a burst is detected.

Another important point to note is how the IPGw average works as a threshold to separate the normal and contributing flows. It occurs because, in general, the link utilization is about 10% for 90% of the time (ZHANG *et al.*, 2017), which means that when a burst starts, the tendency is that we have several heavy flows that make the average become a threshold between flows with normal rates and heavy flows.

To summarize, our microburst detection algorithm operates entirely within the data plane and is designed to identify microbursts and the corresponding contributing flows. It sends relevant reports about these heavy flows to the monitoring server in the control plane using INT. This approach reduces the information overhead in the control plane, as the monitoring server only receives pertinent details, such as information about flows that caused burst events. This enables the network operator to promptly apply network policies when necessary, resulting in more efficient network management.

3.1.3.3 Limitations

We will now summarize the primary limitations encountered in the developed solution, considering aspects related to implementation, hardware, and language.

- **Hash Collisions:** We use a hash function to identify and index the flows in the registers (which use source and destination IPs/ports and the protocol). However, when we have collisions, what happens in practice is that we have two or more flows updating the same register slot, decreasing the accuracy of the detection of contributing flows (i.e., reporting flows that should not be reported).
- **Fixed Threshold:** Similar to existing solutions, our solution requires the prior definition of a threshold for queue occupation and, consequently, detection of bursts. This makes the solution less dynamic since this threshold cannot be changed at runtime.
- **False Positives:** While utilizing the IPG average as a dynamic threshold effectively helps in identifying contributing flows, it can still lead to some false positives. This is particularly noticeable when dealing with a limited number of heavy flows. However, it is important to note that our strategy still outperforms state-of-the-art approaches by reporting fewer flows overall.

3.2 Inferring QoE In The Data Plane

This section introduces and discusses our second use case, called `QoEyes`: estimating QoE entirely on the data plane. This is still a developmental use case, as it is the first effort in the literature to perform QoE estimation entirely on the data plane.

3.2.1 Problem Description

Typically, the QoE is estimated by utilizing methods of measurement or prediction in either the user plane or the control plane side. These methods are known for their accuracy in estimating the user's QoE and generally involve measuring various QoS metrics and utilizing machine learning or deep learning algorithms. Despite providing accurate results, these techniques may not always be as quick in their measurement due to the extent and complexity of the network metrics that need to be measured and the efficiency of the machine learning or deep learning algorithms employed. As a result, it becomes difficult to promptly implement network policies to address potential problems or quality degradation.

To address this limitation, in this work, we propose a first step to the QoE estimation entirely in the data plane. The data plane QoE assessment enables us to make decisions at a nanosecond level, enhancing our reaction time to problems and improving the user QoE. By processing and evaluating QoE metrics closer to the source of network traffic, latency can be minimized, resulting in faster response times and improved user satisfaction. For instance, if we detect a loss in the QoE of a video session, we can use strategies like (IURIAN *et al.*, 2022) to prioritize that flow and thus recover the desired QoE.

Furthermore, performing QoE estimation in the data plane reduces the need for exposing sensitive data to the control plane. By keeping QoE analysis within the data plane, user privacy can be better protected as sensitive information does not need to be transmitted to external systems for analysis.

Finally, from a network perspective, performing QoE estimation in the data plane reduces the burden on the control plane by offloading computational tasks. This leads to more efficient utilization of network resources and enables the control plane to focus on higher-level network management functions.

3.2.1.1 Challenges

Despite all the benefits, there are several challenges to performing QoE inference on the data plane. From an implementation perspective, and similarly to the challenges encountered in detecting microbursts, we have challenges like (i) hardware resource and operation constraints; and (ii) limited information about data flow. In the P4 programmable hardware targets, such as the Tofino hardware, we have limited storage capacity, memory access restrictions, and limitations with arithmetic and comparison operations. As previously discussed, the register can only be accessed once during the lifespan of a packet, comparisons are restricted to a set number of bits and arithmetic operations such as division and multiplication can only be performed using bit shifting.

Additionally, implementing in the data plane, we have other conceptual challenges, like limited knowledge about the data flow. For example, we do not possess information about the application level, including aspects such as video resolution, buffer size, and segment size. So, below we describe some of these main challenges:

- **Lack of End-to-End Visibility:** The data plane operates at a lower layer of the network stack, which means it may not have complete visibility into higher-level network layers and end-to-end factors that can affect QoE, such as congestion in the network or application-specific metrics.
- **Limited Contextual Information:** The data plane typically focuses on packet forwarding and processing, lacking access to critical contextual information about user behavior, application requirements, or device characteristics. This can hinder accurate QoE estimation, as these factors play a significant role in determining user experience. Furthermore, the QoE estimation for VR applications involves intricate algorithms and analysis of media-specific parameters (e.g., video quality, audio clarity) that are not available in the data plane.
- **Traffic Classification:** In order to estimate the QoE of an application in the data plane, it is crucial to identify that specific application accurately. This necessitates using effective traffic classification algorithms that can seamlessly integrate with the proposed solution.
- **Scalability and Performance:** Implementing comprehensive QoE estimation algorithms entirely in the data plane using P4 code can introduce scalability and performance challenges. This is primarily due to the inherent complexity of these algorithms, which often rely on sophisticated techniques such as machine learning.

3.2.2 Proposed Approach

So, we introduce `QoEyes`, a QoE estimation technique that uses IPG calculation to carry out the QoE estimation entirely in the data plane. Our strategy is driven by recent academic studies that have already utilized IPG as the primary metric for inferring QoE, demonstrating a direct correlation between IPG and QoE (MUSTAFA; ROTHENBERG, 2022). We argue that we can get a QoE estimation directly in the data plane by calculating the IPG as an EWMA.

Figure 3.2 presents the desired workflow for estimating QoE entirely in the data plane. In that workflow there are some stages that are described below:

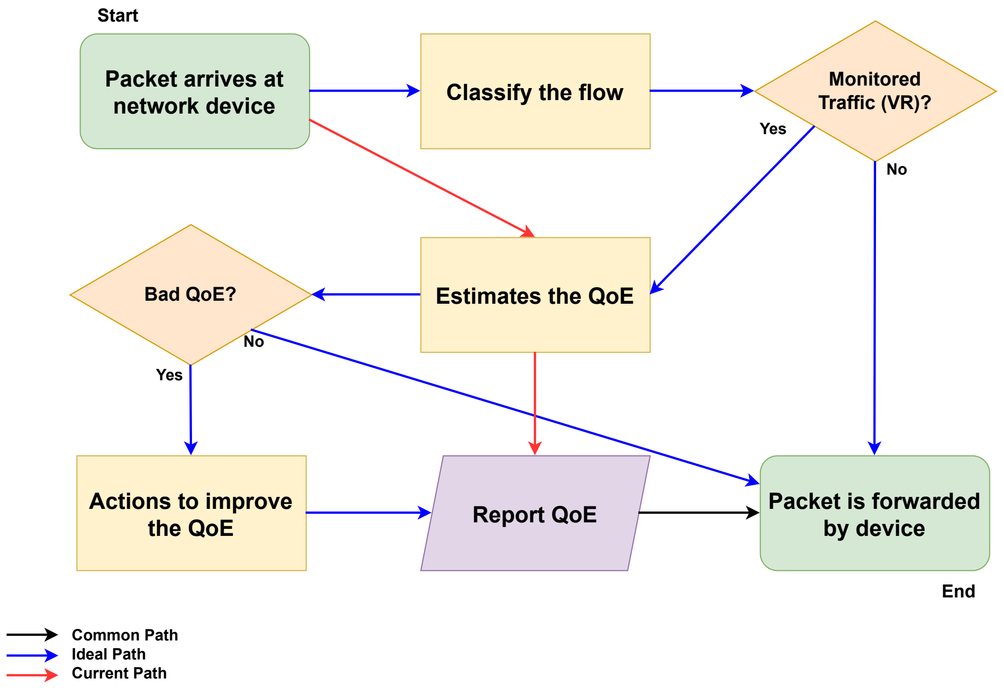

Figure 3.2 – QoE estimation workflow

- **Packet arrives:** The packet arrives at the network device, then the packet is pre-processed, identifying, for example, which flow it belongs to.
- **Classify the flow:** At this stage, the traffic is classified, identifying its type of application (e.g., VR video streaming).
- **Monitored Traffic?:** In this state we check if the traffic that was classified is a VR traffic. If the test results in true, the workflow proceeds to QoE estimation, and if false, it simply forwards the traffic normally.
- **Estimates the QoE:** In this state, the QoE estimate is calculated, which in our case is done by calculating the IPGw.
- **Bad QoE?:** After calculating the current QoE estimate, we check whether this estimate is good or bad. If the QoE has a good estimate, the traffic is forwarded normally. In case there is a bad QoE, we move to the state of taking action to improve the QoE.
- **Actions to improve the QoE:** In this state, actions are taken to try to improve the QoE that was detected as bad. These actions may include routing changes, and queue management actions, as proposed in (IURIAN *et al.*, 2022).
- **ReportQoE:** It then reports the estimated QoE to the control plane so that it also has updated information on the quality of the QoE.

- **Packet forwarding:** Performs post-processing of the packet and normally forwards it to the network.

Note that in Figure 3.2, there are paths with blue, black, and red arrows. The blue arrows represent the ideal workflow, where the packet goes through all the described stages, being classified, calculating the QoE estimate, and taking actions when necessary. However, the red arrow represents the current path of the proposed strategy, where we still don't have a traffic classifier algorithm, and also, we don't take action when we detect a bad QoE, we just focus on estimating the QoE and reporting it to the control plane. The black arrow only represents the shared path.

With the current `QoEyes` limits well defined, Figure 3.3 shows an example of `QoEyes` application. In the figure, we can see some VR video clients connecting to a streaming video server. In this process, the QoE of the video sessions is being estimated directly in the data plane (switches) and reported to the control plane when necessary. In this way, devices can act more quickly in the event of a problem (i.e., QoE degradation), improving the end-to-end user experience.

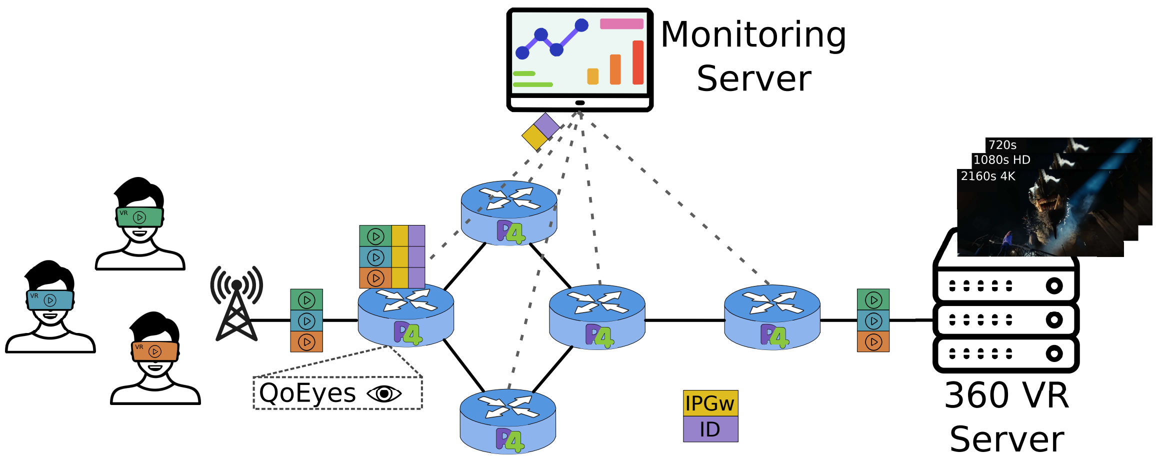

Figure 3.3 – Illustrating an use case of `QoEyes` in a P4-based network.

3.2.2.1 Implementation

So, in this section, we discuss about the `QoEyes` implementation details. The Algorithm 2 illustrates how we implemented the `QoEyes` strategy to perform the IPGw measurement in the Tofino architecture. Similarly to what is done in detecting microbursts, we perform all the calculations in the ingress pipeline, using registers to store the IPGw and the last noted timestamp (TS_l). In addition, we use register actions to perform the approximate EWMA calculation, which is done through bit operations exactly as done before. If you want more details about the IPG calculation, revisit Section 3.1.3.2.2.

Next, unlike the microburst strategy, we check whether the flow in question is a monitored flow. The monitored flows are decided in the control plane, assuming we know which VR flows we want to monitor. So when we find a monitored flow, we clone the packet to send it to the control plane with the desired information (IPG header).

This IPG header has the measured IPGw and the corresponding flow ID and is reported to the control plane. In practical applications, these reports can be conducted at regular intervals or only when necessary to monitor the IPGw. Additionally, the reporting can be triggered only when a drift in the IPGw is detected, which signifies a decline in the QoE.

```
Control IngressPipeline():
    RegisterAction ComputeIPG($IPG_c$):
        $IPG_w$ = $\alpha$ * $IPG_w$ + (1 - a) * $IPG_c$
        return IPGw
    Register<bit<16>> $IPG_w$
    Register<bit<32>> $TS_l$
    Apply:
        flowIndex = computeFlowIndex()
        $IPG_c$ = $TS_c$ - $TS_l$
        $IPG_w$ = ComputeIPG($IPG_c$)
        if monitoredFlow.hit() then
            PacketClone()
        end

Control EgressPipeline():
    if Cloned Packet then
        InsertIPGHeader()
    end
```

Algorithm 2: QoEyes data plane algorithm

3.2.2.2 Limitations

In addition to the hardware and IPG calculation limitations already discussed in section 3.2.1.1, here we discuss other limitations that our QoE estimation strategy has:

- **Traffic Classification:** As highlighted in the challenges, one of the prerequisites for performing QoE inference in the data plane is traffic classification, particularly in scenarios where the objective is to identify specific applications or services, such as video sessions in a VR video streaming. However, in our implementation, we do not implement any traffic classification strategy in our implementation, and we assume that we know in advance which flows we want to analyze.
- **Simplicity of the Method:** While our approach demonstrates a strong correlation with actual QoE, as we will discuss in detail in Caption 5, it is important to note that we rely on a single network metric to establish this QoE estimate. We are aware

that QoE is a complex metric to measure, and as such, our approach may not always provide an accurate estimate in every case.

- **Objective Approach:** As we only use the IPG (network metric) to estimate the QoE, we are making an estimate only with objective methods that can have limitations because QoE is a subjective metric that depends on individual user perception and preferences. Objective metrics can not capture the entire user experience, as they do not consider the subjective aspects of perception and user satisfaction.

4 P4 Programmable Patch Panel (P7)

We faced a challenge in conducting our evaluations due to the limited availability of Tofino switches (just one is available in our infra). Conducting experiments on larger-scale network topologies while maintaining high-fidelity results was difficult. So to validate our use cases in different networking scenarios, performing line-rate and high-fidelity experiments for evaluation, we participated in developing the open-source P4-based network emulator called P7. More specifically, we worked on developing and improving features such as latency, jitter, packet loss, forwarding, and the user's P4 code support. This chapter discusses P7 in detail, exploring its goals, architecture, and implementation details, focusing mainly on the features contributed within the scope of this dissertation.

4.1 Motivation and Objectives

Existing network experiment solutions, such as virtual and emulation-based environments, encounter issues with performance fidelity, scalability constraints, and trade-offs. Therefore, there is a growing demand for a realistic experimental platform that offers high-fidelity performance, scalability, and flexibility.

Existing software-based solutions like Mininet (LANTZ *et al.*, 2010), NSX (KOPONEN *et al.*, 2014) and CrystalNet (LIU *et al.*, 2017) may not be sufficient for testing and evaluating large-scale production networks, as they can suffer from performance and scalability limitations. On the other hand, hardware solutions like BNV (KANNAN *et al.*, 2018), SimBricks (LI *et al.*, 2020), and specially TurboNet (CAO *et al.*, 2020) are hardware solutions for network emulation, however are limited by factors like, not provide support for emulate all link characteristics and the support of a custom P4 code.

To meet this demand, we plan to utilize the power of P4-based hardware, specifically Tofino, to create a realistic experimental platform that provides high-fidelity performance, scalability, flexibility, and support for data plane programmability through a hardware-based environment. P7 leverages the programmability and capabilities of new-generation P4 hardware to emulate network links and instantiate a network topology, allowing line-rate traffic to operate using a single physical P4 switch. With P7, it is possible to provide realistic emulation of network topologies using programmable hardware pipeline features such as recirculations, port configurations, different match+action tables, and even Direct Attach Copper (DAC) cables. Furthermore, the user or experimenter can connect physical servers to inject custom traffic (e.g., PCAP-based or Tofino-based) to the emulated networking scenario. Additionally, we increased the level of customization by supporting user-defined P4 codes on each "emulated" switch.

4.2 Design and Implementation

P7 is a high-end, affordable network emulation platform that realistically emulates network topologies using programmable hardware features. P7 overcomes the shortcomings of traditional testbed emulation approaches limited by small-scale environments, software-based/virtualization environments, or simulation-based approaches, compromising different aspects such as fidelity with real networks, flexibility, scalability, and the customizability of experiments.

P7 allows users to define a network topology, including the link metrics, in a user-friendly script, similar to defining topologies using the popular Mininet. From the user-defined topology, P7 internally generates all the necessary files to transform a single P4 hardware switch into a P7 emulator that can realistically run different scenarios. The design of P7 prioritizes simplicity, making it a hardware-based emulation testbed that allows speeds of 10G, 25G, or 100G.

4.2.1 Architecture

The P7 architecture presented in figure 4.1 illustrates the high-level components and their interconnections.

Input: The user defines the topology in the P7 main script. The setup includes the number of links and their characteristics (link metrics), the number of nodes, the custom P4 code (user P4 code), and table configuration.

P7: The central part of our tool, where all the data is processed, and the corresponding files are generated.

- **User P4 Code:** The P4 code is parsed to identify the principal parts of the code such as the parser, ingress, and tables, then the necessary modifications are made, including index in the table and P7 header parser), and the Modified User P4 code is generated.

- **Packet Forwarding:** To perform the routing, we predefine the routing using the Dijkstra algorithm to define the internal forwarding routes based on the shortest path.

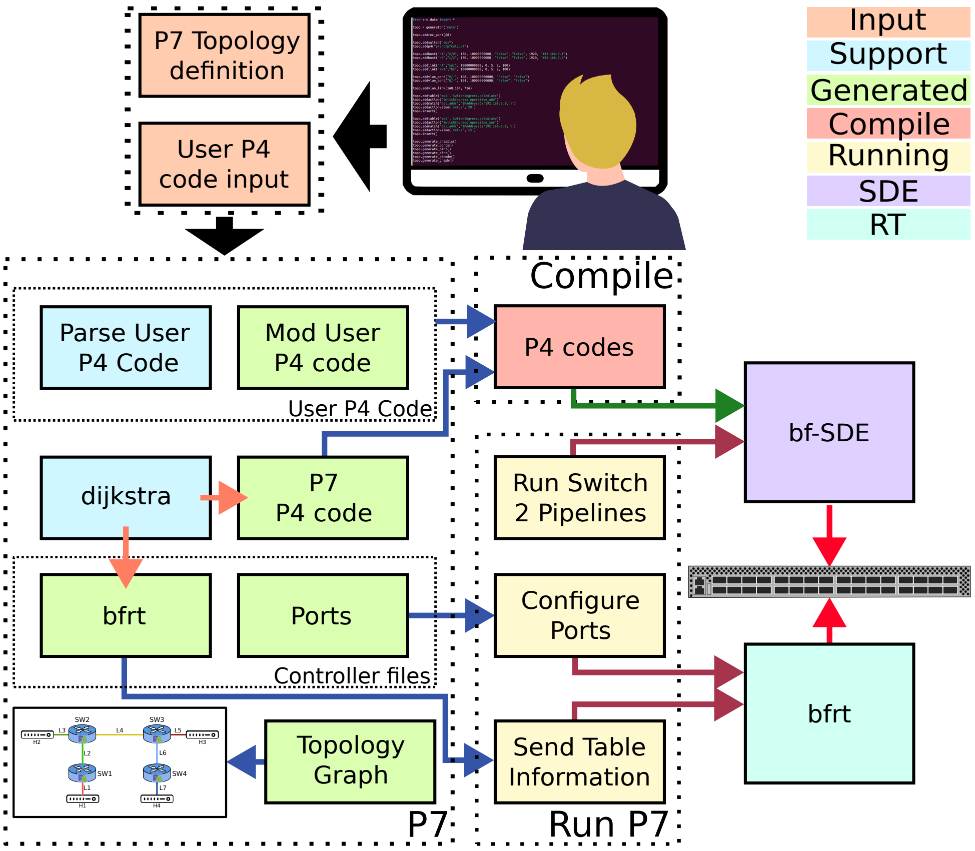

Figure 4.1 – P7 high-level architecture and workflow.

- **Controller files:** Using this information, the table information for the P7 P4 code is prepared using the barefoot runtime (bfrt) format. In addition, The bfrt file includes table information from the user's P4 code. Port mapping and configuration files are also generated from the port configuration input in the main P7 file.
- **P7 P4 Code:** In this part of the architecture, the heart of P7 is also built. This file contains all the recirculation, metrics, headers, and forwarding information running in the Tofino switch.
- **Topology Graph:** In addition to the generated files, a topology graph is also created based on the user configuration.

Run P7: To run P7, it is necessary to compile the generated P4 codes (i.e., User, P7) using the Software Developer Environment (SDE) tools and run the Tofino switch (with the correct pipeline configuration) from the P4 studio environment. On the other hand, it is necessary to set the port configuration and the table information using the bfrt.

From a hardware perspective, the P4 codes deployment is introduced in figure 4.2. To distribute resources, each P4 code (i.e., P7, user) runs in separate pipes (0 and 1). One pipe allocates the P7 P4 code, including all link characteristics logic and forwarding process to the corresponding "emulated" switch. In a different pipe, the user P4 code

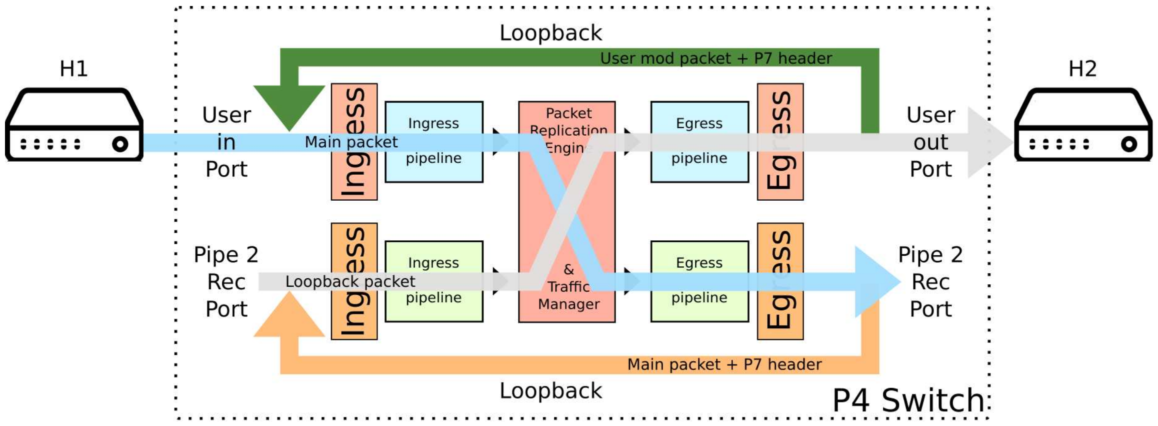

Figure 4.2 – P7 P4 architecture.

is set. The TNA recirculation feature is used to create internal switch representations. When packets need to be forwarded to a switch, a recirculation + pipeline change occurs. Then, the packet is sent back to the P7 pipe to continue with the topology logic.

4.2.2 Main Features

P7 offers several features and characteristics that make it a reliable and accessible tool for network emulation. These features are summarized below:

P4 programmable: P7 is built on P4 hardware, which provides programmable characteristics such as packet recirculation, port configuration, and match+action table abstractions to offer realistic emulation of network topologies.

Affordable: P7 provides high-end emulation capabilities at an affordable cost, making it accessible to researchers with limited budgets and access to a single P4 hardware.

Realistic emulation: P7 allows users to define network topologies with predefined link metrics (Implementation details in table 4.1). The following link metrics can be defined:

- **Bandwidth**: Users can define the bandwidth limit (in Mbps) for each individual link in the topology.
- **Latency**: Users can define per-link latency in milliseconds.
- **Packet loss**: Users can define the probability of dropping a packet as a percentage.
- **Jitter**: In addition to latency, users can define per-link jitter (in ms) plus a probability (per packet) that the jitter will be applied.

Programmable data plane emulation: P7 allows users to add custom P4 code to the internal switches. The user can define the table information using the same P7 script.

Custom traffic traces: P7 allows users to inject custom traffic flows from traffic or trace generators, emulating real-world network scenarios.

Simplicity: P7 provides a user-friendly interface for defining network topologies and autogenerates all the necessary files.

High-speed interfaces: P7 supports high-speed interfaces, such as 10G, 25G, and 100G.

Open source: P7 is publicly available under the Apache License 2.0.

Table 4.1 – P7 link characteristics and P4/TNA implementation approaches.

Link characteristic	P4/TNA implementation approach
Connectivity	Dijkstra algorithm to calculate the routes Internal recirculation to the same pipe represents a link
Latency [**ms**]	Timestamp-based timer Recirculation via internal port until the timer reach the desired latency
Jitter [**ms**]	Random number generator to vary the latency Lookup table with mathematical functions to perform the calculations
Packet loss [%]	Random probability generator to decide the packet discard probability Realistic definition of packet loss to define the good or bad state
Re-ordering [%]	TNA Traffic Management (TM) features Per-packet probability based recirculation
Bandwidth [**bps**]	Rate limit using the Traffic manager feature Port shaping configuration of the ports
Background Traffic [**bps**]	Tofino packet generation engine Up to 100G per pipeline of custom traffic profiles

4.3 Use Case

In this section, we first provide an example of a topology that can be instantiated and devices that can be connected to a network. Next, we detail how the topology is configured within the P7 environment and its features, such as user code and network forwarding.

Figure 4.3 illustrates a high-level overview of a network configuration emulated with P7. In this example, there is a set of interconnected switches (i.e., SW1 to SW5) with different link characteristics (i.e., L1 to L8). Additionally, physical hosts (i.e., H1 to H4) are attached to the switch ports. Custom traffic is sent from a host and passes over the emulated links and switches. Further, Figure 4.4 describes the data plane approach to achieve the network topology mentioned above in the Tofino switch. In P7, link characteristics are implemented in the P7 P4 code. The P4 tables in this pipe define the routing logic and connection to the switches. Recirculations and pipe changes are used to define the routing rules in the P4 code according to the desired topology. We leverage the available physical ports to connect external devices to the switches to forward the traffic to the corresponding link. The user also has the flexibility to customize the P4 code that can

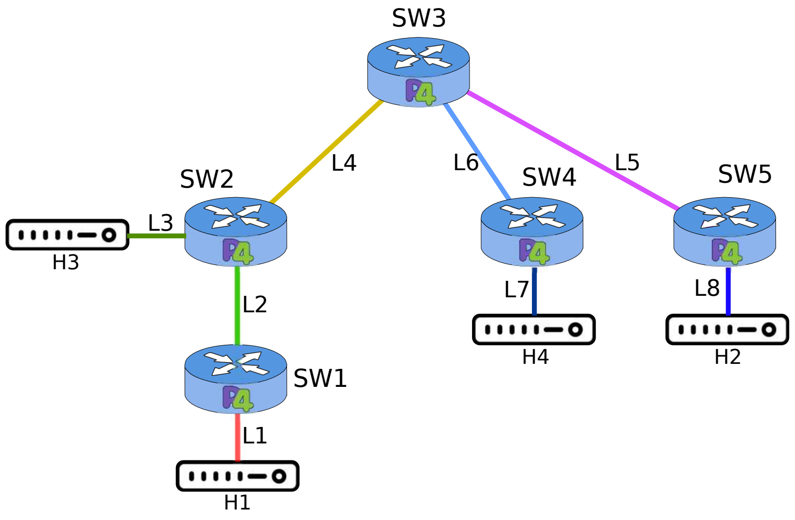

Figure 4.3 – P7 example network topology.

be "emulated" in each internal switch together with the table configuration, thus allowing greater customization of the data plane forwarding logic at each point in the network.

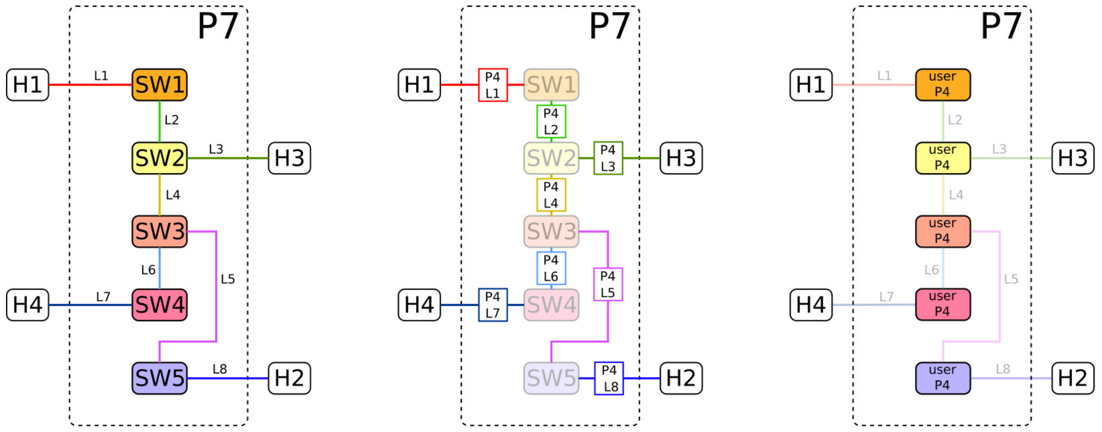

Figure 4.4 – P7 network topology.

4.4 Experimental Evaluation

To evaluate the proposed metrics (shown in Table 4.1), we can use custom network topology, for instance, the one represented in Figure 4.3. End-to-end latency and jitter can be obtained by running a `ping` command between two hosts, such as H1 and H2. Packet loss can be measured either by `ping` or a traffic generator tool by setting independent loss probabilities for each link. Traffic rate and available bandwidth can be estimated by using a traffic generator tool, such as `T-Rex`, to send data between two hosts at a pre-defined bandwidth.

We conducted a series of experiments to evaluate the performance of the proposed metrics, and we can confirm that the experiments are working as expected. We used the topology illustrated in Figure 4.3 and ran various tests to measure the end-to-end latency, jitter, packet loss, and available bandwidth. We used standard tools like `ping` and `T-Rex` to generate traffic and measure the network parameters.

The results obtained from these experiments can be viewed in the demo papers and materials published at (RODRIGUEZ *et al.*, 2022)(CESEN *et al.*, 2023)(RODRIGUEZ *et al.*, 2023), and were consistent with our expectations and matched the theoretical values we calculated. We verified that the proposed metrics accurately reflect the network performance and can be used to diagnose and optimize it.

4.5 Summary

This chapter highlights the significance of P7 in affordable 100G experimental platforms. P7 is a user-friendly and cost-effective network emulator that caters to traditional networking, advanced programmable networking research, and teaching purposes. Additionally, it supports scenarios where metric variation is necessary for tests. As a programmable high-fidelity testbed, P7 offers the advantage of facilitating repeatable and reproducible research. Overall, with our experiments, we are confident that we provide reliable and accurate network performance measurements. The proposed metrics can be used to evaluate and improve the performance of real-world networks.

5 Evaluation

In this chapter, we describe the results found in our evaluations. We divided the results in our two use cases, the microburst detection, and the QoE estimation (described in the Sections 3.1 and 3.3). and evaluate its overhead in the control and data plane. For both use cases, we evaluate aspects such as their impact on the data plane (in terms of resource utilization), and their accuracy in doing what they are intended to do. We also separate in a different section the experiments performed using our P7 emulation tool. Then, after the evaluations, we also held a discussion of the results obtained, highlighting the main points.

5.1 Microburst Detection

In this section, we present our results related to the microburst detection use case. Firstly, we describe the environment we used to perform the experiments, emphasizing the settings relevant to the experiment. In sequence, we demonstrate the use of resources in our strategy, comparing it with the use of resources in state-of-the-art strategies. Then, we present the results that demonstrate the accuracy of our strategy, also buying with other state-of-the-art strategies. Finally, we discuss the obtained results, highlighting the benefits and limitations of our strategy.

5.1.1 Experimental Environment

Here we describe the environment where we have performed the experiments. Our environment contains a *Barefoot Tofino Switch 1* (Edgecore Wedge 100BF-32X) and four servers (Intel Xeon E5-2620v2, dual-port 10G Intel X540-AT2 Network Interface Card (NIC), and 64GB of memory running Ubuntu 20.04) connected via 10G Small Form Pluggable (SFP)+ interfaces. Figure 5.1 shows the organization of the environment used. In the environment, the Tofino switch runs our P4 code strategy (described in Section 3.1) and connects to the 4 servers. We use servers for the following functions:

- **Sender:** Our sender is used to send "normal" traffic, maintaining the link utilization at 10%, as described in (ZHANG *et al.*, 2017).
- **Burster:** Our burster sends bursts at a rate of 3 Gbps to a link with a capacity of 1GB, following the burst data distribution available in (ZHANG *et al.*, 2017).

- **Receiver:** Our receiver is responsible for receiving all this traffic, both normal traffic and bursts.
- **Monitoring server:** The monitoring server is the server that receives reports when a burst is detected, being warned about the burst and the contributing flows.

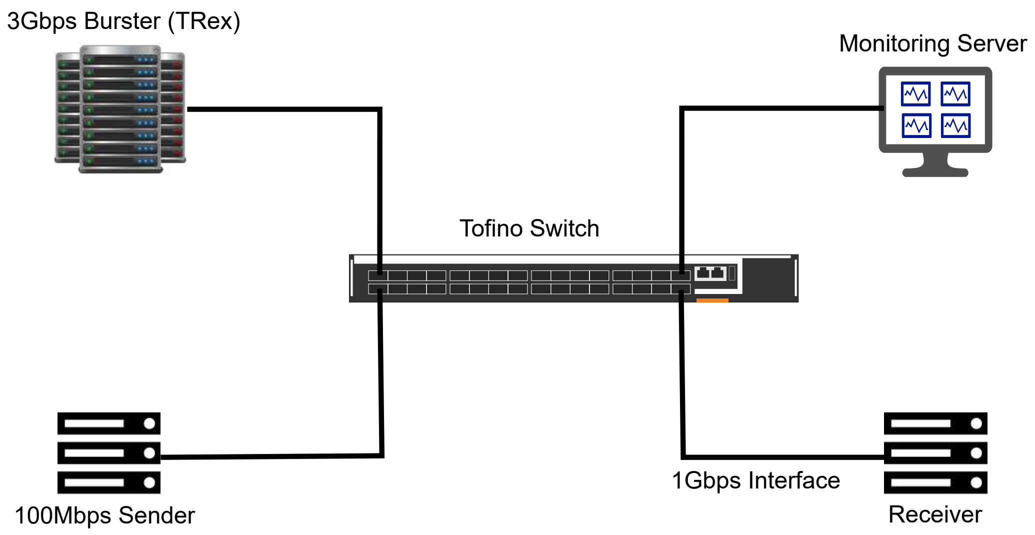

Figure 5.1 – Experimental environment for the microburst detection use case

5.1.2 Resource Utilization

In this section, we present the data plane overhead included by the IPGNET microburst detection algorithm. In this case, the data plane overhead is measured regarding resource utilization of our P4 code compiled for the Tofino switch (NETWORKS, 2018) using the SDE version 9.9. This information is acquired with the P4i tool and shows the utilization of the main resources in the Tofino switch.

Table 5.1 compares the resource utilization of IPGNET with the switch.p4 (CONSORTIUM, 2018) (a baseline P4 program for a switch), and BurstRadar (JOSHI *et al.*, 2018), the state-of-the-art in microbursts detection in the data plane. In the table, we consider the BurstRadar with a ring buffer[1] size of 1k entries, while IPGNET is computing the top-2k flows. Besides that, in the case of IPGNET and BurstRadar, we consider the cost of strategies added to the switch.p4 cost. So, we can see that our proposal uses fewer hardware resources than BurstRadar in all the analyzed resources. Additionally, we can observe that our strategy does not include a huge increase in Switch.p4 resources, with a maximum increase of 3.4%.

[1] Data struct necessary in BurstRadar to report the burst packets

Table 5.1 – Hardware resource utilization

Resource	Switch.p4	BurstRadar	**IPGNET**
Hash Bits	32.3%	37.2%	34.4%
SRAM	29.8%	33.9%	31.1%
TCAM	28.4%	29.2%	28.4%
VLIW Actions	34.6%	39.3%	38.0%
Stateful ALUs	15.6%	28.2%	15.6%

This low use of resources is important and allows other solutions to be implemented along with ours. For example, if we have algorithms for detecting problems that are not addressed in this work, it is possible that we will still implement them because we have a considerable amount of resources available. In addition, we can also use these resources to improve our own method, for example, increasing the number of stored flows or even supporting new features.

5.1.3 Results

In this section, the idea is to present the evaluations carried out by purchasing the proposed strategy with BurstRadar, the state-of-the-art of microburst detecting in the data plane. The main idea here is not to evaluate the accuracy of burst detection, as works like (JOSHI *et al.*, 2018), (GAO *et al.*, 2022), and (CHEN *et al.*, 2018b) already have accurate results in this regard, and moreover, this depends on other factors (i.e., what level of queue occupancy is a burst in your scenario). So, in our experiments, we evaluate how the strategies behave when they detect bursts and what data they report to the control plane.

Carefully providing reports to the control plane provides benefits such as low control plane overhead (in terms of data reports) and low processing overhead, as we can process fewer packets on both the data plane and the control plane. Therefore, our objective is to report to the control plane only the flows that contribute to the burst, so that with this data the control plane can take actions to mitigate these bursts.

To evaluate this, we divided the traffic into burst flows (flows from our burst sender following the burst distribution presented in (ZHANG *et al.*, 2017)) and normal flows, and then measured the reported percentage of those flows. For the queue occupancy threshold (used to detect the bursts), we used the configuration that found the best results in burst detection accuracy (5% of the RTT).

Figure 5.2 shows the number of flows reported for each strategy in a burst scenario. A burst scenario is when the device has already detected the burst (queue occupancy > threshold), and is now deciding which data to report to the control plane.

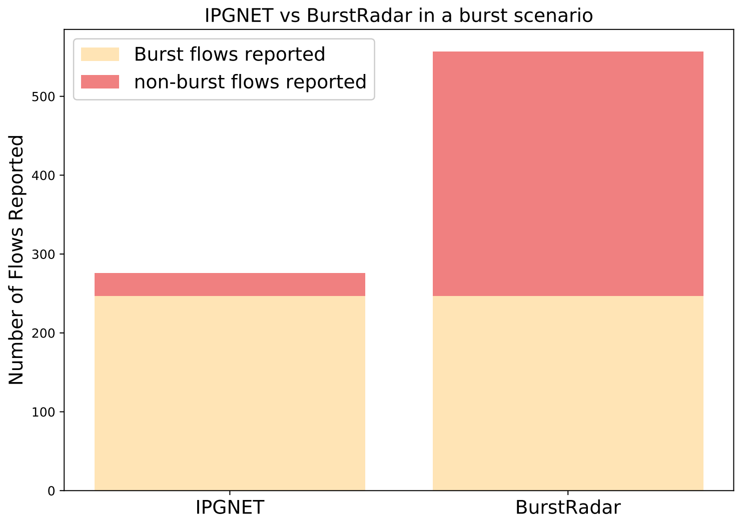

Figure 5.2 – Number of flows reported in a burst scenario.

In the figure, we can see that both strategies report the same number of contributing flows. However, the BurstRadar strategy reports 310 normal flows, while our proposal just reports 29.

Table 5.2 presents the comparison of results on a percentage scale. Then, we can see that both strategies report 100% of burst flows, but IPGNET only reports 9.03% of the non-burst flows, while BurstRadar reports 100%. Even though it manages to report all burst flows, IPGNET reports 50.63% fewer flows than BurstRadar, significantly reducing the control plane overhead.

Table 5.2 – Flows reported to the controller in a burst

Flows	BurstRadar	**IPGNET**
Burst Flows Reported	100%	100%
Non Burst Flows Reported	100%	9.03%
Total Flows Reported	100%	49.37%

The impact of this reduction in the number of reported flows is significant for the total amount of data reported. For example, assume that flows are reported using a package clone. Also, consider that our reports contain a size of 64 bytes each (minimum Ethernet frame size). So, the impact of this reduction on the number of reported flows will depend on the number of packets per flow which can be seen below.

In Figure 5.3, we can see the total amount of data reported to the control plane with the increase of flows reported and packets per-flow. As we can see, with the reporting of 100 additional flows, we can have an increase of 62.5 KB to up to 6.25 MB of data reported, depending on the number of packets per-flow (in this case, ranging from 10 to 1000). As in our experiment BurstRadar reported 310 flows more than necessary

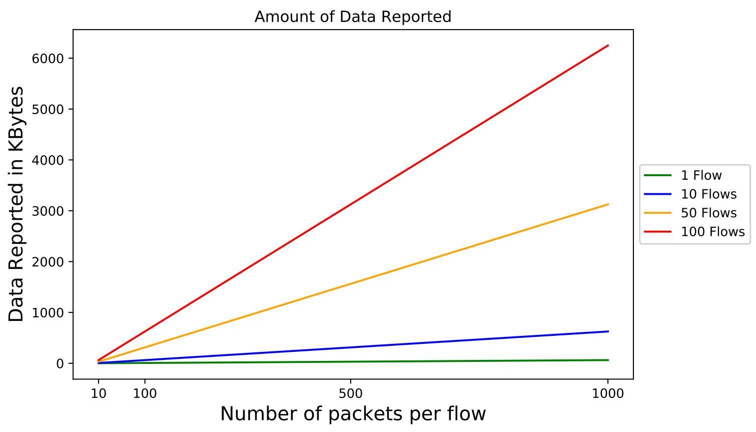

Figure 5.3 – Total amount of data reported with the increase of packets per flow.

(normal flows), and in general, IMIX(IMIX, 2021) uses small flows, with an average of 12 packets, BurstRadar reported approximately 19,4 KB more than necessary. On the other hand, our strategy only reported 19 more flows than necessary, resulting in approximately 1KB of extra data reported.

5.1.4 Discussion

Based on the executed results, we can observe some interesting points about the comparison between BurstRadar and our strategy. First, we can observe that our strategy consumes fewer resources from the Tofino switch, which is interesting because it opens up space for us to use other algorithms in conjunction with the proposed solution. Nevertheless, BurstRadar can also be considered a low-cost strategy, even using more hardware resources, as it does not use 40% of the available resources. Two factors that may influence our use of fewer resources than BurstRadar are: (I) the compiler version and (ii) the ring buffer in BurstRadar.

BurstRadar was implemented and compiled on an ancient version of the SDE compiler, which may influence your resource allocation/usage. However, another factor that influences is the composition of the ring buffer, which is the data structure used to carry out the reports in BurstRadar, which indeed uses more resources than our data structures for storing the IPG.

Regarding reporting contributing flows in burst situations, our algorithm and BurstRadar successfully detect all contributing flows to microbursts. However, BurstRadar reports many more packets to the control plane, which adds network overhead. This occurs because BurstRadar is not concerned with reporting the flows that most contributed to the burst but all the packets in the queue during the occurrence of a burst.

Nonetheless, despite our algorithm showcasing improved accuracy, we acknowledge the presence of false positives, as indicated by the evaluation results. These false positives are likely influenced by factors previously discussed in Section 3.1.3.3.

5.2 Data Plane QoE Estimation

In this section, we present our evaluation of our second use case: the QoE estimation entirely in the data plane. As discussed in Section 3.2, we seek to perform QoE inference entirely in the data plane, and for that, our first implementation considers the IPGw as QoE estimation. Then, in this section, we demonstrate the correlation between the proposed estimate with QoE measured outside the data plane. The structure of the section follows the same as the results on microburst detection, initially presenting the environment and the resources utilization and moving on to the results and discussion.

5.2.1 Experimental Environment

Similarly to our microburst strategy, we evaluate the performance of `QoEyes` using a *Barefoot Tofino Switch* (Edgecore Wedge 100BF-32X) and four servers (Intel Xeon E5-2620v2, dual-port 10G Intel X540-AT2 NIC, and 64GB of memory running Ubuntu 20.04) connected via 10G SFP+ interfaces. However, in this case, we also use the P7 emulator running on the Tofino to simulate network metrics like bandwidth and latency. Figure 5.4 illustrates the setup of our testbed. Then, in addition to our P4 code and P7 running on Tofino, in this case, despite using the same physical servers, they now assume a different behavior. These new functions are described below:

- **VR-EXP Player:** This server runs the VR-EXP player (FILHO *et al.*, 2019) to request 360-degree video content. In addition to downloading the video in a tile-based scheme, this player saves objective QoE metrics to calculate the QoE after the end of the video session.
- **Traffic generator:** This traffic generator server generates multiple video session requests in parallel as background traffic.
- **Apache server:** This server is equipped with an Apache server that hosts the VR video content and responds to requests from clients in the network.
- **Monitoring Server:** The monitoring server is the server that receives the QoE reports of the monitored flows (VR video sessions).
- **P7 Emulator:** Unlike the experiments carried out for detecting microbursts, in these experiments, the Tofino switch, and running the P4 code of our strategy, also

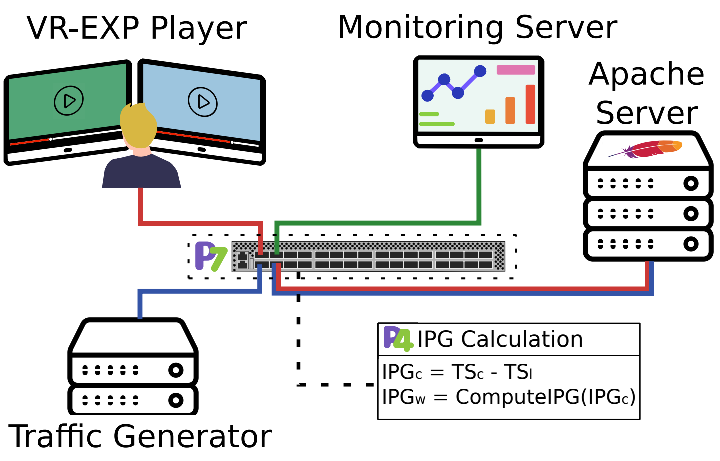

Figure 5.4 – Experimental environment for the QoE estimation.

runs the P7 emulator. This is done so that some network metrics (e.g., bandwidth and latency) can be emulated.

Furthermore, in the experiments, we rely on the same traces used by Filho et al.(FILHO *et al.*, 2019). The publicly available traces comprise two 360-degree videos: Google Spotlight and Freestyle Skiing (WU *et al.*, 2017). Each VR video is encoded with 720p, 1080p, and 4K qualities. VR video sessions are initiated using a random user file (also from (FILHO *et al.*, 2019)), which contains a series of movements (described in axis x, y, z) of user interactions with the VR.

5.2.2 Resource Utilization

In this section, we present the resource utilization of our algorithm compiled for Tofino using the SDE version 9.9. Table 5.3 shows the resource utilization of the switch.p4 (baseline of p4 code for switching) and the cost of the extra logic `QoEyes` imposes on the switch.p4 (i.e., the cost of `QoEyes` + switch.p4). Observe that `QoEyes` adds not much to the switch's overall physical resource utilization, with a maximum increase of 4.2% in VLIW actions.

However, unlike what is done in the microburst detection results, we did not compare the resources used by our strategy with any other method. This is because *QoEyes* is the first step in QoE estimation entirely in the data plane. Note also that our QoE estimation strategy working independently consumes fewer resources than the microburst detection strategy. This is because to estimate QoE, we do not need some data structures that are used for microburst detection (e.g., IPGw average).

Table 5.3 – *QoEyes* hardware resource utilization

Resource	**Switch.p4**	**QoEyes**
Hash Bits	32.3%	34.2%
SRAM	29.8%	30.6%
TCAM	28.4%	28.4%
VLIW Actions	34.6%	38.8%
Stateful ALUs	15.6%	15.6%

5.2.3 Results

First, we evaluate the relationship between the IPGw measured in the data plane by `QoEyes` and the QoE measured by the user side. For the QoE user side model, we use as a baseline the QoE model provided by Filho et al. (FILHO *et al.*, 2019), which utilizes the output provided by the VR-EXP player to calculate the MOS. In our case, we consider for the calculation of the ideal MOS (MOS = 5), the case where we have only one active video session with the maximum available link capacity. In all our experiments, we varied the physical port capacity (with Tofino port shaping) of 10Gbps, 1Gbps, and 100Mbps. Also, we varied the number of active VR video sessions running on the network from 100 to 2000. These parameters were applied in our testbed to have different network conditions.

Table 5.4 – Number of tiles per zone received with 100 sessions in parallel

	10Gb			1Gb			100Mb		
	z1	z2	z3	z1	z2	z3	z1	z2	z3
720p	1	12	1375	1	12	1375	1	12	1375
4K	59	468	5	59	468	5	59	468	5

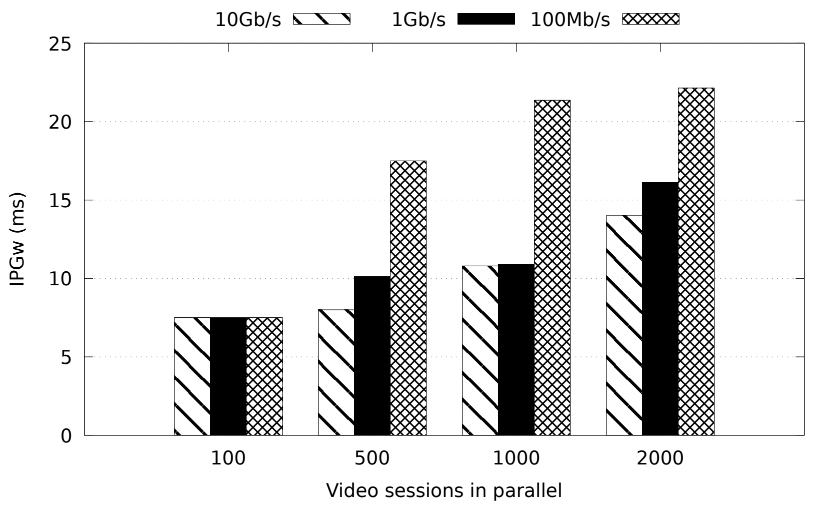

Figure 5.5 – IPGw with the increasing number of VR video sessions.

Figures 5.5 and 5.6 illustrate the IPGw and QoE measured in the experiment by `QoEyes`. As observed in Figure 5.6, the QoE appears to decline with an increase in the

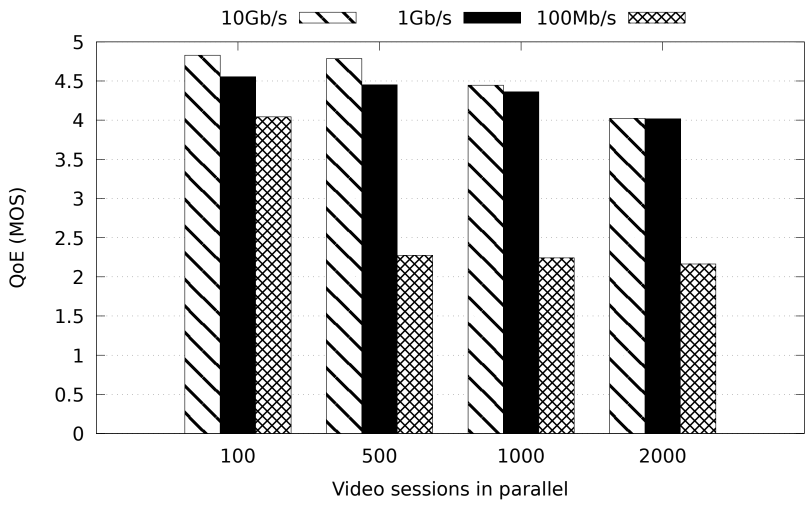

Figure 5.6 – QoE (MOS) with the increasing number of VR video sessions..

number of concurrent active sessions or a decrease in transmission capacity. In Figure 5.5, the IPG shows a similar behavior (but increasing) when the number of active sessions increases, or the transmission capacity decreases. The only exception can be seen when there are 100 sessions, where the IPG remains constant, but the QoE varies. It occurs because with a small number of sessions, the traffic can be forwarded in the minimum IPG time (less than 1 ms), and there is only a gap between the establishment and closing of new TCP connections. However, the QoE model employed calculates the QoE based on the bitrate and thus produces better results with higher transmission capacity, even if all tiles arrive with the same quality (see Table 5.4).

In Figure 5.7, we can see the IPGw behavior over 2000 reports for a flow with a measured QoE MOS greater than 4.5 (blue color) and another flow with a QoE MOS less than 2.5 (red color). Note that most of the time, the flow with QoE below 2.5 has a higher IPGw than the flow with good QoE. Furthermore, the IPGw of the flow with lower QoE has higher peaks and a longer recovery time (time to return to a low IPG). In both experiments (Figures 5.5, 5.6 and 5.7), we can observe a strong correlation between the IPGw measured in the data plane and the QoE MOS measured on the user side.

Figure 5.8 shows the IPGw and MOS measured in a latency increase scenario. The blue line represents the IPGw measured, while the green line is the QoE. Both are measured with a latency increase between 0 and 20 ms. As we can see, the QoE decreases from 5 to 4.12, while the IPG increases from 0.98 to 2.4. This reinforces IPGw's correlation with the measured QoE, where both present a similar behavior even in a scenario of increased latency. Remember that using IPGw for QoE estimation is only a first step for QoE estimation entirely in the data plane, but it already demonstrates exciting results.

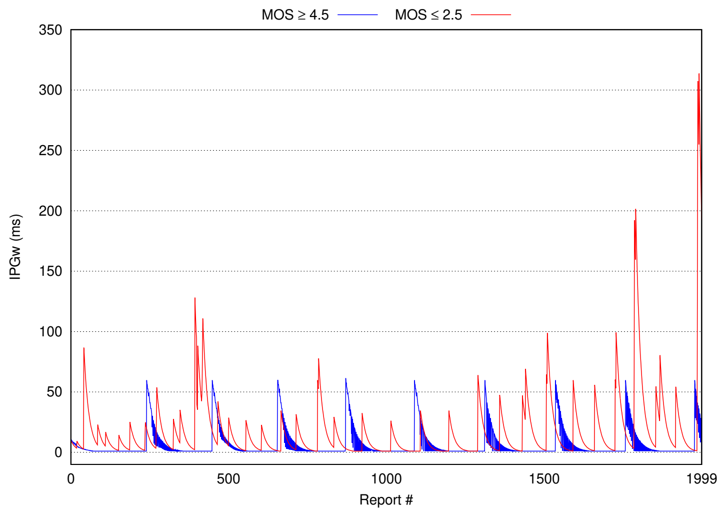

Figure 5.7 – Calculated IPGw over VR video sessions with different MOS.

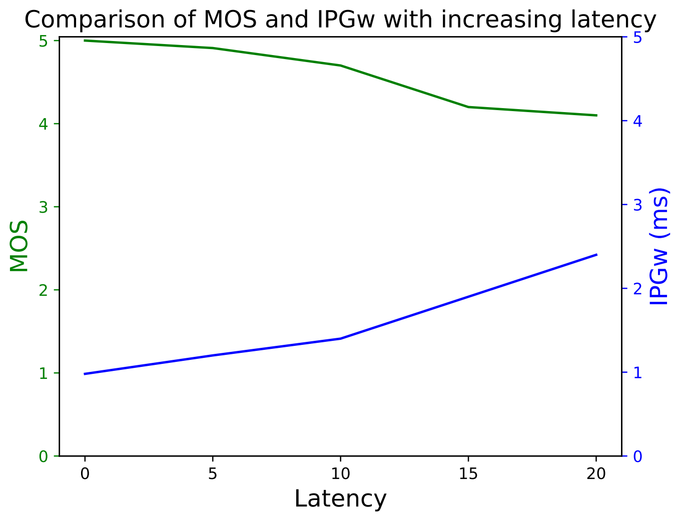

Figure 5.8 – Correlation between IPGw and MOS with the latency increase.

5.2.4 Discussion

Based on the experiments performed, we can get some insights about the data plane QoE measurement. Firstly, we can conclude that it is possible to implement methods/metrics that are used in the literature for QoE estimation, such as the use of the IPGw proposed in (MUSTAFA; ROTHENBERG, 2022). Second, we can observe that even using a simple method, we can have a reasonable estimate of the QoE for many cases and thus take advantage of a QoE inference in the data plane. Third, we also observed that using only the IPGw as a QoE estimate, despite presenting a good correlation, does not necessarily reflect the QoE in all cases, presenting some limitations and, consequently, the need to be extended. Finally, the results demonstrate that the proposal has great potential, but there are still many challenges and limitations to be overcome and much room for further research.

6 Conclusions

6.1 Final Remarks

In conclusion, this research focused on addressing the challenges posed by the rapid growth of network traffic and the increasing demands for network capacity, latency, and quality of service. The study proposed the use of IPGNET as a low-cost autonomous data plane solution for detecting network anomalies and improving the QoE in VR video streaming.

The main contributions of this work include the development of a data plane autonomous solution that operates without the need for intervention from the control plane. The implementation of a microburst detection strategy was also presented, allowing for the identification and reporting of contributing flows. Additionally, a design for in-network QoE estimation in VR video streaming was provided, enabling targeted optimizations to enhance latency, video quality, and overall user experience.

To evaluate the proposed techniques, high-fidelity experiments were conducted using physical hardware, specifically the Tofino switch. However, due to limited physical network infrastructure, the research also introduced P7 as an emulation tool capable of simulating different events with high fidelity. This allowed for the evaluation of the proposed strategies in various network scenarios.

The objectives of this research were successfully achieved, exploring the capabilities of IPG as a network metric, developing algorithms and methodologies for microburst detection, investigating the correlation between IPG metrics and QoE in VR video streaming, and evaluating the performance and effectiveness of the proposed techniques through extensive experimental simulations and real-world network deployments.

The open-source nature of the implemented solutions, including the use cases and the P4-based network emulator, ensures that the findings and contributions of this research can be further developed, extended, and adopted by the broader research community. By addressing the challenges of network monitoring and performance optimization, this study contributes to the ongoing efforts in enhancing network operations and delivering a satisfactory user experience in the face of evolving network demands.

6.2 Future Work

As future works, researchers can continue exploring the capabilities of the IPG metric, deriving it in other metrics, and applying it in different contexts. While this work focused on microburst detection, there is potential for expanding the scope of anomaly detection. Future research can explore the use of IPG and other network metrics to identify and address additional network anomalies, such as Distributed Denial-of-Service (DDoS) attacks, traffic congestion, and network faults.

In the QoE estimation side, future work can focus on adapting the QoE estimation techniques for different network conditions, including varying network bandwidth, latency, and congestion levels. For that, it is possible to extend our QoE estimation method, considering other QoS metrics and even using more complex models (e.g., ML algorithms). Additionally, research efforts can be devoted to intend to work on other parts of QoE inference (described in Section 3.2.2), such as traffic classification and actions to improve the bad QoE.

By pursuing these future directions, researchers and practitioners can continue to enhance network performance, optimize resource utilization, and deliver exceptional user experiences in the face of evolving network demands and challenges.

Bibliography

AGARWAL, K.; ROZNER, E.; DIXON, C.; CARTER, J. Sdn traceroute: Tracing sdn forwarding without changing network behavior. In: *Proceedings of the third workshop on Hot topics in software defined networking*. [S.l.: s.n.], 2014. p. 145–150. Cited on page 16.

BASAT, R. B.; RAMANATHAN, S.; LI, Y.; ANTICHI, G.; YU, M.; MITZENMACHER, M. Pint: Probabilistic in-band network telemetry. In: *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. [S.l.: s.n.], 2020. p. 662–680. Cited 2 times on pages 37 and 39.

BENSON, T.; AKELLA, A.; MALTZ, D. A. Unraveling the complexity of network management. In: *NSDI*. [S.l.: s.n.], 2009. p. 335–348. Cited on page 21.

BHAT, D. *et al.* Application-based qoe support with p4 and openflow. In: IEEE. *IEEE INFOCOM*. [S.l.], 2019. p. 817–823. Cited 2 times on pages 36 and 39.

BOSSHART, P.; DALY, D.; GIBB, G.; IZZARD, M.; MCKEOWN, N.; REXFORD, J.; SCHLESINGER, C.; TALAYCO, D.; VAHDAT, A.; VARGHESE, G. *et al.* P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, ACM New York, NY, USA, 2014. Cited 2 times on pages 16 and 23.

CANOFRE, R.; CASTRO, A.; LORENZON, A.; ROSSI, F.; LUIZELLI, M. Towards efficient selective in-band network telemetry report using smartnics. In: SPRINGER. *International Conference on Advanced Information Networking and Applications*. [S.l.], 2022. p. 271–284. Cited 3 times on pages 16, 37, and 39.

CAO, J. *et al.* Turbonet: Faithfully emulating networks with programmable switches. In: *IEEE 28th ICNP*. Madrid, Spain: IEEE, 2020. p. 1–11. Cited on page 55.

CESEN, F. E. R.; VOGT, F. G.; CASTRO, A. G. D.; ROTHENBERG, C. E. Towards multiple pipelines network emulation with p7. In: IEEE. *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. [S.l.], 2023. p. 290–292. Cited on page 61.

CHEN, S.; ZHANG, Y.; LI, Y.; CHEN, Z.; WANG, Z. Spherical structural similarity index for objective omnidirectional video quality assessment. In: IEEE. *2018 IEEE international conference on multimedia and expo (ICME)*. [S.l.], 2018. p. 1–6. Cited 2 times on pages 35 and 39.

CHEN, X.; FEIBISH, S. L.; KORAL, Y.; REXFORD, J.; ROTTENSTREICH, O. Catching the microburst culprits with snappy. In: *Proceedings of the Afternoon Workshop on Self-Driving Networks*. [S.l.: s.n.], 2018. p. 22–28. Cited 4 times on pages 35, 39, 43, and 64.

CISCO. *Cisco Visual Networking Index: Forecast and Trends, 2017-2022*. [S.l.], 2019. Cited on page 16.

CONSORTIUM, P. L. *Baseline switch.p4*. 2018. <https://github.com/p4lang/switch>. Cited on page 63.

FILHO, R. I. T. D. C.; LUIZELLI, M. C.; PETRANGELI, S.; VEGA, M. T.; HOOFT, J. V. d.; WAUTERS, T.; TURCK, F. D.; GASPARY, L. P. Dissecting the performance of vr video streaming through the vr-exp experimentation platform. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, ACM New York, NY, USA, v. 15, n. 4, p. 1–23, 2019. Cited 5 times on pages 35, 39, 67, 68, and 69.

FILHO, R. I. T. da C. *et al.* Predicting the performance of virtual reality video streaming in mobile networks. In: *Proceedings of the 9th ACM Multimedia Systems Conference*. [S.l.: s.n.], 2018. p. 270–283. Cited 2 times on pages 35 and 39.

GAO, K.; LI, D.; WANG, S. Bandwidth-efficient microburst measurement in large-scale datacenter networks. 2022. Cited 5 times on pages 35, 39, 41, 43, and 64.

GUO, C.; YUAN, L.; XIANG, D.; DANG, Y.; HUANG, R.; MALTZ, D.; LIU, Z.; WANG, V.; PANG, B.; CHEN, H. *et al.* Pingmesh: A large-scale system for data center network latency measurement and analysis. In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. [S.l.: s.n.], 2015. p. 139–152. Cited on page 16.

HAO, M. K. F.; LAKSHMAN, T. V. *Line-rate, real-time traffic detector*. 2011. US Patent 8,054,760. Cited 2 times on pages 34 and 39.

HARRINGTON, D.; PRESUHN, R.; WIJNEN, B. *An architecture for describing simple network management protocol (SNMP) management frameworks*. [S.l.], 2002. Cited on page 16.

HOHEMBERGER, R.; CASTRO, A.; VOGT, F.; MANSILHA, R.; LORENZON, A.; ROSSI, F.; LUIZELLI, M. Orchestrating in-band data plane telemetry with machine learning. *IEEE Communications Letters*, IEEE, v. 23, n. 12, p. 2247–2251, 2019. Cited on page 16.

HUANG, Q.; SUN, H.; LEE, P. P.; BAI, W.; ZHU, F.; BAO, Y. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. In: *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. [S.l.: s.n.], 2020. p. 404–421. Cited 2 times on pages 37 and 39.

IMIX. *IMIX Traffic*. 2021. <ttps://en.wikipedia.org/wiki/Internet_Mix>. Cited on page 66.

IURIAN, C.-M.; BOTEZ, R.; IVANCIU, I.-A.; DOBROTA, V. Video streaming evaluation using priority queuing in p4 programmable networks. In: IEEE. *2022 21st RoEduNet Conference: Networking in Education and Research (RoEduNet)*. [S.l.], 2022. p. 1–5. Cited 4 times on pages 36, 39, 49, and 51.

JIA, C.; PAN, T.; BIAN, Z.; LIN, X.; SONG, E.; XU, C.; HUANG, T.; LIU, Y. Rapid detection and localization of gray failures in data centers via in-band network telemetry. In: IEEE. *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. [S.l.], 2020. p. 1–9. Cited 2 times on pages 37 and 39.

JOSHI, R.; QU, T.; CHAN, M. C.; LEONG, B.; LOO, B. T. Burstradar: Practical real-time microburst monitoring for datacenter networks. In: *Proceedings of the 9th Asia-Pacific Workshop on Systems*. [S.l.: s.n.], 2018. p. 1–8. Cited 5 times on pages 34, 35, 39, 63, and 64.

KANNAN, P. G. *et al.* Bnv: Enabling scalable network experimentation through bare-metal network virtualization. In: *USENIX*. USA: [s.n.], 2018. (CSET'18), p. 6. Cited 2 times on pages 17 and 55.

KIM, Y.; SUH, D.; PACK, S. Selective in-band network telemetry for overhead reduction. In: IEEE. *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*. [S.l.], 2018. p. 1–3. Cited on page 16.

KOPONEN, T. *et al.* Network virtualization in multi-tenant datacenters. In: *USENIX NSDI*. Seattle, WA: USENIX Association, 2014. p. 203–216. ISBN 978-1-931971-09-6. Disponível em: <https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/koponen>. Cited on page 55.

KREUTZ, D.; RAMOS, F. M.; VERISSIMO, P.; ROTHENBERG, C.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, Ieee, v. 103, n. 1, p. 14–76, 2014. Cited on page 16.

KREUTZ, D.; RAMOS, F. M. V.; VERíSSIMO, P. E.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, v. 103, n. 1, p. 14–76, Jan 2015. ISSN 0018-9219. Cited 2 times on pages 9 and 23.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: Rapid prototyping for software-defined networks. In: *SIGCOMM*. NY, USA: ACM, 2010. (Hotnets-IX). ISBN 9781450304092. Disponível em: <https://doi.org/10.1145/1868447.1868466>. Cited 2 times on pages 17 and 55.

LI, H. *et al. Enabling End-to-End Simulation for Host Networking Evaluation using SimBricks*. arXiv, 2020. Disponível em: <https://arxiv.org/abs/2012.14219>. Cited on page 55.

LIU, H. H. *et al.* Crystalnet: Faithfully emulating large production networks. In: *26th SOSP*. NY, USA: ACM, 2017. ISBN 9781450350853. Disponível em: <https://doi.org/10.1145/3132747.3132759>. Cited on page 55.

LIU, Z.; BI, J.; ZHOU, Y.; WANG, Y.; LIN, Y. Netvision: Towards network telemetry as a service. In: IEEE. *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. [S.l.], 2018. p. 247–248. Cited 2 times on pages 16 and 28.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>. Cited on page 22.

MOLERO, E. C.; VISSICCHIO, S.; VANBEVER, L. Fast in-network gray failure detection for isps. In: *Proceedings of the ACM SIGCOMM 2022 Conference*. [S.l.: s.n.], 2022. p. 677–692. Cited 2 times on pages 37 and 39.

MUSTAFA, R. U.; ROTHENBERG, C. E. Machine learning assisted real-time dash video qoe estimation technique for encrypted traffic. In: *Proceedings of the 1st Mile-High Video Conference*. [S.l.: s.n.], 2022. p. 123–123. Cited 5 times on pages 29, 34, 39, 50, and 71.

NETWORKS, B. *Tofino Switch*. 2018. <https://goo.gl/cdEK1E>. Cited on page 63.

P4.ORG. *Open-Source P4 Implementation of Features Typical of an Advanced l2/l3 Switch*. 2015. <https://github.com/p4lang/switch>. Cited on page 46.

P4.ORG APPLICATIONS WORKING GROUP. *In-band Network Telemetry (INT) Dataplane Specification*. [S.l.], 2020. Cited on page 29.

PIRATLA, N.; JAYASUMANA, A.; SMITH, H. Overcoming the effects of correlation in packet delay measurements using inter-packet gaps. In: IEEE. *Proceedings. 2004 12th IEEE International Conference on Networks*. [S.l.], 2004. Cited 3 times on pages 17, 34, and 39.

RODRIGUEZ, F.; VOGT, F. G.; CASTRO, A. G. D.; SCHWARZ, M. F.; ROTHENBERG, C. P4 programmable patch panel (p7) an instant 100g emulated network on your tofino-based pizza box. In: *Proceedings of the SIGCOMM'22 Poster and Demo Sessions*. [S.l.: s.n.], 2022. p. 4–6. Cited on page 61.

RODRIGUEZ, F.; VOGT, F. G.; CASTRO, A. G. D.; SCHWARZ, M.; ROTHENBERG, C. Network emulation with p7: A p4 programmable patch panel on tofino-based hardware. In: SBC. *Anais Estendidos do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.], 2023. p. 40–47. Cited on page 61.

SA-INGTHONG, J.; PHONPHOEM, A.; JANSANG, A.; JAIKAEO, C. Probabilistic analysis of packet losses in dense lora networks. In: IEEE. *2021 13th International Conference on Knowledge and Smart Technology*. [S.l.], 2021. Cited 3 times on pages 17, 34, and 39.

SCHWARZMANN, S.; MARQUEZAN, C. C.; TRIVISONNO, R.; NAKAJIMA, S.; BARRIAC, V.; ZINNER, T. Ml-based qoe estimation in 5g networks using different regression techniques. *IEEE Transactions on Network and Service Management*, IEEE, v. 19, n. 3, p. 3516–3532, 2022. Cited 2 times on pages 35 and 39.

SINGH, S. K.; ROTHENBERG, C. E.; LUIZELLI, M. C.; ANTICHI, G.; GOMES, P. H.; PONGRÁCZ, G. Hh-ipg: Leveraging inter-packet gap metrics in p4 hardware for heavy hitter detection. *IEEE Transactions on Network and Service Management*, IEEE, 2022. Cited 7 times on pages 17, 29, 34, 38, 39, 43, and 46.

SOMMER, R.; FELDMANN, A. Netflow: Information loss or win? In: *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. [S.l.: s.n.], 2002. p. 173–174. Cited on page 16.

STORCH, I.; AGOSTINI, L.; ZATT, B.; BAMPI, S.; PALOMINO, D. Fastinter360: A fast inter mode decision for hevc 360 video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, IEEE, v. 32, n. 5, p. 3235–3249, 2021. Cited 2 times on pages 35 and 39.

SUN V. XU, S. A. Y.; DAFTARY, K. *Measuring network performance using inter-packet gap metric*. 2015. US Patent 9,178,783. Cited 2 times on pages 34 and 39.

TAN, C.; JIN, Z.; GUO, C.; ZHANG, T.; WU, H.; DENG, K.; BI, D.; XIANG, D. {NetBouncer}: Active device and link failure localization in data center networks. In: *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. [S.l.: s.n.], 2019. p. 599–614. Cited 2 times on pages 36 and 39.

TAN, L.; SU, W.; ZHANG, W.; LV, J.; ZHANG, Z.; MIAO, J.; LIU, X.; LI, N. In-band network telemetry: A survey. *Computer Networks*, Elsevier, v. 186, p. 107763, 2021. Cited on page 16.

UPENIK, E.; RERABEK, M.; EBRAHIMI, T. On the performance of objective metrics for omnidirectional visual content. In: IEEE. *2017 ninth international conference on quality of multimedia experience (QoMEX)*. [S.l.], 2017. p. 1–6. Cited 2 times on pages 36 and 39.

VIDHYA, R.; KARTHIK, P.; JAMADAGNI, S. Anticipatory qoe mechanisms for 5g data analytics. In: IEEE. *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*. [S.l.], 2020. p. 523–526. Cited 2 times on pages 35 and 39.

VOGT, F. G.; RODRIGUEZ, F.; ROTHENBERG, C.; PONGRÁCZ, G. Innovative network monitoring techniques through in-band inter packet gap telemetry (ipgnet). In: *Proceedings of the 5th International Workshop on P4 in Europe*. [S.l.: s.n.], 2022. p. 53–56. Cited on page 29.

WU, C.; TAN, Z.; WANG, Z.; YANG, S. A dataset for exploring user behaviors in vr spherical video streaming. In: *Proceedings of the 8th ACM on Multimedia Systems Conference*. [S.l.: s.n.], 2017. p. 193–198. Cited on page 68.

WU, P.; CUI, Y.; WU, J.; LIU, J.; METZ, C. Transition from ipv4 to ipv6: A state-of-the-art survey. *IEEE Communications Surveys & Tutorials*, IEEE, v. 15, n. 3, p. 1407–1424, 2012. Cited on page 21.

YU, M.; LAKSHMAN, H.; GIROD, B. A framework to evaluate omnidirectional video coding schemes. In: *2015 IEEE International Symposium on Mixed and Augmented Reality*. [S.l.: s.n.], 2015. p. 31–36. Cited 3 times on pages 33, 36, and 39.

ZAKHARCHENKO, V.; CHOI, K. P.; PARK, J. H. Quality metric for spherical panoramic video. In: SPIE. *Optics and Photonics for Information Processing X*. [S.l.], 2016. v. 9970, p. 57–65. Cited on page 33.

ZHANG, Q.; LIU, V.; ZENG, H.; KRISHNAMURTHY, A. High-resolution measurement of data center microbursts. In: *Proceedings of the 2017 Internet Measurement Conference*. [S.l.: s.n.], 2017. p. 78–85. Cited 3 times on pages 48, 62, and 64.

ZHAO, Y.; YANG, K.; LIU, Z.; YANG, T.; CHEN, L.; LIU, S.; ZHENG, N.; WANG, R.; WU, H.; WANG, Y. *et al.* {LightGuardian}: A {Full-Visibility}, lightweight, in-band telemetry system using sketchlets. In: *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. [S.l.: s.n.], 2021. p. 991–1010. Cited 2 times on pages 36 and 39.

ZHOU, Y.; SUN, C.; LIU, H. H.; MIAO, R.; BAI, S.; LI, B.; ZHENG, Z.; ZHU, L.; SHEN, Z.; XI, Y. *et al.* Flow event telemetry on programmable data plane. In: *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. [S.l.: s.n.], 2020. p. 76–89. Cited 2 times on pages 36 and 39.