Marcelo Ribeiro Nascimento

Proposta e Validação de Nova Arquitetura de Roteamento IP com Separação de Planos

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Orientador: Maurício Ferreira Magalhães
Co-orientador: Christian Rodolfo Esteve Rothenberg

Campinas, SP
2012

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

N17p

Nascimento, Marcelo Ribeiro
Proposta e validação de nova arquitetura de roteamento IP com separação de planos / Marcelo Ribeiro Nascimento. – Campinas, SP: [s.n.], 2012.

Orientador: Maurício Ferreira Magalhães
Coorientador: Christian Rodolfo Esteve Rothenberg.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Comutação de pacotes. 2. Redes de computadores - Protocolos. 3. Arquitetura de rede de computador. 4. Sistemas distribuídos. 5. Sistemas de transmissão de dados. I. Magalhães, Maurício Ferreira, 1951-. II. Rothenberg, Christian Rodolfo Esteve. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título

Título em Inglês:	Proposal and evaluation of a new IP routing architecture with separation of planes
Palavras-chave em Inglês:	Packet switching, Computer networks - Protocols, Architecture of computer network, distributed systems, System of data transmission
Área de concentração:	Engenharia de Computação
Titulação:	Mestre em Engenharia Elétrica
Banca Examinadora:	Cesar Augusto Cavalheiro Marcondes, Marco Aurélio Amaral Henriques
Data da defesa:	21/06/2012

Marcelo Ribeiro Nascimento

Proposta e Validação de Nova Arquitetura de Roteamento IP com Separação de Planos

Tese de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.
Aprovação em 21/06/2012

Banca Examinadora:
Prof. Dr. Cesar Augusto Cavalheiro Marcondes - UFSCAR
Prof. Dr. Marco Aurélio Amaral Henriques - UNICAMP
Prof. Dr. Maurício Ferreira Magalhães - UNICAMP

Campinas, SP
2012

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Marcelo Ribeiro Nascimento

Data da Defesa: 21 de junho de 2012

Título da Tese: "Proposta e Validação de Nova Arquitetura de Roteamento IP com Separação de Planos"

Prof. Dr. Maurício Ferreira Magalhães (Presidente): _____

Prof. Dr. Cesar Augusto Cavalheiro Marcondes: _____

Prof. Dr. Marco Aurélio Amaral Henriques: _____

Resumo

Os roteadores atuais implementam uma arquitetura verticalmente integrada composta de uma camada de software e um hardware proprietários. Este modelo resulta em soluções de alto custo e inviabiliza a experimentação de novas ideias. Em contrapartida, existem alternativas de alta flexibilidade baseadas em software e, consequentemente, de baixo custo. Entretanto, essas soluções apresentam baixo desempenho. Motivado pela disponibilidade de uma API aberta para programação do plano de encaminhamento (ex. OpenFlow), esta dissertação apresenta uma proposta de arquitetura de roteamento IP com separação de planos. Trata-se de uma abordagem que procura combinar o alto desempenho de hardwares de prateleira (commodities) com a flexibilidade de uma pilha de roteamento executada remotamente em computadores de uso geral. O grande desafio é garantir confiabilidade, escalabilidade e desempenho à rede, a partir de um controle remoto e centralizado sobre uma arquitetura que permita maior flexibilidade no mapeamento entre os elementos de controle e encaminhamento. O resultado corresponde a uma nova proposta de roteamento IP com perspectivas promissoras do ponto de vista do custo e da flexibilidade. Com o objetivo de avaliar a arquitetura proposta foi desenvolvido um protótipo com base em uma versão simplificada do modelo. Os resultados da avaliação apresentados nesta dissertação comprovam a viabilidade da arquitetura.

Palavras-chave: Comutação de pacotes, Redes de computadores - Protocolos, Arquitetura de rede de computador, Sistemas distribuídos, Sistemas de transmissão de dados.

Abstract

Today's networking gear follows the model of computer mainframes, where closed source software runs on proprietary hardware. This approach results in expensive solutions and prevents equipment owners to put new ideas into practice. In contrast, recent alternatives of highly flexible software-based routers promise low cost and programmability at the expense of low performance. Motivated by the availability of an open API to control packet forwarding engines (i.e., OpenFlow), we propose a commodity IP routing architecture that combines the line-rate performance of commercial hardware with the flexibility of open source routing stacks (remotely) running on general-purpose computers. The challenge is to ensure reliability, scalability and performance to a network running a remote and centralized control plane architecture that allows a flexible mapping between the control and forwarding elements. The outcome is a novel point in the design space of cost-effective IP routing solutions with far-reaching implications. The initial experimental evaluation of our prototype implementation validates the feasibility of the design.

Keywords: Packet switching, Computer networks - Protocols, Architecture of computer network, distributed systems, System of data transmission.

Agradecimentos

Ao meu orientador e co-orientador, Prof. Dr. Maurício Ferreira Magalhães e Dr. Christian Rodolfo Esteve Rothenberg, sou grato pela orientação, contribuição e empenho que dedicaram para que esta dissertação se tornasse possível.

Ao Dr. Marcos Rogério Salvador pela ajuda na escolha do tema desta dissertação e constante apoio e motivação no desenvolvimento deste trabalho.

Aos colegas Rodrigo Ruffato Denicol e Eder Leao Fernandes pela contribuição na montagem do cenário de avaliação e execução dos testes.

A minha noiva Letícia Andréa Bocchi Silva com quem compartilhei toda essa jornada e que foi fundamental com seu incentivo, motivação e preocupação, com seu amor, alegria e constância e por tudo o que abdicou ao meu lado para que este trabalho pudesse ser realizado.

A minha família por todo afeto, educação e incentivo que me permitiram traçar uma trajetória de sucesso.

A minha futura sogra Roseli Aparecida Amadore Bocchi Silva e meu futuro sogro Gilberto Antônio da Silva pelo apoio e momentos de descontração que me ajudaram a relaxar durante este período de trabalho intenso.

Ao CPqD pelo patrocínio da pesquisa.

À Unicamp pelo ensino de qualidade.

Aos meus pais, irmãos e a minha noiva: esta é uma pequena recompensa por todas as horas que me afastei do vosso convívio, absorvido neste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

ACL - Access Control List

API - Application Programming Interface

ARP - Address Resolution Protocol

ARPA - Advanced Research Projects Agency

BGP - Border Gateway Protocol

CEC - Controlador do Elemento de Controle

CEE - Controlador do Elemento de Encaminhamento

DHT - Distributed Hash Table

DNS - Domain Name System

DROP - Distributed SW ROuter Project

EC - Elemento de Controle

EE - Elemento de Encaminhamento

ER - Elemento de Rede

FIB - Forwarding Information Base

IMP - Interface Message Processor

IS-IS - Intermediate System to Intermediate System Protocol

LPM - Long Prefix Match

LSA - Service Level Agreement

MV - Máquina Virtual

MV - Máquina Virtual

NAT - Network Address Translation

OF - OpenFlow

OSPF - Open Shortest Path First

OVS - Open vSwitch

PC - Plano de Controle

PD - Plano de Dados/Encaminhamento

PDI - Protocolo de Descoberta de Interface

PIF - Physical Interface

QoS - Quality of Service

RF - RouteFlow

RFP - RouteFlow Protocol

RIB - Routing Information Base

RTT - Round-trip Time

RV - Roteador Virtual

SLA - Service Level Agreement

SNMP - Simple Network Management Protocol

TCAM - Ternary Content Addressable Memory

TLV - Type-Length-Value

VIF - Virtual Interface

VR - Virtual Router

Trabalhos Publicados Pelo Autor

1. Marcelo R. Nascimento, Christian E. Rothenberg, Rodrigo R. Denicol, Marcos R. Salvador, Maurício F. Magalhães. "RouteFlow: Roteamento Commodity Sobre Redes Programáveis". RB-RESD - Revista Brasileira de Redes de Computadores e Sistemas Distribuídos, vol. 4, no. 2, Dez 2011 - 1983-4217.

2. Marcelo R. Nascimento, Christian E. Rothenberg, Marcos R. Salvador, Carlos Corrêa, Sidney Lucena and Maurício F. Magalhães. "Virtual Routers as a Service: The RouteFlow Approach Leveraging Software-Defined Networks". In 6th International Conference on Future Internet Technologies 2011 (CFI 11), Seoul, Korea, June 2011.

3. Marcelo R. Nascimento, Christian E. Rothenberg, Rodrigo R. Denicol, Marcos R. Salvador, Maurício F. Magalhães. "RouteFlow: Roteamento Commodity Sobre Redes Programáveis"XXIX Simpósio Brasileiro de Redes de Computadores - SBRC'2011, Campo Grande, MS, Brazil, May 2011.

4. Marcelo R. Nascimento, Christian E. Rothenberg, Rodrigo R. Denicol, Marcos R. Salvador, Maurício F. Magalhães. "QuagFlow: Partnering Quagga with OpenFlow". In Proceedings of the ACM SIGCOMM 2010 Conference on SIGCOMM (New Delhi, India, August 30 - September 03, 2010). SIGCOMM '10. ACM, New York, NY, 441-442.

Eventos e Divulgação

1. Demo no Open Networking Summit 2011 - Stanford (CA), EUA.

2. Ganhador do concurso proposto pelo projeto MyFIRE no Futurecom 2011 como melhor proposta brasileira para executar sobre o tested europeu OFELIA - São Paulo, Brasil.

3. Tutorial no CHANGE & OFELIA Summer School 2011 - Berlin, Alemanha.

4. Demos no Super Computing 2011 - Seattle (WA), EUA:

    - SC11 SCinet Research Sandbox: "RouteFlow: Virtualized IP Routing Services in OpenFlow".
    - Convidados pela Universidade de Indiana a dividir o estande deles para execução da Demo do RouteFlow.

5. Demo no Open Networking Summit 2012 - Santa Clara (CA), EUA.

Capítulo 1

Introdução

A Internet tem revolucionado o mundo da computação e da comunicação como nada visto antes. A invenção do telégrafo, telefone, rádio e computador define o cenário para esta integração de tecnologias sem precedentes. A Internet é atualmente uma rede mundial de comunicação, um mecanismo de disseminação de informação e um meio de colaboração e interação entre indivíduos e seus computadores sem levar em conta a localização geográfica. Essa rede representa um dos maiores exemplos de sucesso dos benefícios do investimento a longo prazo e do compromisso com a pesquisa e desenvolvimento de infraestrutura de informação [3]. Iniciada a partir de uma pesquisa em comutação de pacotes, o governo, a indústria e a academia têm trabalhado em parceria na evolução e no uso desta tecnologia de sucesso.

A rede mundial de computadores tem suas raízes na ARPANET, a primeira rede de comutação de pacotes do mundo e fundada pela ARPA (*Advanced Research Projects Agency*) com o propósito de pesquisa [4]. No seu início, a rede era utilizada apenas por especialistas em computação, engenheiros e cientistas, o que levou ao desenvolvimento de um sistema nada amigável. Não existiam usuários domésticos, escritórios ou empresas conectados à rede, e aqueles que fossem utilizá-la teriam que aprender a operar um sistema extremamente complexo.

A ARPANET foi constituída a partir de um pequeno, mas crescente número de usuários interconectados por comutadores de pacotes conhecidos como IMPs (*Interface Message Processors*), que posteriormente evoluíram para os roteadores IP. Esses IMPs eram conectados por linhas alugadas de baixa velocidade e longa distância em conexões ponto-a-ponto.

Ao passar de décadas essa arquitetura rudimentar de rede sofreu sofisticações até atingir uma arquitetura hierarquicamente distribuída extremamente tolerante a falhas e única em sua capacidade de coordenar um grande número de redes heterogêneas [5]. No entanto, essa distribuição tem seu preço. Notadamente, o monitoramento e o gerenciamento da rede são difíceis e suscetíveis a erros. Além disso, otimizações de desempenho da rede requerem uma visão global do estado da rede e

ações corretivas coordenadas pelos roteadores, o que é dificilmente conseguido em uma arquitetura distribuída.

O desenvolvimento da Internet esteve diretamente relacionado à evolução dos comutadores de pacotes, que aconteceu de forma bastante simples: de sistemas com um único processador, para sistemas multiprocessados altamente especializados. Curiosamente, esse desenvolvimento arquitetural tem acontecido quase que exclusivamente para o plano de dados, enquanto o plano de controle tem se mantido praticamente o mesmo: software rodando sobre um processador de uso geral [5].

A arquitetura atual dos roteadores é claramente baseada em um modelo monolítico e verticalmente integrado. São dispositivos proprietários, fechados e de alto custo, cuja arquitetura básica é concebida da combinação de um *chip* dedicado ao processamento de pacotes, responsável por garantir o alto desempenho, e uma camada de software de controle, que inclui uma imensa pilha de protocolos suportados, visando atender o universo das redes de forma maximizada [6]. No entanto, torna-se evidente a necessidade de especialização da lógica de controle de acordo com cada tipo e objetivo de rede, principalmente corporativa. No âmbito da pesquisa, essa exigência cresce com a necessidade de experimentações de novos protocolos [7]. A ausência de flexibilidade e o alto custo da infraestrutura vigente são barreiras que dificultam o avanço das redes e a inovação [8].

Com o objetivo de suprir tais necessidades, existem alternativas como *software routers* (ex: RouteBricks [9], [10], [11], [12]), que são soluções que utilizam "hardware de prateleira" (*commodity*) com grande capacidade de processamento (ex.: x86) e onde todo processamento de pacotes é realizado no nível de software, o que torna a proposta bastante flexível, porém acarreta em um grande prejuízo de desempenho [13]. Alternativas baseadas em FPGAs apresentam bom desempenho, entretanto, são de alto custo e o tempo de desenvolvimento é bastante elevado. Este último fator é ainda mais crítico nas alternativas baseadas em *network processors* (NP) [14]. Propostas recentes [15] têm explorado também o uso de *graphics processing units* (GPUs) para acelerar o processamento de pacotes nos *software routers*.

Diante deste cenário problemático e engessado, existe um grande interesse da comunidade de pesquisa de redes em rearquitetar a distribuição de funções nas redes IP/Ethernet. Esses esforços podem ser descritos como uma refatoração da funcionalidade dos elementos de rede (roteador e *switch*) em componentes modulares e interfaces bem definidas com o objetivo de permitir maior extensibilidade nas redes. Uma infraestrutura de rede extensível permitiria o desenvolvimento de novos serviços de redes, unificação das funcionalidades dos planos de controle e gerência, especialização de redes para diferentes aplicações e clientes (virtualização de redes) e maior facilidade no gerenciamento da rede. Os esforços neste sentido vão desde APIs abertas para a gerência diferenciada de funcionalidades dos planos de dados e de controle até a definição de uma plataforma (ou sistema operacional) aberta, como por exemplo roteadores baseados em software, *firmware* aberto e plataformas de hardware aber-

tas. Este conceito de elementos de rede programáveis promete acelerar a inovação e implantação de novos serviços/funcionalidades em redes. Porém, ao mesmo tempo, uma maior programação pode exacerbar as tarefas já complexas do gerenciamento de rede.

A centralização do controle e da lógica de rede é uma alternativa à abordagem tradicional para atacar simultaneamente as questões relacionadas aos desafios (i) da definição de planos de controle multi-tecnologia, (ii) da complexidade da gerência, (iii) da evolução gradual da rede em função das demandas dos clientes (por exemplo, acesso a serviços em nuvem) e (iv) da evolução das tecnologias de rede (por exemplo, novos padrões de transmissão).

A principal motivação por trás dessa abordagem é a redução na complexidade por meio da (i) separação da funcionalidade de controle dos dispositivos físicos de transmissão de dados e (ii) da utilização de algoritmos centralizados para o plano de controle de rede em vez de implementações distribuídas do mesmo.

Uma centralização da tomada de decisões sobre as operações permite de forma natural implementações mais simples e fornece uma interface única para o gerenciamento e especificação de políticas no nível global da rede. Esta abordagem promete uma melhoria significativa na gerência de redes de grande porte, onde o controle individual de milhares de elementos de rede é uma tarefa difícil, demorada, cara e propensa a erros de configuração.

Centralização e distribuição de controle de rede é um dos debates clássicos na comunidade de pesquisa e desenvolvimento de tecnologias de rede. Vale a pena notar que a escolha por uma solução ou outra depende de múltiplos fatores, entre eles o domínio de aplicação (ex., multi-domínio, operador único), os modelos de serviço (ex., controle aberto a clientes/terceiros) e o estado da arte das tecnologias (ex., capacidades dos links de transmissão, desempenho/custo de memórias rápidas).

Motivado pelas vantagens do modelo de controle de rede logicamente centralizado e pela disponibilidade de equipamentos de rede programáveis junto a um padrão aberto de API, o trabalho proposto tem como objetivo abordar, simultaneamente, as questões de desempenho, flexibilidade e custo das redes de computadores, conforme esboçado em [16]. A estratégia baseia-se no uso da recente tecnologia de *switches* programáveis [17], o que possibilita mover o plano de controle, antes embarcado no equipamento, para um dispositivo externo [18]. A finalidade, no caso, é permitir a programação remota do plano de encaminhamento. O resultado consiste em uma solução flexível de alto desempenho e comercialmente competitiva, denominada **RouteFlow**, a partir da combinação de recursos disponíveis, tais como: dispositivos programáveis de baixo custo e software embarcado reduzido; pilha de protocolos de roteamento *open source* e servidor de prateleira de alto poder de processamento e baixo custo.

O RouteFlow armazena a lógica de controle dos *switches* programáveis utilizados na infraestrutura de rede através de uma rede virtual composta por máquinas virtuais (MVs), cada uma executando um

código (*engine*) de roteamento de domínio público (*open source*). Esses conjuntos formados por MVs mais *engines* de roteamento são caracterizados como roteadores virtual (RVs), que podem ser interconectados de maneira a formar uma topologia lógica espelhando a topologia de uma rede física correspondente. O ambiente virtual é armazenado em um servidor externo, ou um conjunto deles, que se comunicam com os equipamentos do plano de dados através de um elemento chamado de controlador, cuja responsabilidade é transportar para o plano de encaminhamento as decisões tomadas pelo plano de controle.

A arquitetura proposta procura atender os seguintes requisitos: (a) integridade e sincronização das informações de configuração trocadas entre o ambiente físico e virtual; (b) mecanismo eficiente para detecção de atualizações no plano de controle, com o objetivo de minimizar o atraso da respectiva implantação no plano de dados; (c) isolamento máximo entre as instâncias de roteamento (RVs) de modo que os recursos sejam idealmente compartilhados; (d) gerenciamento dinâmico das conexões entre os RVs e, por último, (e) seleção apropriada do tráfego que deve ser mantido no plano de dados e aquele que necessita ser processado pelo plano de controle.

Deve ser destacado que a arquitetura do RouteFlow promove a efetiva separação entre os planos de controle e de encaminhamento. Essa separação traz inúmeras vantagens com relação ao isolamento de falhas e, também, às questões relacionadas à segurança [19]. Outro ganho está na flexibilidade do mapeamento e operação entre os equipamentos físicos e as instâncias de controle virtuais, que permite múltiplos modos de operação: separação lógica, multiplexação e agregação.

Outros destaques da arquitetura proposta são a de flexibilidade de configuração e implementação das funções de controle (roteamento) nos roteadores virtuais e o desempenho, uma vez que o tráfego de dados é processado em hardware na taxa de transferência das interfaces (*line rate*). Podemos ressaltar ainda a utilização de equipamentos comerciais de prateleira, o que resulta em uma implementação de baixo custo. Desta forma, é possível construir uma rede bastante flexível e que suporte a interoperabilidade com redes legadas.

Os resultados alcançados na avaliação do protótipo sugerem o grande potencial do RouteFlow como solução de roteamento para redes de campus/corporativas com o ganho de flexibilidade e poder de inovação. As questões pertinentes ao desempenho não inviabilizam a proposta, uma vez que otimizações já identificadas e a maturidade do protocolo OpenFlow trarão significativas melhorias de desempenho.

O restante desta dissertação está organizada da seguinte forma: o capítulo 2 discute as tecnologias e trabalhos relacionados a esta tese; o capítulo 3 apresenta a arquitetura e os componentes do RouteFlow; o capítulo 4 descreve a implementação de um protótipo aderente à arquitetura RouteFlow e os testes elaborados para avaliação e, por último, o capítulo 5 apresenta as conclusões e os trabalhos futuros.

Capítulo 2

Tecnologias e Trabalhos Relacionados

2.1 Arquitetura clássica de roteamento

A arquitetura das redes de comunicação está organizada atualmente em três planos funcionais que em conjunto determinam a operação da rede como um todo, são eles: plano de dados (ou encaminhamento), plano de controle e plano de gerência. A Figura 2.1 ilustra essa organização com mais detalhes.

O plano de dados envolve funções que operam em um curto período de tempo e envolvem substancialmente operações de tratamento de pacotes na velocidade em que eles chegam. Por exemplo, o plano de dados realiza o encaminhamento de pacotes, incluindo o *longest-prefix match* que identifica o *link* de destino para cada pacote, bem como listas de controle de acesso (ACLs) que filtram pacotes baseados em seus campos de cabeçalho. O plano de dados também implementa funções como tunelamento, gerenciamento de filas e escalonamento de pacotes. Do ponto de vista físico, o plano de dados é implementado por um hardware dedicado internamente a cada equipamento de rede.

O plano de controle envolve funções que operam em um período de tempo mais longo e fornecem informações requeridas pelo plano de dados de forma automatizada. Esse plano consiste de algoritmos que enviam e analisam pacotes de controle, a partir dos quais inferem parte dos estados da rede. Por exemplo, o plano de controle gera e processa mensagens de atualização do BGP, bem como o *Inter Gateway Protocol* (tal como OSPF), seus avisos de estado de *link* (LSAs) e o algoritmo de menor caminho Dijkstra's. Estes protocolos constroem a tabela RIB, que permite o encaminhamento de pacotes no plano de dados. Funções do plano de controle podem ser orientadas-a-dado, isto é, desencadeadas por eventos do plano de dados (ex.: ARP) ou serem puramente orientadas-a-controle (ex.: OSPF).

O plano de gerência envolve funções que facilitam o monitoramento, o gerenciamento e *troubleshooting* de redes e trabalham em um horizonte de tempo maior ainda que as funções do plano

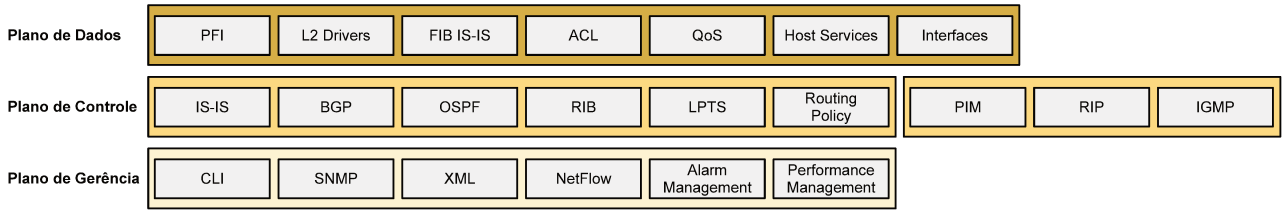


Fig. 2.1: Planos funcionais de uma rede de comunicação.

de controle. O plano de gerência armazena e analisa dados de medições da rede e gera ações de configuração nos elementos de rede, assim como, relatórios (logs) e alarmes para os administradores da rede. Por exemplo, o plano de gerência coleta e combina estatísticas SNMP (em português, Protocolo Simples de Gerência de Rede), registros de tráfego, LSAs do OSPF e fluxo de atualização do BGP. Uma ferramenta que configura a largura do *link* para o OSPF e políticas do BGP para satisfazer as metas de engenharia de tráfego seria parte do plano de gerência. Similarmente, um sistema que analisa medições do tráfego para detectar ataques do tipo DoS e configura ACLs para bloqueio de tráfego malicioso seria também parte do plano de gerência.

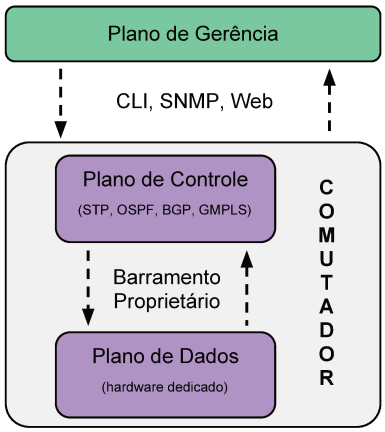


Fig. 2.2: Interação entre os planos funcionais de uma rede.

O acoplamento e a interação entre os planos funcionais da rede podem ser visualizados na Figura 2.2.

Nota-se a forte integração dos planos de controle e dados em um único dispositivo, consequência do modelo de controle distribuído que somado a uma arquitetura monolítica, verticalmente integrada e fechada adotada pelos fabricantes de equipamentos de rede, tornam o sistema engessado e criam uma barreira para inovação.

O plano de controle acessa e programa o plano de dados através de uma API proprietária fornecida pelo fabricante do hardware dedicado ao processamento de pacotes. Desta forma, a programação do software de controle torna-se específica para o plano de dados, amarrando os novos desenvolvimentos

ao mesmo fabricante. O acesso ao equipamento limita-se ao plano de gerência como forma de proteção às informações relacionadas ao projeto/arquitetura dos equipamentos de rede. A interface externa disponibilizada pelo fabricante oculta as camadas de hardware e software e restringe a utilização do equipamento a comandos de configuração para as funcionalidades desenvolvidas. Por consequência, o operador não tem qualquer abertura para alterar a lógica de controle e menos ainda para desenvolver novas funcionalidades. Este modelo acaba por retardar a evolução das redes, uma vez que o desenvolvimento está restrito ao fabricante do equipamento, que não tem condições de acompanhar a necessidade de criação de novos serviços gerada pelo rápido crescimento das redes e criatividade dos usuários no desenvolvimento de ferramentas. Este cenário dificulta a experimentação de novas propostas em ambientes de rede reais.

2.2 Redes definidas por software

O grande sucesso da Internet acarretou na acelerada expansão da infraestrutura de rede e evolução das tecnologias de transmissão, que hoje atingem taxas da ordem de Tbps. No entanto, a arquitetura, representada por um modelo em camadas e pelos protocolos do modelo TCP/IP, não acompanhou essa evolução e tem se arrastado pelos últimos vinte anos. A Internet tornou-se comercial e os equipamentos de rede tornaram-se “caixas pretas”, ou seja, implementações integradas verticalmente baseadas em software fechado sobre hardware proprietário contribuindo, desta forma, para o já reconhecido engessamento da Internet [20].

Diante deste cenário, a comunidade científica tem investido em alternativas para tornar as redes mais flexíveis e abertas à inovação. Em contraste com a arquitetura original da Internet, caracterizada por um plano de controle (PC) distribuído, os avanços na padronização de APIs independentes do fabricante do equipamento permitem mover grande parte da lógica de tomada de decisão dos dispositivos de rede para controladores externos, que podem ser implementados com o uso da tecnologia de servidores comerciais, um recurso abundante, escalável e barato. Essa “lobotomia” da inteligência do equipamento da rede para controladores logicamente centralizados possibilita a definição do comportamento da rede em software não apenas pelos fabricantes do equipamento, mas também por fornecedores ou pelos próprios usuários, como, por exemplo, operadores de rede.

2.2.1 OpenFlow

O OpenFlow é uma proposta criada pela Universidade de Stanford para trazer inovação às redes de computadores [17]. A essência está na definição de um protocolo de comunicação para acesso ao plano de encaminhamento das redes de pacotes, permitindo a execução de um controle externo a partir

de um conjunto padronizado de instruções (API). Para programação do plano de encaminhamento, a proposta explora um recurso comum aos dispositivos de redes, uma tabela de hardware (TCAM) para suporte de ACLs. O processamento dos pacotes ocorre com base em regras e conjuntos de instruções instaladas no hardware de rede por meio do elemento externo, denominado controlador, que pode ser implementado em um servidor comum, conforme ilustrado na Figura 2.3. Hoje, a tecnologia OpenFlow está presente em comutadores, roteadores e pontos de acesso sem fio de diversos fabricantes incluindo os maiores do setor.

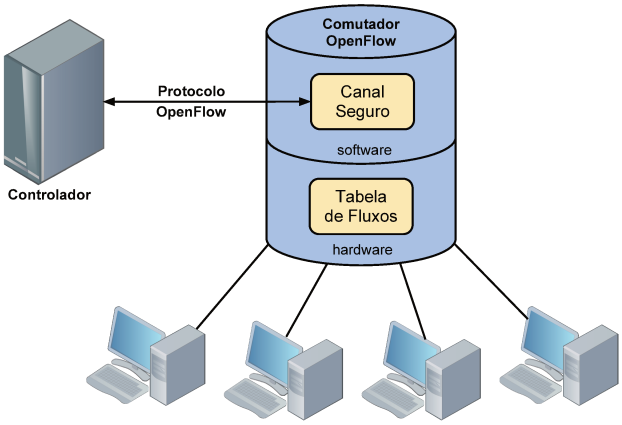


Fig. 2.3: Arquitetura de uma rede OpenFlow.

O paradigma de encaminhamento especificado no OpenFlow está centrado no conceito de fluxo. Um fluxo é definido por uma tupla formada por uma regra a qual está associado um conjunto de instruções. A regra é composta da combinação de campos do cabeçalho do pacote, que compreendem as camadas de enlace, de rede e de transporte, segundo o modelo TCP/IP, conforme a Figura 2.4. O conjunto de instruções associado à regra pode compreender inúmeras ações dentre as quais estão: encaminhar, descartar ou enviar pacotes ao controlador, editar campos do pacote, etc.

in_port	Ethernet			VLAN		IP				TCP/UDP	
	src	dst	type	id	pri	src	dst	proto	ToS	src	dst

Fig. 2.4: Campos de cabeçalho suportados pelo OpenFlow.

Quando um pacote chega no *switch* OpenFlow, a lógica de processamento realiza a identificação e separação dos valores dos campos para serem comparadas com as regras das entradas de fluxo da tabela. Caso o pacote não se enquadre em algum fluxo, ele poderá ser enviado ao controlador de rede, descartado ou continuar para uma próxima tabela, dependendo da configuração do controlador. No caso do pacote que se enquadrar em algum fluxo, a lógica de processamento deve atualizar os

contadores e executar as ações correspondentes ao fluxo. Como comentado anteriormente, dependendo da ação, o pacote poderá ser encaminhado para uma próxima tabela. O diagrama descrevendo o processamento de um pacote quando entra no *pipeline* de um *switch* OpenFlow está representado na Figura 2.5.

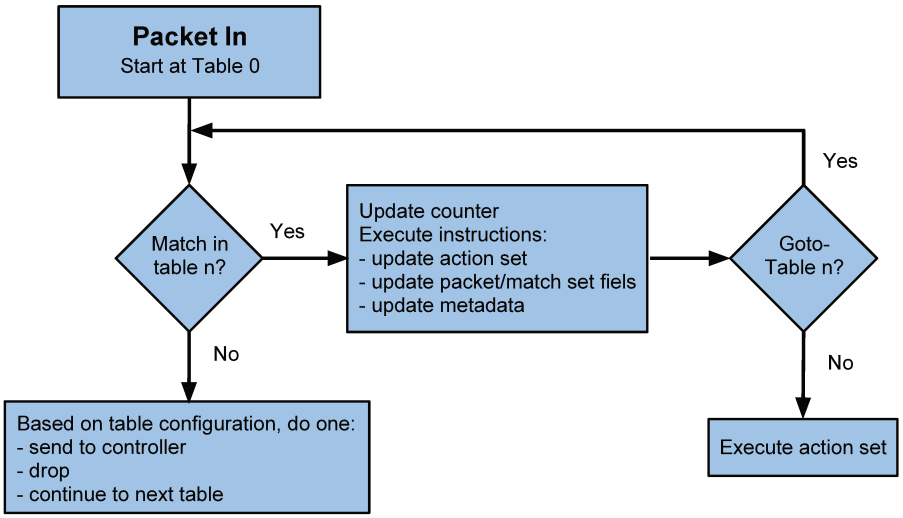


Fig. 2.5: Diagrama de tratamento de um pacote no *pipeline* de um *switch* OpenFlow.

Deve-se enfatizar que a abstração da tabela de fluxos ainda está sujeita a refinamentos com o objetivo de oferecer uma melhor exposição dos recursos do hardware e, nesse caso, permitir a concatenação de várias tabelas já disponíveis, como, por exemplo, tabelas IP/Ethernet/MPLS [21]. Nesse sentido, a contribuição mais importante do paradigma do OpenFlow é a generalização do plano de dados - qualquer modelo de encaminhamento de dados baseado na tomada de decisão fundamentada em algum valor, ou combinação de valores, dos campos de cabeçalho dos pacotes, pode ser suportado.

De forma simplificada, a arquitetura de uma rede programável com OpenFlow pode ser segmentada em quatro componentes principais:

- **Canal seguro:** é o canal de comunicação entre o controlador da rede e os dispositivos programáveis. Como prática de segurança, recomenda-se a utilização de um canal criptografado com o protocolo SSL para evitar ataques de elementos mal-intencionados na rede. Existem outras alternativas como o protocolo de conexão TCP ou ainda socket local, que podem ser utilizadas em ambientes experimentais, onde a segurança não representa um fator de preocupação;

- **Tabela de Fluxos:** é a tabela em hardware utilizada no processamento dos pacotes, cujas entradas contêm as características que representam os fluxos. De maneira geral, utiliza-se a memória TCAM dos equipamentos de rede para implementação desta tabela, pois é um recurso comumente disponível nos hardwares dedicados ao processamento de pacotes e que permite realizar

análises especializadas sobre os cabeçalhos dos pacotes. Cada entrada da tabela é formada por um campo que define a regra a ser aplicada sobre o tráfego e um conjunto de ações a serem executadas sobre os pacotes que se encaixarem na regra. Para cada entrada existe também um contador associado, responsável por armazenar as estatísticas correspondentes ao fluxo;

- **Protocolo OpenFlow:** é um protocolo aberto que define e padroniza a interface externa de programação dos equipamentos com suporte ao OpenFlow. O protocolo especifica mensagens de controle para monitoramento do canal, de requisição de informações do dispositivo, de comandos para programação dos elementos de encaminhamento, de eventos de rede, de transmissão de pacotes entre outras;

- **Controlador:** é o software responsável pela tomada de decisões com base nos eventos de rede e pela manipulação das entradas das tabelas de fluxos de acordo com o objetivo desejado. O controlador exerce o papel de uma camada de abstração sobre a infraestrutura física de rede, facilitando o desenvolvimento de aplicações e serviços que gerenciam os fluxos na rede. Este modelo assemelha-se ao computacional, onde o sistema operacional cria uma abstração dos recursos físicos e disponibiliza uma interface em alto nível para o desenvolvimento de aplicações. O controlador de rede segue a mesma abordagem, o que facilita a criação de novas aplicações e, por consequência, aumenta a agilidade e o poder de inovação sobre as redes de computadores.

2.2.2 Sistema Operacional de Rede

No contexto de redes definidas por software, é comum a utilização de um software centralizado (Figura 2.6) responsável por prover uma camada de abstração dos recursos físicos da rede, coordenar a execução das aplicações, fornecer um conjunto de bibliotecas para auxiliar no desenvolvimento de aplicações, dentre outras funções. Este conceito se equivale ao sistema operacional dos sistemas computacionais e passa a ter também um papel fundamental nas redes definidas por software.

O NOX [18] foi a primeira iniciativa independente de um controlador para redes OpenFlow e nasceu com o objetivo de prover uma plataforma simples para o desenvolvimento de aplicações de rede utilizando as linguagens C++ e Python. O *framework* ainda traz um conjunto de aplicações auxiliares com intuito de disponibilizar ao desenvolvedor algumas tarefas comuns e muito úteis na criação de novas aplicações, como o monitoramento do estado dos enlaces, descoberta de topologia, autenticação dos terminais de rede, entre outras.

O controlador trabalha com um modelo de comunicação, entre contextos de execução, orientado a eventos. As aplicações se registram nos eventos de interesse e passam então a receber as informações pertinentes à sua lógica de controle. As aplicações podem também gerar seus próprios eventos. As aplicações de controle têm completo acesso às tabelas de fluxos dos equipamentos do plano de dados,

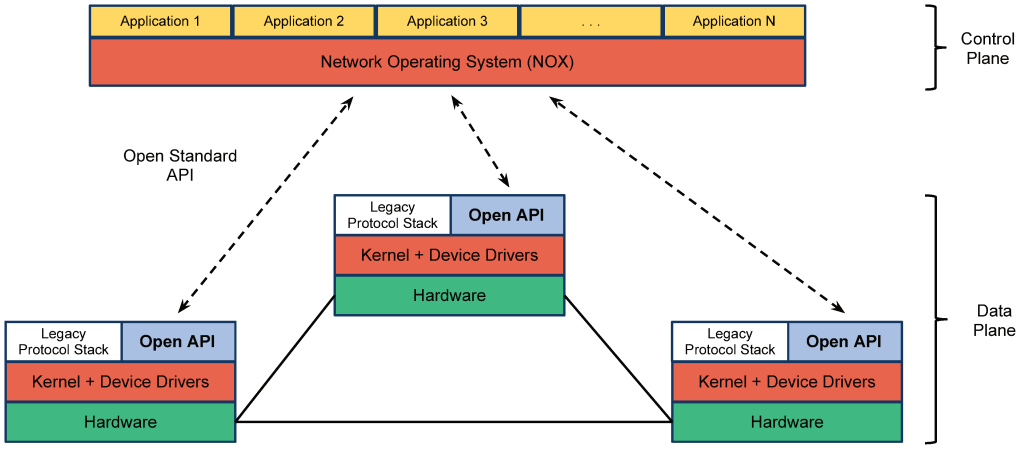


Fig. 2.6: Modelo de rede definida por software com sistema operacional de rede.

dessa forma é possível criar, apagar e modificar os fluxos, alterando assim o encaminhamento do tráfego dentro da rede. O gerenciamento dos fluxos é de completa responsabilidade das aplicações, ou seja, o NOX não segmenta os recursos da infraestrutura para manter um isolamento entre o domínio de atuação das aplicações. Portanto, uma aplicação pode alterar o comportamento da rede imposto por outra.

Depois do NOX e a partir do amadurecimento do padrão OpenFlow, outras propostas e soluções de sistemas operacionais para redes OpenFlow têm surgido. Algumas delas são: Beacon [22], Onix [23], Trema [24], Floodlight [25].

2.3 Virtualização de redes

O conceito de virtualização é comumente conhecido como a habilidade de compartilhar recursos físicos de um mesmo sistema computacional entre diversas instâncias independentes e isoladas de sistemas operacionais. Esta abordagem é normalmente aplicada quando se tem um grande poder de processamento local aliado à necessidade de executar tarefas independentes. Portanto, multiplexar essas tarefas em um mesmo equipamento reduz o uso do espaço físico e o custo de manutenção de hardware, além de otimizar o aproveitamento dos recursos ociosos.

Integrando o paradigma de redes definidas por software à tecnologia de virtualização computacional é possível estender o conceito de virtualização para as redes de comunicação, herdando as inúmeras vantagens de sua aplicação nos sistemas computacionais. Um exemplo ilustrativo de virtualização de redes pode ser visto na Figura 2.7.

A virtualização de redes pode diminuir a ossificação da Internet atual e estimular inovação com a possibilidade de diversas arquiteturas de rede coexistirem sobre um mesmo substrato físico compar-

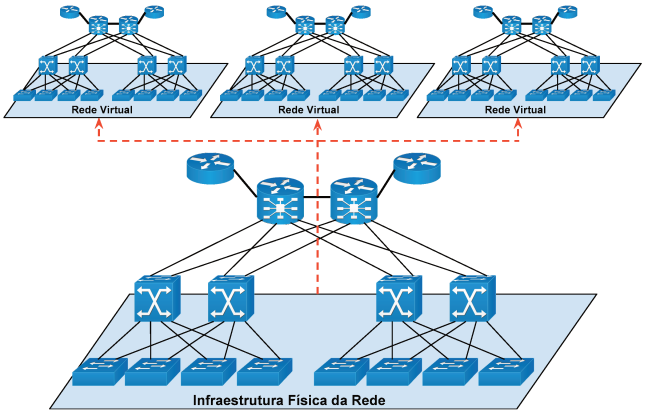


Fig. 2.7: Conceito de virtualização de redes.

tilhado [26][27].

O conceito de diversificação através da separação entre política e mecanismo é um princípio bem testado na literatura computacional. Propostas similares têm sido sugeridas para redes virtuais [27][28]. Neste caso, os tradicionais provedores de serviços de Internet (IPSs) foram divididos em dois: provedores de infraestrutura responsáveis pelo gerenciamento da infraestrutura física, e provedores de serviços, responsáveis pela criação de redes virtuais através da agregação de recursos de múltiplos provedores de infraestrutura e pelo oferecimento de serviços aos usuários finais. Tal modelo, ilustrado na Figura 2.8, irá promover o desenvolvimento de múltiplas arquiteturas de redes heterogêneas que não herdarão as limitações da Internet existente [29].

O objetivo global de permitir múltiplas redes virtuais heterogêneas pode ser subdivido em objetivos menores: flexibilidade, gerenciabilidade, escalabilidade, isolamento, segurança, programabilidade, heterogeneidade, experimentação e suporte a redes legadas. Estes objetivos provêem um guia para o desenvolvimento de protocolos ou algoritmos para virtualização de redes.

2.3.1 Comutador/Switch Virtual

A virtualização de redes implica na construção de topologias lógicas a partir de nós virtuais que executam os algoritmos de controle sobre uma infraestrutura compartilhada. Cada nó da rede lógica é representado por uma instância de máquina virtual. Portanto, a construção da malha de rede é realizada a partir da inter-conexão de MVs, que podem ser instanciadas em um mesmo servidor ou estar isoladas fisicamente.

No entanto, criar uma rede virtual através da interconexão de MVs não é uma tarefa simples e representa grandes desafios, considerando o perfil dinâmico deste ambiente. Um dos recursos interessantes da virtualização é a migração de MVs, no entanto o protocolo IP clássico não suporta mobilidade. Redes com endereçamento plano como Ethernet não escalam sem segmentação em múl-

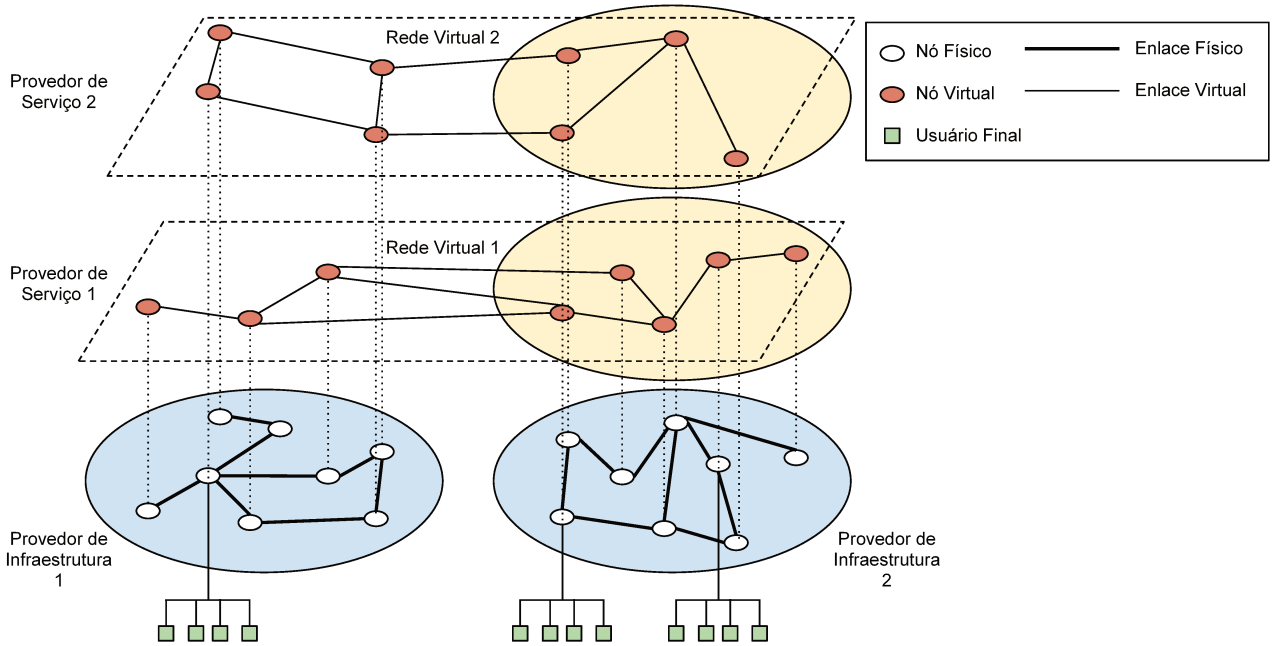


Fig. 2.8: Arquitetura de Redes Virtuais.

tiplas sub-redes, o que inviabiliza topologias virtuais com muitos nós. Outro ponto importante é o fato que estados de configuração de rede tais como isolamento, qualidade de serviço e políticas de segurança não se adaptam dinamicamente às mudanças de topologia.

Para atacar essas limitações, uma nova camada de acesso de rede está surgindo para prover inter- e intra-conectividade das máquinas virtuais [1]. Ao passo que a tecnologia de virtualização acarreta requisitos mais rigorosos, ela traz funcionalidades que facilitam o gerenciamento das redes. Por exemplo, a camada de virtualização (*Hypervisor*) pode prover informações sobre a dinâmica do espaço virtual simplificando o tratamento da mobilidade dos nós virtuais.

Neste contexto, o Open vSwitch (OVS) surge como uma ferramenta para atuar na camada de acesso de rede e prover conectividade em ambientes virtualizados, como ilustrado na Figura 2.9. A partir dos *switches* virtuais torna-se possível interconectar as MVs através das interfaces virtuais (VIFs) e permitir o acesso externo, que é conseguido através da associação das interfaces reais (PIFs) aos *switches* virtuais.

OVS é um *switch* virtual multi-camadas desenvolvido sobre o conceito de flexibilidade e portabilidade. Ele suporta funcionalidades de um *switch* de borda avançado: visibilidade de fluxo de tráfego com NetFlow, sFlow e *Port mirroring*[1] ; granularidade fina de ACL (*Acess Control Lists*) e políticas de QoS (*Qualidade de Serviço* baseadas em L2, L3 e L4; e suporte a controle centralizado com

[1] *Port mirroring* é a capacidade de um *switch* de enviar uma cópia dos pacotes de uma porta de rede para uma outra porta. Esta funcionalidade é geralmente utilizada no monitoramento do tráfego de rede.

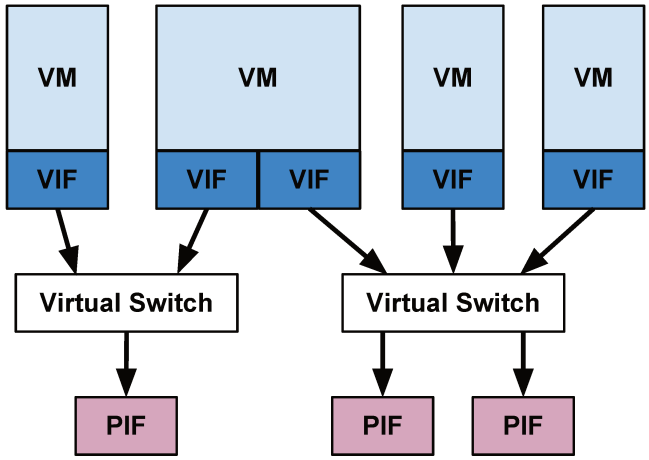


Fig. 2.9: Os *switches* virtuais conectam as interfaces virtuais às interfaces físicas. [1]

OpenFlow.

2.3.2 FlowVisor

Dentro do contexto de virtualização de redes, o OpenFlow provê uma interface padrão para programação do plano de encaminhamento, possibilitando a execução da lógica de controle em um plano externo. Portanto, o padrão OpenFlow contribui para aplicação do conceito de virtualização de redes, apesar de não promover a virtualização propriamente dita.

Uma funcionalidade do OpenFlow também alinhada com o paradigma de virtualização de redes é a possibilidade do compartilhamento dos recursos de um mesmo dispositivo de rede. Através da consideração das portas de um *switch* OpenFlow como recurso, o padrão permite criar instâncias isoladas de *datapaths*, cada qual utilizando portas diferentes do equipamento e tabelas isoladas. Enquanto isso, o controlador de rede não enxerga o dispositivo físico, mas sim os *datapaths*.

Apesar do protocolo OpenFlow não permitir a segmentação da rede em instâncias isoladas, uma camada intermediária entre os *switches* OpenFlow e o controlador de rede pode promover essa funcionalidade. Dessa forma, múltiplos controladores podem compartilhar uma mesma infraestrutura de dispositivos programáveis a partir de políticas bem definidas. Essa camada é implementada pelo FlowVisor [2].

Esta camada atua de forma transparente entre os elementos dos planos de controle e de dados, interceptando as mensagens OpenFlow e as reencaminhando conforme as políticas definidas para cada fatia da rede (*slice*). A operação está demonstrada na Figura 2.10.

O FlowVisor objetiva o compartilhamento da infraestrutura programável reservando para cada fatia da rede recursos como largura de banda, topologia, tipo de tráfego, processamento e tabelas de encaminhamento. Todas essas características são definidas por um sistema flexível de políticas.

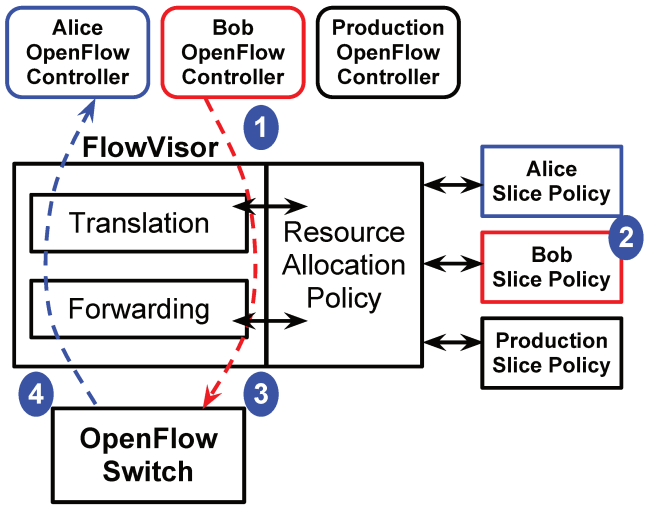


Fig. 2.10: O FlowVisor intercepta as mensagens OpenFlow dos controladores (1) e, usando a política de fatia do usuário (2), re-escreve transparentemente (3) a mensagem para controlar somente uma fatia da rede. As mensagens dos switches (4) são encaminhadas somente para o controlador definido pela política da fatia de rede. [2]

Portanto, dentro de uma rede OpenFlow, o FlowVisor é responsável por virtualizar os recursos de forma transparente para os elementos da rede e garantir isolamento entre os recursos associados às topologias lógicas.

2.4 Propostas

A proposta apresentada nesta dissertação (RouteFlow) alinha-se com a tendência de centralizar o controle da rede unificando a informação do estado da rede e desacoplando-a (encaminhamento e configuração) dos elementos de hardware [30]. Esta abordagem reduz a complexidade de software em grande parte dos elementos de encaminhamento, aumentando assim a confiabilidade das redes [19]. Outro benefício está na flexibilidade de modificar o plano de controle, facilitando a adição e criação de novos serviços. Existe um número significativo de trabalhos recentes que aderem a esse paradigma.

2.4.1 Tesseract

O Tesseract [31] apresenta-se como um sistema experimental que permite o controle direto de um equipamento de rede que está sob um único domínio administrativo. A flexibilidade herdada do Modelo 4D [32], sobre o qual o sistema foi construído, facilita o desenvolvimento de aplicações e serviços, promovendo inovação. Outro destaque está na simplicidade dos dispositivos de rede, que

precisam implementar apenas três funções distribuídas: serviço de descoberta de vizinhos; serviço de disseminação e serviço de configuração de nó.

A proposta baseia-se em uma arquitetura de rede com controle logicamente centralizado, cujo objetivo é redesenhar os diferentes aspectos de controle e gestão de redes.

A partir do modelo de arquitetura 4D, podem ser identificados quatro planos funcionais de uma rede de comunicações (como ilustrado na Figura 2.11): 1) plano de decisão, que é responsável por criar a configuração da rede (por ex..calculando as tabelas FIB para cada elemento de rede); 2) plano de disseminação, que (i) coleta informações sobre o estado da rede (por ex.. link up/down), (ii) repassa para o plano de decisão, e (iii) distribui as decisões para os elementos de rede; 3) plano de descoberta, que permite aos dispositivos descobrir os vizinhos diretamente interligados; e 4) plano de dados, que é responsável pelo encaminhamento do tráfego da rede.

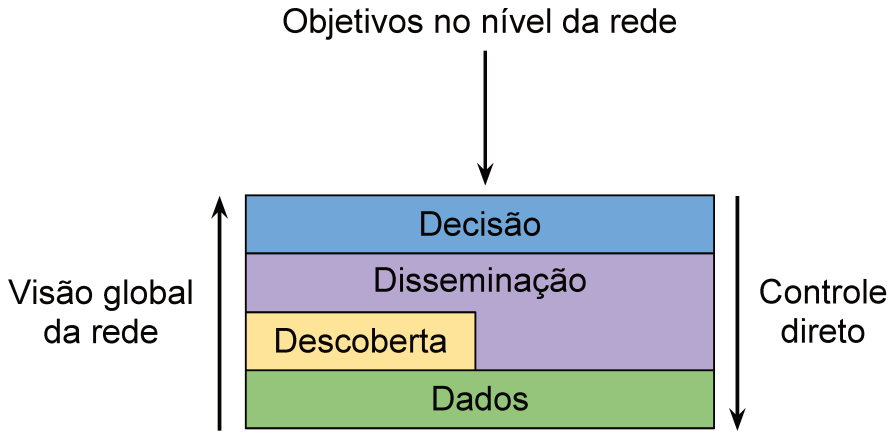


Fig. 2.11: Representação do paradigma do modelo 4D.

A realização prática desses princípios cobre três aspetos fundamentais:

1. Desacoplamento da lógica de decisão: Separação das funções complexas do plano de controle (por ex., processamento dos protocolos de roteamento) dos elementos de encaminhamento de pacotes no plano de dados.

2. Elementos Controladores: A implementação das funções de processamento do plano de controle é feita em servidores externos dedicados.

3. Interface padrão: Definição de um protocolo padrão para as comunicações entre os elementos de controle (servidores/controladores) e os elementos de encaminhamento (switch/roteador/access point/comutador óptico).

É importante notar que o paradigma 4D introduz apenas um modelo que redistribui as funcionalidades dos planos de controle e gerência em três novas áreas de atuação. Portanto, sua aplicação

permite múltiplas abordagens, sendo o protótipo Tesseract simplesmente uma forma de implementar o modelo dentro do universo de soluções possíveis.

2.4.2 SoftRouter

A proposta do SoftRouter [33] alinha-se ao paradigma da separação entre os planos de controle e encaminhamento das redes de comutação de pacotes com o propósito de reduzir a complexidade na adição de novos serviços/funcionalidades na rede. Nesta arquitetura, todas as funções do plano de controle são implementadas em servidores de propósito geral chamados de elementos de controle (ECs), que podem estar a múltiplos saltos dos elementos de encaminhamento (EEs). A associação entre ECs e EEs resulta no elemento de rede (ER) e é realizada de forma dinâmica, tornando a definição do ER bastante flexível.

Em alto nível, um ER é um agrupamento lógico das portas de rede e os respectivos ECs que irão controlá-las. O administrador da rede pode configurar diferentes restrições para agrupamento das portas com os ECs, resultando em um conjunto de ERs permitidos.

Do ponto de vista da infraestrutura física, uma rede SoftRouter é formada por nós interconectados por enlaces. Existem dois tipos de nós: os elementos de encaminhamento e os elementos de controle. Um EE é similar a um roteador tradicional em termos de construção: pode conter múltiplos cartões de linha [2] e uma matriz de interconexão para rotear o tráfego entre os cartões. A principal diferença para um roteador tradicional é que o EE não tem uma sofisticada lógica de controle executando localmente, ao contrário, a lógica de controle executa no EC.

No modelo SoftRouter, os controles de roteamento dos ERs estão desagregados dos EEs e os protocolos executam nos ECs. A amarração entre um EE e um EC implica que o EC irá executar funções particulares de controle para o EE. Como múltiplos protocolos podem ser necessários para a operação de um EE, o EE pode associar-se e mais de um EC. Um exemplo de rede SoftRouter está ilustrado na Figura 2.12.

2.4.3 FIBIUM

Com objetivos semelhantes ao trabalho proposto nesta dissertação, o FIBIUM [34] busca preencher o espaço entre roteadores comerciais de alto custo e roteadores puramente em software, que sofrem com o problema de baixo desempenho. A proposta apresenta uma arquitetura alternativa de roteador que combina um encaminhamento em software, rodando em um computador *commodity*, com um hardware programável responsável pela maior parte do encaminhamento de pacotes. O foco

[2] Um cartão de linha, ou *line-card*, refere-se a um conjunto de interfaces de rede. É um padrão utilizado em sistemas que seguem o modelo de chassi.

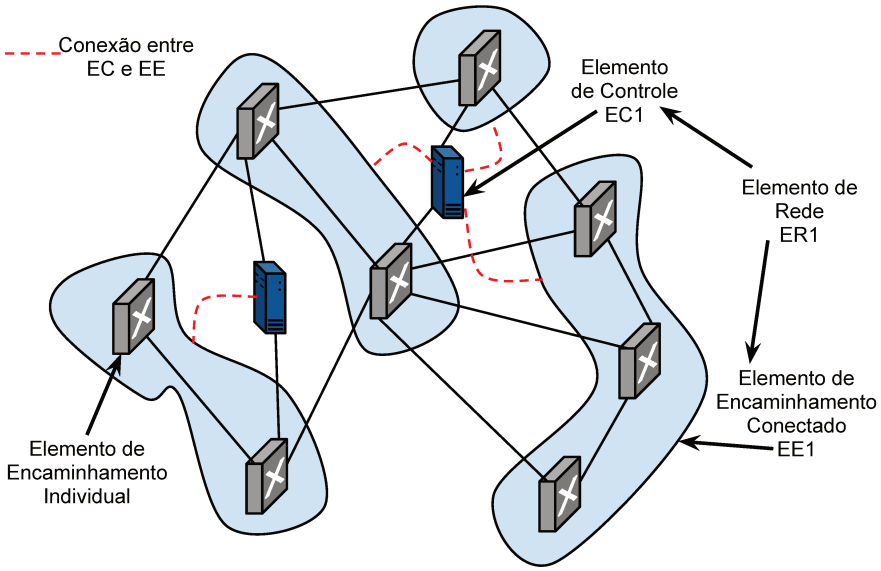


Fig. 2.12: Exemplo de rede SoftRouter.

do trabalho é a otimização do número de entradas instaladas em hardware baseada na observação da popularidade das mesmas.

Roteadores puramente implementados em software rodam em computadores de uso geral, portanto seus maiores inconvenientes incluem (a) o desempenho baixo no encaminhamento de pacotes e (b) a limitação da densidade de portas. Neste contexto, o FIBIUM propõe um novo modo de juntar um roteador em software com um *switch* Ethernet, como pode ser visto na Figura 2.13.

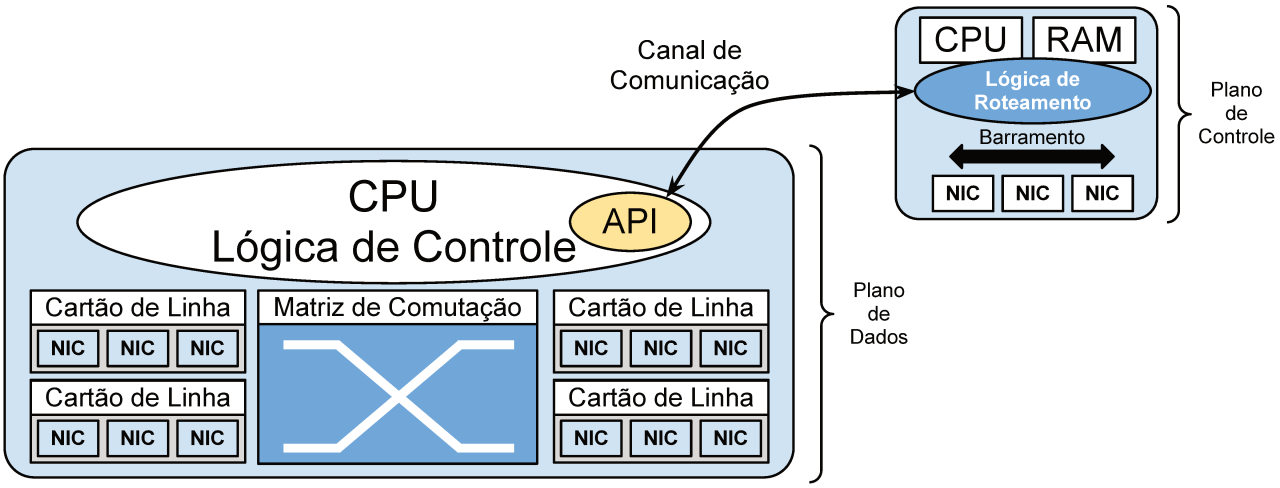


Fig. 2.13: Arquitetura proposta pelo FIBIUM.

O *switch* provê um encaminhamento acelerado em hardware assim como uma maior densidade de portas com relação aos computadores de uso geral. Este hardware é utilizado para o encaminhamento

da maior parte do tráfego. O computador é utilizado não somente como controlador, mas também para monitorar o tráfego e manter o canal de comunicação com o *switch* para incluir na FIB os prefixos de destinos mais populares. Além disso, ele provê um encaminhamento em software para o tráfego restante que não foi tratado pelo *switch*. O protótipo do FIBIUM pode, em princípio, tratar o volume de tráfego de uma rede de agregação de um provedor.

2.4.4 DROP

Outro trabalho que também visa paradigmas avançados e flexíveis sobre roteadores baseados em software é o DROP(*Distributed SW ROuter Project*) [35]. Este trabalho fundamenta-se nas diretrizes do padrão IETF ForCES [36] e tem como foco principal a criação de nós lógicos de rede através da agregação de múltiplos elementos de encaminhamento e uma clara separação entre os elementos de controle e de encaminhamento. O DROP provê uma solução arquitetural interna capaz de esconder a complexidade da distribuição dos nós de rede de uma maneria autônoma e oferecer uma interface única de controle e gerenciamento para os usuários. Tal arquitetura, que pode ser visualizada na Figura 2.14, está apoiada nos seguintes conceitos do padrão IETF ForCES:

- **Elementos de Controle (EC):** elementos de rede dedicados às funcionalidades de controle e aptos a tomar decisões de roteamento e gerenciamento através de uma *engine* de roteamento;
- **Elementos de Encaminhamento (EE):** elementos de rede destinados às operações do plano de dados, que são ajustados e adaptados especificamente para desempenhar o processo de encaminhamento IP com eficiência;
- **Rede Interna/Privada:** uma rede para conexão de todos os elementos de controle e encaminhamento. Esta rede é completamente dedicada à conectividade interna dos elementos distribuídos e não pode ser alcançada por *hosts* externos.

Na implementação da arquitetura proposta foram desenvolvidas duas aplicações principais em software, chamadas de Controlador do Elemento de Controle (CEC) e Controlador do Elemento de Encaminhamento (CEE). O principal objetivo dessas aplicações é gerenciar as interações entre os planos de controle e dados permitindo a distribuição das funcionalidades do roteamento IP de um modo automatizado e eficiente.

De modo geral, as propostas apresentadas nesta seção alinham-se com o paradigma de redes definidas por software, através do qual promove-se o desacoplamento entre os planos de encaminhamento e controle. Esse desacoplamento tem como objetivo principal trazer flexibilidade e inovação para a lógica de controle das redes de computadores. Neste contexto, o FIBIUM segue uma abordagem

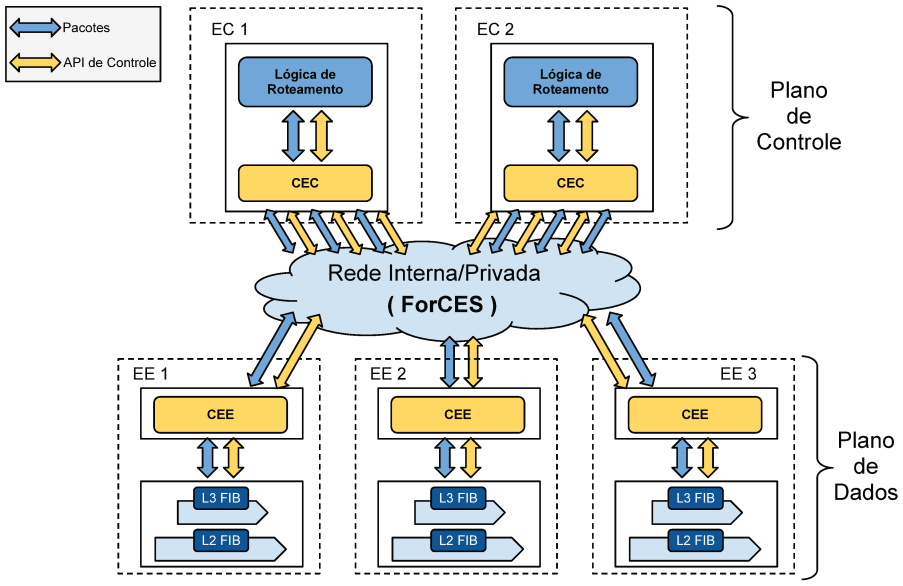


Fig. 2.14: Proposta de arquitetura do DROP.

mais prática com foco em desempenho e escalabilidade do roteamento, enquanto as demais propostas exploram aspectos mais teóricos e conceituais de SDN.

A partir de uma abordagem pragmática, o presente trabalho tem o objetivo de executar lógicas de controle legadas para roteamento IP em redes SDN sem promover alterações nos protocolos. A arquitetura proposta, que será apresentada no próximo capítulo, foi concebida para operar com topologias de rede virtuais no plano de controle, permitindo assim explorar os benefícios da virtualização de redes. Vale ressaltar que apesar do modelo arquitetural abrir caminho para a exploração de novas abordagens no plano de gerência, esse não é o foco do trabalho.

Capítulo 3

RouteFlow

O trabalho apresentado nessa dissertação visa a concepção de uma nova arquitetura de roteamento IP denominada RouteFlow. A arquitetura caracteriza-se pela centralização da lógica de controle, que executa externamente (ou remotamente) ao dispositivo de rede, resultando no desacoplamento entre os planos de encaminhamento e controle.

Como consequência da separação dos planos, as redes IP tornam-se mais flexíveis pela facilidade da adição, remoção e especialização dos protocolos e algoritmos de controle. Outras implicações da arquitetura RouteFlow para as redes IP estão no baixo custo da infraestrutura associada, no gerenciamento flexível e, principalmente, no alto desempenho de comutação, devido ao processamento de pacotes por um hardware dedicado.

O RouteFlow move a lógica de controle dos equipamentos de rede, até então embarcada, para um controlador de rede remoto, cuja responsabilidade é tomar decisões de encaminhamento e aplicá-las aos elementos do plano de encaminhamento através de uma interface de programação. Portanto, um pré-requisito da proposta é a existência de elementos de rede (hardwares) no plano de encaminhamento que ofereçam uma interface de programação de aplicação (API). Um exemplo de API para programação dos dispositivos de rede é o padrão OpenFlow.

A programação dos elementos de rede baseia-se nas tomadas de decisão do plano de roteamento o qual, por sua vez, é composto por roteadores virtuais (RVs) conectados de forma a construir topologias lógicas/virtuais de roteamento. Neste sentido, a arquitetura proposta possibilita que a topologia física da rede seja utilizada como um recurso compartilhado pelas topologias virtuais. A Figura 3.1 ilustra uma visão geral da arquitetura RouteFlow.

No plano virtual, as topologias lógicas são construídas a partir de roteadores virtuais construídos a partir de máquinas virtuais. Cada RV roda uma *engine* de roteamento padrão, ou seja, o mesmo software de controle (protocolos) que estaria embarcado em um roteador. As interfaces dos RVs representam as portas do roteador físico através das quais os pacotes dos protocolos são enviados

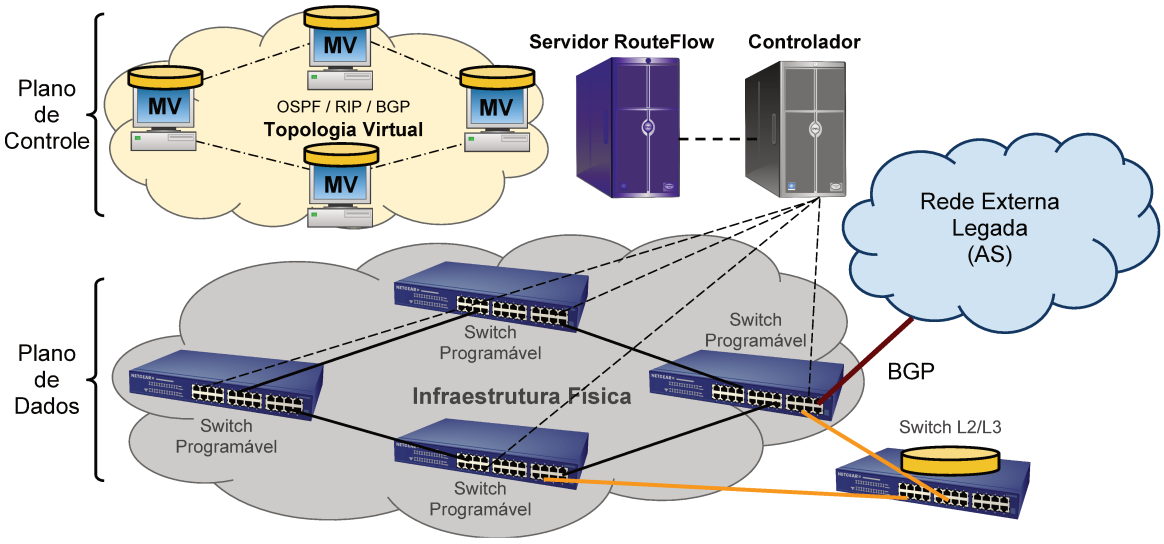


Fig. 3.1: Visão geral do RouteFlow.

e recebidos pela lógica de controle. Ainda, a arquitetura proposta permite configurar o tráfego de controle para que fique confinado na topologia lógica (entre os RVs) ou para que seja enviado ao plano de encaminhamento. A manutenção dos pacotes dos protocolos confinados na topologia virtual torna-se interessante para efetuar a separação física entre os planos de controle e dados. As decisões tomadas pelas *engines* de rotamento dentro dos RVs são enviadas para o controlador, que as instala nos respectivos elementos físicos da rede. Além da instalação de rotas no plano de dados, o controlador faz o gerenciamento dos roteadores virtuais bem como a conexão entre eles, isto é, a construção das topologias lógicas. Um ponto de destaque da solução refere-se à não modificação tanto do sistema operacional como da *engine* de roteamento, com isso evita-se amarrar a arquitetura a uma plataforma específica. Portanto, o RouteFlow fica independente da pilha de protocolos, da distribuição do sistema operacional Linux e da tecnologia de virtualização utilizada. Esse aspecto possibilita explorar o universo de opções de aplicações de código aberto.

Apesar do controle estar fisicamente centralizado, ele continua distribuído logicamente na topologia virtual. Desta forma, não faz-se necessária qualquer alteração dos protocolos de roteamento existentes. Além disso, a solução pode tornar-se mais escalável no futuro com a distribuição de processamento e armazenamento através do uso de nuvens computacionais (*Cloud Computing*).

Não é novidade a questão da separação dos planos de controle e dados e seus benefícios para as redes [19]. Com esse desacoplamento, a evolução dos planos de controle e dados pode ocorrer de maneira independente de acordo com a necessidade específica de cada um, propiciando maior flexibilidade e dinamismo no avanço das tecnologias de redes. Com a lógica de roteamento externa ao equipamento, a infraestrutura de encaminhamento pode ser composta por equipamentos cujo software embarcado seja bastante reduzido. Essa simplificação resulta na redução de custos dos elementos de

encaminhamento e ainda permite a utilização de protocolos de código aberto, o que também acarreta na diminuição de custo.

Uma outra vantagem é o ganho de flexibilidade na especialização da pilha de software, que agora pode ser facilmente modificada com a adição de novos protocolos, correção de erros (bugs) e otimizações. Outro ganho importante refere-se à realização de modificações de forma pontual para a rede como um todo, devido justamente ao fato do controle ser centralizado.

Com o RF existe a possibilidade de virtualização da rede por meio da criação de mais de uma topologia lógica de roteamento operando sobre a mesma infraestrutura física, permitindo explorar o conceito de plataforma como serviço (PaaS) e otimizar a utilização dos recursos da rede.

Contudo, a arquitetura proposta apresenta grandes desafios do ponto de vista de escalabilidade. Alguns destes estão relacionados ao grau de maturidade das tecnologias ligadas ao paradigma de redes definidas por software, que são bastante recentes. Outros dizem respeito ao modelo arquitetônico de controle centralizado. Contornar essas questões exige um estudo mais aprofundado tanto da tecnologia OpenFlow quanto de processamento paralelo, que não foram abordados neste trabalho.

O restante deste capítulo está organizada da seguinte forma: a seção 3.1 apresenta a arquitetura proposta; a seção 3.2 descreve os componentes do RouteFlow; a seção 3.3 apresenta as mensagens do protocolo interno à arquitetura RouteFlow; a seção 3.4 discute os mecanismos utilizados pelos componentes para troca de informação; a seção 3.5 descreve as opções de configuração para tratamento do tráfego de controle e, por último, a seção 3.6 apresenta os casos de uso e modos de operação da arquitetura RouteFlow.

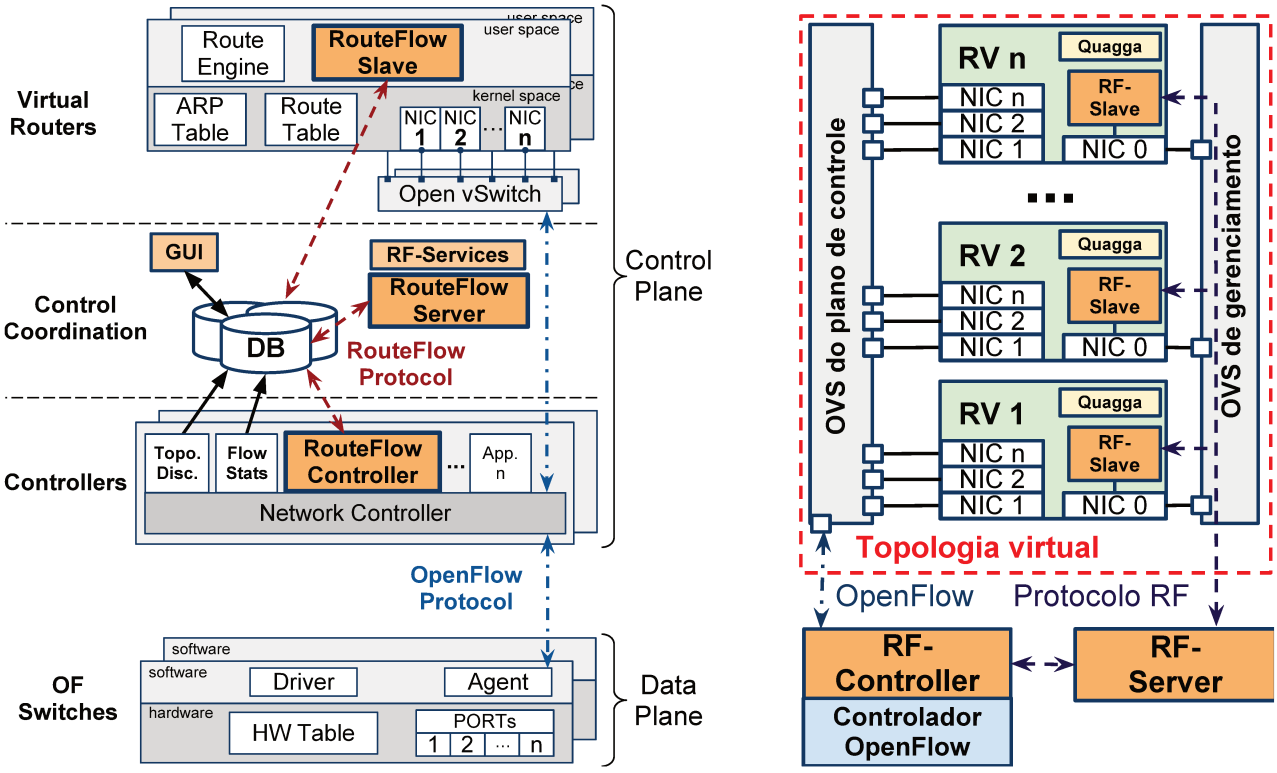
3.1 Arquitetura

3.1.1 Modelo

O modelo de arquitetura de rede para roteamento IP proposto neste trabalho foi elaborado com o propósito de construir um sistema robusto que atenda aos requisitos apresentados na Seção 3.1.2. A estratégia utilizada na concepção da arquitetura seguiu os seguintes conceitos: (i) agrupar as informações do estado da rede em uma base de dados centralizada, o que simplificaria a comunicação entre os componentes e aumentaria a redundância/confiabilidade da informação; (ii) implementar uma camada de abstração entre os planos de dados e controle de forma que a arquitetura interna seja transparente para a lógica de controle; (ii) manter a estrutura do plano de controle dentro do paradigma de sistemas distribuídos, o que possibilitaria o reuso dos protocolos legados sem qualquer alteração e; (iv) criar uma rede flexível e dinâmica no plano virtual de controle. O objetivo principal do trabalho é mover o plano de controle, que tradicionalmente executa embarcado no equipamento de rede, para uma

entidade externa/remota ao plano de dados. Para que essa separação ocorra de forma transparente é necessário que tanto o plano de dados como o de controle não tenham conhecimento desta mudança de arquitetura e continuem operando sem qualquer alteração. Neste sentido, foi criada uma camada intermediária para atuar na interconexão entre os planos de dados e controle de maneira a abstrair a separação.

Essa camada de indireção, responsável por tornar a mudança transparente aos planos de dados e controle, foi construída a partir da composição de três entidades: o RouteFlow-Slave, o RouteFlow-Controller e o RouteFlow-Server. A comunicação entre as entidades do sistema ocorre por meio de um protocolo que define mensagens de comandos e eventos para transmissão de informação sobre as alterações de estado do plano de controle para o plano de dados e vice-versa. O desenho da arquitetura está ilustrado na Figura 3.2. Cada um dos módulos será detalhado na Seção 3.2 e o protocolo na Seção 3.3.



(a) Visão global da arquitetura proposta.

(b) Detalhamento do plano virtual.

Fig. 3.2: Desenho da arquitetura proposta.

Cada roteador físico (*switch* programável) tem sua lógica de controle executando em uma instância de roteador virtual. Os RVs trabalham em contextos isolados e independentes entre si. Dessa forma é possível minimizar a interferência entre as instâncias das lógicas de controle em termos de consumo de recursos computacionais e ainda isolar falhas. Contudo, o desempenho do plano vir-

tual está intimamente ligado ao virtualizador utilizado. Estão disponíveis uma grande variedade de ferramentas para virtualização e a escolha de qual utilizar é uma questão bastante pertinente ao desempenho do sistema. O trabalho apresentado em [37] faz uma análise comparativa entre alguns virtualizadores.

O canal para troca de tráfego entre os planos de controle e encaminhamento foi pensado com foco na otimização de desempenho e flexibilidade. Nessa direção, adotou-se um *switch* programável implementado em software para atuar como ponto de interconexão entre os RVs e estabelecer uma ponte com o plano de encaminhamento. A funcionalidade de programabilidade do *switch* virtual possibilita a construção de topologias entre os RVs no plano virtual, que podem ou não ser iguais à topologia física de rede. Ainda, a programação do *switch* virtual permite configurar o destino dos pacotes com base em fluxos, tornando possível o confinamento do tráfego de controle (pacotes de protocolos) na topologia virtual. Essas características dão abertura para explorar os conceitos de virtualização de redes e roteamento como serviço, que serão abordados na Seção 3.6.

A arquitetura proposta utiliza uma base de dados central para o armazenamento de informações persistentes para eventual compartilhamento entre as entidades do sistema. A estratégia de centralizar a informação evita problemas de sincronismo e, ao mesmo tempo, facilita a replicação da informação com o propósito de aumentar a confiabilidade do sistema. Dessa forma, manter cópias atualizadas da base de dados como backup torna-se uma tarefa simples e contribui para que o sistema atenda aos requisitos de alta disponibilidade, ou seja, baixo tempo de recuperação após falha. Em contra partida, a concentração da tarefa de armazenamento em um único elemento acarreta em um alto fluxo de requisições, sendo necessário o uso de um sistema com alta capacidade de processamento.

A presente arquitetura não é vinculada a uma *engine* de roteamento específica, sendo possível operar com qualquer pilha de protocolos que tenha suporte para sistemas operacionais Linux, independentemente da distribuição. A interoperabilidade entre protocolos de diferentes *engines* habilita a utilização de uma rede com pilhas de protocolos mista. Essa flexibilidade permite também a customização de protocolos sem qualquer impacto para a proposta RouteFlow. O mesmo vale para as RVs com sistemas Linux distintos.

Dentre os módulos apresentados na arquitetura RF da Figura 3.2, apenas o RF-Slave, RF-Server e RF-Controller, que estão representados em destaque na ilustração da Figura 3.2, necessitam ser desenvolvidos para a construção do sistema. Os demais módulos são aplicações prontas de código aberto disponíveis na Internet.

3.1.2 Requisitos

A arquitetura RouteFlow foi projetada sobre o paradigma de redes definidas por software com o objetivo de mover a lógica de controle embarcada no equipamento de rede para um elemento externo.

Para isso, o projeto baseou-se nos seguintes requisitos sistêmicos:

- **Transparência:** abstrair a arquitetura interna dos planos de controle e dados;
- **Interoperabilidade:** manter compatibilidade com as redes legadas;
- **Escalabilidade:** suportar o crescimento da rede sem perda de desempenho;
- **Confiabilidade:** permitir mecanismos de recuperação de falhas no plano de controle;
- **Portabilidade:** executar sobre diferentes sistemas operacionais de rede (controladores OpenFlow).

Em resposta ao modelo engessado e verticalmente integrado do plano de controle atualmente embarcados nos equipamentos de rede tradicionais, o RouteFlow tem como premissa criar uma arquitetura aberta à inovação. Nesta direção, a ideia é conceber uma arquitetura flexível e que seja transparente para o plano de controle (*engine* de roteamento) e, ao mesmo tempo, para o plano de dados (*switches* programáveis), e que permita executar pilhas de protocolos de roteamento sem qualquer alteração de código. Dessa forma, a interoperabilidade com redes legadas torna-se uma consequência direta.

Na arquitetura proposta, o plano de controle executa centralizado sobre o controlador da rede. Esse aspecto exige alto poder de processamento do elemento central e pode trazer problemas de escalabilidade. Por essa razão, torna-se interessante construir o RouteFlow sobre uma arquitetura modular que permita a execução distribuída de seus componentes. Dessa forma, o problema de escalabilidade é minimizado com a distribuição da carga de processamento.

Apesar da centralização afetar negativamente a escalabilidade do sistema, ela facilita a construção de mecanismos de proteção contra falhas, pois é mais simples proteger um único ponto do que coordenar a proteção de muitos deles. Neste sentido, manter as informações de estado da rede em uma base de dados centralizada e protegida contribui para alta disponibilidade do sistema.

Uma questão também importante no desenvolvimento de sistemas de software diz respeito à portabilidade. É interessante que as aplicações sofram um impacto mínimo com a alteração do sistema base. Neste sentido, a arquitetura do RouteFlow foi concebida para minimizar o impacto com a troca do controlador de rede. Com essa finalidade, o módulo RF-Controller atua como um *proxy*, ou seja, como uma camada responsável pelo acoplamento do sistema RF com o controlador de rede. Este componente será apresentado com detalhes na Seção 3.2.3.

3.2 Componentes

Uma visão global dos componentes do RouteFlow foi apresentada na Figura 3.2. A seguir, apresenta-se a descrição de cada um dos componentes da arquitetura proposta.

3.2.1 RouteFlow-Slave

Cada RV do plano de controle executa um processo (*daemon*) responsável pelo cumprimento das seguintes funções:

- realizar a descoberta das interfaces de rede disponíveis no sistema;
- registrar o roteador virtual no RF-Server como recurso da topologia virtual;
- executar o Protocolo de Descoberta de Interfaces (PID);
- aplicar configurações ao ambiente do RV (interfaces de rede, *engine* de roteamento);
- detectar as atualizações das tabelas ARP e de roteamento do sistema;
- enviar informações de rotas e vizinhos ao RF-Server a serem instaladas no plano de dados.

A primeira tarefa executada pela aplicação RF-Slave é examinar o sistema em busca das interfaces de rede disponíveis. Esse conhecimento é importante para que o RV assuma o controle de um dispositivo de rede com um número de portas compatível. Essa análise de compatibilidade deve ser feita considerando que uma das interfaces do RV é reservada para o canal de gerência, ou seja, é através dela que o RF-Slave comunica-se com o RF-Server. As demais interfaces estarão disponíveis para representarem as portas do roteador. Portanto, o RV que executar a lógica de controle de um roteador de N portas deve ter ao menos N + 1 interfaces de rede.

Após o levantamento das interfaces de rede do sistema, o RF-Slave conecta-se ao RF-Server a partir da interface de gerência e registra-se como um recurso disponível na topologia virtual, que futuramente será alocado para executar a lógica de controle de um roteador. Na etapa de registro são trocadas informações de configuração (número de interfaces de rede) e executado o Protocolo de Descoberta de Interfaces(PDI).

O objetivo do PDI é informar ao RF-Server em quais portas do software *switch* as interfaces de rede do RV estão conectadas. Esta informação é necessária para que o RF-Server encaminhe, quando necessário, o tráfego do plano de dados ao plano de controle e vice-versa. Considerando que os roteadores virtuais são instanciados e liberados conforme a demanda, as conexões das interfaces de rede dos RV com o software *switch* ocorrem dinamicamente, portanto torna-se necessário um mecanismo em tempo de execução para descobrir esse mapeamento.

O funcionamento de PDI é simples. Na etapa de registro do RF-Slave no RF-Server, o RF-Slave envia um frame Ethernet (ilustrado na Figura 3.3) em cada uma das interfaces de rede do RV, com exceção da gerência, com as seguintes informações:

- Ethernet_Type = 0x0A0A;

- identificador do RV;
- identificador da interface;

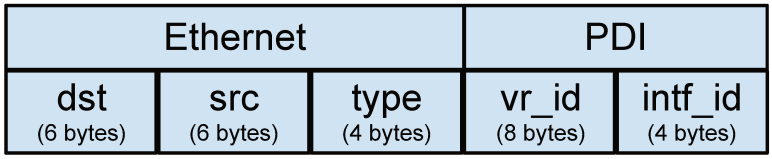


Fig. 3.3: Frame do Protocolo de Descoberta de Interfaces.

O frame Ethernet do PDI chega ao RF-Server através do controlador de rede carregando também a informação da porta de entrada do software *switch* programável, pois este encaminha o frame ao plano de controle. Desta forma, o RF-Server obtém as informações necessárias para o mapeamento que será descrito na Seção 3.2.2.

Após concluir o mapeamento entre as interfaces do RV e as portas do software *switch*, o RF-Server pode associar o RV a um elemento do plano de dados, configurando seu estado como ativo. Caso ainda não exista demanda dos elementos de encaminhamento, o RF-Slave é configurado no estado inativo e ficará aguardando por demandas futuras. Quando no estado inativo, as interfaces de rede são desabilitadas, com exceção da gerência.

No momento em que o RF-Slave entra no modo ativo, ele deve receber a informação do número de portas que serão utilizadas para que as mesmas sejam habilitadas no sistema e então a *engine* de roteamento pode ser inicializada. Concluída essa etapa, o RV está pronto para operar.

De forma análoga, com a saída de um elemento de encaminhamento da rede, a instância de controle respectiva deve ser liberada. O RF-Slave recebe a mensagem para entrar em modo inativo e esta operação implica em desabilitar as interfaces do sistema, com exceção da gerência, e as informações de rotas e vizinhos são removidas das tabelas do sistema.

Durante o modo de operação ativo, os pacotes referentes ao tráfego de controle da rede e os pacotes de dados com destino ao próprio roteador são encaminhados e recebidos pelo RV para processamento (por exemplo, ARP, ICMP, Telnet, SSH, OSPF, RIP, etc.). Esses pacotes chegam aos RVs pelas interfaces do sistema para que sejam devidamente tratados. Pacotes como ARP, ICMP são tratados pela pilha TCP/IP do Linux, enquanto os pacotes dos protocolos de controle são utilizados pela *engine* de roteamento para cálculo de rotas. O caminho inverso ocorre do mesmo modo, ou seja, os pacotes injetados pelo Linux, ou pela *engine* de roteamento, nas interfaces do RV são recebidos pelo *software switch* responsável por encaminhar os pacotes para o controlador ou, dependendo da configuração, para outros RVs.

Concomitantemente ao processamento de pacotes do RV, um mecanismo executa a tarefa de verificar as tabelas ARP e ROUTE do sistema em busca de atualizações que devem ser reproduzidas

no plano de dados. Quando atualizações são detectadas, as informações correspondentes são encaminhadas ao RF-Server através do canal de gerência para ser tratado.

3.2.2 RouteFlow-Server

É o componente central e de maior complexidade da arquitetura proposta, pois possui o conhecimento de todo o sistema sendo, portanto, o único elemento capaz de gerenciar o acoplamento entre o plano de dados e a topologia virtual (plano de controle). As principais responsabilidades do RF-Server são:

- conectar-se à aplicação RF-Controller;
- processar os eventos de rede recebidos por meio do RF-Controller;
- identificar e cadastrar o software *switch* programável;
- gerenciar os recursos (RVs) disponíveis na topologia virtual;
- configurar o software *switch* para construção da topologia lógica da rede;
- gerenciar as associações entre *switches* programáveis e RVs;
- processar as mensagens recebidas com informações de rotas provenientes do RF-Slave.

Por atuar no meio de campo entre os planos de controle e dados, o RF-Server deve concentrar e gerenciar toda a informação de rede necessária para tornar essa camada intermediária transparente para ambos os planos. Isso significa que os eventos e pacotes de controle provenientes do plano de dados devem ser entregues ao plano de controle e vice-versa, da mesma maneira que ocorre em um roteador clássico com a lógica de controle embarcada. A partir deste requisito, garante-se que a lógica de controle, que agora executa nos roteadores virtuais, não sofre qualquer alteração em relação à sua operação tradicional, o que permite a utilização de qualquer pilha de protocolos de roteamento que suporte o sistema operacional dos roteadores virtuais.

A questão chave deste componente está no mapeamento entre os elementos dos planos de dados e controle. No entanto, não é suficiente resolver o mapeamento apenas entre os dispositivos de rede e os RVs correspondentes, sendo necessário considerar também as portas/interfaces de cada um deles para que o controle atue adequadamente.

O encaminhamento de pacotes entre os planos é realizado por intermédio de um software *switch* programável, como ilustrado na arquitetura da Figura 3.2. Por se tratar de um elemento de encaminhamento programável, o software *switch* conecta-se ao controlador de rede da mesma forma que

os dispositivos físicos programáveis do plano de dados. Então, é necessário que o RF-Server faça a diferenciação de quais são os elementos do plano de dados e qual é o *switch* responsável pela interconexão dos RVs do plano virtual, pois este será gerenciado como uma entidade própria da arquitetura proposta.

Por atuar na troca de tráfego entre os planos de controle e dados, o software *switch* está diretamente relacionado com o mapeamento entre as portas dos *switches* físicos e as interfaces dos RVs. Para resolver esse mapeamento, o RF-Server precisa ter conhecimento do acoplamento das interfaces dos RVs com as interfaces do software *switch* para encaminhar o tráfego corretamente. Por se tratar de um acoplamento dinâmico, o mapeamento é conseguido através do Protocolo de Descoberta de Interfaces executado pelo componente RF-Slave no momento do registro no RF-Server, como descrito na Seção 3.2.1. Um exemplo do mapeamento entre os elementos do plano virtual e da infraestrutura física pode ser visto na Figura 3.4.

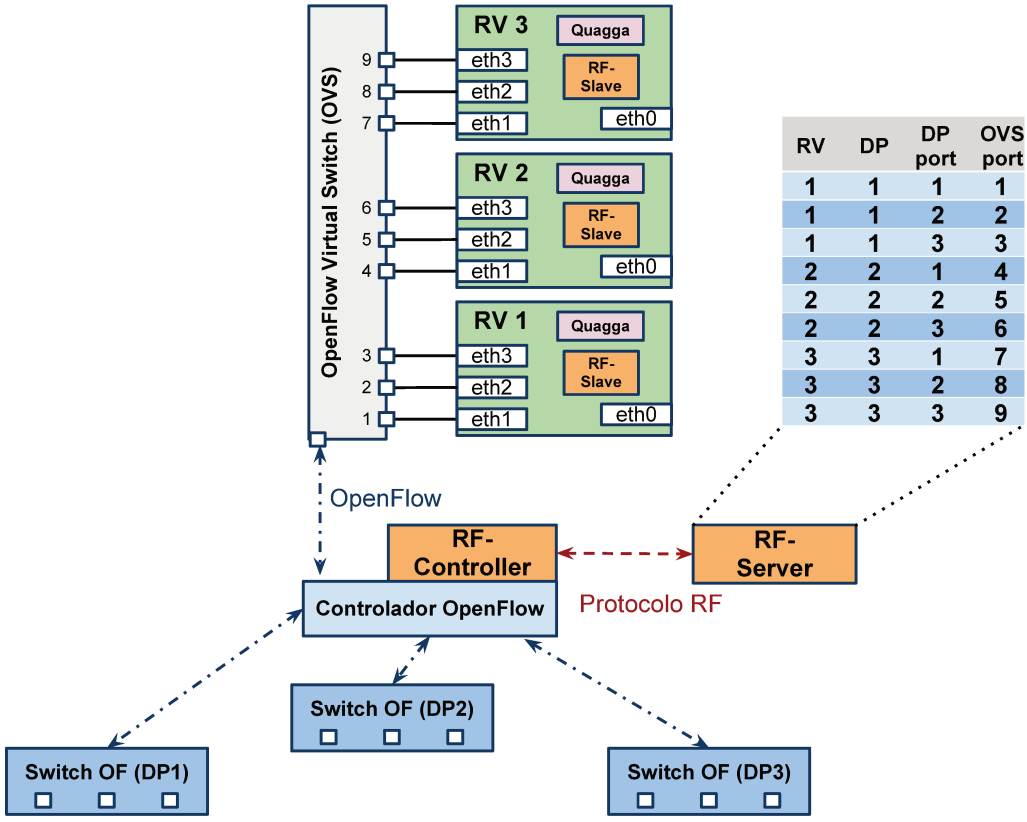


Fig. 3.4: Exemplo de mapeamento para arquitetura proposta.

Os frames PDI transmitidos nas interfaces dos RVs atingem o software *switch* e são destinados ao controlador de rede para serem tratados. Este processo resulta em um evento de entrada de pacote que chega ao RF-Server através de uma mensagem composta por: (i) identificador do dispositivo que recebeu o pacote, (ii) identificação da porta de entrada e (iii) o pacote recebido. Complementando

essas informações com aquelas carregadas no frame PDI (descritas na Seção 3.2.1), é possível mapear em quais portas do software *switch* as interfaces dos RVs estão conectadas.

Com o mapeamento estabelecido, os pacotes de controle recebidos pelos dispositivos do plano de dados e encaminhados para o plano de controle são destinados ao controlador de rede, que utiliza o mapeamento do RF-Server para encaminhar o pacote na porta correspondente do software *switch* e chegar até o roteador virtual, onde será tratado. Os pacotes dos RVs que devem ser entregues ao plano de dados atravessam o mesmo processo no sentido contrário.

Além da função de encaminhar pacotes, o software *switch* também permite manter o tráfego de controle confinado em uma topologia virtual através da criação de fluxos que direcionem determinado tráfego de um RV diretamente para outra sem passar pelo plano de dados. Essa interconexão dos RVs pode ser feita estaticamente através de um arquivo de configuração permitindo, dessa forma, a configuração de uma topologia virtual independente da topologia física. Alternativamente, a configuração pode ser dinâmica com base em um mecanismo de descoberta de topologia governado pelo controlador de rede. Neste último caso, a rede virtual será uma réplica da topologia física. Outra possibilidade consiste em agregar um conjunto de nós físicos como, por exemplo, *switch stacking/trunking*, em um único RV no plano virtual, ou ainda ter várias *engines* de roteamento atuando sobre um mesmo elemento físico, o que remete ao conceito de roteadores virtuais. Esses conceitos serão explorados na Seção 3.6.

As informações de rotas provenientes do RF-Slave para sincronismo com o plano de dados são tratadas no RF-Server. Ao receber as mensagens de eventos para remoção ou inclusão de rotas, o primeiro passo é converter essa informação do formato de rota tradicional para o formato de fluxo, no qual se baseia o encaminhamento dos *switches* programáveis. O procedimento de conversão está demonstrado no diagrama da na Figura 3.5.

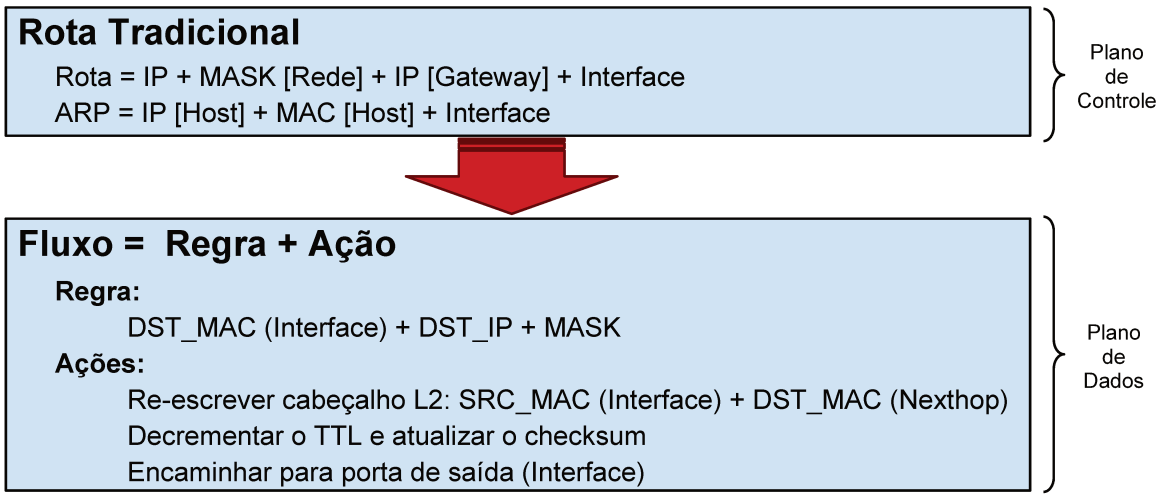


Fig. 3.5: Conversão de rotas em fluxos.

3.2.3 RouteFlow-Controller

Esta é a aplicação que executa sobre o controlador de rede. Ele é a interface entre o componente central do sistema (RF-Server) e os elementos do plano de dados, atuando como um *proxy*. O objetivo principal é isolar o sistema do controlador de rede facilitando a adaptação do RouteFlow para operar com múltiplos controladores. As principais responsabilidades deste componente são:

- registar-se no controlador de rede para receber eventos de entrada de pacotes de conexão e desconexão de *switches* e do estado dos enlaces;
- encaminhar os eventos de rede ao RF-Server;
- instalar/remover os fluxos dos equipamentos do plano de dados;
- enviar os pacotes aos planos de dados e de controle, este último através do *software switch*.

Os eventos de rede são importantes para garantir o sincronismo entre os planos de controle e dados. O primeiro passo é tratar os eventos de entrada e saída dos dispositivos de rede no plano de dados para que o plano de controle instancie ou libere um RV da topologia lógica. Dentro deste contexto, o RF-Controller encaminha os eventos ao RF-Server, onde serão tratados conforme descrito na Seção 3.2.2. O diagrama de sequência da Figura 3.6 ilustra este processo.

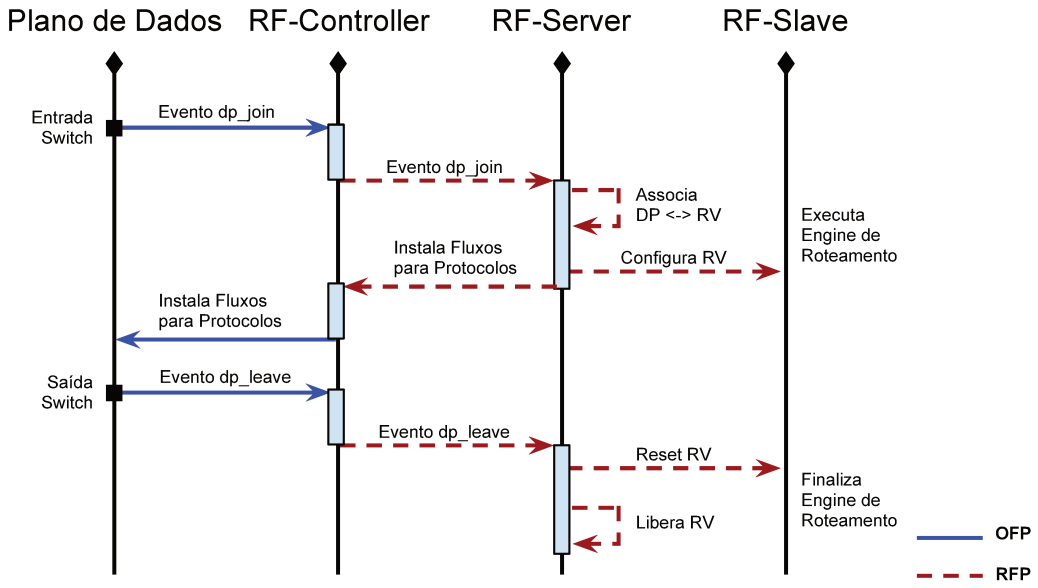


Fig. 3.6: Diagrama de sequência do processo de entrada e saída de *datapaths*.

Após a associação dos *datapaths* com os RVs, são tratados os eventos relacionados ao encaminhamento do tráfego entre os planos. Os pacotes que necessitam subir do plano de dados até o plano

de controle são recebidos pelo RF-Controller através do evento de entrada de pacotes (*packet-in*). Esse pacote chega com as informações do *switch* remetente e da porta de entrada. O pacote é então armazenado em um *buffer* e um evento de entrada de pacote é enviado ao RF-Server, cuja responsabilidade é resolver o mapeamento entre os elementos dos planos de dados e de controle. O resultado desse mapeamento é devolvido ao RF-Controller com a informação da porta do software *switch* na qual o pacote deve ser enviado para ser entregue ao RV na porta correspondente, como ilustrado no diagrama de sequência da Figura 3.7. O processo reverso ocorre de forma recíproca.

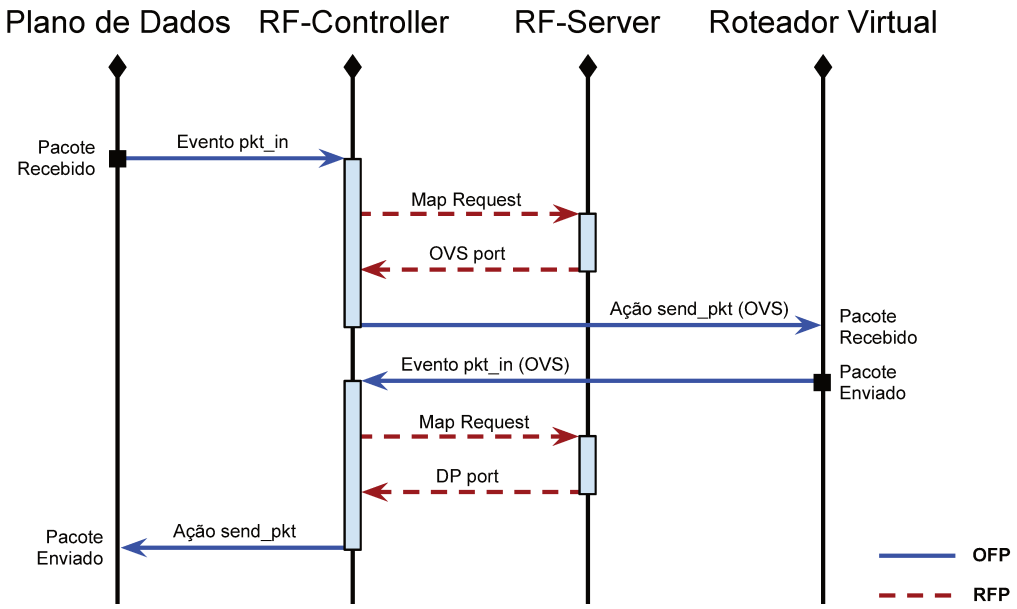


Fig. 3.7: Diagrama de sequência da troca de tráfego entre os planos.

3.3 Protocolo RouteFlow

As interfaces de comunicação entre os componentes da arquitetura proposta utilizam as mensagens especificadas pelo protocolo RouteFlow (RFP). As mensagens são estruturadas sobre o modelo Type-Length-Value (TLV) e definem eventos e comandos com o objetivo de sincronizar as informações entre os planos de controle e dados. Este processo de sincronismo necessita de mensagens para tratar: mudanças topológicas da rede, programação dos dispositivos de rede, obtenção de estatísticas de fluxos, gerenciamento do ambiente virtual, encaminhamento de tráfego entre os planos de controle e dados, e o mapeamento RV-Switch.

O protocolo RouteFlow utiliza uma comunicação assíncrona sem confirmação de recebimento de mensagem. Portanto não existe uma mensagem de *acknowledge* definida para notificar o recebimento,

uma vez que a utilização de um canal de comunicação orientado à conexão garante a entrega da mensagem. Esse canal pode utilizar o protocolo TCP, ou ainda, protocolos como SSH ou SSL para prover maior segurança ao sistema.

O formato da mensagem definida para o protocolo RFP está representado na Figura 3.8. Abaixo segue a descrição geral para cada um dos campos definidos.

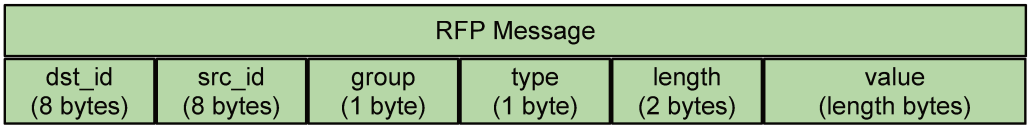


Fig. 3.8: Formato das mensagens do protocolo RouteFlow.

- **dst_id**: identificador único de 64 bits do destinatário;
- **src_id**: identificador único de 64 bits do remetente;
- **group**: classificação das mensagens em grupos de funcionalidades;
- **type**: identificação da mensagem dentro do grupo especificado;
- **length**: tamanho da informação carregada no campo *value* da mensagem;
- **value**: carrega as informações específicas de cada tipo de mensagem (argumentos de comandos, respostas de comandos, informações de eventos, informações adicionais sobre erros, entre outros).

Os campos *dst_id* e *src_id* são necessários para identificação do RF-Slave, do RF-Server e do RF-Controller. A necessidade de identificação do RF-Slave fica bastante evidente na arquitetura proposta, pois o plano de controle trabalha com inúmeras instâncias de roteadores virtuais para representação dos dispositivos do plano de dados. Entretanto, apesar do RF-Server e do RF-Controller serem únicos do ponto de vista lógico, eles podem ser implementados de forma fisicamente distribuída com o propósito de escalabilidade e alta disponibilidade. Ambos requisitos mostram-se imprescindíveis no cenário de redes para as próximas evoluções, portanto estão sendo consideradas no modelamento do protocolo de comunicação entre os componentes da arquitetura.

O agrupamento de mensagens em categorias bem definidas facilita a visualização e o entendimento do protocolo. Consequentemente, deve existir um campo que especifique o tipo da mensagem dentro do grupo. Seguindo níveis lógicos hierárquicos, o grupo deve ser processado primeiro para então o tipo ser analisado dentro do contexto do grupo definido.

Os campos *length* e *value* terminam a mensagem definindo o comprimento e o conteúdo da informação relacionada ao tipo de mensagem que está sendo carregada. Vale destacar que o comprimento diz respeito apenas ao campo *value* e não à mensagem como um todo. Considerando que as menagens podem exigir informações com tamanhos distintos, manter o tamanho da mensagem flexível exige um processamento extra para conhecer o tamanho da mensagem, porém apresenta economia do canal de comunicação em relação à utilização de um tamanho fixo definido pela maior mensagem existente.

As mensagens do protocolo RFP foram definidas com base nas necessidades das duas interfaces de comunicação presentes na arquitetura proposta: a interface entre os componentes RF-Server e RF-Slave, e a interface entre os componentes RF-Server e RF-Controller. Cada uma delas apresenta características particulares que serão abordadas nas seções seguintes.

3.3.1 Comunicação entre o RF-Slave e o RF-Server

A comunicação entre os roteadores virtuais e o elemento central do sistema é caracterizada pelo modelo cliente-servidor. O RF-Server assume a função de servidor enquanto as instâncias do RF-Slave fazem o papel de clientes. Dessa forma, a conexão é iniciada pelo RF-Slave, que deve registrar-se no RF-Server como um recurso da topologia virtual para ser alocado para executar a lógica de controle dos elementos físicos programáveis da infraestrutura de rede.

Ao receber um pedido de registro, o RF-Server deve buscar pela identificação do RV com o objetivo de verificar sua unicidade dentro do sistema. Não pode existir outro RV já registrado com o mesmo identificador. Caso o RV tenha autorização para registrar-se, o RF-Server envia ao RV uma mensagem de RESET de configuração, o que coloca o RV em modo de operação inativo. A sequência de mensagens está ilustrada no diagrama da Figura 3.9.

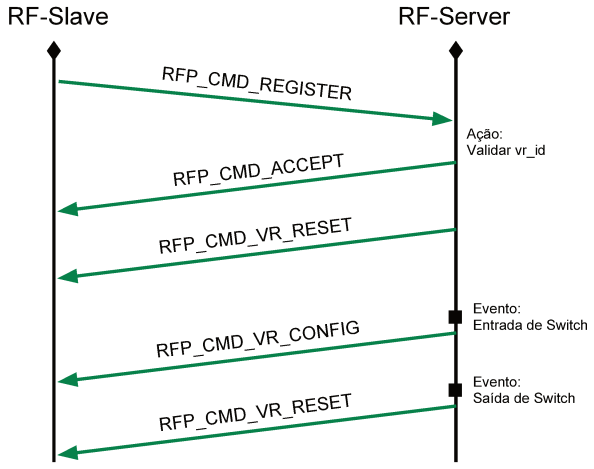


Fig. 3.9: Diagrama de sequência das mensagens de conexão e configuração do RV.

Após receber a mensagem de RESET e entrar no modo de operação inativo, o RV estará pronto para ser alocado para executar a lógica de controle de um elemento de encaminhamento. Quando um novo dispositivo de rede programável entra na rede o RF-Server envia uma mensagem de configuração ao RF-Slave para que este entre no modo ativo, ou seja, inicialize a *engine* de roteamento e comece a monitorar as tabelas de roteamento do sistema. De forma recíproca, quando o equipamento é removido da rede ou perde conexão com o controlador, o RV é liberado para ser utilizado por novos *switches* programáveis e uma mensagem de RESET é recebida.

Depois que a associação RV-*Switch* estiver concluída, o protocolo RouteFlow é utilizado para garantir o sincronismo entre as tabelas de rotas das *engines* de roteamento e as tabelas de fluxos dos dispositivos do plano de dados. Este processo consiste do monitoramento das entradas de roteamento e quando houver alteração, tanto adição quanto remoção, o evento deve ser enviado ao RF-Server para que o dispositivo de rede seja devidamente programado.

As ações responsáveis por disparar mensagens de eventos de atualização das tabelas são:

- Adição de novas rotas na tabela de roteamento;
- Remoção de rotas da tabela de roteamento;
- Aprendizado de novos *hosts* das redes diretamente conectadas.

Para todos eventos citados acima, a aplicação RF-Slave deve enviar uma mensagem de evento ao RF-Server informando as alterações. Vale destacar que as entradas ARP que expiram (usualmente, quando ociosas por 300s) não precisam ser removidas da tabela de fluxos do *switches*, pois uma vez que o tráfego está passando apenas pelo plano de dados a ociosidade detectada pelo plano de controle não representa a realidade, portante este evento é ignorado.

Todo sequenciamento de mensagens pode ser observado no diagrama da Figura 3.9.

A Tabela 3.2 apresenta as mensagens definidas de acordo com a discussão acima sobre a interface de comunicação entre os componentes RF-Slave e RF-Server.

A maior parte das mensagens do grupo de comandos não necessitam de argumentos, portanto o campo *length* será zero e o *value* é omitido. As demais mensagens requerem argumentos variados o que resulta em mensagens com tamanhos distintos. O tamanho do campo de endereço MAC é de 6 bytes, os endereços IP são de 4 bytes, o número de portas é de 2 bytes, a porta de destino é de 4 bytes e a máscara de rede é de 4 bytes.

3.3.2 Comunicação entre o RF-Server e o RF-Controller

Na arquitetura RouteFlow, o componente RF-Controller atua apenas como um *proxy* entre o RF-Server e o controlador de rede. Por essa razão, o RF-Controller não armazena estados e também

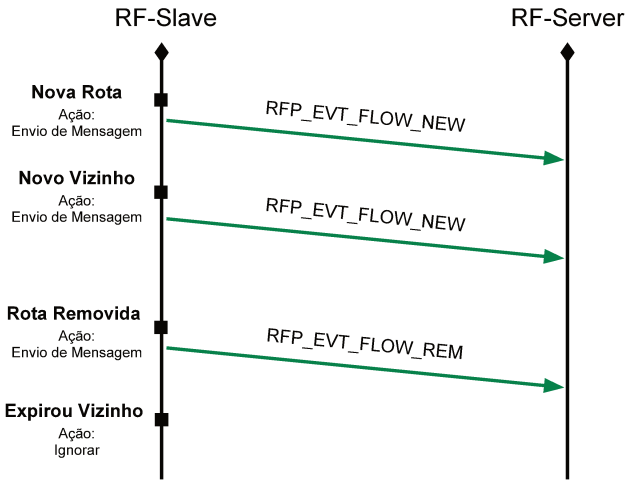


Fig. 3.10: Diagrama de sequência das mensagens de atualização de rotas.

Tab. 3.1: *Definições das mensagens para interface de comunicação RF-Server e RF-Slave.*

Grupo	Tipo	Origem	Destino	Argumentos
Comandos	RFP_CMD_REGISTER	RF-Slave	RF-Server	(null)
	RFP_CMD_ACCEPT	RF-Server	RF-Slave	(null)
	RFP_CMD_REJECT	RF-Server	RF-Slave	(null)
	RFP_CMD_UNREGISTER	RF-Slave	RF-Server	(null)
	RFP_CMD_VR_RESET	RF-Server	RF-Slave	(null)
	RFP_CMD_VR_CONFIG	RF-Server	RF-Slave	(num_ports)
Eventos	RFP_EVT_FLOW_NEW	RF-Slave	RF-Server	(my_mac, nh_mac, dst_ip, net_mask, dst_port)
	RFP_EVT_FLOW_REM	RF-Slave	RF-Server	(my_mac, dst_ip, net_mask)

não executa análises sobre os dados. A lógica implementada no componente fica encarregada de encaminhar todos os eventos produzidos pelo controlador de rede para o RF-Server e receber deste último os comandos para serem aplicados no plano de dados.

As primeiras mensagens de eventos a serem definidas estão relacionadas à entrada e saída dos dispositivos da rede, ou seja, qualquer mudança topológica deve ser informada ao plano de controle para que este adeque-se ao novo estado da rede. Outra informação pertinente à mudança topológica diz respeito aos enlaces que conectam os equipamentos formando a malha do plano de dados. A informação de estado de enlace pode ser utilizada pelo RouteFlow para montar topologias lógicas virtuais que reflitam as conexões da infraestrutura física, ou ainda, utilizar essa informação para notificar diretamente a *engine* de roteamento sobre as alterações de estado dos enlaces, eliminando assim o processo de detecção de falha de enlace executado por protocolos de roteamento baseados em estado de enlace.

Além das mudanças topológicas, o plano de controle necessita receber os pacotes de controle que serão utilizados pelas *engines* de roteamento para calcular as rotas para o encaminhamento do tráfego. Esse processo é realizado através do mapeamento realizado pelo RF-Server, como descrito na Seção 3.2.2. A Figura 3.11 apresenta um diagrama de sequência do processo de encaminhamento de pacotes.

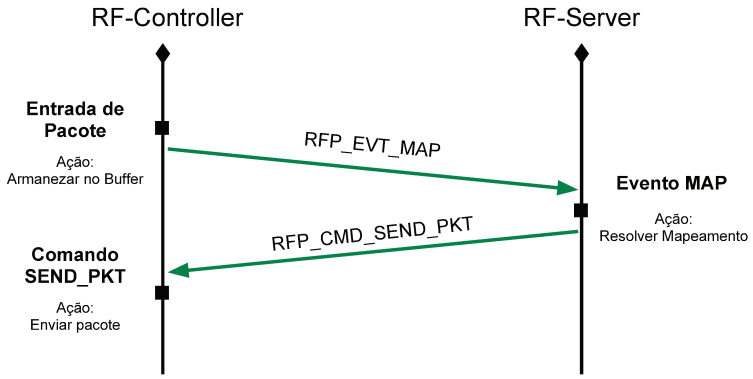


Fig. 3.11: Diagrama de sequência das mensagens no encaminhamento de pacotes no RouteFlow.

Após definir as mensagens para transmitir o estado e os pacotes do plano de dados para o plano de controle, são necessárias as mensagens para sincronizar o estado das tabelas de roteamento dos RVs com as tabelas dos dispositivos programáveis e, por fim, a transmissão dos pacotes de controle nesta mesma direção. Estas serão as mensagens de comando do ponto de vista da interface de comunicação entre o RF-Server e o RF-Controller.

As mensagens de eventos provenientes do RF-Slave que chegam ao RF-Server são transformadas em comandos para o RF-Controller. As informação retiradas das tabelas de roteamento dos RVs são convertidas no formato de fluxo para que possam ser programadas nos dispositivos do plano de dados.

A Tabela 3.2 apresenta as mensagens definidas de acordo com a discussão acima sobre a interface de comunicação entre os componentes RF-Server e RF-Controller.

3.4 Mecanismos Internos de Troca de Informação

Internamente à arquitetura proposta, os pacotes e mensagens trafegam entre os planos de dados e controle através de dois tipos de interface de comunicação, como mostrado na Figura 3.12. A comunicação entre os *switches* programáveis e o controlador de rede ocorre pela API padrão de configuração, neste caso o protocolo OpenFlow. Apesar do software *switch* não pertencer conceitualmente ao plano de dados, ele se conecta ao controlador de rede e, portanto, utiliza o canal de comunicação via o protocolo OpenFlow. Por outro lado, a comunicação entre os componentes do RouteFlow ocorre através

Tab. 3.2: *Definições das mensagens para interface de comunicação RF-Server e RF-Controller.*

Grupo	Tipo	Origem	Destino	Argumentos
Comandos	RFP_CMD_FLW_ADD	RF-Server	RF-Controller	(dp_id, flw_rules, flw_actions)
	RFP_CMD_FLW_REM	RF-Server	RF-Controller	(dp_id, flw_rules)
	RFP_CMD_FLW_MOD	RF-Server	RF-Controller	(dp_id, flw_rules, flw_actions)
	RFP_CMD_SEND_PKT	RF-Server	RF-Controller	(dp_id, port_out, pkt_id)
Eventos	RFP_EVT_DP_JOIN	RF-Controller	RF-Server	(dp_id, num_ports, hw_desc)
	RFP_EVT_DP_LEAVE	RF-Controller	RF-Server	(dp_id)
	RFP_EVT_LINK	RF-Controller	RF-Server	(reason, dp1_id, port_1, dp2_id, port_2)
	RFP_EVT_MAP	RF-Controller	RF-Server	(vm_id, vm_port, ovs_port)

do protocolo RFP, como apresentado nas seções anteriores.

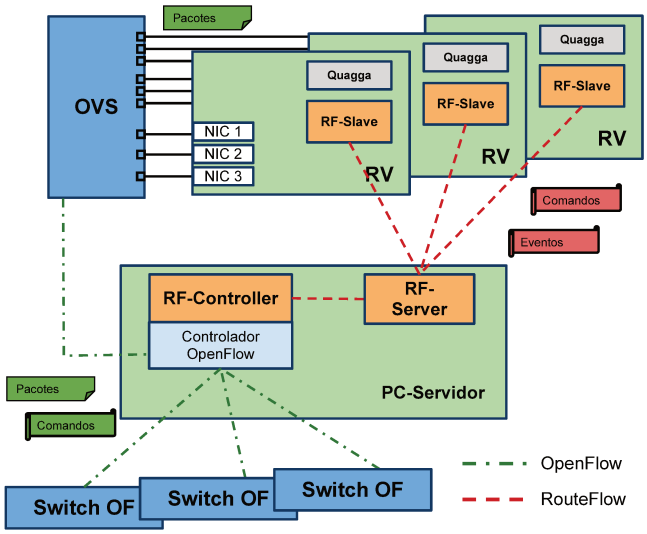


Fig. 3.12: Tráfego de mensagens e pacotes entre os planos de dados e controle.

Abaixo segue a descrições dos processos de encaminhamento de tráfego e mecanismos de sincronismo de informações entre os equipamento do plano de dados e os roteadores virtuais do plano de controle virtual.

3.4.1 Tráfego de Controle

O tráfego de controle aplica-se tanto no sentido do PC para o PD quanto do PD para o PC, pois os pacotes de protocolos que são recebidos pelo dispositivo de rede devem ser encaminhados para análise pelo controle, ou seja, pela *engine* de roteamento correspondente. De forma recíproca, a

lógica de controle também produz pacotes de protocolos para disseminar a informação na rede, o que exige que esse tráfego saia dos RVs e seja recebido nos elementos de rede correspondentes.

Dentro do RV, a *engine* de roteamento injeta os pacotes nas respectivas interfaces do sistema que estão diretamente conectadas ao software *switch*. Ao entrar pela porta desse *switch*, que é programável, o pacote é encaminhado ao controlador de rede via protocolo OpenFlow e consequentemente chega ao componente RF-Controller. Este armazena o pacote em uma fila e envia a mensagem RFP_EVT_MAP para o RF-Server via o RFP. A mensagem questiona qual é a porta e o dispositivo do plano de dados pela qual o pacote deve ser enviado. O RF-Server resolve o mapeamento e envia a mensagem RFP_CMD_SEND_PKT ao RF-Controller que então remove o pacote da fila e o encapsula em uma mensagem OpenFlow para ser enviada ao plano de dados.

O tráfego de pacotes do PD para o PC ocorre de forma equivalente trilhando o mesmo percurso, porém no sentido contrário. O pacote chega então ao controlador pelos *switches* do plano de dados e, após a resolução do mapeamento no RF-Server, o pacote é direcionado ao PC através do software *switch*.

3.4.2 Sincronismo da FIB

O sincronismo ocorre a partir da detecção das atualizações da tabela de roteamento no sistema operacional do RV. Após detectadas as atualizações, a informação deve ser propagada até o RF-Server, que então se encarregará de efetuar a programação do elemento de encaminhamento.

De acordo com o procedimento de conversão de rotas em fluxos, que foi apresentado na Seção 3.2.2, as informações da tabela de roteamento não são suficientes, sendo então necessário combinar com as informações de vizinhos obtidas da tabela ARP. Após coletados os dados, uma mensagem RFP_EVT_FLOW_NEW ou RFP_EVT_FLOW_REM é criada e enviada via protocolo RFP ao RF-Server, que então converte a mensagem para RFP_CMD_FLW_ADD ou RFP_CMD_FLW_REM, respectivamente, e encaminha também via RFP ao RF-Controller. Por fim, a mensagem é colocada no formato da API OpenFlow e enviada ao elemento de encaminhamento correspondente para que a programação seja efetivada.

Configurar fluxos com regras que caracterizem sub-redes IP não é suficiente para garantir que o tráfego IP será encaminhado pelo *switch* programável de forma adequada respeitando a lógica do *Long Prefix Match* (LPM). O problema pode ocorre porque dentro de uma mesma prioridade os pacotes que se encaixarem na regra de mais de um fluxo serão encaminhados seguindo a ordem do primeiro fluxo adicionado, o que não refletiria o comportamento do LPM. Para resolver essa questão pode-se utilizar o valor da máscara de rede para compor a prioridade da entrada de fluxo, com isso garante-se que em casos de múltiplos *matches* a prioridade implicará na escolha do fluxo de acordo com o LPM.

3.4.3 Descoberta de Vizinhos (ARP)

O ARP é o protocolo responsável por conectar a camada 3 à camada 2 do modelo OSI, ou seja, vincular o endereço IP ao endereço físico da interface de rede (endereço MAC). Para se comunicar com outro dispositivo dentro da rede local é necessário conhecer o endereço MAC, que pode ser obtido através do protocolo ARP a partir do endereço IP conhecido. Caso o endereço IP destino esteja fora da rede local, a comunicação deve ser feita através do *gateway* da rede, cuja função é isolar domínios de *broadcast* realizando o encaminhamento na camada 3 (roteamento). Portanto, o endereço MAC de destino utilizado na comunicação externa é sempre o do *gateway*, enquanto o endereço IP é mantido o do destino original. Neste contexto, o *gateway* atua como elemento de borda entre domínios de rede IP.

O mapeamento entre os endereços IP e MAC é mantido na tabela ARP. As entradas podem ser de dois tipos: dinâmicas ou estáticas. Como o próprio nome sugere, as entradas dinâmicas são preenchidas automaticamente pelo protocolo ARP e expiram após um intervalo de tempo determinado (*aging time*) de inatividade (300 segundos por padrão). Já as entradas estáticas são inseridas e removidas manualmente.

Cada interface, ou porta, de um roteador pertence a um domínio IP distinto. Por essa razão o dispositivo apresenta uma tabela ARP para cada uma das redes locais. A função deste elemento de rede é encaminhar os pacotes no nível de endereçamento IP, com a finalidade de promover a comunicação entre as redes. Se o endereço IP de destino do pacote a ser encaminhado pertencer a alguma das redes diretamente conectadas ao roteador, ele realiza o encaminhamento através das tabelas ARP. Caso o destino do pacote esteja fora do alcance do roteador, ele o encaminha, através da tabela de roteamento, para um roteador (*next-hop*) que estiver em uma das suas redes.

Nexte contexto, o RouteFlow precisa também sincronizar as entradas da tabela ARP com o plano de dados para implementar o roteamento. Porém, devido ao *aging time*, esse sincronismo apresenta uma característica diferenciada quando comparada à tabela FIB. Um aspecto bastante relevante está no fato da tabela ARP do plano de controle deixar de detectar atividade após o sincronismo com o plano de dados, pois o tráfego não chegará mais ao RV. Com isso, as entradas ARP do plano de controle irão expirar. Porém, isso não significa que a entrada deve ser removida do dispositivo do plano de dados, pois o tráfego pode estar ativo. Portanto, as atualizações de remoção da tabela ARP devem ser ignoradas no plano de controle e o próprio plano de dados deve implementar um *timeout* de 300 segundos, repassando então a função do *aging time* para o equipamento de rede.

Vale ressaltar que as mensagens utilizadas para sincronizar as entradas da tabela ARP bem como o procedimento de conversão de rotas em fluxos, são equivalentes ao da tabela FIB, considerando agora que os fluxos referentes aos vizinhos apresentam valor máximo de máscara de rede e por isso têm maior prioridade, como explicado na Seção 3.4.2.

3.5 Tratamento do tráfego de controle

A efetiva separação entre os planos de controle e dados possibilita aplicar um tratamento personalizado sobre o tráfego de controle. Um grande aliado nesse processo é o software *switch* programável que faz a interconexão entre os roteadores virtuais do plano de controle. Tal programabilidade permite caracterizar o tráfego de controle e aplicar ações individuais para cada um deles.

A decisão de encaminhamento dos pacotes de protocolos pelo elemento que atua entre os planos de controle e dados permite manter determinado tráfego confinado na topologia virtual. Isso porque os pacotes de protocolos destinados aos roteadores vizinhos podem ser entregues diretamente aos RVs sem precisar trafegar pelo plano de dados.

No entanto, a decisão de manter os pacotes de protocolos confinados no plano de controle impossibilita a detecção direta de falhas na rede. Desta forma, em caso de falha de um enlace no plano de dados, os pacotes de controle continuariam a ser entregues sem refletir o estado atual da rede. Mas essa situação pode ser resolvida a partir de um mecanismo dedicado em manter o plano de controle sincronizado com o estado do plano de dados.

Utilizar mecanismos mais eficientes de monitoramento do estado da rede, como por exemplo o protocolo IEEE 802.1ag [38], pode trazer grande benefício na otimização dos protocolos de roteamento. Quanto mais rápido a falha for detectada e informada à *engine* de roteamento, mais rápido ocorrerá o processo de convergência da rede e, consequentemente, o serviço ficará menos tempo fora.

3.6 Casos de uso e modos de operação

A arquitetura de roteamento IP proposta neste trabalho promove a efetiva separação entre os planos de controle e encaminhamento. Esta característica, combinada aos recursos da tecnologia de virtualização utilizada no plano de controle, possibilita explorar novas aplicações e serviços de rede. A flexibilidade no mapeamento entre os equipamentos físicos e as instâncias de controle virtuais (RVs) abre possibilidades de variações na combinação entre os elementos agregando, dessa forma, um grande potencial de inovação à arquitetura. Dentre as combinações possíveis, foram identificados três casos de uso, que estão ilustrado na Figura 3.13.

A seguir serão apresentados em detalhe cada um dos casos de uso identificados.

3.6.1 Separação Lógica

Seguindo o modelo clássico, no qual cada equipamento de rede executa sua própria lógica de controle, o mapeamento 1:1 entre os *switches* físicos e os RVs reflete o cenário atual sendo que, no

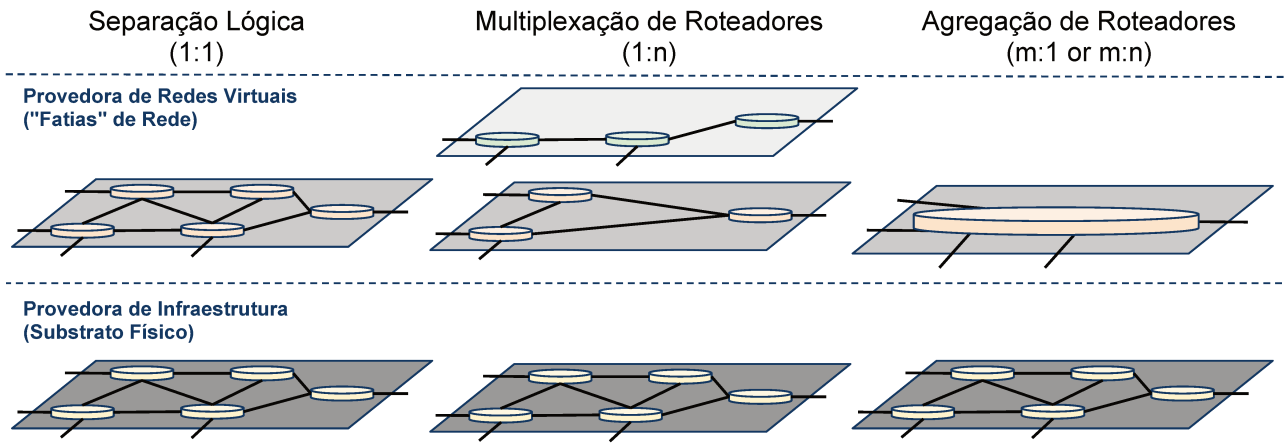


Fig. 3.13: Casos de uso da arquitetura RouteFlow.

caso, o controle executa remotamente ao dispositivo. Neste modo de operação, a topologia virtual é construída por meio do espelhamento da infraestrutura física.

Apesar de não apresentar inovação do ponto de vista lógico da arquitetura de roteamento, este modo de operação oferece uma grande flexibilidade sobre a lógica de controle, que abrange desde da facilidade em adicionar e remover funcionalidades até a customização dos protocolos para aplicações específicas de rede.

Além do alto potencial de personalização da lógica de controle, o mapeamento 1:1 mostra-se bastante interessante para os seguintes cenários de aplicação:

- A utilização de *switches* programáveis em uma infraestrutura de rede legada exige que estes equipamentos executem os mesmos protocolos da rede. Uma alternativa seria adquirir os *switches* programáveis com os protocolos embarcados, no entanto o custo seria elevado. Nesse cenário, a arquitetura RouteFlow permitiria que os novos equipamentos fossem integrados à rede de forma simples e com baixo custo.

- A interoperabilidade de um sistema autônomo (AS), cuja infraestrutura é exclusivamente composta de *switches* programáveis, com outros ASes legados. Neste cenário a arquitetura RouteFlow possibilita que os equipamentos de borda executem o protocolo BGP e compartilhem as rotas desejadas com os demais ASes. Enquanto isso, o núcleo da rede executaria seus próprios protocolos para realizar o encaminhamento interno ao AS.

3.6.2 Multiplexação

Compartilhar a utilização dos recursos da infraestrutura de rede é sempre uma opção eficiente, principalmente considerando o perfil de tráfego não linear da Internet. Portanto, operar instâncias

isoladas de redes lógicas sobre uma mesma topologia física torna-se um recurso bastante importante para as redes de próxima geração.

A arquitetura proposta possibilita realizar o mapeamento 1:n, ou seja, um único *switch* programável pode ser operado por n instâncias de roteadores virtuais, cada qual executando sua lógica de controle de forma isolada. Então, através desta funcionalidade a arquitetura RouteFlow habilita o compartilhamento da infraestrutura física entre diferentes topologias virtuais de rede operando isoladamente.

A partir deste modo de operação, a arquitetura proposta torna possível aplicar um novo modelo de negócio que promova uma divisão de responsabilidades na oferta de serviços sobre uma infraestrutura de rede. Neste modelo, uma categoria de provedores assume a responsabilidade pela infraestrutura física da rede enquanto outra categoria ficaria incumbida de fornecer serviços de conectividade aos usuários. Portanto, o conceito de roteamento como serviço, sugerido em [39][40], pode ser explorado sobre a arquitetura RouteFlow.

3.6.3 Agregação

Um domínio de rede implementado sobre uma infraestrutura de *switches* programáveis opera sobre um novo paradigma em relação às arquiteturas tradicionais de rede. Enquanto as redes atuais operam com controle distribuído, as redes definidas por software são controladas por um sistema operacional de rede logicamente centralizado. Portanto, os protocolos de rede clássicos, que foram desenvolvidos sobre o paradigma de sistemas distribuídos, não são eficientes para uma arquitetura centralizada.

Contudo, no universo de redes heterogêneas, a interoperabilidade é requisito fundamental para que as novas redes não tornem-se ilhas isoladas. Nesta direção, a arquitetura de roteamento IP proposta neste trabalho permite que redes programáveis interoperem com as redes legadas quando operando no modelo de agregação.

O trabalho apresentado em [41] aborda esse modo de operação através de uma plataforma de roteamento como serviço sobre a arquitetura RouteFlow. O mapeamento m:1 possibilita que a rede programável como um todo seja vista pelas redes legadas como um único roteador, ou seja, todos os _m_ *switches* programáveis estarão sobre o controle de uma única lógica de controle (RV), como mostra a Figura 3.14.

As sessões BGP com as redes adjacentes cuidariam do encaminhamento do tráfego externo. Porém, o encaminhamento do tráfego interno pode ser realizado por qualquer outra lógica que rode no controlador de rede. Desse modo, uma rede programável pode se comunicar com as redes legadas e ao mesmo tempo implementar um algoritmo eficiente para o encaminhamento interno sobre o paradigma de controle centralizado.

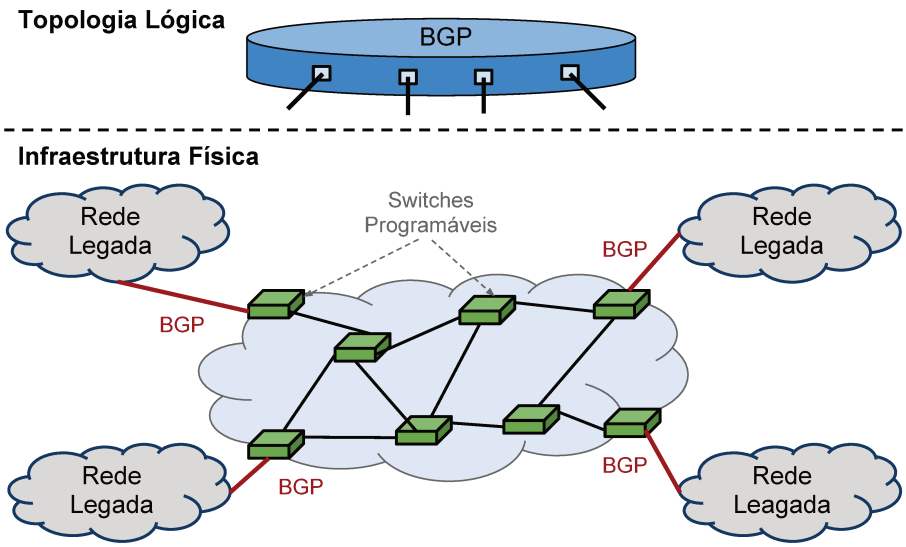


Fig. 3.14: Arquitetura RouteFlow operando em modo de Agregação.

3.7 Resumo do Capítulo

Este capítulo apresentou em detalhes a arquitetura proposta para roteamento IP com separação entre os planos de controle e encaminhamento sobre redes programáveis. Esse mesmo conceito é também explorado nos trabalhos do SoftRouter, DROP e FIBIUM, expostos na Seção 2.4. No entanto, os dois primeiros abordam o paradigma de separação entre os planos de dados e controle de forma mais conceitual através da definição dos elementos da rede e suas responsabilidades, sem entrar nos detalhes de implementação. Já o FIBIUM, embora apresente uma aplicação prática, tem foco no algoritmo de controle de um elemento de encaminhamento de forma individual com o objetivo de otimizar a utilização da tabela de fluxos em hardware.

Diferentemente dos trabalhos anteriores, o RouteFlow é uma proposta pragmática de arquitetura de roteamento IP que abrange a rede como um todo. O trabalho detalha a especificação dos componentes responsáveis pela construção de uma plataforma flexível que mantém compatibilidade com as redes legadas.

Apresentou-se, conceitualmente, os mecanismos e algoritmos construídos com o intuito de viabilizar a execução da lógica de controle externa ao equipamento. Contudo, existem *overheads* tanto da troca de informação entre os planos como do processamento na nova camada intermediária responsável pelo acoplamento entre eles. Logo, é importante que esses *overheads* sejam mensurados na prática para uma análise refinada de viabilidade do projeto. Neste contexto, o próximo capítulo apresenta o protótipo e os testes desenvolvidos para avaliação do mesmo.

Capítulo 4

Protótipo e Avaliação

Com o objetivo de demonstrar a viabilidade de um roteador em operar com uma lógica de controle remota a partir da arquitetura proposta, um protótipo foi construído considerando uma versão simplificada da arquitetura apresentada no Capítulo 3. Este trabalho de avaliação não tem a preocupação com a otimização de desempenho na implementação do protótipo. O foco está em validar o modelo, identificar pontos de atenção e explorar os desafios no contexto das redes definidas por software.

A modularidade da arquitetura não amarra a implementação à lógica de controle. Essa característica flexibiliza o uso da pilha de protocolos de roteamento, que pode ser substituída sem qualquer alteração em código. No caso do protótipo foi utilizado o Quagga [10], uma conhecida *engine* de roteamento *open source* com suporte aos principais protocolos de roteamento (RIP, OSPF, BGP). Dentre as opções *open source* disponíveis poderiam ser utilizadas também as pilhas de roteamento BIRD [42] e XORP [11]. A escolha baseou-se no conhecimento prévio sobre a *engine* Quagga, que foi utilizada no desenvolvimento de um protótipo de roteador clássico.

Outro ponto que apresenta grande flexibilidade é a escolha do virtualizador. Existe uma variedade de ferramentas para virtualização que seguem paradigmas diferenciados. Cada virtualizador apresenta características particulares que podem ser exploradas dentro do contexto de cada aplicação. Nesta implementação foi utilizado o QEMU, que é uma ferramenta de virtualização de código aberto e opera com o modo de virtualização completa.

A técnica de virtualização completa é caracterizada pelo forte isolamento do ambiente virtualizado e simplicidade na instalação, pois o sistema operacional da MV não tem ciência de que executa em um ambiente virtualizado, portanto, não sofre alteração. No entanto, o consumo de recursos é mais elevado e pode apresentar um desempenho inferior quando comparado com virtualizadores que operem com para-virtualização.

Ao contrário da virtualização completa, a técnica de para-virtualização exige que o sistema operacional tenha ciência da virtualização. Com isso, otimizações são feitas para que o sistema apresente

melhor desempenho. Em contrapartida, essa técnica apresenta menor isolamento e exige sistemas operacionais próprios.

Outras opções de virtualizadores são: LXC [43], VMWare [44], VirtualBox [45] e Xen [46]. Um estudo comparativo entre alguns destes é apresentado em [37].

Como plataforma para programabilidade remota dos equipamentos de rede utilizou-se o OpenFlow na versão 1.0 [1], que define uma interface padrão de código aberto e com foco em redes de pacotes. Uma grande vantagem desse protocolo é a abordagem *multi-vendor*. Isto significa que independentemente do fabricante e do modelo do equipamento que compuser a rede, desde que haja suporte ao protocolo OpenFlow, não serão necessárias mudanças no controlador nem nas aplicações de rede que executam sobre ele.

A infraestrutura do plano de dados utilizada nos testes é formada por NetFPGAs, que são hardware programáveis com quatro interfaces de rede Gigabit e com módulo OpenFlow. A NetFPGA é uma alternativa de baixo custo ao *switch* OpenFlow quando o cenário de aplicação não requer muitas interfaces de rede, pois apresenta um número bastante limitado. No entanto, o processamento dos pacotes ocorre em velocidade de linha devido à aceleração no processamento de pacotes pela FPGA. Outra característica importante é a atualização do protocolo OpenFlow, que por tratar-se de um código programável na FPGA, pode ser facilmente atualizado para novas versões.

4.1 Implementação

A versão implementada do protótipo, cujo principal objetivo é validar a arquitetura proposta, foi desenvolvida a partir de uma simplificação do modelo completo apresentado na Figura 3.2. A redução da complexidade da arquitetura baseou-se nos requisitos mínimos necessários para que o protótipo pudesse ser aplicado à rede experimental construída no laboratório. A arquitetura simplificada está ilustrada na Figura 4.1.

Na comparação entre as arquiteturas, as modificações aparecem com bastante evidência. O componente RF-Server não está mais presente, pois toda a lógica desse elemento está concentrada no RF-Controller. O motivo da separação é, principalmente, simplificar a portabilidade para diversos controladores. No entanto, esse requisito não se mostra necessário no caso do protótipo.

Outra modificação significativa está na ausência do software *switch* que realiza a função de interconexão dos roteadores virtuais e também atua na troca de tráfego entre os planos de controle e dados. No protótipo, o tráfego de pacotes compartilha o canal de gerência dos RVs, que é utilizado para troca de mensagens do protocolo RouteFlow. Com essa alteração, o *daemon* RF-Slave fica agora

[1] Foi utilizada a versão do OpenFlow mais recente implementada, apesar desta não apresentar suporte ao decremento do TTL. No entanto, este recurso está disponível a partir da versão 1.1.

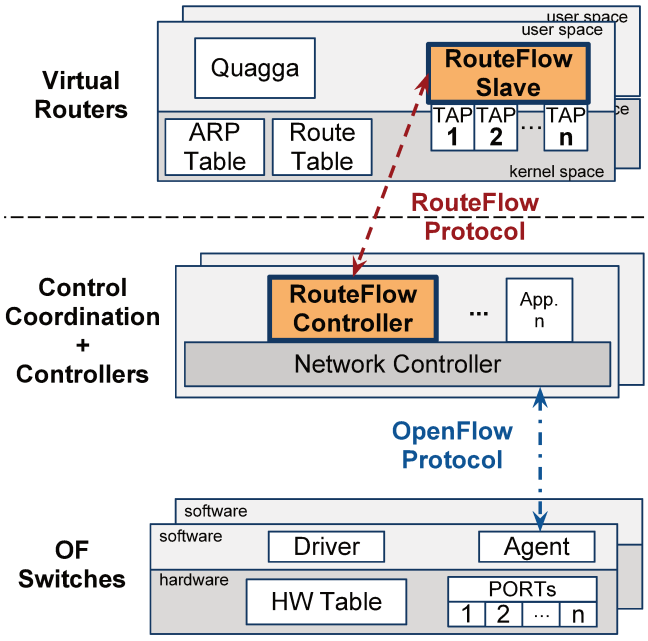


Fig. 4.1: Arquitetura simplificada implementada para o protótipo.

responsável também por encaminhar o tráfego de pacotes entre os planos de controle e dados.

Sem o software *switch* no plano virtual a interconexão dinâmica dos roteadores virtuais não é mais possível. Neste caso, os pacotes de controle trocados entre os RVs devem trafegar via plano de dados. Contudo, essa característica é desejada para simplificar o teste de avaliação da convergência da rede após falha, pois uma falha na infraestrutura física de rede será percebida automaticamente pelo plano de controle.

A base de dados para o armazenamento persistente do estado da rede também não foi implementada no protótipo. Este recurso é fundamental para o RouteFlow operar em um ambiente distribuído, com mais de um servidor para virtualização e com mais de um controlador de rede. Uma base de dados também mostra-se interessante para o compartilhamento das informações do estado da rede que podem ser utilizadas por interfaces de gerência pelo operador de rede. Porém, nenhum desses aspectos são necessários para demonstrar a viabilidade dos equipamentos de rede em operar com uma lógica de controle externa.

A seguir serão apresentadas as decisões de implementação de forma individual para os componentes RF-Slave e RF-Controller desenvolvidos para o protótipo.

4.1.1 RouteFlow-Slave

O componente RF-Slave é um aplicativo que executa em modo usuário e em *background* durante todo período de vida dos roteadores virtuais. Cada RV carrega o sistema operacional Linux com su-

porte ao módulo de *kernel* Universal TUN/TAP [47], que é responsável por criar dispositivos virtuais de rede. Juntamente com o *daemon* RF-Slave, o RV também executa a *engine* de roteamento Quagga, que é composta pelo aplicativo central zebra, além dos *daemons* que implementam os protocolos de roteamento IP: ospfd, ripd e bgpd. O detalhamento do esquema interno do RV está ilustrado na Figura 4.2.

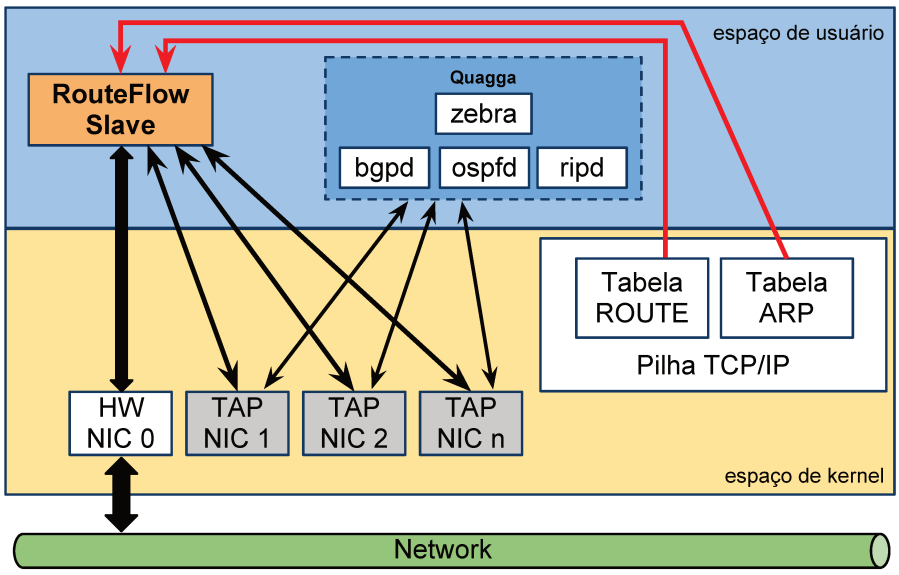


Fig. 4.2: Arquitetura interna do RV rodando o *daemon* RF-Slave.

No desenvolvimento do protótipo adotou-se dispositivos virtuais de rede TAP para troca de pacotes entre os planos de controle e dados. Através dos TAPs é possível injetar quadros no *kernel* do Linux com o propósito de serem tratados pela pilha TCP/IP ou pela *engine* de roteamento IP. Esses quadros são recebidos pelo RF através do canal de gerência que utiliza o protocolo RouteFlow.

As mensagens RouteFlow, que são recebidas do controlador de rede, são tratadas e os frames desencapsulados para serem injetados nas interfaces virtuais TAPs por meio de um *raw socket*. Após o RF-Slave escrever o conteúdo dos frames no *socket*, os pacotes são recebidos pelo sistema operacional como se estivessem chegado da rede. De forma recíproca, os pacotes provenientes da *engine* de roteamento ou da pilha TCP/IP do Linux são lidos do *socket* pelo RF-Slave e encapsulados em mensagens RouteFlow para serem enviados ao controlador de rede.

Para que os pacotes enviados ao plano de controle sejam escritos nas interfaces TAPs correspondentes às portas de rede dos equipamentos do plano de dados, torna-se necessário que a mensagem RouteFlow carregue a informação da porta de rede pela qual o quadro foi recebido. De forma equivalente, as mensagens contendo os pacotes enviados do plano de controle ao plano de dados devem conter a informação do TAP proveniente.

Um ponto de atenção para este modelo de troca de tráfego utilizando TAPs está na transição das

informações entre o espaço de *kernel* e usuário do sistema operacional.

A partir do estabelecimento da troca de tráfego entre os planos de controle e dados, ocorre o aprendizado dos vizinhos (tabela ARP) e a convergência dos protocolos resulta no preenchimento da tabela de roteamento. O sincronismo dessas informações com o plano de dados é realizado por um mecanismo de *polling* configurado com um intervalo de 100 milissegundos[2].

Para obter apenas as informações referentes às modificações que ocorrem nas tabelas, o mecanismo de *polling* executa *shell scripts*, que são responsáveis por:

- filtrar o conteúdo das tabelas com base nos nomes das interfaces TAPs;
- ordenar as entradas das tabelas;
- comparar as entradas atuais com aquelas obtidas na iteração anterior;
- identificar as modificações que devem ser sincronizadas com o plano de dados.

O mecanismo de detecção das atualizações das tabelas ARP e ROUTE adotado no protótipo visa manter a independência da arquitetura sobre a pilha de protocolos de roteamento. A obtenção das informações diretamente do sistema operacional possibilita substituir a *engine* de roteamento sem qualquer alteração de código. No entanto, a estratégia de *polling* não é um método eficiente. O mais adequado seria utilizar um mecanismo baseado em eventos. Das rotas aprendidas pelo Quagga são extraídas as informações necessárias para o RF-Controller converter uma rota em um fluxo OpenFlow, como discutido na Seção 3.2.2. Além da tabela de rotas IP, as entradas da tabela ARP também devem ser instaladas no plano de dados, pois é através delas que torna-se possível alcançar os nós das redes diretamente conectadas ao roteador. Quando no formato de fluxo, a diferença básica entre uma entrada da tabela FIB e uma entrada na tabela ARP é a máscara, que neste último será sempre 32 bits (*exact match*) e para as rotas irá depender da sub-rede em questão.

4.1.2 RouteFlow-Controller

O RF-Controller é a aplicação que executa sobre o controlador de rede NOX. Esse é um controlador OpenFlow *open source* cujo objetivo é prover uma plataforma simples para programação de aplicações de redes OpenFlow.

A troca de informações entre o controlador de rede e as aplicações baseia-se em um modelo de comunicação orientado a eventos. Para receber as informações referentes à mudança de estado da rede, a aplicação deve inscrever-se nos eventos de interesse a partir do registro de uma função de

[2]Valor definido empiricamente de modo que a frequência do *polling* não prejudique as demais tarefas executadas pelo RF-Slave, considerando o poder de processamento do servidor utilizado para virtualização.

callback. Além dos eventos de rede provenientes do núcleo do controlador, as aplicações também podem gerar eventos para outras aplicações.

Relativamente à proposta apresentada nesse trabalho, a aplicação RF-Controller precisa ter conhecimento da entrada e saída dos equipamentos de rede e também tratar os pacotes que forem encaminhados do plano de dados para o controlador. Portanto, a aplicação se inscreve para receber informações dos seguintes eventos de rede: Datapath_join_event, Datapath_leave_event, Packet_in_event.

A *callback* registrada para tratar os eventos de entrada de um novo *switch* programável (Datapath_join_event) é responsável por buscar um roteador virtual disponível para executar a lógica de controle do dispositivo de rede. Se houver um RV disponível, a associação é efetivada, o RV é informado e entradas de fluxos OpenFlow são instaladas no *switch* com o propósito de encaminhar explicitamente os pacotes de protocolos (RIP, OSPF e BGP) para o controlador. Caso não exista um RV disponível, o *switch* é adicionado a uma lista de espera e irá aguardar até a entrada de um novo roteador virtual.

Quando um *switch* OpenFlow deixa a rede, o evento Datapath_leave_event é gerado. Esta mudança de topologia implica na liberação do roteador virtual correspondente, que volta a ser um recurso disponível da topologia virtual. O RF-Controller envia um comando de *reset* ao RV para que toda configuração antiga seja removida.

Após a associação do *switch* programável com o roteador virtual que irá executar a lógica de controle, o conjunto está pronto para operar. A partir deste ponto, o evento Packet_in_event fica encarregado de tratar a subida do tráfego de pacotes do plano de encaminhamento para o plano de controle. Com o evento, chega também a informação por qual dispositivo de rede o pacote foi recebido e em qual porta. Com isso, é possível encaminhar o pacote encapsulado em uma mensagem RouteFlow para o RV correspondente.

Além dos eventos provenientes da rede, o RF-Controller atua como servidor para as instâncias de roteadores virtuais. Ele é responsável por registrar os RVs e tratar os pacotes recebidos dos mesmos RVs. Ambas as tarefas são implementadas em contextos de execução logicamente disjuntos com a utilização de duas *threads*.

No NOX, as *thread* operam no modelo operativo, ou seja, executam no mesmo contexto do ponto de vista do sistema operacional. Nesse modelo, o próprio controlador fica responsável por gerenciar a execução das *threads*. No entanto, o programador de aplicações deve se preocupar em não utilizar funções bloqueantes, o que causaria o bloqueio de todo controlador de rede.

A utilização de bibliotecas de *threads* fora do NOX causaria problemas de concorrência pelos recursos disponibilizados pelo controlador como, por exemplo, o canal de acesso aos dispositivos de rede. Por essa razão, o RF-Controller utiliza a biblioteca de *thread* cooperativa do NOX e se preocupa em utilizar funções com opção de *timeout* para não travar o processo geral.

Uma das *threads* fica responsável pelo estabelecimento das conexões com os RVs. Utiliza-se um *socket* TCP, que escuta uma porta configurável à espera de tentativas de conexão dos RVs. Após a conexão, as mensagens são tratadas pela outra *thread* que monitora todas as instâncias de conexões estabelecidas com cada RV do plano virtual.

4.2 Ambiente Experimental

O protótipo foi avaliado em uma rede experimental composta por cinco nós de encaminhamento conectados em uma topologia *mesh* e três terminais de rede, como ilustrado na Figura 4.3. Os elementos de encaminhamento são servidores Intel Core2Duo equipados com placas NetFPGA [48] com suporte ao OpenFlow versão 1.0. Os nós do plano de controle são virtualizados em um servidor Intel Core2Quad, onde também é executado o controlador OpenFlow, NOX.

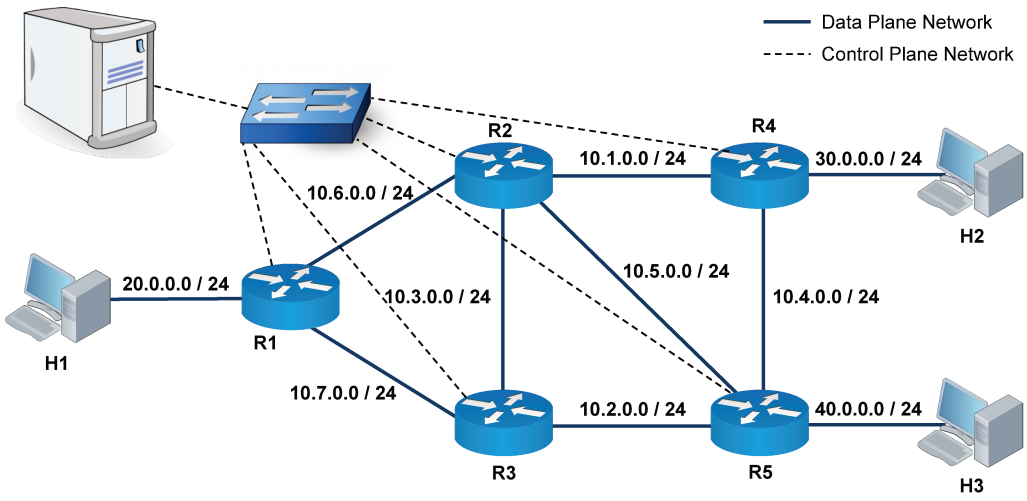


Fig. 4.3: Rede de teste montada com NetFPGAs.

A lógica de encaminhamento é programada na FPGA, que para o caso específico deste ambiente de avaliação executa a lógica de encaminhamento OpenFlow disponível no repositório oficial de Stanford [49].

4.3 Testes de validação

O objetivo princial dos testes foi avaliar o protótipo em uma rede OpenFlow real e não simulada, por isso a rede utilizada como *test-bed* apresenta um número limitado de nós devido à quantidade de equipamentos NetFPGAs disponíveis no laboratório. Entretanto, a quantidade de nós é suficiente

para uma avaliação de viabilidade da arquitetura de roteamento proposta, a partir da qual poderá ser explorado o modelo de roteamento como serviço.

A metodologia de avaliação está focada na análise detalhada das trocas de mensagens entre os planos de encaminhamento e de controle, que no modelo clássico (embarcado) resume-se à comunicação direta dos planos através de um barramento proprietário, enquanto na arquitetura proposta, diferentemente, a comunicação atravessa níveis intermediários.

O *overhead* na troca das mensagens pode ser quantificado através da comparação entre os tempos de convergência após falha de enlace de uma rede IP executando o protocolo OSPF no modelo embarcado e na arquitetura proposta. Este cenário mostra-se interessante, pois envolve a troca de inúmeras mensagens de controle entre os elementos de encaminhamento. Uma outra forma de quantificar o *overhead* é através da medida do RTT para o encaminhamento em *slow path*, no qual os pacotes de dados precisam ir até o plano de controle para serem processados.

4.3.1 Caminho das Mensagens e Pacotes

Antes dos testes, torna-se importante uma análise mais aprofundada sobre os caminhos, internamente à arquitetura, percorridos pelas mensagens e pacotes na implementação do protótipo a ser avaliado. Detalhes da construção dos componentes e das conexões entre eles interferem fortemente no desempenho da solução. Vale ressaltar que o desempenho não foi o foco do desenvolvimento do protótipo, onde o objetivo é validar o projeto e arquitetura do modelo proposto.

A Figura 4.4 apresenta os caminhos percorridos pelas mensagens e pacotes que transitam entre os planos de dados e controle na arquitetura proposta e implementada conforme descrito na Seção 4.1. Ainda na mesma figura, encontra-se uma comparação com o modelo clássico embarcado.

Na arquitetura proposta, os pacotes que necessitam ser processados pelo plano de controle devem percorrer três níveis de tratamento até chegar ao ponto onde serão processados. Os níveis podem ser identificados da seguinte forma:

- Encapsulamento na mensagem OpenFlow (1º nível);
- Mapeamento do RV e encapsulamento na mensagem RouteFlow (2º nível);
- Desencapsulamento e inserção do pacote na pilha TCP/IP do Linux (3º nível).

Quando o pacote é processado pelo *pipeline* do hardware OpenFlow e direcionado ao controlador (porta de CPU), ele é recebido pela aplicação OpenFlow embarcada responsável por encapsulá-lo em uma mensagem do protocolo para ser encaminhado ao controlador da rede. Este é o primeiro nível de tratamento do pacote em software.

No controlador de rede, o pacote é recebido pela aplicação RF-Controller que deve resolver o mapeamento do *datapath* pelo qual o pacote chegou no roteador virtual correspondente para o qual

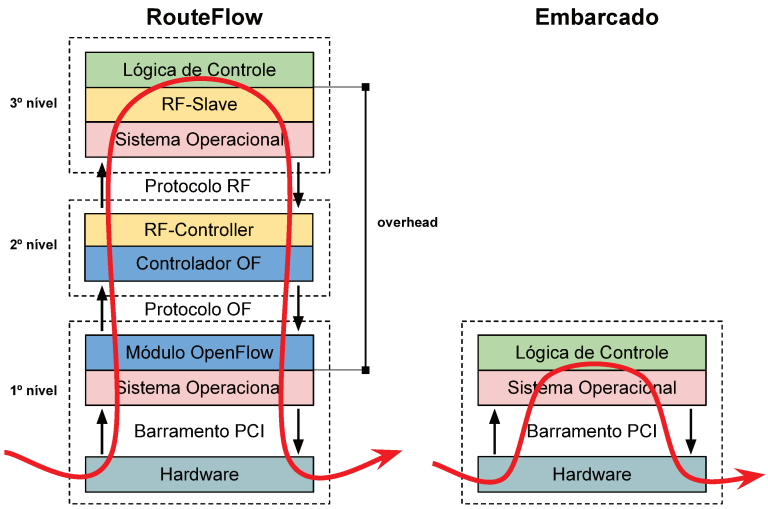


Fig. 4.4: Fluxo de pacotes na implementação do protótipo RouteFlow.

o pacote deve ser redirecionado. Antes de ser enviado, o pacote é encapsulado em uma mensagem RouteFlow. O segundo nível de tratamento ocorre no controlador OpenFlow e mais especificamente na aplicação RF-Controller.

Por último, o pacote é recebido no roteador virtual que representa o plano de controle do elemento do plano de dados. No entanto, as mensagem são recebidas pelo canal único de gerência. Por isso, o RF-Slave precisa da informação da mensagem RouteFlow para identificar em qual interface virtual o pacote deve ser injetado. A partir de então, o pacote chega à pilha TCP/IP do Linux, que irá processá-lo. O terceiro nível de tratamento, antes de chegar ao plano de controle, ocorre já no roteador virtual no *daemon* RF-Slave.

O caminho percorrido pelos pacotes que são provenientes do plano de controle e devem chegar ao plano de encaminhamento é o mesmo da subida, porém, no sentido inverso. Os pacotes que são encaminhados pela lógica de controle passam duas vezes em cada um dos níveis, na subida para o plano de controle e na descida para o plano de dados.

Em comparação, quando a lógica de controle encontra-se embarcada no equipamento, o pacote percorre um único nível referente à aplicação que controla o hardware e recebe o pacote para injetá-lo na pilha TCP/IP do sistema operacional. Portanto, o tempo gasto no caminho do pacote a partir da entrada no dispositivo até atingir a lógica de controle é bastante reduzido em relação à arquitetura proposta.

O objetivo da avaliação é quantificar o tempo "a mais" (*overhead*) gasto nos níveis intermediários até que o pacote seja devidamente tratado pela lógica de controle. A representatividade deste *overhead* no tempo total da tomada de decisão impacta diretamente na análise de viabilidade do modelo proposto.

4.3.2 Tempo de convergência com tráfego *line rate*

O objetivo do teste é medir o tempo de convergência de uma rede IPv4, rodando o protocolo de roteamento OSPF, após a queda de um enlace e, a partir da análise das trocas de mensagens entre os planos de controle e encaminhamento, quantificar o *overhead* gerado pela arquitetura RouteFlow nesta implementação.

Para tornar o cenário de rede mais próximo do real foi utilizado um equipamento gerador e analisador de tráfego configurado para transmitir pacotes IPv4 de 1500 bytes em ambos os sentidos (*fullduplex*) a uma taxa de 1 Gbps. Desta forma, garante-se que esta solução de roteamento com lógica de controle remota sobre uma rede programável é capaz de operar com tráfego em velocidade de linha uma vez que os pacotes são processados por um hardware dedicado (FGPA).

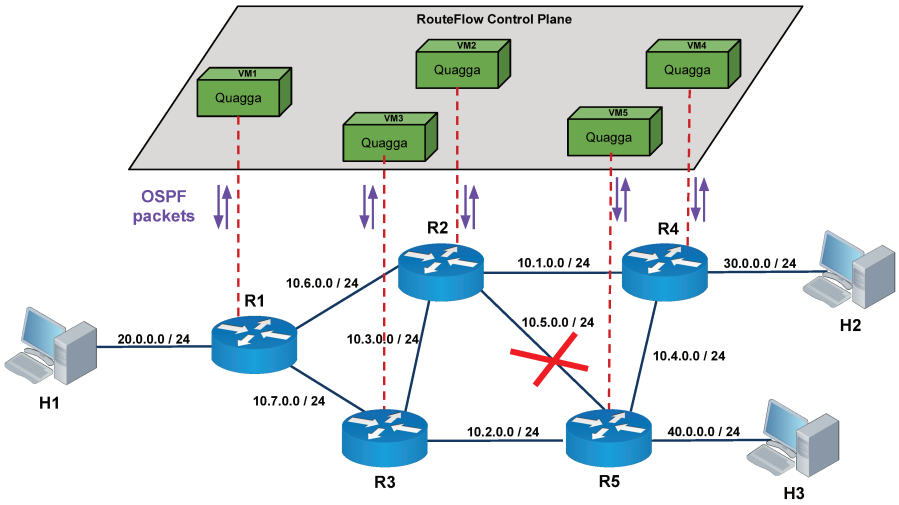
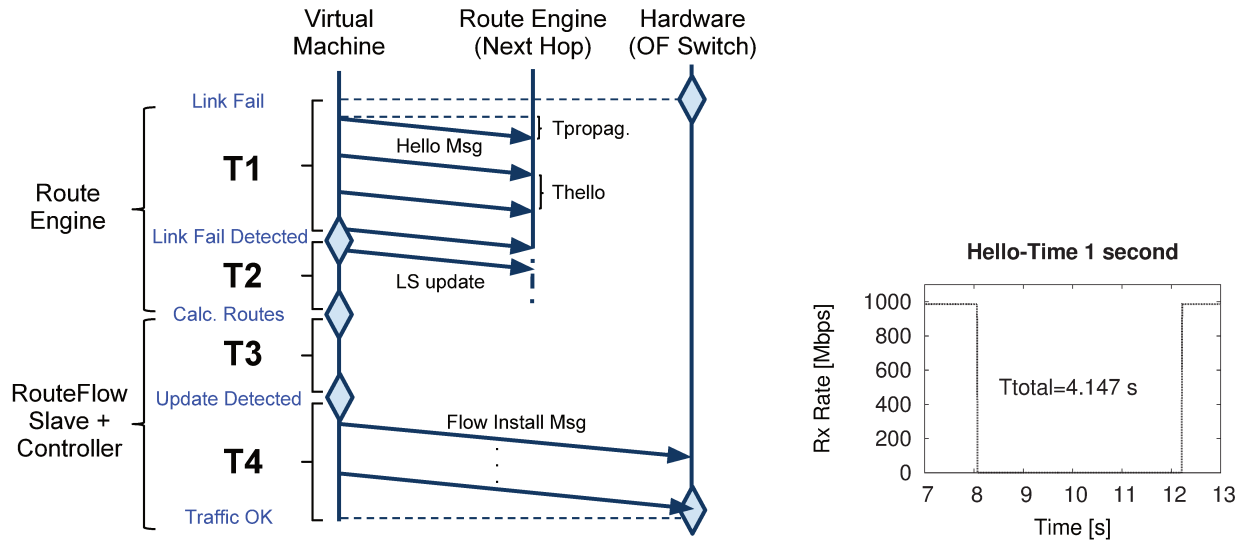


Fig. 4.5: Configuração do ambiente para teste do tempo de convergência.

Os tempos de convergência estão ligados ao parâmetro de configuração *hello-time* do OSPF, porém, é necessário uma análise mais detalhada para identificarmos o impacto de utilizar uma pilha de protocolos remota ao equipamento. Neste sentido, fragmenta-se o tempo de convergência em quatro partes (Figura 4.6(a)), a saber: (T1) tempo que o protocolo leva para identificar a falha de *link*; (T2) tempo gasto pelo OSPF com as trocas de mensagens e o cálculo de novas rotas; (T3) tempo necessário para o RouteFlow-Slave detectar as modificações na tabela de roteamento no Linux; e (T4) tempo requerido para o RouteFlow-Slave encaminhar as mensagens de instalação de fluxos para o RouteFlow-Controller e este, através do controlador NOX, instalar os fluxos em hardware. De fato, os dois primeiros tempos (T1 e T2) são inerentes do protocolo de roteamento, portanto o impacto do RouteFlow está claramente presente nos tempos T3 e T4.

O gráfico da Figura 4.6(b) mostra o tempo de convergência da rede após a falha de um *link* com o protocolo OSPFv3 configurado com o *hello-interval* em 1s. Este é o principal parâmetro do OSPF



(a) Fluxo de eventos durante convergência.

(b) Exemplo de tempo de convergência.

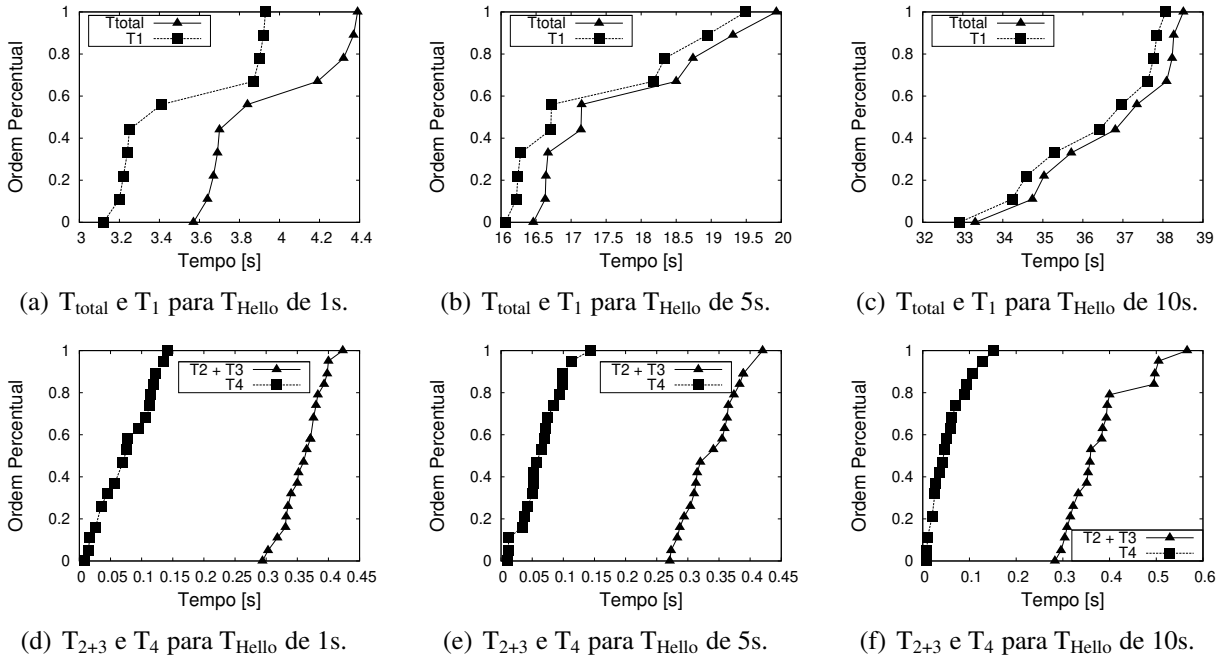
Fig. 4.6: Convergência do OSPF após falha.

diretamente relacionado ao tempo de detecção de falhas. O *hello-interval* foi configurado para 1 segundo, 5 segundos e 10 segundos, correspondendo aos valores típicos da indústria para enlaces Ethernet [50]. O *dead-interval* usado foi o recomendado de quatro vezes o *hello-interval*.

Com o intuito de fragmentar os tempos mostrados no gráfico da Figura 4.6(b), foram feitas análises das mensagens enviadas e recebidas pelo hardware OpenFlow para o controlador. Nos resultados está incluso o tempo gasto pelas mensagens para transitar entre o dispositivo de encaminhamento, o controlador e o RV. O tempo T1 pode ser obtido a partir do intervalo entre o instante em que ocorre a interrupção do tráfego até a primeira mensagem de LS-Update gerada pela *engine* de roteamento ao detectar a falha; em seguida é necessário o tempo T2 + T3 para que o RF-Slave inicie o envio da primeira mensagem de alteração de fluxo e, por último, o T4 finaliza o processo com o restabelecimento do tráfego. Nos gráficos da Figura 4.7 e na Tabela 4.1, apresenta-se os tempos conforme a fragmentação proposta anteriormente, em que cada um deles possui o valor de tempo abaixo do qual se encontram metade dos resultados e o valor de tempo abaixo do qual se encontram 90% dos testes num total de 20 repetições para cada uma das três configurações de *hello-time*.

Tab. 4.1: *Tempos de convergência após falha.*

Hello Time	T_1 [s]		T_2+T_3 [s]		T_4 [s]		T_{total} [s]	
OSPF	$T_{med.}$	$T_{90\%}$	$T_{med.}$	$T_{90\%}$	$T_{med.}$	$T_{90\%}$	$T_{med.}$	$T_{90\%}$
1 sec.	3.249	3.923	0.360	0.398	0.070	0.123	3.700	4.373
5 sec.	16.713	18.937	0.320	0.389	0.057	0.099	17.135	19.308
10 sec.	36.406	37.846	0.358	0.497	0.042	0.106	36.807	38.266



(a) T_{total} e T_1 para T_{Hello} de 1s. (b) T_{total} e T_1 para T_{Hello} de 5s. (c) T_{total} e T_1 para T_{Hello} de 10s.

(d) T_{2+3} e T_4 para T_{Hello} de 1s. (e) T_{2+3} e T_4 para T_{Hello} de 5s. (f) T_{2+3} e T_4 para T_{Hello} de 10s.

Fig. 4.7: Fragmentação do tempo de convergência do OSPF.

Conforme os dados apresentados na Tabela 4.1, o tempo total de convergência é bem próximo do tempo T1, ou seja, o tempo de detecção da falha. Portanto, o tempo restante gasto com o cálculo de rotas, detecção de modificações da tabela de roteamento pelo RouteFlow-Slave e a instalação do fluxo têm pouco impacto no tempo de convergência. Apesar dos tempos T2 e T3 não terem sido analisados separadamente, sabe-se que o tempo de *polling* para verificar atualizações da tabela de roteamento e ARP do Linux é fixo em 100 ms, ou seja, da soma T2 + T3, o T3 representa até 100 milissegundos no pior caso. Vale ressaltar que foi utilizada a estratégia de *polling* por simplificação; no entanto, podem ser feitas otimizações para reduzir substancialmente o tempo T3. Uma opção seria fazer com que a aplicação se registrasse no Zebra (módulo central do Quagga) para receber notificações sobre a RIB, por consequência, a informação chegaria ao RF-Slave de forma muito mais rápida e eficiente.

Dada a baixa representatividade dos tempos T3 e T4 sobre o tempo total, é razoável afirmar a viabilidade da solução RouteFlow dentro do contexto proposto.

4.3.3 Encaminhamento em *Slow Path* e *Fast Path*

Uma outra forma de avaliar o impacto no desempenho de um roteador que roda uma pilha remota de protocolos de roteamento é quantificar os tempos para encaminhamento do tráfego em *slow path* e *fast path*. *Slow path* refere-se ao encaminhamento lento que ocorre no plano de controle, em software, por exemplo, quando o roteador precisa se comunicar com um nó de uma rede diretamente

conectada a ele e ainda não ocorreu o aprendizado do endereço MAC do destino, que é necessário para o encaminhamento em hardware [3]. Este cenário tipicamente ocorre para os nós que entraram na rede ou que ficaram sem comunicação por um longo período, por isso a relevância desta operação nos roteadores é baixa, embora necessária. O *fast path* refere-se ao encaminhamento rápido, em hardware, que ocorre após o aprendizado dos endereços dos nós vizinhos e o preenchimentos das tabelas em hardware. Portanto, numa transmissão de dados, o *slow path* ocorre, quando necessário, apenas para o primeiro pacote, a partir do qual será realizado o aprendizado e o restante dos pacotes serão processados em *line rate*.

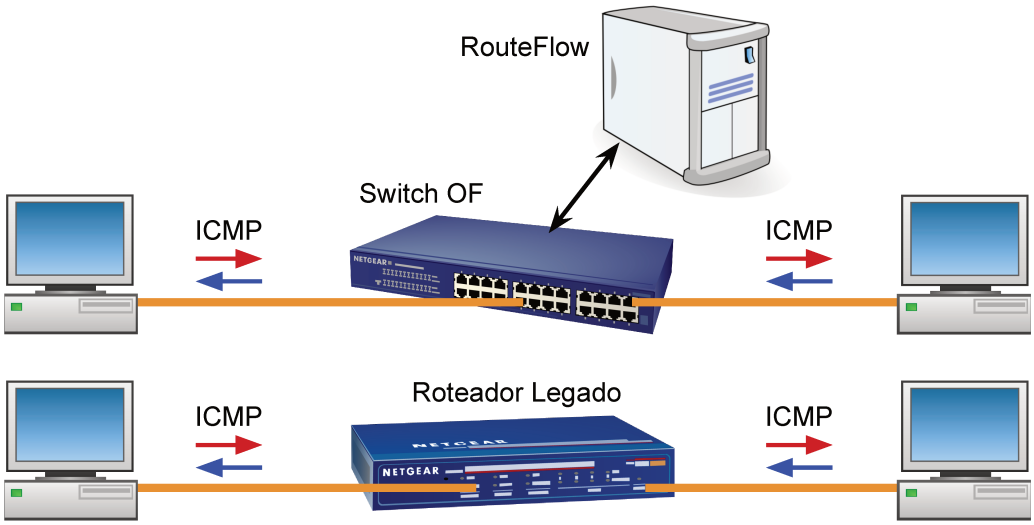


Fig. 4.8: Cenário de teste para encaminhamento de tráfego em *Slow* e *Fast Path*.

Através deste teste é possível avaliar a diferença de tempo entre um encaminhamento por software e por hardware de um roteador com protocolos embarcado e da arquitetura proposta, cujo plano de controle executa remotamente.

Com este objetivo foram realizados testes de PING (ICMP) entre dois *hosts* diretamente conectados a portas distintas, de um mesmo roteador, configuradas em redes diferentes e partindo-se do ponto no qual os *hosts* não têm conhecimento do endereço MAC dos seus *gateways* e o roteador também não contempla ambos *hosts* em suas tabelas. Após o aprendizado dos endereços, são obtidos os valores de *fast path*. Foram utilizados os seguintes *switches layer 3*: CISCO 3560-e Catalyst, Extreme x450-e, CPqD Enterprise (protótipo) e a arquitetura proposta. Os resultados dos testes estão apresentados na Tabela 4.2.

Foi observado nos testes com a arquitetura proposta que tanto o primeiro pacote ICMP *request* quanto o *reply* são encaminhados por software. O motivo deste comportamento está relacionado

[3]Conceitualmente, um elemento de rede pode tratar o cabeçalho de uma camada (ex.: IP) apenas se o endereço de destino da camada inferior (ex.: Ethernet) for o dele próprio.

Tab. 4.2: *Tempo de resposta ICMP.*

Equipamento	Slow Path [ms]		Fast Path [ms]	
	$T_{med.}$	$T_{90\%}$	$T_{med.}$	$T_{90\%}$
CISCO 3560-e Catalyst	5.46	7.75	0.100	0.130
Extreme x450-e	11.30	14.00	0.106	0.141
CPqD Enterprise	14.20	17.30	0.101	0.147
Arquitetura Proposta	116.00	138.00	0.082	0.119

principalmente ao tempo de *polling* de 100ms para que as atualizações nas tabelas do Linux sejam detectadas. No momento da resposta ICMP, os fluxos ainda não estão instalados e o pacote precisa novamente ser direcionado ao controlador para ser encaminhado por software. Portanto, com uma otimização da forma com que o RF-Slave passa a ter conhecimento das atualizações, é razoável esperar um tempo consideravelmente menor para este teste, como exemplificado na Seção 4.3.2. Outro ponto a ser destacado sobre o resultado de *slow path*, consideravelmente superior quando comparado aos dispositivos com software embarcado, é o desempenho do controlador OpenFlow (NOX) utilizado nos testes, cuja proposta é a de prover uma plataforma simples de desenvolvimento de aplicações de rede sem o compromisso de manter o foco em desempenho. Neste sentido já foram identificadas possíveis otimizações que devem trazer ganhos significativos no tempo de processamento das mensagens no controlador. Uma delas seria tornar o controlador multi-tarefa de forma a realizar o processamento paralelo das mensagens. Ainda, a implementação de mensagens que agreguem mais informação resultaria em um número de chaveamentos de contextos menor da aplicação para processar cada pacote recebido.

Em contrapartida, pode-se observar o melhor desempenho de *fast path* da arquitetura proposta, que é a forma de encaminhamento mais relevante. Este fato pode ser explicado pelo rápido encaminhamento da tabela de fluxos quando comparado ao *longest prefix match*, que é utilizado nas tabelas de encaminhamento IP.

Capítulo 5

Considerações Finais

5.1 Discussão

Nesta seção serão discutidos brevemente alguns aspectos que merecem considerações adicionais.

- **Isolamento:** Durante os testes de avaliação foi possível notar variações de desempenho do sistema que, de modo geral, podem estar relacionadas à questão do isolamento entre as instâncias de roteamento virtualizadas. Problemas de isolamento entre MVs já foram identificados na literatura, principalmente na análise do processamento do sistema em qualquer solução montada em cima de ambientes virtuais. No entanto, algumas medidas para mitigar instabilidades de desempenho dos roteadores virtuais podem ser aplicadas. Uma delas é a utilização de (um *cluster* de) servidores com grande poder de processamento, o que evitaria atingir a capacidade máxima da CPU mesmo quando todos os RVs necessitarem de processamento para o cálculo de rotas durante a convergência da rede. As inúmeras ferramentas e tecnologias de virtualização disponíveis atualmente trazem uma grande diversidade de funcionalidades que acarretam em impactos significativos de desempenho. A escolha de um virtualizador mais robusto pode também contribuir para estabilidade e bom desempenho do ambiente virtual.

- **Escalabilidade:** O *testbed* esteve limitado ao número de equipamentos disponíveis, o que inviabilizou uma avaliação de escalabilidade do RouteFlow neste trabalho. É clara a necessidade de uma infraestrutura virtual com servidores distribuídos e alta capacidade de processamento para suportar o aumento do tamanho da rede sem qualquer prejuízo ao funcionamento do sistema. Porém, sabe-se que a capacidade de processamento disponível em soluções comerciais aumenta rapidamente com o passar dos anos (lei de Moore), além de uma redução progressiva dos custos envolvidos, contribuindo assim com a relação custo-eficiência e a escalabilidade do RouteFlow. Outra questão pertinente é o gargalo observado nas implementações disponíveis do

OpenFlow quanto ao tamanho da tabela de fluxos (entre 2 e 4 mil) e a capacidade de instalação de novas entradas por segundo (valor em torno de 100 fluxos/seg.). Entendemos que estas limitações são transitórias e serão contornadas com a maturidade da tecnologia, incluindo o acesso a múltiplas tabelas de hardware a partir da versão 1.1 do OpenFlow.

- **Virtualização de redes:** A partir de uma arquitetura baseada na separação entre os planos de controle e dados e no isolamento, tanto do tráfego na infraestrutura física quanto nas instâncias de controle, torna-se possível explorar os casos de uso abordados na Seção 3.6, por exemplo o roteamento como serviço [40]. Desta forma podemos ter topologias lógicas independentes sobre uma mesma rede e que executem protocolos distintos, resultando em uma abordagem de virtualização com um aproveitamento melhor dos recursos da infraestrutura, que agora pode ser compartilhada com diferentes propósitos em alinhamento com as propostas de arquiteturas pluralistas [51].

- **Confiabilidade:** A confiabilidade é um parâmetro crítico para os sistemas de comunicação da atualidade. A rede deve manter-se em operação o máximo de tempo possível sem interrupção dos serviços. Esse requisito denomina-se alta disponibilidade. Um sistema apresenta alta disponibilidade se a soma dos períodos de interrupção forem inferiores a 5,26 minutos em um ano, ou seja, mantiver uma disponibilidade de 99,999%. Para alcançar essa meta, os sistemas utilizam técnicas de replicação de dados e redundância de operação. O RouteFlow apresenta uma arquitetura adequada ao processo de replicação de dados com a utilização de uma base de dados para armazenar os estados da rede. Diversos bancos de dados possuem funcionalidades de replicação e sincronismo com garantia da integridade dos dados. Dessa forma, os estados da rede estariam protegidos contra falha dos servidores de armazenamento. Outra medida de prevenção a falhas está relacionada à redundância do contexto de execução dos roteadores virtuais e do RF-Server. A proteção dos roteadores virtuais pode ser obtida através de virtualizadores que apresentam essa funcionalidade. Já o RF-Server pode executar também em uma máquina virtual, o que tornaria o processo de redundância bastante simples. Outra alternativa seria a utilização de um *framework* alta disponibilidade, como exemplo o OpenSAF, que é uma implementação de código aberto da especificação do *Service Availability Forum* (SAForum).

5.2 Conclusões e Trabalhos Futuros

A principal contribuição desta tese consistiu em uma abordagem de redução da complexidade individual dos elementos de rede, direcionando a evolução das redes de pacotes para uma arquitetura baseada em um conjunto de elementos mais simples e com uma distribuição maior de responsabilida-

des. Com o plano de controle externo ao dispositivo de rede, passa a existir maior independência entre este plano e o de encaminhamento, de forma que ambos escalem e evoluam separadamente. Nesse sentido, a tese apresenta um modelo de plano de controle externo ao comutador de rede sobre uma infraestrutura de equipamentos programáveis, apoiando uma evolução das redes onde a conectividade IP pode se tornar um recurso comum ofertado em um modelo de plataforma como serviço [40].

Os resultados alcançados na avaliação do protótipo sugerem o grande potencial do RouteFlow como solução de roteamento para redes de campus/corporativas com o ganho de flexibilidade e poder de inovação. As questões pertinentes ao desempenho não inviabilizam a proposta, uma vez que otimizações já identificadas e a maturidade do protocolo OpenFlow trarão significativas melhorias de desempenho.

No site do projeto RouteFlow[1] encontra-se uma lista atualizada dos trabalhos em andamento. Recentemente foi integrado o uso do *software switch* na interconexão dos RVs, dando suporte a ambientes dinâmicos e fisicamente distribuídos. Também foi implementado o mecanismo de detecção de atualizações da tabela de roteamento destacado na secão 4.3.2.

Dentre os próximos passos identificados, será dada ênfase ao protocolo BGP, que tem sido um dos grandes desafios e causa de problemas nas arquiteturas de roteamento IP. Uma nova abordagem baseada no modo de agregação discutido na Seção 3.6.3 trará otimizações para o roteamento IP inter-domínio e ao mesmo tempo simplificará o plano de gerência. Além do modo de agregação, a multiplexação apresentada na Seção 3.6.2 será explorada no contexto de virtualização de redes e roteamento como serviço.

Em paralelo, serão realizados trabalhos relativos aos recursos da tecnologia de virtualização com foco na otimização do ambiente virtual [37]. A utilização de técnicas avançadas para a migração e o gerenciamento do estado (ex: *checkpointing*, *rollback*) das MVs combinadas ao uso de múltiplos servidores irão contribuir para aumentar a disponibilidade do sistema e minimizar os pontos fracos de um controle logicamente centralizado.

[1] http://go.cpqd.com.br/routeflow

Referências Bibliográficas

[1] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending networking into the virtualization layer. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[2] Rob Sherwood, Glen Gibb, Kok-kiong Yap, Guido Appenzeller, Martin Casado, Nick Mckeown, and Guru Parulkar. Flowvisor : A network virtualization layer flowvisor. *Network*, page 15, 2009.

[3] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jonathan B. Postel, Lawrence G. Roberts, and Stephen S. Wolff. A brief history of the internet. *CoRR*, cs.NI/9901011, 1999.

[4] L. G. Roberts. The evolution of packet switching. *Proceedings of the IEEE*, 66(11):1307–1313, November 1978.

[5] András Császár, Gábor Enyedi, Gábor Rétvári, Markus Hidell, and Peter Sjödin. Converging the evolution of router architectures and ip networks. *IEEE Network*, 21(4):8–14, 2007.

[6] James Hamilton. Networking: The last bastion of mainframe computing. Disponível em: <http://perspectives.mvdirona.com/2009/12/19/NetworkingTheLastBastion OfMainframeComputing.aspx>. Acesso em: 12 maio 2012, Dec 2009.

[7] Muhammad Bilal Anwer, Murtaza Motiwala, Mukarram bin Tariq, and Nick Feamster. Switchblade: a platform for rapid deployment of network protocols on programmable hardware. SIGCOMM '10, pages 183–194, New York, NY, USA, 2010. ACM.

[8] Kok-Kiong Yapy Guido Appenzeller Martin Casado Nick McKeowny Guru Parulkary Rob Sherwood, Glen Gibby. Can the production network be the testbed? OSDI'10, pages 1–14. USENIX Association, 2010.

[9] Dobrescu and et al. Routebricks: exploiting parallelism to scale software routers. SOSP '09, pages 15–28, New York, NY, USA, 2009. ACM.

[10] GNU Quagga Project. Disponível em: <http://www.quagga.org>. Acesso em: 19 janeiro 2012.

[11] The XORP Project. extensible open router platform. Disponível em: <http://www.xorp.org>. Acesso em: 19 janeiro 2012.

[12] Vyatta. Series 2500. Disponível em: <http://vyatta.com/downloads/datasheets/vyatta_2500_datasheet.pdf>. Acesso em: 11 agosto 2011.

[13] Katerina Argyraki, Salman Baset, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Eddie Kohler, Maziar Manesh, Sergiu Nedevschi, and Sylvia Ratnasamy. Can software routers scale? PRESTO '08, pages 21–26, New York, NY, USA, 2008. ACM.

[14] Tammo Spalink, Scott Karlin, Larry Peterson, and Yitzchak Gottlieb. Building a robust software-based router using network processors. *SIGOPS Oper. Syst. Rev.*, 35:216–229, October 2001.

[15] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. Packetshader: a gpu-accelerated software router. SIGCOMM '10, pages 195–206, New York, NY, USA, 2010. ACM.

[16] Marcelo Ribeiro Nascimento, Christian Esteve Rothenberg, Marcos Rogério Salvador, and Maurício Ferreira Magalhães. QuagFlow: partnering Quagga with OpenFlow. *SIGCOMM Comput. Commun. Rev.*, 40:441–442, August 2010.

[17] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38:69–74, 2008.

[18] Gude and et al. NOX: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, 2008.

[19] Ramachandran Ramjee, Furquan Ansari, Martin Havemann, T. V. Lakshman, Thyagarajan Nandagopal, Krishan K. Sabnani, and Thomas Y. C. Woo. Separating control software from routers. In *COMSWARE*. IEEE, 2006.

[20] N. M. Mosharaf Kabir Chowdhury and Raouf Boutaba. Network virtualization: state of the art and research challenges. *Comm. Mag.*, 47:20–26, July 2009.

[21] OpenFlow Switch Speci
cation Version 1.1.0 Implemented (Wire Protocol 0x02), February 28, 2011. Disponível em: <http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf>. Acesso em: 10 abril 2012.

[22] Beacon Concluence. Disponível em: <https://openflow.stanford.edu/display/Beacon/Home>. Acesso em: 19 janeiro 2012.

[23] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A distributed control platform for large-scale production networks. In *In Proc. OSDI*, 2010.

[24] Trema OpenFlow Controller. Disponível em: <https://github.com/trema/trema>. Acesso em: 19 janeiro 2012.

[25] Floodlight OpenFlow Controller. Disponível em: <http://floodlight.openflowhub.org>. Acesso em: 12 abril 2012.

[26] Thomas Anderson, Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming the internet impasse through virtualization. *Computer*, 38:34–41, April 2005.

[27] Jon Turner and David Taylor. Diversifying the internet. In *In Proc. IEEE GLOBECOM*, pages 755–760, 2005.

[28] Nick Feamster, Lixin Gao, and Jennifer Rexford. How to lease the internet in your spare time. *ACM SIGCOMM Computer Communication Review*, 37:61–64, 2007.

[29] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Trans. Netw.*, XX:XX, XX 2011.

[30] Martin Casado, Teemu Koponen, Rajiv Ramanathan, and Scott Shenker. Virtualizing the network forwarding plane. In *Presto '10*, August 2010.

[31] Hong Yan, David A. Maltz, T. S. Eugene Ng, Hemant Gogineni, Hui Zhang, and Zheng Cai. Tesseract: A 4d network control plane. In *in Proc. Networked Systems Design and Implementation*, 2007.

[32] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A Clean Slate 4D Approach to Network Control and Management. *SIGCOMM Comput. Commun. Rev.*, 35(5):41–54, 2005.

[33] R. Ramjee K. Sabnani T. Woo T.V. Lakshman, T. Nandagopal. The softrouter architecture. In HotNets, 2004.

[34] Nadi Sarrar, Anja Feldmann, Steve Uhlig, Rob Sherwood, and Xin Huang. FIBIUM - towards hardware accelerated software routers. In *EuroView 2010*, August 2010.

[35] Raffaele Bolla, Roberto Bruschi, Guerino Lamanna, and Andrea Ranieri. Drop: An open-source project towards distributed sw router architectures. In *GLOBECOM*, pages 1–6. IEEE, 2009.

[36] Forwarding and Control Element Separation (ForCES) Protocol Specification. Disponível em: <http://tools.ietf.org/html/rfc5810>. Acesso em: 12 maio 2012.

[37] Carlos N. A. Corrêa, Sidney Lucena, Christian E. Rothenberg, and Marcos R. Salvador. Desempenho de soluções de virtualização para plano de controle de roteamento de redes virtuais. In *SBRC 2011*, May 2011.

[38] Norman Finn, Dinesh Mohan, and Ali Sajassi. Connectivity fault management, Sep 2007.

[39] Karthik Lakshminarayanan, Ion Stoica, and Scott Shenker. Routing as a service. Technical Report UCB/CSD-04-1327, EECS Department, University of California, Berkeley, 2004.

[40] Eric Keller and Jennifer Rexford. The 'Platform as a Service' model for networking. In *INM/WREN 10*, April 2010.

[41] Carlos Corrêa, Sidney Lucena, Christian Rothenberg, Marcos Salvador, editor. *Uma plataforma de roteamento como serviço baseada em redes definidas por software*, Workshop de Gerência e Operação de Redes e Serviços (WGRS). SBRC'12, Ouro Preto, MG, Brazil, Abril 2012.

[42] The BIRD Project. Bird internet routing daemon. Disponível em: <http://bird.network.cz>. Acesso em: 19 janeiro 2012.

[43] Lxc Linux Containers. Disponível em: <http://lxc.sourceforge.net>. Acesso em: 19 janeiro 2012.

[44] VMware Virtualization Software for Desktops, Servers & Virtual Machines for Public and Private Cloud Solutions. Disponível em: <http://www.vmware.com>. Acesso em: 19 janeiro 2012.

[45] Oracle VM VirtualBox. Disponível em: <https://www.virtualbox.org>. Acesso em: 19 janeiro 2012.

[46] Xen Hypervisor, the powerful open source industry standard for virtualization. Disponível em: <http://xen.org>. Acesso em: 19 janeiro 2012.

[47] Universal TUN/TAP driver. Disponível em: <http://vtun.sourceforge.net/tun>. Acesso em: 19 janeiro 2012.

[48] NetFPGA. Disponível em: <http://netfpga.org>. Acesso em: 28 abril 2012.

[49] OpenFlow - Enabling Innovation in Your Network. Disponível em: <http://www.openflow.org>. Acesso em: 10 abril 2012.

[50] VPN 3000 Series Concentrator Reference Volume I. Disponível em: <http://www.cisco.com/en/US/docs/security/vpn3000/vpn3000_47/configuration/config.html>. Acesso em: 11 agosto 2011.

[51] Natalia Fernandes, Marcelo Moreira, Igor Moraes, Lyno Ferraz, Rodrigo Couto, Hugo Carvalho, Miguel Campista, Luís Costa, and Otto Duarte. Virtual networks: isolation, performance, and trends. *Annals of Telecommunications*, pages 1–17, October 2010.