UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

RAFAEL EMERICK ZAPE DE OLIVEIRA

KEYFLOW: UMA PROPOSTA PARA O PROVIMENTO DE CONECTIVIDADE FABRIC NO NÚCLEO DE REDES DEFINIDAS POR SOFTWARE

VITÓRIA
2014

RAFAEL EMERICK ZAPE DE OLIVEIRA

KEYFLOW: UMA PROPOSTA PARA O PROVIMENTO DE CONECTIVIDADE FABRIC NO NÚCLEO DE REDES DEFINIDAS POR SOFTWARE

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Magnos Martinello.

Co-orientador: Prof. Dr. Moisés Renato Nunes Ribeiro.

VITÓRIA
2014

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Setorial Tecnológica,
Universidade Federal do Espírito Santo, ES, Brasil)

O48k

Oliveira, Rafael Emerick Zape de, 1985-
KeyFlow : uma proposta para o provimento de conectividades *fabric* no núcleo de redes definidas por software / Rafael Emerick Zape de Oliveira. – 2014.
100 f. : il.

Orientador: Magnos Martinello.
Coorientador: Moisés Renato Nunes Ribeiro.
Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Redes de computadores – Protocolos. 2. Comutação de pacotes (Transmissão de dados). 3. Redes definidas por software. 4.OpenFlow. I. Martinello, Magnos. II. Ribeiro, Moisés Renato Nunes. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 621.3

RAFAEL EMERICK ZAPE DE OLIVEIRA

KEYFLOW: UMA PROPOSTA PARA O PROVIMENTO DE CONECTIVIDADE FABRIC NO NÚCLEO DE REDES DEFINIDAS POR SOFTWARE

Dissertação submetida ao programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para a obtenção do Grau de Mestre em Engenharia Elétrica.

Aprovada em 26 de junho de 2014.

COMISSÃO EXAMINADORA

Prof. Dr. Magnos Martinello
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Moisés Renato Nunes Ribeiro
Universidade Federal do Espírito Santo
Co-orientador

Prof. Dr. Michael Anthony Stanton
Universidade Federal Fluminense

Prof. Dr. Anilton Sales Garcia
Universidade Federal do Espírito Santo

A Ana, Márcia, Rita e Adão.

Agradecimentos

Chega ao fim mais uma etapa desta minha missão atual. Agradeço a Deus pela oportunidade de concluir mais este objetivo e por permitir que este momento chegasse com a ajuda de pessoas que foram fundamentais nesta caminhada. Dentre elas, o destaque maior fica para minha família. A minha mãe Rita e ao meu Pai Adão, por me criar, educar, incentivar e orientar no melhor caminho. A minha companheira Márcia por todo apoio e incentivo em todos os momentos que achei que não conseguiria chegar até aqui. Agradeço também à minha pequena Ana Luiza pela sua compreensão durante alguns períodos de minha ausência e pela motivação diária que sua existência trouxe para alavancar esta minha atual passagem terrena.

Agradeço também aos meus orientadores, prof. Magnos e prof. Moiséis, por todas as oportunidade de trocas de ideias, por compartilharem comigo oportunidades acadêmicas que foram muito importantes para meu amadurecimento durante esta fase. Agradeço também ao Romulo Vitoi pelo grande apoio no desenvolvimento deste trabalho. Aos amigos do PoP-ES, agradeço pelo grande incentivo e pela disponibilidade em ajudar durante os experimentos que lá realizei. Aos amigos pessoais - que não me arriscarei a citar os nomes aqui pela falta de espaço - por me ajudarem em diversos momentos de descontração e incentivo. Agradeço a todos os amigos que me intuíram e orientaram para que eu chegasse até aqui.

Não posso deixar de agradecer também à Universidade Federal do Espírito Santo, onde me formei, sendo fundamental para a transformação pessoal e profissional pela qual passei nestes últimos anos. Ao professores do PPGEE com quem pude aprofundar meus conhecimentos em várias disciplinas. Agradeço também à FAPES, e aos professores envolvidos, pelo suporte oferecido para o desenvolvimento deste trabalho por meio do Projeto Cidades Digitais.

Muito obrigado a todos vocês!

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Abreviaturas

ACL	*Access Control List*
API	*Application Programming Interface*
ASIC	*Application Specific Integrated Circuit*
ATM	*Asynchronous Transfer Mode*
CAM	*Content Access Memory*
CE	*Customer Edge*
CLI	*Command Line Interface*
CPU	*Central Processing Unit*
DCAN	*Develved Control of ATM Networks*
E/S	Entrada e Saída
GSMP	*General Switch Management Protocol*
IETF	*Internet Engineering Task Force*
ISP	*Internet Service Provider*
L3VPN	*Layer 3 Virtual Private Network*
LTS	*Long Term Support*
MV	Máquina Virtual
ONF	*Open Networking Foundation*
OXM	*OpenFlow eXtensive Match*
PBB	*Provider Backbone Bridge*
QoS	*Quality of Service*

RAM	*Random Access Memory*
RFC	*Request For Comments*
RPC	*Remote Procedure Call*
SDN	*Software Defined Networking*
SNMP	*Simple Network Management Protocol*
TCAM	*Ternary Content Access Memory*
TI	Tecnologia da Informação
TLV	*Type-Length-Value*
VLAN	*Virtual Local Area Network*
VPN	*Virtual Private Network*
XML	*eXtensible Markup Language*

Resumo

O grande volume de pacotes/fluxos nas futuras redes de núcleo exigirá um processsamento altamente eficiente dos cabeçalhos nos elementos de comutação. Simplificar a operação de busca(*lookup*) nos elementos de comutação de núcleo é fundamental para o transporte de dados a taxas elevadas e com baixa latência. O hardware de rede flexível combinado com controle de rede ágil também são propriedades essenciais para as futuras redes definidas por software. Argumenta-se que somente com mais desacoplamento entre o plano de controle e plano de dados que haverá a flexibilidade e agilidade nas redes definidas por software para as novas soluções de redes de núcleo. Este trabalho propõe uma nova abordagem denominada KeyFlow para construir um modelo de rede flexível *fabric*. A idéia é substituir a busca na tabela na máquina de encaminhamento por operações elementares com base em um sistema numérico de resíduos. Isso fornece ferramentas para projetar uma rede sem estado no núcleo, com a utilização do controle centralizado do OpenFlow. Como prova de conceito, um protótipo é validado usando o ambiente de emulação Mininet e OpenFlow 1.0. Os resultados indicam redução de RTT acima de 50%, especialmente para redes com tabelas de fluxo densamente ocupadas. KeyFlow alcança mais de 30% de redução no volume de estado relativo aos fluxos ativos na rede.

Abstract

The large bulk of packets/flows in future core networks will require a highly efficient header processing in the switching elements. Simplifying lookup in core network switching elements is capital to transport data at high rates and with low latency. Flexible network hardware combined with agile network control is also an essential property for future Software-Defined Networks (SDN). We argue that only further decoupling between control plane and data plane will unlock that flexibility and agility in SDN for the design of new network solutions for core networks. This paper proposes a new approach named *KeyFlow* to build a flexible network-fabric-based model. It replaces the table lookup in forwarding engine by elementary operations relying on residue number system (RNS). This provides us tools to design a stateless core network by still using *OpenFlow* centralized control. A proof of concept prototype is validated using the *Mininet* emulation environment and *OpenFlow* 1.0. The results indicate RTT reduction above 50%, especially for networks with densely populated flow tables. *KeyFlow* achieves above 30% reduction in keeping active flow state in the network.

1 Introdução

O crescimento da Internet requer que a infraestrutura de núcleo que a compõe possua cada vez mais flexibilidade e agilidade no encaminhamento dos dados. Desde seu surgimento, sobre a infraestrutura de telefonia, até os dias atuais, o seu núcleo é compostos por nodos de roteamento e encaminhamento organizados em redes que se interconectam. Essa interconexão vem a cada dia ficando menos hierarquizada, pois a dependência de grandes empresas de telecomunicações é cada vez menor devido ao crescimento no número de conexões diretas entre diversas organizações, seja por pontos comuns de troca de tráfego, por conexões contratadas de terceiros ou por infraestrutura própria. Com mais opções de caminhos, as extremidades das redes tem mais opções para melhor atender às aplicações e aos usuários finais.

Essas conexões diretas permitiram o crescimento de redes de distribuição de conteúdo e de nuvens de serviços (e.g Amazon, Google, NetFlix, etc.) pela maior oferta de meios de interconexão. Isto viabilizou que provedores de aplicações se conectassem diretamente com diversos outros provedores de Internet regionais, incluindo Redes Acadêmicas. Houve, assim, uma redução nos altos custos de transporte de dados, bem como nas restrições políticas impostas pelas operadoras tradicionais. Com isso, houve uma significativa melhora na ”experiência do usuário” devido ao fornecimento de conexões por caminhos menores e com melhores capacidades de vazão.

O provimento de conexões deixou de ser exclusividade dos detentores de recursos de telecomunicações. Atualmente é comum o fornecimento de conexões lógicas, ou virtuais, interligando diferentes organizações. **Por provimento de conectividade, entende-se aqui, a capacidade de transporte de pacotes/quadros entre dois nodos**. Isto pode ser estabelecido por meio tecnologias tanto de camada de rede (L3) e de enlace (L2) quanto por tecnologias de camadas superiores, diferenciando-se entre si pela latência inerente de processamento/computação. A conectividade por sistemas de telecomunicações, realizada diretamente pelo transporte de bits, possui latência menor, associadaa às limitações do meio físico de transmissão. Este tipo de conectividade é, geralmente, menos flexível e possui um maior custo de implementação e operação.

1.1 Motivação e Justificativa do Trabalho

A demanda de conectividade já ultrapassa o relacionamento institucional entre os sistemas autônomos para atendimento massivo. Grupos de usuários da rede demandam por comunicabilidade versátil de maneira transversal entre as redes que compõe a Internet. A inteligência das pontas da rede já necessita de uma infraestrutura fim a fim capaz de transportar a sua informação com mais eficiência e com menor custo. O poder de computação de nodos interconectados pela Internet se encontra limitado pela capacidade de comunicação dos nodos remotos, tanto no sentido de capacidade de transmissão quanto pela latência inerente ao sistema de transporte dos pacotes. Esta limitação se deve, entre outros fatores, à características do sistema de transmissão e também às limitações do Protocolo IP, que requer um alto grau de processamento por nodo de transmissão. Este protocolo atua com entrega de pacotes sem garantias e como elemento de fronteira entre as redes de telecomunicações e as redes de computadores, por onde navegam o conteúdo das nuvens. Uma melhor integração entre estas redes se torna importante para viabilizar conexões mais flexíveis e eficientes.

Muito se investe na capacidade de transmissão dos enlaces da rede, mas pouco se pode fazer para a redução da latência de comunicação sobre a rede IP. Uma iniciativa no Brasil, pela RNP, é o Serviço Experimental CIPÓ (MACHADO; STANTON; SALMITO, 2011), que tem por objetivo aprovisionar circuitos lógicos de forma sobreposta a seu backbone IP. A solução é baseada na utilização de um complexo aparato de ferramentas para a sinalização interna no domínio de gerência para alocação dos recursos para criação do circuito. Na prática, são criados domínios de difusão isolados, por meio de VLANs, transportados de maneira sobreposta a uma infraestrutura MPLS/IP. A complexidade de sinalização para alocação dinâmica de recursos é o principal desafio para propostas de aprovisionamento de circuitos. A reserva e garantia de recursos de forma sobreposta a uma infraestrutura MPLS/IP é algo de difícil implementação, além de sofrer de muitas restrições para co-existência com a rede de produção. Neste sentido, outra iniciativa da rede acadêmica brasileira, em parceria com redes da Europa, é o FIBRE (SALLENT et al., 2012), que tem por objetivo a criação de uma grande rede experimental para desenvolvimento de soluções e inovações que possam mitigar as principais limitações atuais existentes na Internet.

Quanto mais ao centro da rede está o roteador da Internet, maior é o número de rotas que ele precisa armazenar para poder controlar o encaminhamento dos pacotes e viabilizar a interconexão das redes. Estas rotas são armazenadas nas tabelas de roteamento e encaminhamento. Essas tabelas são implementadas em memórias extremamente rápidas e complexas, que são as principais responsáveis pelo encarecimento dos equipamentos de grande capacidade da Internet.

As iniciativas promissoras para viabilizar o redesenho da Internet, de forma a viabiliar uma maior flexibilização na estrutura de roteamento, de encaminhamento e de transmissão de dados, caminham no sentido de prover uma lógica programável e virtualizável para o compartilhamento dos recursos de comunicação. Neste sentido, as Redes Definidas por Software (SDN) ganham espaço nos meios acadêmicos e nas redes de grandes provedores de conteúdo.O OpenFlow é uma implementação dos conceitos de SDN para o controle centralizado da inteligência de transmissão de pacotes, e já está presente em diversos roteadores e comutadores comercializados pela indústria de rede. Por meio de um protocolo de controle bem definido, o OpenFlow viabiliza uma forma programável de controle da forma de comutação dos pacotes pelos elementos OpenFlow da rede.

Pela forte dependência de manutenção de estados nas suas tabelas de fluxos, que possuem um volume de dados maior e mais complexo, os comutadores OpenFlow requerem uma maior complexidade no seu plano de dados, principalmente no seu *hardware* responsável pelas tabelas de fluxos e pela decisão de comutação. De maneira conjunta com a evolução do protocolo IP, esta dependência de memórias rápidas indica que esta tecnologia esbarra na complexidade do seu plano de dados para viabilizar o aprovisionamento de conexões flexíveis e de baixa latência.

1.2 Objetivo Gerais

Este trabalho tem por objetivo estudar e avaliar experimentalmente formas alternativas de comutação e encaminhamento para o provimento de conectividade entre redes, de forma a viabilizar a versatilidade do OpenFlow/SDN no núcleo das redes. Busca-se reduzir a dependência de manutenção de estados e, assim, permitir o aprovisionamento de conexões versáteis e com baixíssima latência no transporte de pacotes. Por meio da redução na quantidade de variáveis para estabelecimento, manutenção e controle de conexões, espera-se alcançar comutadores mais baratos, econômicos e eficientes.

Neste trabalho, encontra-se a primeira implementação real para a criação de Redes *fabric* de Núcleo Definidas por Software, em contraste a recentes propostas puramente conceituais (CASADO et al., 2012; RAGHAVAN et al., 2012). A proposta de comutação *fabric* é baseada num conceito inovador de roteamento/encaminhamento sem manutenção de estados; e apenas com SDN e plataformas abertas, como o Mininet e *OpenFlow*, foi possível viabilizar a implementação destas ideias de forma realística e com baixo custo, de forma desacoplada aos equipamentos fechados tradicionais, validando a proposta para cenários de redes reais.

1.3 Objetivos Específicos

A implementação desenvolvida neste trabalho visa validar uma alternativa de comutação para extensão da especificação do *OpenFlow*/SDN para sua aplicação no núcleo das redes. A proposta se baseia numa estrutura de comutação sem manutenção de estados que permite a criação de circuitos lógicos de baixa latência.

Os caminhos, na proposta deste trabalho, são definidos por rótulos globais e a comutação é controladas por chaves locais instaladas nos comutadores pelo plano de controle da rede. A rede proposta é dividida entre borda, formada por comutadores *OpenFlow*, e por um núcleo *fabric*, ou seja, que permite a interconexões de todos os pontos das suas extremidade de forma especializada no transporte de pacotes entre as bordas.

Esse núcleo é formado por comutadores batizados por KeyFlow, que utilizam uma chave interna para governar a comutação dos pacotes identificados por rótulos globais, sem a necessidade de utilização de tabelas para manutenção de estados. Sendo assim, a proposta cria uma arquitetura semelhante a rede de circuitos, porém sem a necessidade de alocação de recursos pelo caminho e com uma rápida convergência no estabelecimento das rotas. Esse comportamento provê à proposta uma similaridade híbrida entre as redes de comutação de pacotes e a circuito, utilizando-se de conceitos das redes definidas por software para obter maior flexibilidade para o aprovisionamento dos caminhos.

1.4 Metodologia de Desenvolvimento

A metodologia do trabalho consistiu na implementação de um protótipo de comutador para servir de prova de conceito, por meio de uma avaliação experimental da comutação de pacotes em caminhos definidos sem a manutenção de estado nos elementos centrais. Este trabalho apresenta os resultados do desenvolvimento desse protótipo[1], que foram publicados recentemente em (OLIVEIRA et al., 2013; MARTINELLO et al., 2014), como também em (OLIVEIRA et al., 2013; MARTINELLO et al., 2014).

O protótipo desenvolvido é executado em ambiente virtual, sobre a plataforma Mininet[2]. Os testes consistiram do provimento de conexão entre duas máquinas físicas por meio um caminho de comutadores virtuais, avaliando o desempenho em um cenário puramente baseado em manutenção de estados com OpenFlow comparativamamente com um cenário com o núcleo

[1] http://gtif-ufes.github.io/keyflow/
[2] http://mininet.github.com/

sem manutenção de estado com o protótipo do KeyFlow. Este protótipo foi desenvolvimento a partir da implementação de referência do comutador *OpenFlow* 1.0 [3].

No decorrer do trabalho, no Cap. 2 são apresentados os conceitos básicos e toda fundamentação teórica. O Cap. 3 apresenta de forma detalhada a arquitetura de comutação KeyFlow e, na sequência, o Cap 4 apresenta a prova de conceito realizada e os resultados obtidos de análise experimentais da escalabilidade da proposta. O Cap.5 apresenta as conclusões do trabalho e aponta para trabalhos futuros pertinentes para a evolução da proposta.

[3] http://archive.openflow.org/downloads/openflow-1.0.0.tar.gz

2 *Fundamentação Teórica:*

2.1 Introdução

A conectividade por transporte de pacotes viabiliza conexões mais versáteis e econômicas, muitas vezes pelo compartilhamento de vários recursos de infraestrutura de diferentes provedores. Com a redução nos custos para provimento de conectividade, mais conexões existirão e permitirão aumentar a resiliência e a capacidade de comunicação pela Internet.

Existe, contudo, muitos problemas relacionados às conexões lógicas devido a sua característica computacional para o controle da conectividade. Este controle é realizado por meio da manutenção de estados em memórias, como as tabelas de encaminhamento e roteamento, que se implementadas indevidamente em memórias lentas, implicam em severas restrições de latência para as conexões lógicas, inviabilizando um conjunto considerável de aplicações, como a transmissão de voz, de vídeos, de dados de tempo real, etc. Além disto, a necessidade de manter o estado das conexões em memória é o principal gargalo para a escalabilidade da infraestrutura lógica de provimento de conectividade, tanto pela limitação física da memória quanto seu alto custo para implementações eficientes.

A criação de conectividade virtual de forma mais fácil e flexível no núcleo das redes ainda não é realidade, principalmente quando se trata do encaminhamento de fluxos que necessitam de características especiais. Do lado do mundo de telecomunicações, uma infinidade de recursos e de tecnologias existem para o estabelecimento de conexões, contudo as opções existentes muitas das vezes são caras e extremamente complexas. Isso dificulta e distancia a integração das redes de tecomunicações com o as redes de dados, ou de computadores, por onde trafegam os conteúdos dos usuários. Do lado das redes de computadores, existe um grande engessamento do seu principal protocolo de conectividade, o Protocolo Internet. Sua falta de serviços de garantia de qualidade, seus problemas de endereçamento e, principalmente, sua grande dependência de manutenção de estados para manutenção de caminhos, frente ao crescimento no número de redes interconectadas, são algumas de suas principais limitações a serem superadas.

2.2 Técnicas de virtualização para o provimento de conectividade

Desde o substrato físico até a conexão lógica fim-a-fim de fato, existem diversas possibilidade de estabelecimento de conexão. As tecnologias de multiplexação de recursos físicos tem historicamente um alto custo de implantação e operação. Nas camadas superiores, a versatilidade das conexões ficam acopladas às limitações do protocolo IP no transporte dos pacotes. Além disso, quanto mais alta a camada de provimento de conectividade, maior a latência computacional e as limitações para o transporte ágil dos pacotes.

As conexões controladas em níveis intermediários entre a camada física e a de rede vem ganhando força no mercado, tecnologias como o Ethernet por meio das redes virtuais vem sendo largamente utilizados em redes metropolitanas. No contexto regional e inter-regional, o MPLS/IP é usado em via de regra para provimento de circuitos virtuais para transporte de pacotes com latência reduzida. A necessidade de manutenção de estados em tabelas, como o caso das de roteamento e encaminhamento, e a grande complexidade do plano de controle para o caso das redes MPLS, são as principais fontes dificuldade para a criação conexões lógicas versáteis, isto é, que são dinamicamente criadas em resposta às demandas das aplicações e da rede.

A ideia de compartilhamento de recursos por meio da coexistência de múltiplas redes pode ser categorizada em quatro classes principais (CHOWDHURY; BOUTABA, 2009): redes locais virtuais (VLANs), redes privadas virtuais (VPNs), redes ativas e programáveis e redes sobrepostas, conforme maior detalhamento a seguir.

2.2.1 Redes Locais Virtuais

O Ethernet, especificado pelo grupo de trabalho IEEE 802.3 [1] se torna cada vez mais um protocolo ubíquo. Diversos dispositivos, desde a indústria até domicílios, se conectam em rede utilizando esse protocolo na comunicação local e, também, como um meio de acesso à Internet.

Em *datacenters* é praticamente um padrão na interconexão de elementos computacionais. Seu sucesso se deve principalmente à sua simplicidade. Em termos operacionais, um fator muito relevante é o suporte a VLAN, ou rede local virtual. Por meio de uma *tag* os quadros são diferenciados por domínio de colisão, dando a impressão lógica de separação desses domínios em múltiplos ”cabos lógicos” que interconectam logicamente um grupo de *hosts* em um mesmo

[1] http://www.ieee802.org/3/

"barramento".

Todos os quadros de uma determinada VLAN são identificados por um ID de VLAN no cabeçalho de controle de acesso ao meio (MAC), e os comutadores habilitados com o suporte a essas *tags* utilizam tanto o endereço MAC de destino como o ID da VLAN para encaminhar os quadros pela rede.

Com a utilização de redes locais lógicas, os quadros dos usuários são alterados e marcados com a TAG de VLAN. Com isso, é possível prover um alto nível de isolamento entre diversas redes locais por meio de conexões estabelecidas de forma virtual. Essa é uma forma simples e eficiente para a administração, a gerência e a reconfiguração das redes tanto num contexto local, de campus ou até mesmo em escopo metropolitano, com algumas limitações.

Para prover maior escalabilidade no contexto metropolitano existe uma extensão do IEEE 802.1q, é o protocolo IEEE 802.1ad [2], ou mais popularmente conhecido por QinQ. Com essa especificação, a noção de LAN virtual foi estendida para o suporte de pontes virtuais. Assim, um grupo de VLANs poderia ser isolada de outro grupo. Isto é realizado, basicamente, pela extensão de uma tag dupla de VLAN ID. Assim, uma tag tem o papel de transportadora, chamada de tag de serviço ou S-TAG, e a outra de transportada, chamada de tag do cliente ou C-TAG. Na passagem pela ponte virtual, a tag transportadora é retirada e o quadro é entregue a rede local virtual do cliente com a devida tag, agora única no pacote. Com a dupla tag de VLAN, é possível criar um isolamento ainda mais sofisticado, de forma semelhante à tuneis, porém aplicados diretamente no segmento de cabeçalho referente às tags de VLAN do quadro Ethernet.

Uma limitação importante das redes virtuais locais baseadas no Ethernet é a necessidade de manutenção de estado para se evitar *loops* na rede. Isso passa a ser crítico na utilização deste tipo de virtualização em redes metropolitanas, umas vez que certos enlaces passa a operar de maneira ociosa, reduzindo o retorno de investimento pela subutilização dos recursos disponíveis.

2.2.2 Redes Privadas Virtuais

Uma Rede Privada Virtual, ou VPN, é uma rede dedicada que conecta múltiplos *sites* usando tuneis privados e seguros sobre redes de comunicações públicas ou compartilhadas, como a Internet.

A conexão lógica pode ocorrer em diversas camadas, considerando o modelo de referência TCP/IP por exemplo, conforme visão geral a seguir.

[2] http://www.ieee802.org/1/pages/802.1ad.html

- **VPN em Camada 1**

  Comparado a um serviço de conectividade entre uma par de dispositivos de borda de clientes (CE), a VPN de camada 1 tem características de serviços adicionais, como o provimento de conectividade entre um conjunto de CEs e um controle e gerenciamento por VPN (TAKEDA; KOJIMA; INOUE, 2004). É formado um tubo totalmente transparente entre um conjunto determinado de portas.

  Ela provê um *backbone* multiserviço onde os clientes podem oferecer seu próprio serviço, e o *payload* pode ser de qualquer camada. Isso garante que cada serviço de rede possui um espaço de endereçamento e uma visão de camada 1 independentes, políticas totalmente separadas e um isolamento completo de outras VPNs (CHOWDHURY; BOUTABA, 2009).

  No aspecto dos requisitos de controle, as VPNs de camada 1 precisam prover requisição de criação e destruição de caminhos, recepção de informação de membros da VPN, de informações topológicas da rede do cliente e do provedor de rede.

  Com as VPN de camada 1, os provedores utilizam da infraestrutura compartilhada como se fosse dedicada a seus clientes. Por meio de tecnologias ópticas como ROADM, uma frequência pode ser alocada para um conjunto de nodos acessíveis para um cliente. Este cliente, sobre esta frequência pode contar com uma rede TDM própria para prover outros caminhos de conectividade para seus clientes. Ou seja, o cliente tem o controle da criação de caminhos na infraestrutura e passa a ser capaz de alocar estes recurso para se tornar provedor de outros clientes de camadas superiores.

- **VPN em Camada 2**

  Como a tecnologia de VPN em camada 1 ainda tem um custo restritivo, é muito comum, principalmente devido a ubiquidade do Ethernet, a necessidade de formação de redes virtuais de entrega de quadros para um conjunto de nodos. Este nível de serviço não provê plano de controle e gerenciamento de caminhos sobre a VPN.

  O encaminhamento de quadros é feito baseado em informações de camada 2, como DLCI, VPI/VCI, MAC, VLAN ID, etc. O usuário tem controle de comunicação apenas em camadas superiores, i.e. IP.

  Para conexões de longa distância, na interconexões de LANs remotas, normalmente é utilizada uma infraestrutura MPLS/IP. No caso do núcleo MPLS/IP, o quadro ou pacote é encapsulado com um rótulo, definido por meio de um protocolo de sinalização que define o caminho fim-a-fim da rede IP e um conjunto de rótulos que devem ser utilizados salto-a-alto. Os quadros ou pacotes fluem pelo núcleo sem a necessidade de roteamento, com o

tráfego organizado em classes de encaminhamento específicas. A definição dos caminhos sobre a rede IP é feita por um protocolo específico de descoberta de recurso e criação de caminhos, que é responsável por toda a sinalização para reserva de recurso e definição de rótulos a cada salto nos roteadores da rede. Assim, não é utilizado nenhum cabeçalho IP no controle do encaminhamento dos quadros de um ponto a outro da rede, apenas os rótulos que definem a forma de comutação do quadro da origem ao destino.

Outra forma de criação de redes privadas virtuais, é por meio da especificação 802.1ah, ou como é mais conhecida a Pontes

No aspecto da segurança da rede privada, a própria restrição de acesso a nível de camada 2 já provê uma segurança básica para a comunicação entre redes remotas, uma vez que a própria infraestrutura de transporte dos quadros provê o isolamento do acesso a cada domínio de *broadcast*. A confiabilidade na infraestrutura do provedor é o referencial de segurança para transmissão de dados pela rede virtual, por ter característica privada. Para casos extremos é comum a proteção dos dados por criptografia em camadas superiores para serviços específicos transportados pela rede.

- **VPN em Camada 3**

  Uma rede privada virtual de camada 3 (L3VPN) é caracterizada pela utilização de protocolos de camada 3 para transportar os dados entres CEs distribuídas.

  Existem duas abordagem na construção de VPN em camada 3: baseada nos equipamentos do cliente, ou fornecida pelo provedor de rede. No primeiro caso, a infraestrutura do provedor de rede fica completamente a parte do estabelecimento da conexão. Os equipamentos do cliente criam, gerenciam e finalizam os tuneis entre si. Um elemento na rede do cliente é responsável por encapsular o pacote e envia-los para a rede do provedor. Na outra ponta do cliente, o pacote é desencapsulado e os dados são extraídos e encaminhados para o alvo de destino da conexão. No caso da VPN de camada 3 fornecida pelo provedor, a infraestrutura de rede entre os equipamentos do cliente fica com a responsabilidade de estabelecer e manter a conexão (CHOWDHURY; BOUTABA, 2009).

  Para as VPNs estabelecidas por meio de túneis a partir da camada 3, é crucial uma maior preocupação com o acesso indevido ao conteúdo dos pacotes, devido às características de melhor esforço do protocolo utilizado para entrega dos pacotes. Para viabilizar uma conexão segura é utilizado o protocolo IPSec, projetado pela IETF, que oferece transferência segura de informações fim-a-fim por meio de uma infraestrutura IP pública ou privada.

- **VPN em Camada 4 ou superior**

A criação de VPN utilizando camadas mais altas, como a de transporte, de sessão ou de aplicação, também é possível. É muito comum a utilização de conexões virtuais baseadas em sessões SSL/TLS.

A principal vantagem do estabelecimento de redes virtuais baseadas em sessões de camadas superiores é sua característica inerente de conexão fim-a-fim, que provê uma grande granularidade de conexão de usuários distribuídos. Além disso, por terem esta característica fim-a-fim, são de instalação e de utilização mais simples, pois dependem diretamente apenas dos *hosts* finais.

Numa visão geral, as VPNs são uma técnica bastante versátil para viabilização de conectividade, contudo, pela característica de tunelamento utilizar sempre técnicas legadas e tradicionais, os tuneis sofrem das mesmas limitações existentes no processamento do cabeçalho que encapsula o pacote original. Em camada 1 e 2, a principal limitação se encontra na complexidade para a definição e manutenção dos caminhos. Na camada 3, os tuneis têm a limitação natural da manutenção de estado do IP, que impacta severamente na latência e no engessamento do protocolo, que implica na necessidade de utilização de remendos, como o IPSec para provimento de segurança. Nas camadas superiores, as VPNs funcionam acumulando algumas perdas de desempenho relativo a necessidade de processamento das camadas inferiores.

2.2.3 Redes Sobrepostas

Uma Rede Sobreposta, ou rede *Overlay*, é uma rede lógica construída por cima de uma ou mais redes. A própria Internet opera em sobreposição às rede de telecomunicações, o seu surgimento se deu sobre a infraestrutura de telefonia da época. Outras redes sobrepostas são formadas por cima da própria Internet, como é o caso das redes P2P, formadas na camada de aplicação da arquitetura por meio de conexões fim-a-fim pelos protocolos de transporte TCP ou UDP. Contudo, existem várias implementações de sobreposição de redes nas camadas inferiores da pilha de protocolos. Portanto, uma rede *overlay* é uma rede virtual que é construída sobre um outra rede, seja ela uma rede física ou uma outra rede lógica.

Uma outra forma de entender as arquiteturas de rede *overlay* é na visão de camada de serviço e de transporte, por exemplo uma rede IP/MPLS e uma SDH ou WDM, respectivamente. Nesta visão, as redes, apesar de serem construídas uma por cima da outra, possuem mecanismos de proteção e restauração construídos e executados de forma totalmente independentes. Cabe ressaltar que pela natureza desse tipo de sobreposição, há uma significativa complexidade no domínio de controle e sinalização dessas redes para o provimento de caminhos lógicos

independentes.

Redes *overlay* no núcleo da rede são implantadas por Provedores de Rede ou Provedores de Serviço de Internet quando: a) possuem tanto os ativos de roteamento e encaminhamento quanto a infraestrutura de transporte, mas a engenharia, a manutenção, a operação e a evolução são mantidas por organizações separadas; b) possuem apenas os ativos de roteamento e encaminhamento e subcontratam as conexões e a capacidade de transmissão de outros provedores; ou c) oferecem outros serviços além do IP, e por isso requisitam de uma infraestrutura de transporte capaz de acomodar plataformas multi-serviços (BARONI et al., 2000).

As redes *Overlay* têm sido usadas para montar ambientes experimentais, como o PlanetLab, para projetar e avaliar novas arquiteturas. As redes sobrepostas não causam e não requerem nenhuma alteração na rede subjacente. Por isso, são usadas com relativa facilidade e baixo custo para testar novas funções e correções para a Internet.

De forma objetiva, essas redes podem ser caracterizadas por terem sua própria organização de nodos e de enlaces virtuais, com seu próprio plano controle e gerenciamento, tratando de maneira independente a forma de comutação e de transporte dos fluxos em relação a infraestrutura de rede da camada inferior.

2.3 Ecossistema da virtualização de redes

Segundo (CHOWDHURY; BOUTABA, 2009), o ambiente de virtualização de redes é composto basicamente por dois papéis: o provedor de serviço (PS) e o provedor de infraestrutura (PI). Esse ambiente é composto por um conjunto de múltiplas e heterogêneas arquiteturas de rede de diferentes PS, onde cada PS obtém os recursos que precisa de um ou mais PIn para cria sua própria rede virtual (RV) e pode disponibilizar seus próprios serviços a seus usuários finais.

No modelo de negócio tradicional da Internet existe o Provedor de Serviço de Internet (PSI, ou ISP em inglês) que interage com outros ISPs e consegue, assim, viabilizar a conectividade entre os usuários da Internet. No caso do modelo de negócio do ambiente de redes virtualizadas este papel é dividido entre o PS e o PI.

O PI é o responsável por implantar e manter os recursos da camada física de interconexão e prover seus recursos por meio de interfaces programáveis para diferentes PS. Os provedores de infraestrutura se diferenciam pela qualidade dos recursos que são disponibilizados para as redes clientes e pela qualidade do aparato disponibilizado para viabilizar liberdade para seus clientes utilizarem diferentes infraestruturas.

O PS capta recursos de diferentes provedores de infraestrutura para criar e implantar redes virtuais programando recursos de redes alocados para oferecer serviços de conectividade fim-a-fim para seus usuários finais. De maneira recursiva, o provedor de serviços pode aprovisionar frações de seus recursos virtuais para outros provedores de serviço, atuando como um PI virtual, conforme Figura 2.1.

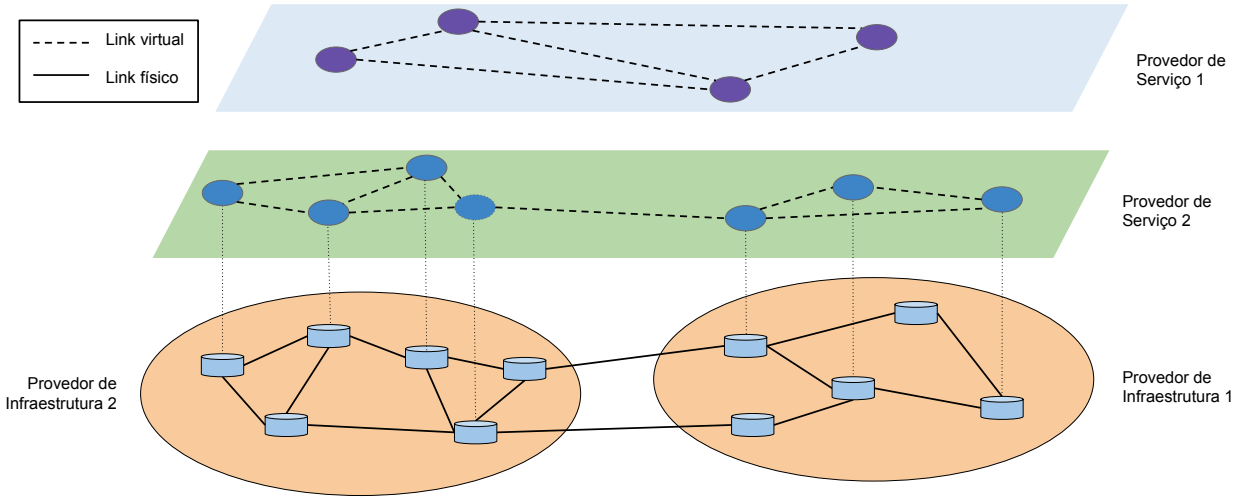


Figura 2.1: Ecossistema de um ambiente de redes virtualizadas

Na Fig. 2.1 tem-se a representação de uma rede totalmente virtualizada. Da infraestrutura física de 2 provedores é criada uma topologia lógica de um provedor de serviço. Tanto os *links* quanto os nodos são elementos lógicos, ou virtuais. A partir da infraestrutura lógica do Provedor de Serviço 2 é criada a topologia do novo provedor em uma camada acima, essa alocação ocorrerá recursivamente até a ocupação total dos recursos virtualizados da camada física.

A criação de *links* lógicos pode ser obtida de várias formas, seja por redes lógicas virtuais, por VPNs ou por sobreposição. Em todos as formas, há uma grande dificuldade para manutenção do estado da rede, principalmente na sinalização para definição e alocação de recursos de cada enlace físico. Essa dificuldade é ainda maior considerando um contexto de manutenção de estado distribuído na rede.

Para a criação dos nodos lógicos, em termos práticos, há uma maior complexidade para separação de unidades lógicas de processamento dos elementos físicos. Essa complexidade também é agravada pela necessidade de manutenção de estados distribuídos, uma vez que diferentes nodos virtuais comportarão um volume diferenciado de estados a serem controlados na forma o roteamento/encaminhamento de diferentes fluxos de isolados por cada infraestrutura.

2.3.1 Virtualização de redes e SDN

As ideias decorrentes da virtualização de redes são bem anteriores e independentes das redes definidas por software, mas essas tecnologias vem se aproximando cada vez mais. A abstração da rede física em termos de uma rede lógica (virtualização de redes) não requer a separação de um plano de controle logicamente centralizado do seu plano de dados (SDN). Contudo, a união dessas características tem catalizado novas áreas de pesquisas, habilitando novas funcionalidades e um novas abordagens para definição de redes flexíveis e inovadoras.

Do ponto de vista das redes virtualizadas, apesar de muito avanço ocorrer com a criação de conexões lógicas desacopladas das conexões físicas, a utilização de nodos virtuais desacoplados dos nodos físicos é algo mais complexo. Nesses sentido, a utilização de um plano de controle logicamente centralizado e desacoplado do plano de dados simplifica a comunicação da sinalização e na visão global da rede. Com isso, o controle da forma de comutação dos nós de maneira diferenciada em relação ao fluxo passa a exercer o papel necessário para o desacoplamento lógico do nodo do recurso físico.

Para as redes definidas por *sofware*, a benefício da virtualização de enlaces complementa a sua capacidade de generalização dos nodos. A criação de segmentos lógicos separados promove um maior isolamento dos fluxos, permitindo um tratamento mais refinado no compartilhamento dos recursos de computação disponíveis na rede programável.

2.3.2 A Inovação no Ambiente de Redes

A baixa abertura para inovação nas redes de computadores e de telecomunicações impulsionou a pesquisa numa nova abordagem para as redes do futuro. Essa nova visão da rede tem por objetivo balancear a capacidade de resiliência, fundamental para o sucesso da Internet, com capacidade de provimento de novas funcionalidades. Uma abordagem para isto está no desacoplamento definitivo do plano de dados do plano de controle no projeto das redes do futuro. Essa ideia, entre outras, norteia o desenvolvimento das Redes Definidas por Software (em inglês, SDN). Recentemente, o *OpenFlow*(MCKEOWN et al., 2008), a primeira especificação de um SDN, tem sido amplamente adotado para identificação e tratamento de fluxos em redes experimentais, acadêmicas e comerciais. A arquitetura baseia-se em um controlador externo aos comutadores *OpenFlow* que centraliza a gerência de encaminhamento de pacotes através de uma visão global e manutenção de estados dos fluxos ativos na rede, conforme apresentado no Cap. 2. Com a possibilidade de controle logicamente centralizado e a capacidade de diferenciação de fluxos de pacotes baseados em múltiplos cabeçalhos, passou-se a ter uma maior

versatilidade no nodo da rede para o processamento dos pacotes, mas ainda não há alternativas para estender esta versatilidades para as conexões que ligam estes nós.

Muitos esforços se concentraram no desenvolvimento de aplicações e modificações sob o ponto de vista do plano de controle das redes *OpenFlow*. Poucos esforços foram direcionados para permitir que a flexibilidade provida no processamento dos pacotes seja também estendida para a criação e manutenção de conectividade lógica. A utilização direta de comutadores *OpenFlow*/SDN no controle do estabelecimento de conexões possui grandes limitações no sentido da necessidade de interação entre o plano de controle e o plano de dados e na própria forma de manutenção de estados, que tem as mesmas restrições de ocupação de memória das tabelas de roteamento do IP potencializadas pelo maior volume de dados armazenados nas tabelas de estado. Além disso, a lógica interna do *hardware* pode ser implementado de diferentes formas, apresentando assimetrias em um parque com diversidade de fabricantes.

É importante, assim, simplificar a máquina de encaminhamento no núcleo das redes, viabilizando uma comutação uniforme ao longo de todo o caminho da rede de núcleo. Rede esta que deve ser especializada no transporte de pacotes e provimento de conectividade entre seus elementos de borda, responsáveis por prover o controle na diferenciação e no roteamento dos fluxos no interior do núcleo. Portanto, o provimento de conectividade lógica carece de alternativas de comutação que permitam o transporte de pacotes sem a manutenção de estados, de forma que a rede de núcleo seja especializada no transporte de pacotes entre todos os elementos de borda, provendo assim, uma função de comutação *fabric* para o núcleo das redes definidas por software.

2.4 Redes Definidas Por Software

O rápido desenvolvimento das Redes Definidas por Software (SDN) está intimamente ligado à evolução dos *datacenters* e dos serviços na "núvem", que demandam cada vez mais flexibilidade da infraestrutura de comunicação.

Inicialmente, haviam os grandes *mainframes* com baixo poder de computação e alto consumo de energia, de espaço e complexos requisitos de refrigeração. Os altos custos desses equipamentos fortificaram a adoção de servidores baseados na arquitetura PC. Esses servidores surgiam com cada vez maior poder computacional e com custos operacionais significativamente inferiores aos dos *mainframes*. Rapidamente dominaram o mercado de TI por serem mais acessíveis e por requisitarem ambiente operacional mais simples. A quantidade e a complexidade dos serviços sobre esses novos servidores cresceu. Os administradores de TI passaram a isolar

os serviços em diferentes servidores. Assim, novamente, novos desafios surgiram relacionados ao consumo de energia, à ocupação de espaço físico, e a necessidade de refrigeração do grande número de servidores instalados nos *datacenters* modernos.

A diversidade e o crescimento do número de serviços hospedados demandou, entre outras coisas, requisitos específicos sobre os sistemas operacionais desses servidores baseados em arquitetura PC. Para acomodar diferentes sistemas e tornar mais flexível a operação e a gerência dos servidores sobre a infraestrutura existente, ganhou força a virtualização nos *datacenters*. Os monitores de máquinas virtuais, ou *"Hypervisors"*, passaram a ter um papel chave. Trouxeram uma grande flexibilidade na operação e manutenção dos servidores existentes, viabilizando o compartilhamento de recursos físicos entre diversas máquinas virtuais (MV) de maneira elástica com relação a ocupação de CPU , memória, armazenamento e E/S.

Os serviços puderam, assim, ter um isolamento eficiente além de maior versatilidade e disponibilidade. Passou a ser possível migrar máquinas pela rede com a mesma facilidade de se mover um arquivo. Essas facilidades foram fundamentais na otimização da utilização de recursos nos grandes provedores de conteúdo que puderam ter mais facilidade para prover serviços com melhor qualidade, por meio de uma utilização elástica dos recursos físicos. O avanço dessas facilidades esbarrou na infraestrutura de rede, que não acompanhou a evolução dos servidores e das máquinas virtuais com a mesma velocidade.

As demandas dos serviços sobre a infraestrutura de virtualização requerem uma maior flexibilidade da rede de comunicação. O padrão de tráfego da rede mudou da estrutura simplista da arquitetura básica de cliente-servidor para a disponibilização de serviços em nuvem. Neste cenário, as aplicações necessitam de acesso a diferentes bancos de dados e servidores espalhados ao redor do mundo. As nuvens públicas ou privadas demandaram por organização de serviços com armazenamento distribuído e baixo tempo de resposta das aplicações. A utilização dos recursos de computação em nuvem acontece, agora, de maneira elástica em relação à ocupação de recursos computacionais. Essa elasticidade deve estar presente não só na utilização do processamento e armazenamento, providos pelas tecnologias de virtualização de servidores, como também nos recursos de rede e comunicação.

A necessidade de uma boa conectividade foi além da relação direta com os usuários do serviço, passou a ser relevante também uma melhor ocupação de enlaces que conectavam diferentes partes da nuvem dos provedores, tornando o ambiente de rede cada vez mais crítico. Enormes conjuntos de dados processados pelos grandes *datacenters* e centros de pesquisa demandaram uma melhor utilização da largura de banda das conexões de longa distância, seja por enlaces dedicados ou sejam por redes sobrepostas à infraestruturas existentes.

A indústria de rede não foi capaz de responder na velocidade necessária a essas novas demandas. O paradigma de comunicação totalmente distribuída contribuiu para aumentar essa inércia, pois novos protocolos precisavam ser chancelados por diversos fabricantes a fim de viabilizar plena compatibilidade de comunicação entre os equipamentos. Isso provocou o surgimento de redes fortemente dependentes de uma empresa específica, o que aumentou os custos das redes do mercado. Além disso, os equipamentos são vendidos como uma espécie de "caixa preta", reforçando a dependência dos fabricantes, e com apenas interfaces básicas de gerenciamento e configuração (SNMP, CLI , *WebServices*), sem espaço para inovações.

O custo de evolução dessas "caixas pretas"passou a ser um empecilho para os grandes *datacenters*, uma vez que possuíam diferentes demandas para distribuição dos seus dados ao redor do mundo. A rede IP e os sistemas de telecomunicações não foram capazes de atender plenamente a esses requisitos de conectividade, principalmente relacionados à maximização da utilização de banda, a redução da latência de comunicação e do custo de projeto e operação das redes.

Os grandes *datacenters* e empresas afins impulsionaram, com isso, o desenvolvimento de uma tecnologia emergente da academia: o OpenFlow, e com ele o desenvolvimento aplicado dos conceitos das Redes Definidas por Software, com o objetivo principal de alterar a arquitetura tradicional das redes, com a redução dos custos operacionais e uma maior abertura para a inovação e para o desenvolvimento de novas tecnologias que atendessem a rápida evolução dos serviços em nuvem e a diminuição da ossificação existente na Internet.

2.4.1 A arquitetura das Redes Definidas por Software

A arquitetura SDN tem por principais características o desacoplamento do plano controle do plano de encaminhamento, ou de dados, e a capacidade de ser programável por uma interface logicamente centralizada. A inteligência da rede é, assim, logicamente centralizada em controladores baseados em software que comandam e interagem diretamente com o plano de dados (ONF, 2013).

Com a manutenção do estado da rede sendo realizada dessa forma, há uma grande simplificação tanto no projeto quanto na operação dessas redes, uma vez que o plano de controle provê uma abstração lógica e uma interface comum para o desenvolvimento das aplicações e serviços da rede. Essa simplificação permite um ganho relativo a independência da indústria de redes na aquisição dos equipamentos do plano de dados, com potencial redução de custos de projeto, de expansão e de operação.

A arquitetura tradicional de comunicação consiste de dispositivos de usuários interconectados por uma infraestrutura de rede, composta por elementos de comutação e encaminhamento, como roteadores e *switches* ou comutadores, interconectados por enlaces de comunicação que transportam os dados entre usuários da rede. Esses dispositivos de comunicação são, normalmente, sistemas fechados e frequentemente possuem interfaces e funcionalidades controladas e limitadas pelos fabricantes. Com isto, as redes de comunicação sofrem enorme dificuldade tanto para evoluir a sua capacidade como as suas funcionalidades. O desenvolvimento de novas versões de protocolos existentes, como o IPv6 ou outros completamente novos, é um obstáculo quase intransponível frente à inércia da cadeia de produção e implantação dos equipamentos de rede. A Internet, como uma rede de redes, sofre deste mesmo obstáculo.

A dificuldade de evolução da Internet, e das redes de comunicação em geral, é atribuída ao forte acoplamento entre os planos de dados e controle. Ou seja, o fato da decisão sobre o fluxo de dados na rede ser feita sobre cada elemento da rede, com as informações de controle compartilhando recursos com a carga útil de dados transportados pela rede. Neste cenário, o desenvolvimento de aplicações que visem melhorar a qualidade da rede é algo altamente complexo e, por isto, demanda um grande ciclo de desenvolvimento, teste e validação. Isto normalmente acontece em um cenário separado do ambiente de produção, quando não em ambientes de simulação, o que retarda o ciclo de inovação e entrega de soluções ao mercado.

Devido à distribuição da inteligência da rede, a configuração e a aplicação de políticas pode exigir uma boa quantidade de esforço devido à falta de uma interface de controle comum para os administradores frente a todos os dispositivos da rede.

As Redes Definidas por Software foram desenvolvidas para facilitar a inovação e permitir controle programável do encaminhamento na rede. A Fig. 2.2 mostra a separação do plano de controle lógico dos elementos de encaminhamento. Na arquitetura tradicional, Fig. 2.2(a), há um sistema de controle distribuído, com cada dispositivo de encaminhamento com seu respectivo elemento de controle associado.

Neste cenário, a implantação de serviços da rede por componentes intermediários, ou middleboxes, como soluções de *firewall*, de balanceamento de carga, de detecção de intrusão, etc., devem ser cuidadosamente configurados separadamente, assim como toda a rede. A aplicação de políticas e implantação de novos serviços passa a ser uma atividade de alta complexidade.

A Fig 2.2(b) ilustra o desacoplamento definitivo entre os elementos de encaminhamento e a lógica de controle da rede, posicionada em um elemento de forma centralizada. Com isso, as políticas de roteamento, os controles de caminhos lógicos, as soluções de proteção e filtragem do tráfego, de balanceamento de carga, etc., são realizadas em aplicações em execução sobre o

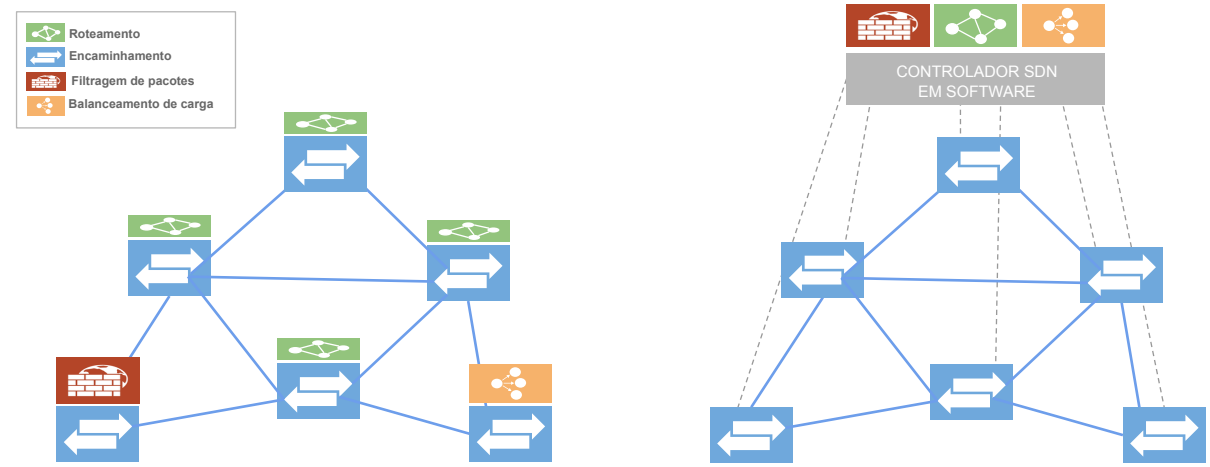


Figura 2.2: Comparação das arquiteturas (a) tradicional, com o controle e aplicações de rede distribuídas (b) SDN, com o controle e aplicações de rede centralizadas em software programável no controlador

controlador. Isso proporciona uma grande simplicidade no projeto e na expansão da rede, tanto em termos de capacidade física, quanto de funcionalidades lógicas.

2.4.2 As Fronteiras de Comunicação para o Controle da Rede

Na arquitetura apresentada na Fig. 2.2(b), o controlador atua como intermediário entre a infraestrutura do plano de dados e as aplicações e serviços. Sendo assim, pode-se dizer que existe uma interface de "fronteira-norte"e outra de "fronteira-sul"ou uma interface superior e outra inferior para o controlador da rede, ou para o sistema operacional da rede, numa analogia à arquitetura de computadores.

Conforme Fig 2.3, pode-se entender a interface na fronteira-norte como aquela provida pelo controlador às aplicações e serviços em execução. Nessa interface transitam as mensagens no sentido controlador-aplicação, como a abstração da topologia, as estatísticas da rede, formas de autorização na rede, etc. No sentido aplicação-controlador, pode-se citar as mensagens de definição de qualidade de serviço (QoS), de roteamento, filtros de pacotes, etc.

Já na interface de fronteira-sul, ainda na Fig. 2.3, existe o protocolo e mecanismo de comunicação entre o controlador e os comutadores da rede. Os comutadores geram as mensagens de estatísticas do plano de dados, informações de eventos na rede e são os responsáveis por encaminhar os pacotes na rede. No sentido controlador-comutador, são enviadas as mensagens de comando de comutação, por meio da definição de estado da tabela de encaminhamento do elemento de rede. O OpenFlow é um exemplo de interface de "fronteira-sul", pois sua especifi-

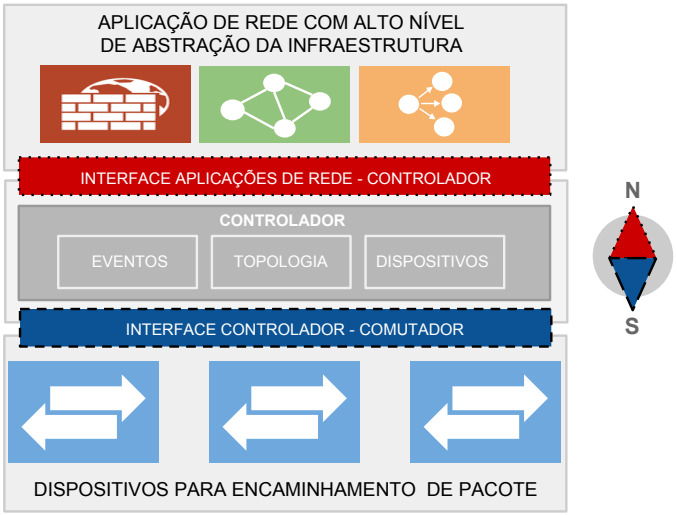


Figura 2.3: Fronteiras norte sul na interface de controle da rede SDN

cação define as características do comutador e todas as mensagens necessárias ao protocolo de comunicação entre o comutador e o controlador da rede.

2.4.3 Modelo de Controle: Centralizado ou Distribuído?

Como já foi visto anteriormente, a separação dos planos de dados e de controle é o pilar principal das redes definidas por software. Contudo, existe ainda muita discussão a respeito da forma ideal dessa separação. Todas as funções de controle precisam estar desacopladas dos equipamentos do plano de dados? Quantas instâncias de controle são necessárias para garantir a disponibilidade e a resiliência da rede? Qual a distância ideal e segura do plano de controle e do plano de dados?

Quanto a separação das decisões de controle e encaminhamento, a centralização do plano de controle sob responsabilidade de um único controlador pode parecer vantajoso do ponto de vista da "fronteira-norte", uma vez que cria uma maior facilidade para execução de aplicações, uma vez que existe uma maior simplicidade nessa interface. Em termos práticos, principalmente tomando-se por base a Internet, a ideia de um controle único cai por terra, uma vez que fere um dos principais fundamentos da Internet: a resiliência da comunicação proporcionada pelo sua arquitetura distribuída.

É fato que a total distribuição do controle da rede é um dos principais fatores para sua limitação atual, tanto do ponto de vista de endereçamento quanto para o próprio plano de roteamento, que carece de roteadores com elevada quantidade de memórias rápidas, que são caras e impactaram diretamente na qualidade de experiência do usuário sobre a rede. No outro ex-

tremo, a centralização total das decisões de controle implica numa enorme fragilidade para a rede, tanto para resiliência de falhas quanto para questões de segurança. Nesse sentido, existe uma escala entre esses extremos que é vista como semi-centralização ou centralização lógica, que consta inclusive nas definições de SDN da ONF.

Uma centralização absoluta, ou literal, também é algo inviável, em termos práticos. Em um datacenter, um único controlador pode até ser capaz de atender às demandas de um grande número de comutadores. Em uma escala maior, a latência de comunicação entre os elementos do plano de dados e o controlador já se tornam limitantes da tecnologia. Além disso, para cada elemento do plano de dados, o controlador deve manter um sessão ativa para controle, a própria memória do controlador já seria um limitante físico para o número de elementos controlados. Além da latência natural, relativa a distância física entre o controlador e os comutadores, existe também a latência de atualização de estado dos comutadores. Em (ROTSOS et al., 2012), nota-se que este valor pode ultrapassar a 1 segundo por elemento da rede.

Na centralização lógica, ou semi-centralização, existe uma descentralização intrínseca. Neste caso, há uma única lógica central mantida por um ou vários controladores. Pode existir vários elementos no plano de controle cuidando da distribuição de estado da rede para um grupo de comutadores. Esses controladores são governados de alguma forma por uma lógica única, provendo assim uma maior escalabilidade e/ou resiliência para a rede.

Onix (KOPONEN et al., 2010) e HyperFlow (TOOTOONCHIAN; GANJALI, 2010) apresentam alternativas para a existência de múltiplos controladores governados por uma lógica central. Com o Onix, tem-se uma plataforma que disponibiliza uma API para as aplicações terem uma visão lógica do estado da rede, sem a responsabilidade de distribuir esse estado pela rede. Já o HyperFlow é um plano de controle distribuído orientado a evento para OpenFlow que permite aplicações de controle realizar decisões localmente por meio de uma visão de rede distribuída de instancias individuais de controladores. A distribuição de estado dentro do plano de controle pode ocasionar, naturalmente, alguns casos de inconsistências para as aplicações (LEVIN et al., 2012), como acreditar ter em uma determinada visão da rede que não esteja correta em uma determinado instante.

Uma abordagem alternativa é apresentada com o Kandoo (YEGANEH; GANJALI, 2012). Com ela há uma abordagem híbrida por meio da utilização de controladores locais para aplicações locais e com um redirecionamento para o controlador global para o caso de decisões que necessitem de uma visão centralizada do estado da rede. Com isto, há uma distribuição da carga de processamento sobre o plano de controle, com o plano de dados com respostas mais rápidas para demandas de escopo local.

Fica claro, assim, que existem algumas alternativas para distribuição da lógica do controle da rede. Essa lógica pode ser setorizada para melhor atender demandas geográficas, para redução de limitações físicas ou para prover maior resiliência e segurança para rede, como também para demandas políticas e administrativas, provendo uma separação dos domínios de administração por meio de protocolo de comunicação entre controladores de diferentes domínios. A escala de centralização ou descentralização dentro do plano de controle poderá ficar a critério dos administradores da rede, que poderão avaliar as relações de perdas e ganhos e escolher o nível ideal de descentralização do seu plano de controle.

2.4.4 A evolução das redes programáveis

As ideias relacionadas a separação dos planos de controle e encaminhamento existe há muitos anos (MENDONCA et al., 2013). Enquanto o OpenFlow recebe considerável atenção da indústria atualmente, nesta seção, apresenta-se uma visão geral dos esforços anteriores relacionados às redes programáveis, baseado no trabalho de (MENDONCA et al., 2013):

REDES ATIVAS. Iniciativa propunha, em meados de 1990, a ideia que uma infraestrutura de rede que pudesse ser programável para serviços customizados. Existiam duas principais abordagens a serem consideradas: (1) comutadores programados por usuários, com a transferência de dados ocorrendo em um canal separado dos canais de gerenciamento e (2) cápsulas, que eram fragmentos de programas que podiam ser transportados na mensagem do usuário. Esses fragmentos poderiam então ser interpretados e executados pelos roteadores. Apesar de considerável atividade motivada, Redes Ativas nunca ganhou massa crítica e foi transferido para utilização muito difundida e desenvolvimento da indústria, principalmente devido a preocupações relativas a segurança e desempenho.

OPENSIG. O grupo de trabalho *Open Signaling* iniciou em 1995 com vários *workshops* dedicados a fazer as redes móveis, ATM, e Internet mais abertas, extensíveis e programáveis. Era defendida a separação entre o *hardware* de comunicação e o *software* de controle como necessários, porém um objetivo difícil de ser alcançado. Esse desafio existia devido a principalmente a roteadores e comutadores verticalmente integrados , dos quais a natureza fechada fazia do desenvolvimento rápido de novos serviços de rede algo impossível. O centro da proposta do grupo de trabalho era prover acesso ao *hardware* da rede por meio de uma interface de rede aberta e programável. Isto permitiria a implantação de novos serviços por meio de um ambiente de programação distribuída.

Motivado por essas ideias, um grupo de trabalho do IETF foi criado, que conduziu a especificação do Protocolo de Gerência de Comutadores Genéricos (do inglês, GSMP), um protocolo de uso geral para controlar um comutador de rótulo. O grupo de trabalho está oficialmente concluído e a último padrão proposto, GSMPv3, foi publicado em junho de 2002.

DCAN. O objetivo do projeto DCAN é projetar e desenvolver a infraestrutura necessária para o controle escalável e gerência de redes multisserviços [3]. É uma iniciativa também dos anos 90. Sua premissa é que as funções de gerência e controle de muitos equipamentos, comutadores ATM, deveriam ser desacoplados dos dispositivos em si e delegados para uma entidade externa dedicada a esse propósito, o que é basicamente o conceito por trás de SDN. Entre o gerente e a rede, DCAN presume um protocolo simplificado, na linha do que do que é hoje o OpenFlow. Além disso, na linha da separação de plano de controle e plano de dados, o compartilhamento da infraestrutura física pelo particionamento de recursos dos comutadores entre múltiplos controladores em uma arquitetura de controle heterogênica já foi prevista em (MERWE et al., 1998).

PROJETO 4D. Com início em 2004, o Projeto 4D (TOWARD, 2004), (GREENBERG et al., 2005), (CAESAR et al., 2005), defendia um redesenho a partir do zero, ou "lousa limpa", da arquitetura da Internet enfatizando a separação entre a lógica de decisão de roteamento e os protocolos de interação entre os elementos de rede. Os três princípios de rede robusta do projeto são [4]:

- **Objetivos de Nível de Rede**: satisfazer os objetivos de desempenho, confiança e políticas que devam ser expressos em metas para a rede como um todo, separadamente do elementos de rede de baixo nível.
- **Visão global da rede**: visão global da rede, de forma precisa e em tempo hábil, da topologia, do tráfego e dos eventos.
- **Controle direto**: A lógica de decisão deve prover operadores com uma interface direta para configurar os elementos de rede. Essa lógica não deve estar implicitamente ou explicitamente embutida fisicamente em protocolos distribuídos entre os comutadores.

O projeto 4D defende que a manipulação descoordenada e distribuída de um grande volume de "estados"entre roteadores e plataformas de gerência introduz uma complexidade

[3] http://www.cl.cam.ac.uk/research/srg/netos/old-projects/dcan/
[4] http://www.cs.cmu.edu/afs/cs/Web/People/4D/

substancial que faz tanto as redes de *backbone* quanto as corporativas cada vez mais frágeis e difíceis de gerenciar. A arquitetura 4D foi decomposta 4 planos de funções de controle: o plano de **decisão**, responsável pela criação da configuração da rede (e.g. computação das tabelas de encaminhamento para cada roteador da rede); o plano de **disseminação** que reúne informação sobre o estado da rede (e.g. se um enlace está funcionando ou não.) para o plano de decisão, e distribuir as decisões do respectivo plano para os roteadores; um plano de **descoberta** que habilita os dispositivos a descobrir seus vizinhos diretamente conectados; e o plano de **dados** para o encaminhamento do tráfego de rede.

NETCONF . Inicialmente publicado pelo grupo de trabalho da IETF pela RFC 4741 (PROTOCOL, 2006) em 2006, e atualizado pela RFC 6241 (ENNS et al., 2011), é um protocolo para configuração de rede que provê mecanismos para instalar, manipular e deletar configurações de equipamentos de rede. O protocolo permite que o equipamento disponibilize uma interface de programação (API) formal e completa. Essa API permite que aplicações possam enviar e receber conjuntos de dados de configurações parciais ou completas.

O protocolo NETCONF usa o paradigma de chamadas de procedimentos remotos (RPC). O cliente codifica um RPC em linguagem de marcação XML e o envia para um servidor usando uma sessão segura orientada a conexão. O servidor, por sua vez, responde com uma resposta codificada também em XML. Um aspecto chave do NETCONF é permitir a funcionalidade do protocolo de gerência de uma forma espelhada muito próxima das funcionalidades dos equipamentos da rede. O NETCONF é bastante similar ao SNMP (STALLINGS, 1998), que surgiu nos anos 80 e teve vários problemas relacionados à segurança, resolvidos em sua última versão. Apesar de corrigir muitas das deficiências do SNMP, de alcançar os objetivos de simplificar a (re)configuração de dispositivos e atuar de forma modular para a gerência, no protocolo NETCONF não há separação entre os planos de dados e de controle. O mesmo acontece com o SNMP.

Uma rede como NETCONF não deve ser considerada como completamente programável com toda nova funcionalidade deve ser implementada em todos os componentes da rede para que possa ser provida. O protocolo foi projetado inicialmente para cuidar da automação das configurações e não para permitir o controle direto do estado da rede e nem viabilizar o rápido desenvolvimento de serviços e aplicações inovadoras.

ETHANE. O predecessor imediato do OpenFlow foi o projeto SANE/Ethane (CASADO et al., 2007), o qual definiu em 2006 uma nova arquitetura para redes corporativas. O foco do projeto estava em utilizar um controlador centralizado para gerência de políticas e de

segurança em uma rede. De forma similar a SDN, Ethane aplicou dois componentes: um controlador para decidir se um pacote deveria ser encaminhado, e um comutador Ethane composto de uma tabela de fluxos e um canal seguro para comunicação com o controlador. Ethane lançou as bases para o que se tornaria Redes Definidas por Software.

É importante ressaltar que a capacidade de ser programável expressa nesta seção está no sentido amplo. O alcance da "programabilidade"pode chegar em ações do plano de controle ou do plano de dados.

A API disponibilizada pelo *OpenFlow*, por exemplo, não expõe diretamente a programabilidade do plano de dados. O OpenFlow viabiliza a exposição da programação do plano de controle. Este possui uma interface bem definida para popular as tabelas de fluxos, como é visto na Seção 2.5, e disponibiliza alguma interface de programação que abstrai os recursos da rede para as aplicações.

Grosso modo, há uma grande similaridade entre a relação dos programas de um computador com seu sistema operacional, e deste para com o *hardware* em si, com a relação das aplicações de rede com o controlador, e deste para com os recursos de infraestrutura de comutação da rede.

2.5 OpenFlow

Para viabilizar a existência da arquitetura de rede SDN, como em qualquer outra, é necessária a adoção de alguns padrões que governem o desenvolvimento e a evolução da interoperabilidade dos componentes dessa arquitetura. Atualmente, o padrão que se tornou muito utilizado na interação entre o plano de controle e de dados é o *OpenFlow* (MCKEOWN et al., 2008).

Este padrão emergiu das Universidades e hoje é mantido pela *Open Networking Fundation* (ONF), uma conglomeração de grandes empresas provedoras de conteúdo, de telecomunicações e de computação em nuvem, com o objetivo de organizar o desenvolvimento de padrões abertos de redes definidas por software. As especificações do *OpenFlow* definem os comutadores *OpenFlow* e o protocolo de comunicação entre o plano de dados e o de controle.

Em termos gerais, os comutadores são compostos por uma ou mais tabelas de fluxos, com as informações que definem a forma de encaminhamento dos fluxos, e por um canal de comunicação seguro com o controlador. Para utilizar este canal, o comutador gera e recebe um conjunto de eventos e mensagens para/do controlador, por meio de uma interface de controle de acordo com a especificação do protocolo. Já este define as estruturas, os eventos e as formas de trocas de informação entre o plano de dados e o de controle.

A arquitetura *OpenFlow* pode ser vista mais claramente na Fig. 2.4. Em uma visão geral, os pacotes, ao chegarem em uma interface, têm os cabeçalhos analisados para a identificação dos fluxos. Se os cabeçalhos identificados corresponderem a alguma regra definida em uma das tabelas - um fluxo previsto pelo plano de controle - as ações são tomadas e as estatísticas atualizadas no comutador. Essa é a principal função do plano de dados. Caso não exista uma regra definida para um determinado fluxo, o comutador utiliza-se do protocolo *OpenFlow* para comunicar-se com o controlador da rede e definir o tratamento para aquele conjunto de pacotes similares subsequentes. Esta comunicação é feita por um canal seguro e, normalmente, separado do plano de dados.

O protocolo governa as trocas de mensagens e informações entre o controlador e os comutadores da rede, e faz o papel de interface de fronteira-sul, ou seja, a relação entre o sistema operacional da rede e a infraestrutura de encaminhamento.

No controlador, as mensagem são tratadas e repassadas para as aplicações de forma simplificada. Essa comunicação de "fronteira-norte"ainda não é padronizada e fica fortemente dependente da API de desenvolvimento de cada controlador. O "controlador *OpenFlow*", na verdade, é qualquer software de controle que dê suporte ao Protocolo e proveja uma interface definida de programação para as aplicações de rede.

As aplicações, por fim, são as responsáveis por definir a lógica de comunicação da rede e por governar, por intermédio do controlador, o comportamento dos pacotes pela rede.

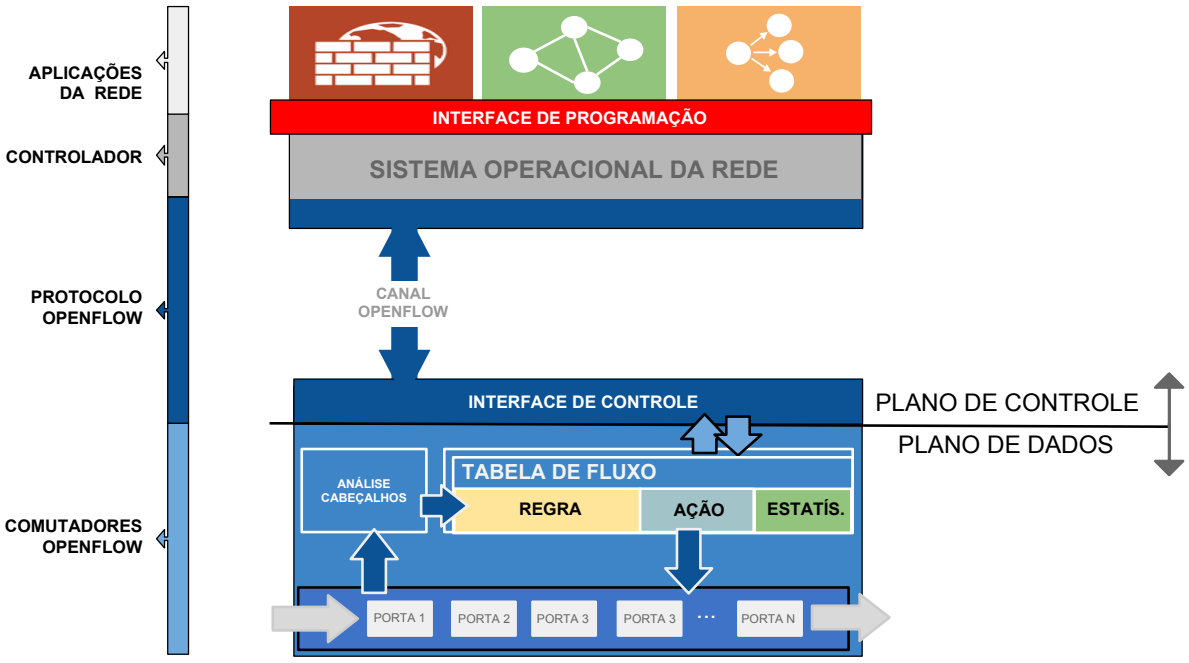


Figura 2.4: Arquitetura básica dos elementos de uma rede *OpenFlow/SDN*

2.5.1 Visão Geral do Protocolo

A troca de mensagens entre o controlador da rede e os comutadores da infraestrutura de encaminhamento é feito pelo Canal *OpenFlow*, ilustrado na Fig. 2.5. Por esse canal é estabelecida uma conexão direta entre um controlador e um comutador. Essa conexão, normalmente, é realizada por uma rede a parte do plano de dados, e estabelecida por uma sessão TCP, que pode ser encriptada ou não. Pelo Canal *OpenFlow* o controlador configura e gerencia o comutador, recebe eventos, e controla a ocupação da tabela de fluxo no plano de dados. Essa tabela é a responsável por manter o estado de encaminhamento do comutador. O formato dessas mensagens e os eventos são definidos pelo Protocolo *OpenFlow*.

Segundo especificações do comutador OpenFlow 1.3.0 (SPEC, 2012), o protocolo suporta três tipos de mensagem: controlador-para-comutador, assíncrona e simétrica, cada uma com múltiplos subtipos. As mensagens controlador-para-comutador são iniciadas pelo controlador e usadas para gerenciar diretamente e inspecionar o estado do do plano de dados. As mensagens assíncronas são iniciadas pelo comutador e usadas para atualizar o controlador sobre eventos e mudanças no estado da rede. As mensagens simétricas podem ser iniciadas tanto pelo controlador quanto pelo comutador e são envidas sem solicitação, como mensagens de início de sessão ou de *keep alive*.

O controlador gerencia múltiplos canais de controle, um para cada comutador. Nas versões mais recentes do protocolo, o comutador também pode gerenciar mais de um canal para prover maior resiliência pela interface com múltiplos controladores. Caso a conexão entre o controlador e o comutador seja interrompida, ainda nas versões mais recentes do protocolo, o comutador entra imediatamente no estado de "modo-seguro"ou em modo-*standalone*", dependendo da configuração e implementação do comutador (SPEC, 2012). No primeiro caso, apenas as mensagens e pacotes destinados ao controlador são rejeitadas, no outro caso, o comutador passa a encaminhar como um comutador Ethernet convencional, e por esta razão esse modo está disponível apenas nos comutadores híbridos, como visto a seguir na seção 2.5.3.

Numa visão geral, estes são os principais aspectos definidos pela especificação *OpenFlow*, maiores detalhes podem ser encontrados na (SPEC, 2012), ou em versões anteriores.

2.5.2 A Rápida Evolução do Padrão

Devido ao grande volume de código disponibilizado para a comunidade e a latente demanda da academia por soluções alternativas para experimentação em redes, o *OpenFlow* tornou-se rapidamente um padrão para redes SDN. Sua especificação passou a ser governada por um con-

Tabela 2.1: A evolução das principais funcionalidades suportadas pelas versões do *OpenFlow* (SPEC, 2013)

Versão	**Lançamento**	**Principais recursos adicionados**
1.0 (0x01)	Dezembro / 2009	• Fatiamento (*Slicing*); • Identificação de cabeçalho Tipo de Serviço/DiffServ do IP; • Substituição de SSL por TLS, para encriptação; • Identificação de cabeçalho IP in pacotes ARP;
1.1 (0x02)	Fevereiro / 2011	• Suporte a múltiplas tabelas; Instruções; tabela de falhas; • Grupos (conjunto de portas); • Suporte a cabeçalhos QinQ e MPLS; • Suporte a portas virtuais; • Adiciona o decremento de TTL;
1.2 (0x03)	Dezembro / 2011	• Identificação e config. de cabeçalho extensível (OXM); • Suporte a básico a IPv6 e Ia CMPv6; • Renomeação das “portas virtuais” para “portas lógicas”;
1.3 (0x04) LTS	Abril / 2012	• Suporte a tag de PBB (Provider Backbone Bridge); • Flexibilização na ordem das tags dos cabeçalhos; • Suporte a empilhamento de tag MPLS (MPLS BoS bit); • Filtro de eventos por canal de controle; • Suporte a cabeçalhos de extensão IPv6;
1.4 (0x05) EXP	Agosto / 2013	• Suporte as propriedades ópticas nas portas dos comutadores; • Monitoramento de fluxos (para multiplos controladores); • Tratamento de tabelas de fluxos cheias; • Mudança da porta padrão TCP do canal OpenFlow p/ 6653;

glomerado de empresas, o que impulsionou um desenvolvimento veloz de novas especificações e alavancou o suporte do protocolo em alguns equipamentos da indústria de redes.

A principal versão largamente adota pela indústria de redes atualmente é a versão 1.0. Alguns fabricantes já prometem atualizações para a nova versão 1.3 do padrão, considerada a versão estável, ou LTS (EWG, 2013). Apesar disso, atualmente a versão mais utilizada em experimentações e com suporte em uma grande variedade de equipamentos é a versão 1.0.

As versões iniciais da especificação do *OpenFlow* foram responsáveis pelo seu amadurecimento durante o seu desenvolvimento acadêmico. Em 28 de março de 2008 foi lançada a versão 0.2. Até a versão 1.0, lançada em 31 de dezembro de 2009, foram 3 versões intermediárias (0.2.x, 0.8.x, e 0.9.x). Nessas versões, foram aprimoradas diversas características do protocolo, como: a adição de mensagem de erros e estatísticas; identificação de fluxo com máscara de rede; suporte a mensagens genéricas para extensões de fabricantes; tratamento de loops na rede por *spanning tree*; melhorias na manipulação das tabelas de fluxos diversas; tratamento do cabeçalho VLAN, suporte rudimentar à falhas no controlador; etc.

Na versão 1.0, o *OpenFlow* teve sua grande expansão no meio acadêmico, transbordando para a indústria. Em março de 2011, logo após o lançamento da especificação 1.1, foi anunciada a formação da ONF [5]. Por haver diversas implementações funcionais tanto do comutador quanto no plano de controle, a versão 1.0 se tornou muito popular no meio acadêmico ao redor do mundo, sendo considerada a primeira versão estável do protocolo. Nessa versão, conforme Tab. 2.1, podemos destacar o inicio do suporte básico ao fatiamento de recurso de rede, pelo suporte de múltiplas filas por porta de saída, o que viabilizou um suporte mínimo a garantia de banda (SPEC, 2013).

No início de 2011 foi lançada a versão 1.1 da especificação. Nessa versão surgiu o suporte a tabelas múltiplas, que viabilizaram a aplicação de um conjunto de ações a um pacote, possibilitando um tratamento mais refinado no encaminhamento dos fluxos no plano de dados. O suporte básico ao tunelamento de Vlans (IEEE 802.1ad ou QinQ) e ao cabeçalho MPLS foram adicionados nessa versão. A partir daí, começou-se a vislumbrar a utilização do *OpenFlow* mais próximo ao núcleo das redes. Cabe destacar a opção de decremento de TTL, que viabiliza a comutação de fluxo baseada em roteamento IP ou encaminhamento MPLS.

Já no final do mesmo ano de lançamento da versão 1.1, em dezembro de 2011, foi disponibilizada a especificação 1.2 do *OpenFlow*. Nessa versão iniciou-se a previsão de suporte básico ao IPv6 e foi iniciado o suporte a OXM , identificação extensiva de cabeçalhos. Nessa nova forma de análise, os cabeçalhos são identificados baseados em estruturas tipo-comprimento-valor (TLV). Nas versões anteriores a identificação dos cabeçalhos era realizada pela identificação dos tamanho estáticos de cada campo, devidamente previstos pela estrutura de dados ofp_match. Com o OXM os campos foram reorganizados, com muitos campos sobrepostos, como no caso dos campos da camada de transporte. Além disso, a identificação baseada em tipo-comprimento-valor permite a utilização de novos cabeçalhos experimentais, o que viabiliza um melhor suporte a inovações na estrutura de computação do plano de dados.

Em 2012 foi lançada a versão 1.3 da especificação. Nessa versão iniciou-se o suporte a empilhamento de tag MPLS e o suporte a tag de PBB, para o provimento de conectividade sobreposta de camada 2. Nessa versão também foi aprimorado o suporte ao IPv6 pela previsão dos cabeçalhos de extensão. O plano de controle passou a dar suporte a filtros de eventos por canal de controle, permitindo que diferentes controladores cuidem de eventos específicos da rede. Essa versão da especificação é considerada LTS, como uma versão estável da especificação, em sucessão à versão 1.0.

Para experimentações e testes, foi lançada a versão 1.4 em agosto de 2013. Nessa especi-

[5]https://www.opennetworking.org

ficação já estão previstas, por exemplo, o início da integração com a camada física em redes ópticas, pelo suporte básico à atributos ópticos nas portas dos comutadores.

Fica evidente que a especificação do *OpenFlow* teve um desenvolvimento rápido, com atualizações da especificação realizadas com grande periodicidade. Fica claro que a evolução do protocolo caminha na integração tanto no acesso da rede, em redes de Datacenter, mas também na utilização do Protocolo mais próxima ao núcleo da rede. Para ser definitivamente aplicado de fato no núcleo das redes, o protocolo ainda precisa endereçar algumas questões importantes relacionadas a centralização do controle e a latência inerente a esta comunicação. Além disto, para aplicação em redes de alto desempenho é necessária o endereçamento de problemas relacionados ao grande volume de estados necessários para controlar o estado de comutação da rede, que serão analisados a seguir.

2.5.3 Visão Geral do Plano de Dados

Como pode ser visto na figura 2.4, os elementos de encaminhamento de uma rede *OpenFlow* são compostos por um *hardware* de encaminhamento acessível por uma interface de controle. Esse *hardware* deve ser o mais simples possível, com a maior parte da lógica e a inteligência do encaminhamento desacoplada e processada remotamente no plano de controle via o Protocolo *OpenFlow*.

Os comutadores *OpenFlow* podem ser de dois tipos: puros ou híbridos. Os comutadores puros não possuem nenhuma funcionalidade legada das redes convencionais e são encaminhadores inteiramente governados pelo controlador da rede. Os híbridos são aqueles que além do protocolo *OpenFlow* dão suporte à operações e protocolos tradicionais (MENDONCA et al., 2013).

O elemento comutador *OpenFlow* é caracterizado por possuir uma ou mais tabelas de fluxo, e uma tabela de grupo - nas versões mais recentes -, que são utilizadas para a definição do estado de encaminhamento para cada fluxo passante pelo equipamento. Esse estado é definido por regras. Estas são definidas por um arranjo de cabeçalhos, que identificam um determinado fluxo, e por uma ação associada a essa regra. Além disto, as regras possuem estatísticas que são atualizadas e servem para gerenciamento por parte do plano de controle.

Se não houver *match* para o pacote na tabela de fluxo ocorre uma "falha de tabela", para implementações a partir da versão 1.3. Nesse caso, o comportamento do pacote é governado pela configuração da tabela de falhas, que controla o tratamento de determinados fluxos não identificados e pode, por exemplo, enviar o pacote para o controlador, rejeitar o pacote ou

envia-lo para uma tabela subsequente. No caso de comutadores híbridos pode acontecer o encaminhamento do pacote para uma porta para a uma rede convencional.

A definição de estado na rede *OpenFlow*, por depender de uma decisão tomada em um ponto remoto, possui um tempo extra para o tratamento dos pacotes. Para reduzir essa latência, a granularidade da rede *OpenFlow* é baseada em fluxo, ou seja, num conjunto de pacotes identificados por cabeçalhos comuns. Assim, apenas o primeiro pacote do fluxo é analisado pelo controlador. Os próximos são tratados diretamente pelo comutador pela regra definida e instalada pelo plano de controle. Essa é a abordagem é conhecida como reativa.

Uma alternativa para reduzir a latência é realizar uma abordagem pró-ativa. Nela, o controlador já define previamente a forma de encaminhamento de um conjunto de fluxos comuns à rede. Com isto, há uma significativa melhora no processo de tratamento dos pacotes, a custo de uma redução na flexibilização no tratamento de fluxos. De fato, os fluxos mais comuns podem ter as regras definidas pró-ativamente com os demais sendo tratados de maneira reativa.

2.5.4 Evolução do Processo de Tratamento do Pacote

Para entender o funcionamento do comutador OpenFlow, é importante conhecer como é realizado o tratamento dos pacotes para identificação do fluxo no plano de dados. A computação envolvida nesse processo e o modo de implementação desse fluxograma impactarão diretamente na eficiência de encaminhamento da rede.

O fluxo de tratamento de pacote, na versão 1.0, está ilustrado na Fig. 2.5. Após a entrada do pacote no comutador são realizadas buscas seguidas nas tabelas de fluxos existentes, e se nenhuma regra ser encontrada, o pacote é enviado para o controlador. Caso contrario, a ação relacionada é diretamente aplicada.

Na versão 1.1 e 1.2, com fluxograma ilustrado na Fig. 2.6, esse processo passou a acontecer em múltiplas tabelas com as ações sendo aplicadas em conjunto no final da busca. O pacote ao ser analisado e ter seu cabeçalho identificado, origina uma busca por regra na primeira tabela. Se essa regra correspondente for localizada, os contadores são atualizados e são executadas três atualizações: do conjunto de ações, dos conjuntos de campos do pacote e dos metadados - utilizados para passar informação entre as tabelas. Se a regra não for localizada,e dependendo da configuração da respectiva tabela, podem ocorrer um de três eventos: o pacote ser enviado ao controlador, ser rejeitado ou a busca continuar na próxima tabela (SPEC, 2011a, 2011b).

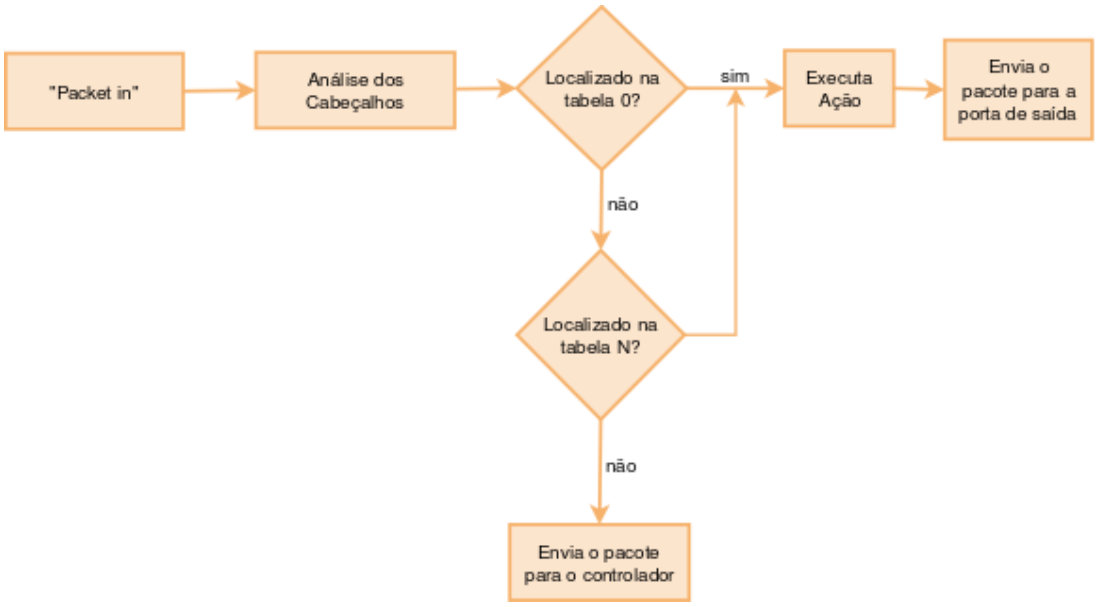


Figura 2.5: Fluxo de tratamento de um pacote no comutador OpenFlow 1.0 (SPEC, 2009).

Apenas na versão 1.3 e 1.4 do *OpenFlow* há uma alteração nesse fluxo. A partir dessa versão surge a tabela de faltas, ou tabela de falhas. As regras definidas nessa tabela determinam como os pacotes não localizados na tabelas de fluxo serão tratados. Eles podem, por exemplo, ser enviados para o controlador, rejeitados ou direcionados para a próxima tabela de fluxo, ou para uma porta conectada em uma rede tradicional. (SPEC, 2012, 2013).

Como pode ser visto, o processo de comutação de pacote com *OpenFlow* possui um crescente volume de computação para identificação dos fluxos e para a realização da comutação. Esse conjunto de condições e testes, com múltiplos processos de busca e localização de regras agrega maior complexidade para o desenvolvimento de *hardware* especializados para as atividades do plano de dados com alto desempenho.

A simplificação gerada pelo *OpenFlow* se concentra no desacoplamento do plano de controle do plano de dados, com a viabilidade de centralização da lógica e padronização da interface de controle da comutação da rede. Sob o ponto de vista da implementação do encaminhamento, a especificação do *OpenFlow* tende a induzir uma maior variabilidade na latência do tratamento de pacotes, que pode ser severamente agravado pela falta de poder computacional no *hardware* de encaminhamento.

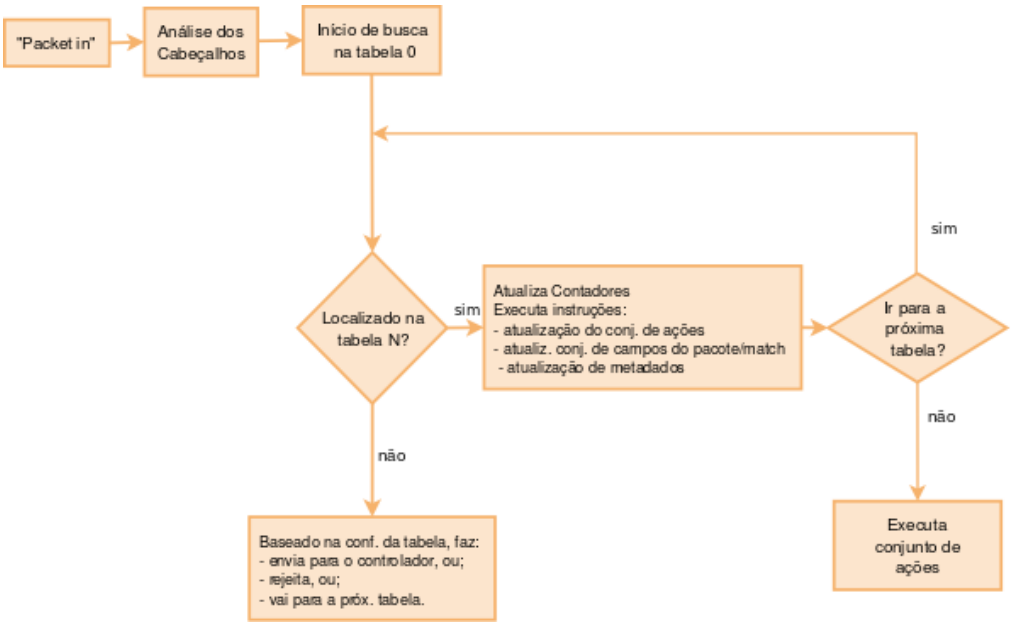


Figura 2.6: Fluxo de tratamento de um pacote no comutador OpenFlow 1.1/1.2 (SPEC, 2011a, 2011b).

2.6 Limitações do OpenFlow no Núcleo das Redes Definidas por Software

2.6.1 O Gargalo no Armazenamento de Estado

Apesar de prover grande simplificação para o plano de controle e suas aplicações, o padrão *OpenFlow* possui uma grande complexidade na sua forma de manutenção do estado da rede no comutador. Isto se deve, principalmente à necessidade de armazenar mais informações, ações e estatísticas para cada regra inserida, como pode ser visto na Fig. 2.8.

O espaço necessário para armazenar e tratar essas informações de maneira eficiente passa a ser um desafio para a arquitetura do comutador, principalmente se o for considerada uma produção de menor custo. Empresas como a Broadcom, a Intel e a Marvell comercializam ASIC com combinados de funções Ethernet que simplificam significativamente o projeto do plano de dados. As funções como análise de cabeçalhos, *buffering*, escalonamento e modificação dos pacotes e toda a busca nas tabelas são realizadas diretamente pelo *hardware* especializado (LU et al., 2012). Os comutadores e roteadores tradicionais possuem seu plano de encaminhamento realizado por estes ASICs, o que simplifica o projeto do plano de dados do equipamento. O desempenho do equipamento cresce, assim como seu preço, quanto mais funções são associadas ao *hardware* especializado.

A utilização desses ASICs com muitas funções incorporadas ocasiona, também, as prin-

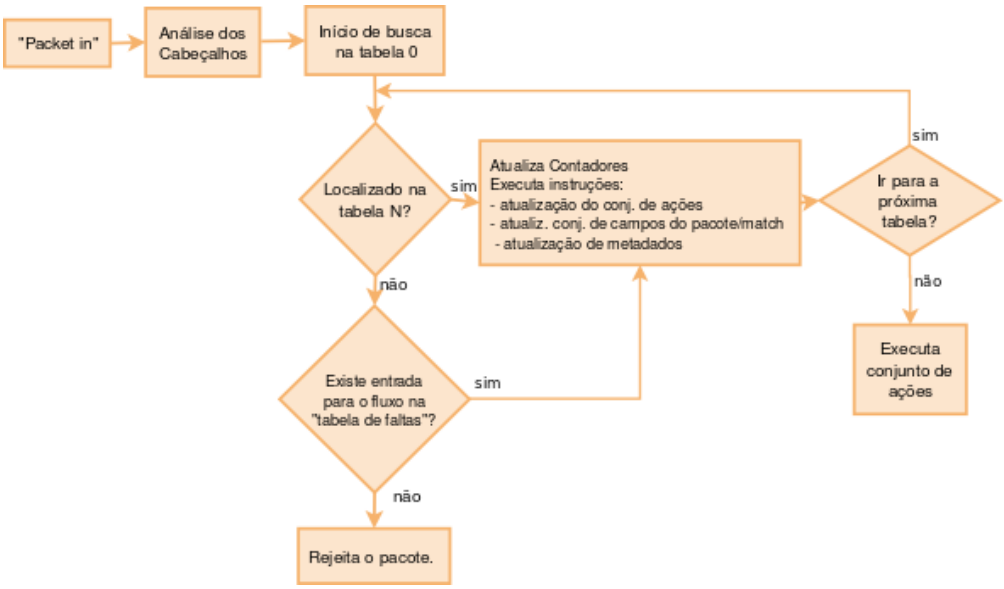


Figura 2.7: Fluxo de tratamento de um pacote no comutador OpenFlow 1.3/1.4 (SPEC, 2012, 2013).

cipais limitações do plano de dados em redes *OpenFlow*. O número de entradas de encaminhamento se torna muito limitado para o tratamento de fluxos. Além disto, a realização de *buffer* dos pacotes no chip destes ASICs limita ainda mais a área do chip e pode ser facilmente inundado por rajadas de tráfego, o que pode degradar significativamente o desempenho da comunicação TCP (LU et al., 2012).

A Fig. 2.8 ilustra comparativamente a manutenção de estado de encaminhamento em diferentes tabelas. Comparando a tabela de manutenção de rotas do IP e a de encaminhamento de quadros do Ethernet com a tabela de fluxos do *OpenFlow*, percebe-se que o volume de informação a ser armazenada é significativamente maior. Para exemplificar, um comutador ASIC da Broadcom de última geração possui cerca de 2 mil entradas para realização de *match* em fluxos com 5-tuplas TCP/IP (LU et al., 2012). Neste estudo é proposto a utilização de CPU mais próxima ao ASIC atuando com um co-processador de tráfego para mitigar estes problemas.

A manutenção de estatísticas feita diretamente na tabela de estados tem uma implementação complexa no ASIC e ocupa uma área significativa. Na versão 1.1 do *OpenFlow*, por exemplo, são especificados três contadores para cada entrada de fluxo cada um com 64 bits, 192 bits (24 bytes) extras de armazenamento para cada entrada na tabela (MOGUL; CONGDON, 2012).

O custo do comutador em ASIC está diretamente ligado com sua área. Assim, a área ocupada pelos contadores impacta na ocupação de outras funções importantes, principalmente para o caso de comutadores *OpenFlow* híbridos (MOGUL; CONGDON, 2012). Como visto na Sec. 2.5.1, o volume de tabelas aumentou com a evolução das versões da especificação do Protocolo,

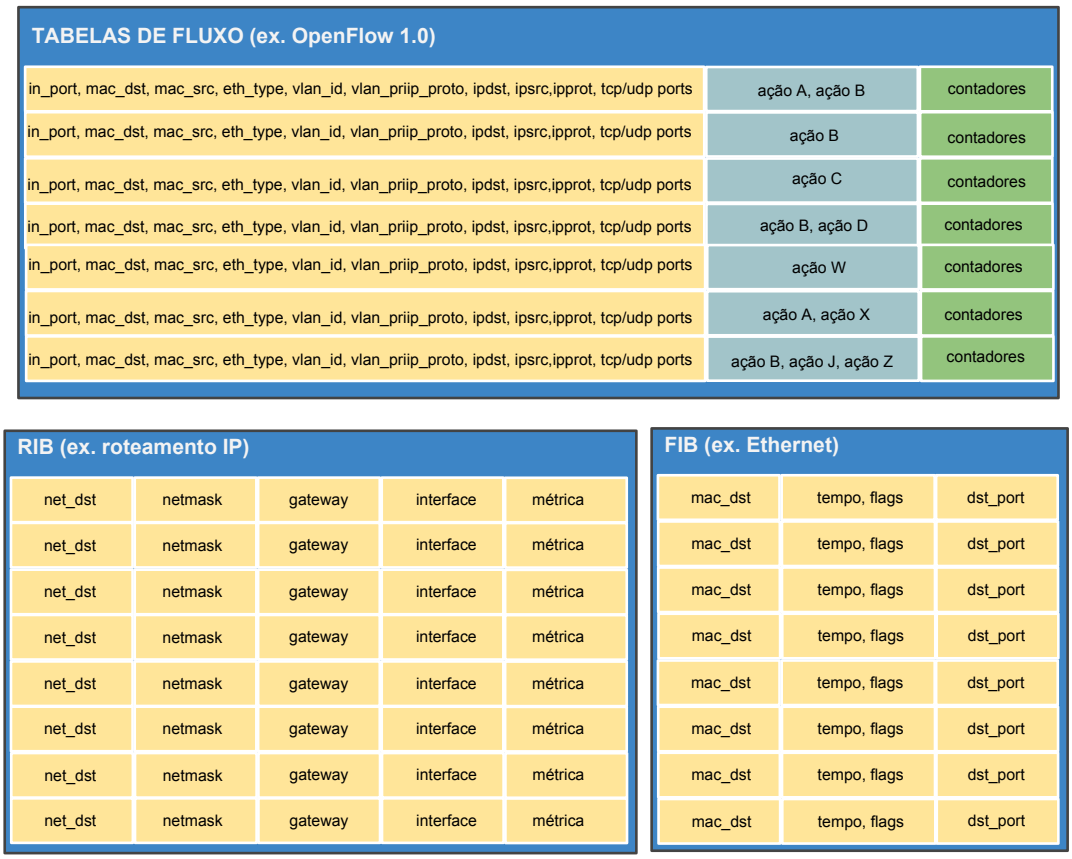


Figura 2.8: Arquitetura básica dos elementos de uma rede *OpenFlow/SDN*

o que reforça a demanda por área para os casos de implementação direta no ASIC.

Para manter as funcionalidades sem alterar significativamente o ASIC, muitas dessas funções são passadas para a unidade central de processamento do equipamento, que possui capacidade bastante limitada. Isto pode ser crítico em função do ambiente de comunicação onde o *OpenFlow* for utilizado. Para ambientes com alta restrição de latência e jitter, a utilização dos comutadores *OpenFlow* deve ser analisada com maior critério.

2.6.2 O Impacto da Arquitetura do Comutador no Plano de Dados

Os comutadores de uma forma geral são formados por uma estrutura de processamento dos pacotes de entrada, uma de enfileiramento e, por fim, uma estrutura que define a porta de saída do pacote realizando esta etapa final. Dentro da diversidade de fabricantes de comutadores *OpenFlow*, os detalhes de implementação dessas estruturas são realizados de diferentes formas, impactando severamente na qualidade, em termos do tempo de resposta, e nos custos finais do equipamento.

Aplicações que necessitem de grande restrições de latência no encaminhamento do plano de dados podem sofrer pela falta de interface com detalhes das implementações dos comutadores e da forma com que a comutação é implementada nos equipamentos. Diferentes fabricantes implementam os detalhes do processamento dos pacotes no plano de dados de diferentes formas, com isso operações determinadas pelo plano de controle podem ter um desempenho diferenciado com relação a implementação do comutador pelo fabricante (ROTSOS et al., 2012).

Isso pode impactar significativamente um plano de dados com equipamentos de diferentes fabricantes. Os detalhes da implementação, da mesma forma que afeta o processamento de pacotes na entrada dos comutadores, também pode afetar na tempo de resposta das memórias responsáveis pelo armazenamento das tabelas de fluxo. Para a atualização de cerca de 1000 fluxos, o tempo de inserção nas tabelas pode variar de 1s até 10s por elemento da rede (ROTSOS et al., 2012), dependendo do fabricante do comutador. Essa variabilidade pode impactar diretamente na convergência de estados no plano de dados.

O padrão *OpenFlow* não apresenta mecanismo de gerenciamento, ou controle, dos detalhes de comutação de cada equipamento. As informações previstas relacionadas pelo protocolo são simplistas para aplicações que requeiram alto desempenho. Em comutadores com múltiplos módulos, por exemplo, as TCAM podem ser centralizadas e gerenciadas por uma única entidade (e.g Cisco Catalyst 4500 Supervisor) ou podem existir em cada placa e serem gerenciadas por uma CPU local (e.g HP ProCurve 5400, Brocade MLX-e).

Normalmente os comutadores são compostos por placas especializadas (ASIC) capazes de comutar com baixíssima latência, ou com "taxa de linha"ou "velocidade do fio". Porém as funcionalizadas dessas placas especializadas é limitada. Toda função não suportada diretamente nas placas de encaminhamento é transferida para a CPU central do comutador. Essas unidades já possuem capacidade limitada, e normalmente são utilizadas apenas para processamento básico das sinalizações de controle. Normalmente utilizam memória de acesso aleatório (SRAM ou DRAM), que possuem respostas mais lentas em relação às implementações das memórias CAMs.

As memórias de acesso de conteúdo (CAM) são posicionadas próximas ao *hardware* especializado e possuem capacidade de armazenamento bastante limitada, devido ao seu custo. Essas memórias diferenciam-se das de acesso aleatório pela sua forma de busca das informações. Enquanto as memórias tipo RAM retornam um conteúdo para um dado endereço de entrada, as memórias tipo CAM retornam um endereço para um dado conteúdo de entrada. Assim, pelo suporte mais dinâmico ao tipo de entrada essas memórias fazem uma busca em todo o conteúdo da memória para retornar o endereço relacionado ao dado fornecido. Por esta maior complexidade

e pela necessidade de prover respostas rápidas, as memórias tipo CAM são caras e representam um fator importante na composição do custo e da eficiência dos comutadores. Elas são utilizadas, por exemplo, nas tabelas de encaminhamento de camada 2, no armazenamento dos MAC que definem as portas de saída dos quadros de entrada.

Um tipo especial de memória CAM é a TCAM (CAM Ternária). Enquanto as memórias CAM podem apenas fazer localização de dois estados na memória (0 ou 1), as CAM ternárias suportam um terceiro estado: "qualquer valor", representado normalmente pelo símbolo * (asterisco). Essas memórias são utilizadas para armazenamento de informações de mais alto nível, como prefixos IP e filtros de listas de controle de acesso (ACL).

A complexidade de implementação das TCAM é ainda maior que da memória CAM. A área do chip necessária para sua instalação é ainda maior e consequentemente, o custo dos equipamentos com maior disposição de TCAM tende a ser significativamente maior.

Para a comutação baseada em *OpenFlow* com alto desempenho, as TCAM têm papel fundamental no armazenamento das tabelas de fluxo. Contudo, o tamanho limitado dessas memórias já é um fator crítico para o volume de fluxos suportados pela rede. O compartilhamento dessas memórias para coexistência de serviços em comutadores híbridos agrava a falta de espaço, pela necessidade eventual de armazenamento de tabelas de prefixo IP, IPv6 ou até ACLs.

A implementação da arquitetura de comutação varia em função das decisões tomadas pelos fabricantes, visando potencializar serviços ou reduzir custos. Assim, comutadores de diferentes fornecedores podem apresentar discrepâncias no desempenho do processamento dos pacotes no plano de dados (YU; WUNDSAM; RAJU, 2014). Funcionalidades não implementadas diretamente em *hardware* especializados são tratadas pelo sistema operacional dos comutadores, com um desempenho pior. A diversidade de formas de implementação do processamento dos pacotes e do armazenamento de regras cria um grande desafio para as aplicações de redes entenderem completamente a capacidade real dos caminhos existentes no plano de dados (YU; WUNDSAM; RAJU, 2014). Essa diversidade não será eliminada, pois existem mercados específicos com diferentes requisitos que podem ser atendidos em função dos custos aceitáveis para determinado setor.

Para o núcleo das redes, reduzir o custo dos ativos implica em prover conectividade fim-a-fim de forma mais barata. Contudo, reduzir o custo pela utilização de memórias mais baratas provoca uma acentuada redução na qualidade do serviço.

Reduzir o volume de informações de estado no núcleo das redes é algo fundamental para a redução do custo dos equipamentos de maneira equilibrada com a manutenção da capacidade

de encaminhamento de pacotes. Essa redução se torna um grande desafio para redes baseadas em *OpenFlow*, devido ao grande volume de estados inerente ao protocolo e ao grande volume de sinalizações necessárias entre os elementos da infraestrutura de comutação e o elemento do plano de controle.

2.6.3 Limitações das Tecnologias Atuais para Conectividade Fabric no Núcleo das Redes Definidas por Software

Em (CASADO et al., 2012) é apresentada a necessidade das SDN suportarem um núcleo "*fabric*", com a preocupação principal de entrega/encaminhamento de pacotes de uma origem a um destino sem utilização do cabeçalho do pacote original para encaminhamento no núcleo. Sugere-se, assim, a utilização de comutadores *OpenFlow* diferenciados em borda e núcleo, onde o núcleo utilizaria de funções comuns ao MPLS. A proposta do KeyFlow, apresentada no Cap 3 se alinha fortemente com as propostas de (CASADO et al., 2012), porém, como detalhado na seção 3.3, no que tange a solução para rede "*fabric*" entende-se que o KeyFlow pode propiciar uma maior simplificação em relação ao MPLS para sinalização e convergência de caminhos lógicos.

O estabelecimento de conectividade por meio da utilização de uma infraestrutura MPLS tem a fragilidade de depender de complexos protocolos de reserva de recursos para definição, reservas e manutenção de caminhos fim-a-fim. Além disso, os equipamentos dessas redes devem suportar a sinalização dos protocolos para reserva de recursos necessária para convergência da conectividade. Isso implica em equipamentos mais elaborados, calcados em sistemas operacionais, o que aumenta o custo operacional da rede. Caso a implementação desses comutadores seja realizada em *hardware* genéricos, haverá uma limitação na escalabilidade da rede, pois não haverá uma especialização e uma otimização para uma rápida convergência de caminhos no núcleo da rede.

Caso o núcleo *fabric*, aquele especializado no transporte de pacotes, seja realizado com comutadores OpenFlow, haverá a limitação relativa ao grande número de interações entre o plano de controle e o plano de dados para manter o estado da rede coerênte frente à demandas dinâmicas. Estas limiteação deve ser evitada no núcleo da rede com o objetivo de reduzir a latência de sinalização e manutenção dos caminhos nas tabelas de fluxos dos comutadores. Para melhorar a eficiência na comutação de pacotes em redes *OpenFlow*, uma estratégia é minimizar as interações dos comutadores com o controlador (CURTIS et al., 2011). Embora as limitações não sejam expressívas quando se utilizam regras fixas pré-instaladas, quando empregadas em redes com muitos elementos e sob demandas muito dinâmicas e aleatórias, a interação com o

controlador no plano de controle ocorre de forma mais acentuada. Além disso, a necessidade de controle frequente das estatísticas de comutação também pode afetar a capacidade final de encaminhamento do comutador, afetando a escalabilidade da rede. Uma estimativa da capacidade deste valor no equipamento desenvolvido pela fabricante HP, o ProCurve 5406z, é de 275 fluxos por segundo. Isto indica que a manutenção de estados completa (*stateful*) dos fluxos ativos é um problema para a escalabilidade das redes definidas por *software* (SDN).

O arranjo OpenFlow-KeyFlow alcança a flexibilidade no tratamento de fluxos pela utilização do OpenFlow nas bordas, com uma alta eficiência e baixo custo de implementação para comutadores especializados no transporte dos pacotes sem manutenção de estado no núcleo da rede, para todos os melhores caminhos definidos para a rede de núcleo, dando ao núcleo da rede uma similaridade com a arquitetura de um grande comutador *fabric* que interconecta suas portas ligadas à "borda"da rede.

2.7 Conclusão

A partir de uma iniciativa acadêmica, por meio do projeto OpenFlow, os conceitos de SDN tomam força para o redesenho das redes do futuro. O desacoplamento dos planos de controle e de dados, com uma interface aberta para manipulação do estado da rede tendem a modificar profundamente a infraestrutura e os serviços disponíveis em rede.

Essa modificação se dá na maneira com que os componentes de tecnologia se organizam, de forma a simplificar o processo de surgimento de inovações para os serviços de rede e para a Internet. A especificação de um padrão para governar os mecanismos de interação entre um elemento facilitador, o controlador, e os recursos de comunicação, o plano de dados, está cada vez mais maduro, principalmente quando analisadas a rapidez da evolução das especificações OpenFlow, e a velocidade com que este padrão já chega à indústria de rede.

Da mesma forma com que questões estratégicas da arquitetura do padrão OpenFlow ainda são discutidas em respeito a seu plano de controle, como os formatos e padrões de interfaces, mecanismos de resiliência do controlador e da rede de dados, interfaces amigáveis para o desenvolvimento de aplicações para rede, etc. Também fica claro que existem questões importantes a serem endereçadas a respeito do plano de dados, e na própria interação deste plano com a inteligência da rede.

Apesar de expor uma abertura para a programabilidade da forma de comutação de pacotes na rede, as especificações OpenFlow/SDN proveem pouca abertura para uma revisão ou para o fornecimento de programabilidade no plano de dados em si. Dessa forma, dificilmente dife-

rentes equipamentos em um parque de rede heterogêneo podem convergir de maneira coerente no estabelecimento de caminhos e no fornecimento de garantias de serviços mínimas para a rede, principalmente, mantendo-se em mente a necessidade de redução de custos de projeto e de operação da infraestrutura de interconexão de redes.

A utilização da estrutura SDN como provedora de caminhos entre redes ainda é algo pouco abordado. Isto se deve principalmente pelas limitações das definições de comutadores fortemente baseados em estruturas dependentes de manutenção de estados em memória, como as tabelas de fluxos. Essa dependência é a raiz dos problemas encontrados para a escalabilidade e a redução de custos das infraestruturas, e já se apresentam nas discussões e em projetos de redes definidas por *software*. Fica evidente que há uma necessidade de utilização de mecanismos eficientes e flexíveis para interconexão dos comutadores OpenFlow, de forma a estender o potencial de SDN para o núcleo das redes. Com isso, será possível a criação de conexões flexíveis aos fluxos transientes na rede, de maneira que os requisitos de qualidade de serviço de conexão possam ser atingidos de maneira mais granular no núcleo. É importante que estes novos mecanismos possam ser realizados por meio de equipamentos especializados no transporte de pacotes, de forma determinística e com implementação simplificada de forma a reduzir tanto o custo CAPEX e OPEX da infraestrutura de transporte *fabric*.

3 *KeyFlow:Conectividade Virtual Fabric para o Núcleo de Redes Definidas por Software*

3.1 Introdução

As recentes aplicações que rodam em nuvens privadas ou públicas são significativamente influenciadas pelo projeto da infraestrutura de rede. Muitas aplicações precisam trocar informações com nodos remotos para efetuar sua computação local. A necessidade de compartilhamento de recursos para redução de custos operacionais é algo importante, mas agora passa a ser fundamental a existência de meios de isolamento de fluxos para a coexistência de diferentes redes operacionais e experimentais sobre a mesma infraestrutura, com diferentes estruturas de conexão lógica interligando variados elementos de comutação ou roteamento independentes.

O grande volume de pacotes/fluxos nas redes baseadas em OpenFlow vai exigir um nível de eficiência no processamento dos elementos de comutação que não é disponível ainda nos métodos clássicos de consulta a tabela. É preciso que a rede possa prover comunicação com largura máxima de banda e atrasos mínimos sem comprometer a qualidade dos serviços transportados.

Um dos elementos críticos nesta infraestrutura é o tamanho das tabelas de encaminhamento nos comutadores. Tradicionalmente, a abordagem para escalar o projeto do tamanho das tabelas tem sido adicionar mais recursos de memória no silício do comutador ou permitir o uso de recursos de memória externa. Entretanto, com o aumento da densidade dos comutadores em redes de *data center* combinado com a necessidade de eficiência energética e custo, há uma demanda importante por novas formas de encaminhamento.

A tabela de encaminhamento ou *Forwarding Information Base* (FIB) implementada em um comutador é usada para roteamento, encaminhamento e funções similares para determinar a interface apropriada para a qual um comutador deve transmitir um pacote. Quando essas tabelas atingem suas capacidades máximas problemas de desempenho ocorrem. Um exemplo é

o aprendizado de endereços MAC: se o conjunto de endereços MAC ativos na rede (função do número de máquinas virtuais na rede) for maior que o tamanho da tabela de encaminhamento, então haverá *flooding* para descobrir os endereços que não estiverem nas tabelas, afetando o desempenho da rede.

O *OpenFlow* tem sido adotado para identificação e tratamento de fluxos em redes experimentais e acadêmicas. A arquitetura baseia-se em um controlador externo aos *switches Openflow* que centraliza a gerência de encaminhamento de pacotes através de uma visão global e manutenção de estados dos fluxos ativos na rede. Assim, o *OpenFlow* pode encontrar problemas de escalabilidade no futuro em função da necessidade de manutenção completa de estados (*fullstate*) dos fluxos ativos e da necessidade, a cada nodo da rede, de comunicação e interação com o controlador da rede. Soma-se a isto o fato do contínuo crescimento da capacidade de transmissão por enlaces pesar sobre o poder de processamento eletrônico dos pacotes para roteamento ou encaminhamento. Dessa forma, iniciativas para o desenvolvimento de técnicas sem manutenção de estados (*stateless*), atingindo um bom compromisso entre eficiência e complexidade, são necessárias. Simplificar os procedimentos de identificação de fluxos do núcleo da rede é um esforço necessário e contínuo para viabilizar o transporte de dados em altas taxas e com baixa latência.

3.2 Os Desafios para a Utilização de OpenFlow/SDN no Provimento de Serviços de Conectividade *Fabric*

Apesar do sucesso do *OpenFlow* na resolução de problemas das redes de datacenter (MYSORE et al., 2009), existe um longo caminho para percorrer para utilização do *OpenFlow* nas redes de núcleo. Recentemente, foi proposto que as redes de núcleo podem se beneficiar da evolução independente entre a comutação "*fabric*" nos elementos do núcleo e as funcionalidades disponíveis nos nodos de borda da rede (CASADO et al., 2012). Uma comutação *fabric* provê uma capacidade de encaminhamento bruta para conectar efetivamente todos os nodos de borda de maneira mais rápida possível e com a melhor relação custo-benefício, mantendo a inteligência mais próxima das extremidades.

Como visto na Seção 2.5, o controlador *OpenFlow* pode popular a tabela de fluxos dos comutadores em dois modos de operação, manutenção de fluxos pró-ativa e reativa. Na primeira, as regras são inseridas na tabela de fluxos para um conjunto de fluxos previstos que passarão pelo comutador. Na forma reativa, é necessária uma comunicação com o controlador para cada novo fluxo que chega ao elemento do plano de dados. Para melhorar a eficiência em redes

OpenFlow, outro requisito importante é minimizar as interações entre o controlador e os elementos do plano de dados (CURTIS et al., 2011), de forma a simplificar e reduzir a latência de encaminhamento.

Embora as limitações de operação do modo pró-ativo não sejam tão relevantes, existem outras limitações para seu uso no núcleo das redes. Por exemplo, a eficiência da memória utilizada para a tabela de fluxo para diferentes fabricantes é fortemente dependente da sua implementação de *hardware* (HUANG; YOCUM; SNOEREN, 2013). Além disso, diferentes formas de implementação dessas memórias para o tratamento dos pacotes no plano de dados podem produzir discrepâncias no desempenho do encaminhamento da rede (YU; WUNDSAM; RAJU, 2014). Em uma infraestrutura composta por elementos de encaminhamento de múltiplos fabricantes, o número total de fluxos ativos fim-a-fim em um determinado caminho é limitado pelo equipamento com maior limitação de recursos de *hardware*, como um equipamento com baixa disponibilidade de memória TCAM.

Para minimizar esta limitação relativa ao número de fluxos ativos, uma solução para o núcleo de redes poderia ser a agregação desses fluxos em uma classificação menos específica para o transporte agregado de dados. Contudo, o espaço para entradas em tabela não exatas, necessário para uma classificação agregada dos fluxos é limitado, principalmente em equipamentos genéricos disponíveis comumente no mercado e no aspecto do serviço de conectividade, o trafego deverá ser qualificado de uma maneira mais genérica, como modelos de qualidade de serviço baseados em serviços diferenciados (DiffServ). Com isso há uma limitação natural na diferenciação de serviços suportados pela infraestrutura de rede.

A principal limitação do *OpenFlow* para atingir as demandas das redes de núcleo estão no modo de operação reativo. Este modo é essencial para redes de núcleo com demandas dinâmicas de tráfego. Atender a requisitos de qualidade de serviço por serviços integrados (IntServ) de uma maneira satisfatória é um desafio justamente pela necessidade de alocação de recursos fim-a-fim na rede para fluxos de alta granularidade. Suportar uma maior granularidade potencializa a infraestrutura de rede a atender melhor os requisitos emergentes de virtualização de recursos, por exemplo. Para poder tratar fluxos granulares no núcleo da redes com melhor qualidade de serviço pelo modo de operação reativo, existem duas grandes preocupações sobre este modo de operação com *OpenFlow*: a taxa de configuração de fluxos no hardware e a latência na interação comutador-controlador.

Em relação à manutenção de fluxos em *hardware*, a forma com que a TCAM e as tabelas em software cuidam da inserção de regras é complexo. Isto pode envolver algorítmos que consideram até a persistência do fluxo, o conjunto de regras instalados, e o rearranjo entre

essas tabelas (HUANG; YOCUM; SNOEREN, 2013). Isto não apenas cria um gargalo para a máxima taxa de configuração de fluxos, mas o tempo de convergência do caminho fim-a-fim para cada fluxo é afetado pela alta variabilidade no tempo de instalação e atualização de regras em cada nodo deste circuito. Experimentos recentes mostraram que apenas algumas centenas de fluxos por segundo podem ser instalados (CURTIS et al., 2011; HUANG; YOCUM; SNOEREN, 2013). (ROTSOS et al., 2012) mostra que o atraso para inserção e modificação de fluxos pode variar de 10ms, para um único fluxo, chegando a 1 segundo para cerca de 1000 fluxos .

Assim como a interação entre o comutador e o controlador é algo preocupante, o tempo necessário para o fluxo estar totalmente funcional é severamente afetado pela latência existente no canal de controle. Isto é particularmente importante em redes de longas distâncias, como as de núcleo, onde o efeito do atraso de propagação intrínseco pode resultar em uma considerável demora para se alcançar a convergência fim-a-fim de um caminho. Por exemplo, evidências estão aparecendo em experimentos emulados como em (PHEMIUS; THALES, 2013), onde se mostra que uma latência no canal de controle de 300 ms resultará em 20 segundos para se alcançar a a vazão máxima em uma rede com 32 comutadores.

O paradigma atual para o projeto de elementos de núcleo em redes convencionais requer implementações de alto desempenho realizadas pela utilização de memórias rápidas, como as do tipo de acesso por conteúdo, que infelizmente é caro e consome muita energia. As implementações de alto desempenho baseadas em *OpenFlow* caíram na mesma armadilha: a necessidade dessas memórias para atender às necessidades *stateful* para tratamento de fluxos ativos. Isto pode levar a problemas relacionados à escalabilidade, à capacidade de resposta, ao custo e ao consumo de energia para aplicações de SDN no núcleo das redes.

3.3 KeyFlow: Conectividade Virtual *Fabric* Sem Manutenção de Estado e Com Roteamento na Origem para redes *OpenFlow/SDN*

Uma rede definida por software ideal deve possuir o *hardware* simples, independente de fabricantes, e ser a prova de inovações futuras e o *software* do plano de controle deve prover total flexibilidade (CASADO et al., 2012). Neste sentido, entende-se que os dispositivos de redes devem ser os mais simples possíveis para evitar discrepâncias de implementação de maneira a surgirem variabilidades no tratamento de pacotes. A independência de fabricantes e a resiliência a inovações estão relacionadas e redução de custos de OPEX e CAPEX no projeto

e operação dessas redes. Partindo da implementação centralizada do plano de controle com o *OpenFlow/SDN* é possível a implementação de redes com alta flexibilidade no tratamento de quadros, porém no que tange o núcleo da rede, há uma carência de uma solução de dispositivos que atendam aos requisitos de uma SDN ideal.

O núcleo de rede *fabric* diz respeito ao conjunto de elementos especializados no transporte dos pacotes, fazendo esta função com máxima eficiência possível. Os demais serviços da rede ficam a critério dos elementos de bordas, responsáveis pela implementação do isolamento e garantias de tráfegos necessárias e eventuais funcionalidades adicionais para a rede. O comutador *Openflow*, conforme visto na Sec 3.2, pode fazer o papel de borda, em vista de sua versatilidade no tratamento de pacotes, mas não pode atender aos requisitos *fabric* obrigatórios no transporte de pacotes no núcleo da rede.

Para cobrir este espaço, este trabalho propõe o comutador KeyFlow. Este comutador utiliza o protocolo OpenFlow para a sinalização e controle, porém altera a forma de definição do encaminhamento de pacotes para uma forma sem armazenamento de estado, com o objetivo simples de encaminhar pacotes por um determinado caminho lógico, independentemente da sinalização de controle das camadas de rede inferiores. O conceito de encaminhamento adotado pelo *KeyFlow* consiste na utilização do Esquema de Informação por Chave (KIS), proposto em (WESSING et al., 2002) para redes ópticas, na criação de topologias *overlay* em redes *OpenFlow*. Os rótulos globais estão relacionados com as chaves locais. O resultado da operação de resto da divisão do rótulo global presente no pacote com a chave local instalada no comutador determina exatamente a porta de saída em cada elemento do caminho, sem a necessidade de interações com o controlador da rede e sem a necessidade de consulta a tabelas. Deste modo, o processamento nos nodos de núcleo é realizado de maneira eficiente, tendendo a ser uma solução escalável com o crescimento da rede e de baixo custo operacional. A limitação no tratamento fica independente do armazenamento de estado nos elementos de comutação, dependendo apenas da sua capacidade de tratamento e encaminhamento de pacotes.

Para geração dos rótulos globais, utiliza-se um sistema numérico de restos que permite que um número inteiro grande possa ser representado por um conjunto de números inteiros menores, sendo o primeiro o rótulo do caminho e os menores as portas de saída em cada nodo deste circuito. Para cada caminho (ou circuito virtual), a computação é efetuada a partir de dois vetores. O primeiro vetor é formado pelas chaves locais de todos os comutadores que compõem o caminho de interesse. O segundo vetor é formado pelo número da porta de saída de cada um desses comutadores. Como condição necessária para a criação de chaves válidas, restringe-se que todas as chaves definidas, em cada caminho, sejam primas entre si, conforme descrito a

seguir na seção 3.4.1.

O plano de controle da rede fica responsável pela distribuição e instalação coerente das chaves nos comutadores. Após esta instalação, e com o controlador conhecendo a topologia de interligação dos comutadores, é possível eleger um conjunto de caminhos de interesse e definir todos os rótulos globais para cada um desses caminhos. De posse dos rótulos de cada circuito, estes podem ser instalados nos elementos de borda de forma pró-ativa ou reativa. A borda, pela inserção do rótulo global em cada pacote de um determinado fluxo define o caminho que este fluxo percorrerá até a extremidade de saída da rede.

O estabelecimento dos caminhos prévios da rede viabiliza uma importante vantagem do KeyFlow em relação ao MPLS. O estabelecimento de caminhos nessas redes é feito por protocolos de sinalização como o RSVP e o LDP. Em ambos os casos, é necessária a manutenção de estados em cada nodo, reserva de recursos, etc. Mesmo considerando que, em alguns casos, esta manutenção de estados tem uma ocupação de recurso insignificante para circuitos lógicos estáveis, é importante ressaltar que no estado transitório de estabelecimento do caminho, esta sinalização pode impactar significativamente na flexibilidade do atendimento à demandas dinâmicas e estabelecimento de caminhos em tempo real para fluxos transitórios na rede. Como no KeyFlow é a borda que define o caminho que cada pacote seguirá, os fluxos poderão seguir diferentes caminhos, com ou sem redundância, independentemente do núcleo tomar conhecimento desta necessidade. Sendo assim, a possibilidade de melhor ocupação dos enlaces disponíveis e a qualidade de serviço a ser garantida para as aplicações tenderá a ser mais flexível e mais simples, com isolamento de fluxos utilizando rótulos simples, como uma tag VLAN, que diferenciem os pacotes na saída do núcleo *fabric*. Com isso, o suporte a virtualização de links sobrepostos é realizado de forma mais simples e com grande flexibilidade.

3.4 O Plano de Controle KeyFlow

O controlador *fabric* é o responsável pelo levantamento da topologia, pela definição e instalação das chaves e pela computação dos rótulos para o conjunto de melhores caminhos definidos.

O protocolo OpenFlow pode ser usado para as interações de fronteira-sul entre os comutadores Keyflow e o controlador. Até a instalação da chave, o KeyFlow opera basicamente encaminhando os pacotes recebidos para o controlador e divulgando pacotes de descoberta na rede, derivando estas funções de um comutador OpenFlow convencional.

No controlador, a partir do levantamento da topologia da rede *fabric*, inicia-se a instalação

das chaves nos comutadores de núcleo. Estas chaves têm por requisito serem todas primas entre si, e terem como valor inicial um número inteiro maior que a quantidade máxima de portas de cada comutador. Durante o processo de inicialização da rede, cada comutador envia para o controle todos seus recursos disponíveis, conforme definição do Protocolo OpenFlow.

Após a instalação das chaves no núcleo, inicia-se o processo de definição dos melhores caminhos. Todos estes eventos até aqui ocorrem na inicialização da rede. Este processo pode ocorrer seguindo as políticas dos administradores da rede. Por exemplo, pode-se aplicar o algoritmo de Dijksta, e escolher os menores caminhos entre cada um dos comutadores de borda da rede. Uma vez definidos e identificados os caminhos, estes serão indexados pelo rótulo global que o identificará. A computação deste rótulo é realizada por meio de um sistema numério residual, por meio do Teorema Chinês dos Restos, conforme explicado a seguir.

3.4.1 Teorema Chinês do Resto (TCR)

Dado um caminho definido pelo conjunto de nodos, representado pelo conjunto de chaves $\bar{k}$ e o conjunto de portas de saída representado pela vetor $\bar{s}$ e considerando que o caminho possui n nodos, tem-se:

$$\bar{k} : (k_1, k_2, \ldots, k_n) \qquad \bar{s} : (s_1, s_2, \ldots, s_n) \tag{3.1}$$

O rótulo global r é um número inteiro tal que, pela sua decomposição em relação ao vetor de chaves $\bar{k}$ é possível restaurar o vetor de portas $\bar{s}$. Para se obter este rótulo, parte-se de um escalar k, obtido pela multiplicação de todas as chaves do caminho escolhido.

$$k = k_1 \cdot k_2 \cdot k_3 \cdots k_n \tag{3.2}$$

Utilizando-se o escalar k e o vetor $\bar{k}$, cria-se o vetor auxiliar $\bar{m}$, de modo que para cada elemento deste vetor, a condição $m_i \; mod \; k_j = 0$ seja atendida para todos $j \neq i$, o que é satisfeito pela seguinte equação:

$$m_i = \frac{k}{k_i}, \quad para \quad i \leqslant n \tag{3.3}$$

Cria-se, então, outro vetor auxiliar $\bar{c}$, baseado nos vetores $\bar{m}$ e $\bar{k}$, de maneira que seus elementos sejam da forma:

$$c_i = m_i(m_i^{-1} mod k_i), \quad para \quad i \leqslant n \tag{3.4}$$

O termo m_i^{-1} representa o inverso multiplicativo de m_i definido por $m_i m_i^{-1} \equiv 1 (\text{mod } k_i)$, e que pode ser obtido de maneira computacional por meio do algoritmo de Euclides Estendido. Sendo assim, tanto m_i e k_i devem ser primos entre si. Como m_i é múltiplo de todos os n elementos de

$\bar{k}$, exceto k_i, verifica-se que todas as chaves, de cada comutador de um dado caminho, devem ser primas entre si. O escalar r é finalmente calculado utilizando-se os vetores $\bar{c}$ e $\bar{s}$, assim:

$$r = (s_1c_1 + s_2c_2 + s_3c_3 + \cdots + s_kc_k) \quad mod \quad k \tag{3.5}$$

Portanto, pelo TCR é possível gerar um escalar r a partir de dois vetores $\bar{k}$ e $\bar{s}$, onde todos os elementos do primeiro devem ser primos par-a-par. A partir do rótulo r, aplicando a operação de ”resto da divisão” com um elemento do vetor de chaves $\bar{k}$, obtem-se o respectivo elemento do vetor $\bar{s}$.

No apêndice A.1 há um exemplo de código em linguagem Python que realiza a computação do rótulo. A partir desta computação, é possível inserir a ”ação OpenFlow” que marque os pacotes dos fluxos. Uma restrição deste método de computação de rótulos baseados no TCR está no fato do rótulo crescer muito rapidamente com o crescimento do número de nodos considerados no caminho. Por isso, faz-se necessário uma análise para definição da melhor forma de utilização desse rótulo. Dependendo do tamanho máximo para uma determinada rede, este rótulo pode ser inserido num campo legado, como o endereço MAC, ou pode ser necessário a criação de uma nova estrutura de cabeçalho para encapsular os pacotes passantes. Neste caso, é necessário instrumentar o comutador OpenFlow de borda a tratar a modificação de pacotes para este cabeçalho. No Cap. 4 é feita uma análise de pior caso para verificar qual é opção mais vantajosa em termos práticos. Um aspecto crítico da utilização do Teorema Chinês do Resto para definição de rótulos está ligado ao rápido crescimento do quantidade de bits necessários para sua representação, devido à multiplicação de números primos. É esperado um crescimento rápido do tamanho do rótulo em função no número de saltos existente no caminho fim-a-fim.

Faz-se necessário uma análise prática para avaliar a quantidades de bits que pode ser utilizada no estabelecimento de caminhos nas redes atuais. Mitigando as limitações do crescimento do rótulo em função do número de saltos do caminho, pode-se obter uma maneira eficiente de definição de rotas dentro de uma infraestrutura de nodos. Esta análise é apresentada no Cap. 4.

3.5 O Plano de Dados KeyFlow

O plano de dados da rede KeyFlow tem a única preocupação de transportar pacotes entre os nodos de borda da rede. Após a fase de inicialização realizada pelo plano de controle, o conjunto de melhores caminhos entre elementos da borda da rede fica disponível no controlador da rede. Este conjunto pode ser representado por uma tabela de rótulos para cada relação origem-destino entre elementos de borda. Estes rótulos são instalados na borda de maneira pró-ativa, reativa ou

de forma híbrida, com os caminhos mais comuns instalados no comutador de borda e os menos comuns obtidos via protocolo de interação comutador-controlador.

Os elementos de núcleo definem a porta de saída de cada pacote pela aplicação direta da função matemática ”resto da divisão” entre o rótulo existente no pacote/quadro e a chave local instalada no comutador. O resultado desta operação determina a porta de saída do pacote. A Fig. 3.1 ilustra o mecanismo de encaminhamento da rede fabric. A partir de dois domínios de rede conectados à núvem fabric, os pacotes chegam ao comutador *OpenFlow* na borda. Na abordagem reativa ilustrada, um evento *OpenFlow* ”packet in” é gerador para o controlador, que responde com a regra de alteração do pacote com inserção do rótulo nas bordas relacionadas ao caminho do fluxo. Este rótulo é armazenado nas tabelas de fluxo. Na origem para modificar o pacote e no destino para regenera-lo na entrega às redes clientes conectadas.

Na Fig. 3.1, o caminho definido para a rede do Domínio A passa pelos nodos #4, #7 e #5. Para o domínio B o caminho definido é o #4, #3 e #5. O algoritmo para comutação da chave com a função geratriz dos rótulos encontra-se no apêndice A.1. A política de escolha dos caminhos para cada domínio fica inteiramente a critério do controlador da rede, no exemplo da figura, apesar da existência do caminho direto entre os nodos #4 e #5, os fluxos seguem caminhos totalmente separados e independentes.

Para o caso de falhas nos caminhos escolhidos, uma vez detectados pelo plano de controle, a recuperação se passa pela sinalização da falha junto as bordas. Isso pode ser feito pela alteração da regra de um caminho para outro, ou pela utilização de um rótulo sobressalente que definirá em tempo real o caminho alternativo para o fluxo, ou seja, a informação de caminho alternativo já estará disponível no elemento de comutação e poderá se aplicado diretamente pelo comutador quando da identificação de evento de queda de enlace em uma de suas portas.

Com a utilização da rede *fabric*, se reduz significativamente o volume de estados instalados nos elementos de encaminhamento. Com isso, espera-se uma redução na latência de processamento duranteo o encaminhamento dos pacotes. Além disto, em termos práticos a organização da rede para provimento de serviço de conectividade pode estar organizada como na Figura 3.2.

O projeto físico do esquema exposto na Fig. 3.1, pode ser organizado na prática em três camadas, sobrepostas obviamente à infraestrutura de transmissão de bits, ilustradas pela conexão direta entre os comutadores Keyflow na Fig 3.2. A camada de operacionalização da rede de núcleo *fabric* é interconectada pelo arranjo de enlaces físicos disponíveis. Sobre esses equipamentos, instala-se a camada de rede de borda, que é responsável pela interface de acesso das redes clientes com a infraestrutura de transporte de pacotes da rede KeyFlow. Nesta camada é concentrada as definições de estado e o controle de acesso aos recursos *fabrics*. Este controle

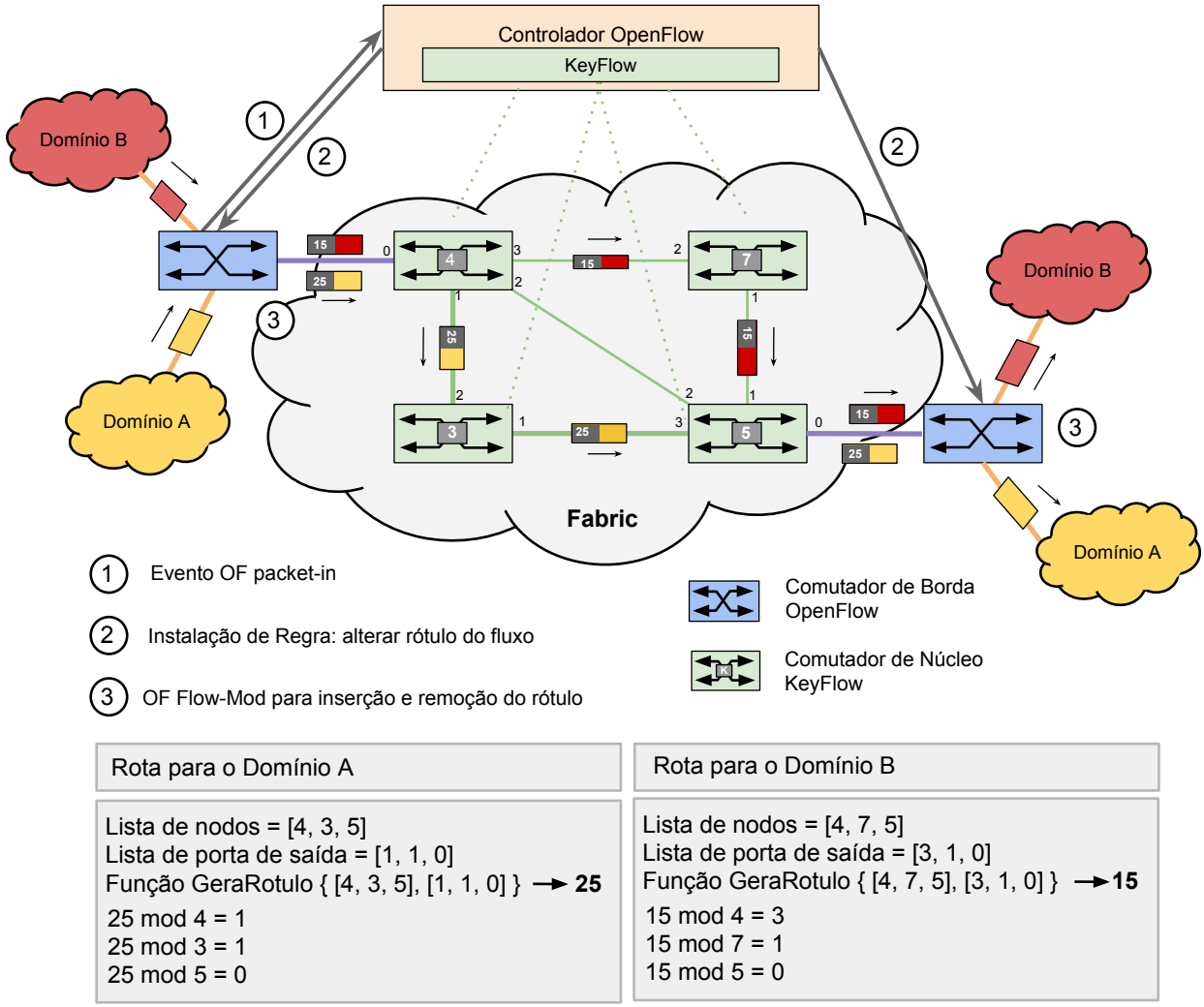


Figura 3.1: Arquitetura para uma rede definida por sofware integrada à solução KeyFlow.

pode ser realizado pela utilização de portas lógicas para cada fluxo, que viabilizariam o controle da taxa de vazão por fluxo. Essas portas lógicas são vinculadas a porta física diretamente conectada ao comutador KeyFlow. Essa porta física é a porta de saída da rede fabric. A tabela de estados, baseada na tabela de fluxo do comutador OpenFlow, faz o controle da definição de caminhos de saída de cada fluxo. No exemplo da Fig. 3.2, o fluxo parte o elemento de conexão A para o C sem a necessidade de manutenção de estados no elemento B. Com isto, espera-se um custo de manutenção de caminhos mais baixo, uma vez que os elementos provedores de recurso na rede (a borda) têm seus recursos alocados exclusivamente para os fluxos originados ou destinados a eles.

Na visão do usuário conectado à rede de borda, considerando a utilização de rótulo diretamente na camada de enlace da conexão, há o estabelecimento de uma adjacência com seus nodos remotos conectados em outra extremidade da rede provedora de conectividade. Caso o

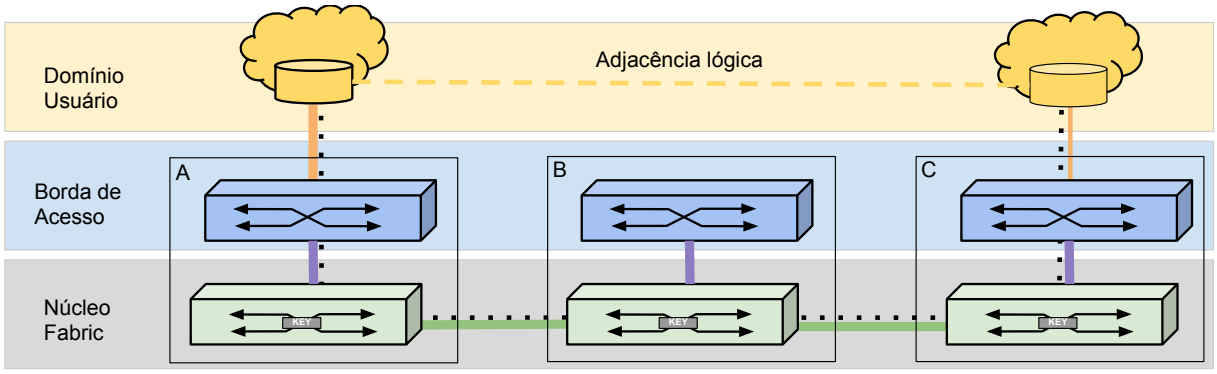


Figura 3.2: Estrutura de ligação e provimento de serviço de conectividade com o KeyFlow

controlador da rede de serviço exponha mecanismos de escolha de caminhos, é possível que o usuário escolha os nodos que deseja conectar diretamente, provendo assim uma forma de programabilidade da estrutura de conexão, sem a necessidade de sobrecarga de sinalização adicional na infraestrutura de rede para tal finalidade.

Com a redução da latência de processamento, espera-se que a rede *fabric* tenha um comportamento híbrido no processamento dos pacotes com características próximas das redes de circuitos, no que se refere à reduzida variação da latência devido a baixa variabilidade derivada pela não manutenção de estado para definições de comutação.

3.6 KeyFlow e as Tecnologias de Provimento de Conectividade Virtual

A ausência de estado para marcação de caminhos possui uma grande vantagem em relação às tecnologias legadas, como, por exemplo, as tecnologias MetroEthernet e MPLS/IP.

Após a configuração inicial, a rede KeyFlow não realiza difusão (*broadcast*) de pacotes entre seus nodos durante o encaminhamento dos fluxos. Não existe, assim, a ocorrência de laços(*loops*). Como isto, todos os enlaces de um elemento podem ser utilizados de maneira balanceada, conforme decisão estabelecida na borda da rede e orientada pelo plano de controle.

O cabeçalho dos quadros Ethernet pode ser uma boa opção para o KeyFlow instalar o rótulo de roteamento dos fluxos. Isto porque o Ethernet é uma tecnologia bastante comum nos equipamentos ditos de ”prateleira”. O OpenFlow, desde as versões iniciais já permite a manipulação de campos dos quadros Ethernet para tratamento de fluxos. Como a decisão de encaminhamento no núcleo KeyFlow não necessita dos dados dos endereços das portas que interconectam um determinado segmento, bastaria que o MAC original do quadro fosse alterado e reconstruído

pelas bordas da rede. Para isto, é necessário realizar uma avaliação do impacto do crescimento do rótulo em relação ao tamanho da rede, conforme apresentado na Sec. 4.3.1, e verificar se o volume disponível de bits no cabeçalho MAC é suficiente para abrigar os rótulos KeyFlow.

Com relação às tecnologias de sobreposição, como o MPLS/IP, há uma significativa vantagem do KeyFlow pela simplificação da estrutura de definição e manutenção dos caminhos. Não é necessário nenhum protocolo de reserva e alocação de recursos, tudo isso é feito logicamente no plano de controle da rede. O controlador define a alocação de recurso por meio do Protocolo OpenFlow nos elementos de borda à rede *fabric* KeyFlow. A utilização do KeyFlow/OpenFlow possibilita a criação de caminhos virtuais lógicos para interconexão de nodos da borda que podem diferenciar explicitamente o fluxo, provendo uma abstração realística de adjacência entre dois pontos conectadas à infraestrutura KeyFlow/OpenFlow.

De maneira semelhante a um roteador *DiffServ* o comutador de borda OpenFlow apenas ”classifica” os pacotes para seus melhores caminhos, contudo com o potencial de diferenciação granular dos fluxos com similaridade a um roteador *Intserv*, sendo novamente a borda responsável por alocar os recursos necessários para o fluxo dentro do caminho do núcleo com melhor capacidade não-bloqueante. Isto pode ser feito limitando-se a taxa de envio de pacotes de um determinado fluxo, ou pela escolha de caminhos mais adequados para um determinado fluxo, com ajuda do plano de controle.

A facilidade de se definir um caminho, pela necessidade de manutenção de estados apenas na borda da rede, faz com que o arranjo OpenFlow/KeyFlow proveja um tratamento dinâmico de escolha de caminho pela diferenciação de fluxos específicos, permitindo que um conjunto de pacotes possa ser roteado por diferentes caminhos no interior do núcleo *fabric* KeyFlow. No caso das redes MPLS/IP, a manutenção de estados distribuída impõe uma severa complexidade para reserva e alocação de caminhos fim-a-fim, inexistente com o KeyFlow. Assim, equipamentos mais baratos e especializados em transportar pacotes entre os elementos de borda podem ser utilizados para provimento de conectividade virtual, sem a necessidade da utilização de protocolos sinalizadores ou de estruturas de encaminhamento associadas a alteração de rótulos no interior da rede. Com isto, espera-se prover conectividade por meio de equipamentos mais eficientes, baratos e com menor consumo de energia.

3.7 Conclusão

A necessidade de infraestruturas de rede mais flexíveis e econômicas no provimento de conectividade lógica se materializa na forte demanda por arquiteturas abertas para redefinição

de roteadores e encaminhadores de pacotes na Internet. Esta demanda resultou em uma nova organização para a evolução de redes abertas em torno de conceitos baseados em Redes Definidas por Software, mais precisamente, sobre o ponto de vista da utilização de padrões abertos que viabilizassem um melhor controle do comportamento da rede pelos seus administradores, e não exclusivamente pela imposição da industria deste mercado. A rápida adoção do OpenFlow pelos diversos fabricantes é resultado disto. Porém, a forma da implementação dessa especificação e sua própria estrutura fortemente baseada em manutenção de estados no encaminhamento, cria um sério entrave para sua adoção próxima ao núcleo das redes especializadas no transporte de pacotes.

O KeyFlow é uma solução para provimento de conexão lógica que viabiliza a aplicação dos conceitos de redes definidas por software em redes de núcleo, pela possibilidade de criação de enlaces lógicos mantidos sem a manutenção de tabelas de estados. A lógica de encaminhamento é baseada em uma propriedade matemática advinda do Teorema Chinês dos Restos, que permite que um rótulo defina globalmente seu caminho dentro de um conjunto de nodos. Estes nodos controlam a comutação de pacotes apenas pela operação matemática do ”resto da divisão” entre o valor presente no rótulo do pacote e uma chave interna previamente instalada pelo plano de controle.

Com isso, o KeyFlow possibilita uma rede de núcleo SDN *fabric* com grande simplicidade dos elementos de encaminhamento especializados no transporte de pacotes com potencial redução da latência e do *jitter* de comutação, e pela diminuição no número de interações necessárias entre o plano de dados e o de controle. Isto proporcionará uma infraestrutura de serviço de conectividade potencialmente mais simples, econômica e com menor consumo de energia.

4 *Prova de Conceito e Análise de Resultados*

4.1 Introdução

O controle da comutação realizado diretamente por uma operação matemática deve produzir uma redução da latência de processamento dos pacotes. Para confirmar essa redução foi montado um ambiente experimental, baseado em virtualização de comutadores, para reproduzir diferentes caminhos, ou circuitos, e avaliar o impacto da redução de carga de processamento em função da busca em memória de regras de controle de comutação. Sendo assim, a primeira pergunta para validação da proposta é: a remoção de busca em tabela por meio de funções hash reduz a latência de processamento em relação à proposta de operação matemática do KeyFlow?

Uma fragilidade da proposta está no rápido crescimento do número inteiro utilizado como rótulo do circuito estabelecido. Para avaliar esta fragilidade de forma pragmática, foi realizado uma análise de pior caso para a utilização deste rótulo em redes reais. Qual seria uma tamanho razoável para utilização da proposta em relação ao tamanho da rede existente no caminho fim-a-fim? No restante deste capítulo é apresentado o protótipo implementado para esta avaliação, o cenário montado e os resultados obtidos.

4.2 Implementação do Protótipo de Comutador KeyFlow

O protótipo [1] foi construído baseado na implementação de referência do comutador OpenFlow de Stanford na versão 1.0 do protocolo [2]. A diferença entre o plano de dados da implementação do protótipo e do OpenFlow está ilustrada na Fig. 4.1.

No caso do comutador OpenFlow 1.0, os campos de cabeçalho dos pacotes são analisados assim que eles chegam para verificar se há alguma regra que case e identifique um fluxo pre-

[1] http://gtif-ufes.github.io/keyflow

[2] http://archive.openflow.org/downloads/openflow-1.0.0.tar.gz

visto na rede. Isto é realizado pela operação de busca em memória em duas tabelas existentes. Primeiramente, uma busca direta é relizada em uma tabela hash para identificação de operações em regras exatas. O comutador OpenFlow prioriza as regras específicas pela busca inicial na tabela hash (tabela 0, máx. 131070 entradas)[3]. Se uma regra não é encontrada, conforme Fig. 4.1, uma busca é realizada na tabela linear (tabela 1, máx. 100 entradas) para entradas não-exatas, ou seja, com *wildcards*. Se uma regra for encontrada, as ações definidas na regra é aplicada. Caso contrário o pacote é encapsulado e enviado ao controlador da rede. As alterações na biblioteca do OpenFlow para suporte ao KeyFlow estão apresentadas no apêndice A.2 e as modificações realizadas no plano de dados do comutador estão disponíveis no apêndice A.3.

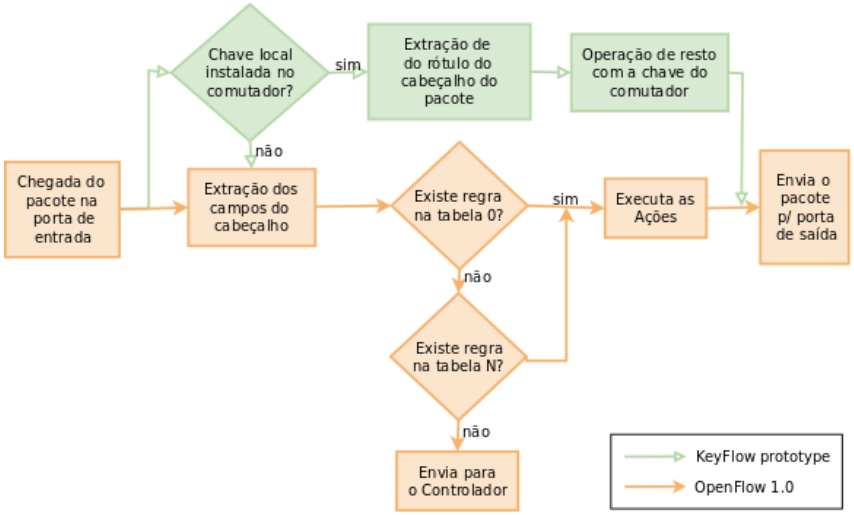


Figura 4.1: Fluxograma do tratamento de pacotes da especificação *OpenFlow 1.0* (SPEC, 2009) e do protótipo de comutador KeyFlow.

Por essa lógica de processamento, é importante realçar a existência natural de uma latência aleatória relativa à busca de regras nas tabelas de fluxos, dependendo: 1) do tipo de entrada; 2) da ocupação da tabela de fluxos analisada; 3) da posição que o fluxo ocupa nesta tabela. Quanto maior o número de entradas na tabela, maior será a variação de tempo necessária para encontrar uma regra e para executar as ações associadas.

Recentes versões do OpenFlow adicionaram suporte a formatos de *match* extensíveis (OXM), comunicação entre múltiplas tabelas por meio de metadados e extensibilidade melhorada para OXM na versão 1.4 da especificação, conforme apresentado na Seção 2.5.4. Toda essa evolução tende a apresentar maiores variações no tempo de encaminhamento para as novas versões, agravado ainda mais pelas diferentes formas de implementação realizadas pelos fabricantes dos comutadores, conforme discutido na Seção 2.6.1. A principal razão está no crescimento, durante a evolução do protocolo, da complexidade para o processamento dos pacotes no tratamento de flu-

[3]http://archive.openflow.org/downloads/openflow-1.0.0.tar.gz

xos ativos em múltiplas tabelas, que também necessitam de suporte para a troca de metadados, ações para o encaminhamento de pacotes, tabelas de grupo e modificações diversas de pacotes. Assim, para uma comparação mais confiável, simplificada e clara, assume-se a implementação de referência da versão 1.0 do OpenFlow na implementação do protótipo proposto.

Para o plano de controle, foi implementada uma modificação na aplicação *dpctl*, que é um utilitário de gerência que permite comandos do plano de controle básicos suficientes para o protótipo KeyFlow. O protótipo foi testado no ambiente do Mininet [4] que permite a criação de redes realísticas virtuais, com execução da implementação real do protótipo. Neste ambiente, o *dpctl* permite uma conexão via interface de *loopback* de cada comutador virtual via sessões TCP em diferentes portas. Para tratar da chave no comutador KeyFlow, foram adicionados dois novos comandos no plano de controle, implementado no comutador e no *dpctl*: key-mod e dump-key. Para isto, também foi implementada duas novas mensagens para o protocolo: *OFPT_KEY_MOD* e *OFPT_KEY*, que são usadas para manipular as chaves dos comutadores. As alterações no utilitário *dpctl* pode ser consultada no apêndice A.4.

O protótipo de comutador KeyFlow é um comutador OpenFlow instrumentado para realizar encaminhamento de pacotes baseado em operações de resto da divisão ao invés de busca de correspondência em tabela. O comutador habilita sua função KeyFlow quando da instalação da chave local. Se esta não for instalada, valor padrão zero, o tratamento de fluxos se dá conforme especificação OpenFlow 1.0. O rótulo utilizado no protótipo inicial do KeyFLow para testes foi o campo de VLAN dos quadros Ethernet. Dessa forma, pode-se isolar e identificar o impacto da substituição do mecanismo de encaminhamento convencional. Além disso, a comparação entre a comutação OpenFlow e KeyFlow foi realizada sobre condições equivalentes.

4.3 Metodologia de Validação

A metodologia de validação é estruturada em duas partes. A primeira parte consiste de uma avaliação analítica da escalabilidade da proposta KeyFlow. Este estudo investiga duas importantes questões para o desenvolvimento do KeyFlow em redes de núcleo: 1) A avaliação de quantos bits seriam necessários para o rótulo a ser inserido no cabeçalho dos pacotes em relação ao tamanho do caminho definido; 2) A redução máxima no volume de dados que pode ser alcançada pela remoção da manutenção de estados do núcleo de rede OpenFlow.

A segunda parte é dedicada a resultados experimentais do protótipo implementado em exe-

[4] http://mininet.org/

cução na plataforma de experimentação Mininet [5]. Os resultados experimentais são focados na latência de comutação com um indicador do gargalo da consulta em tabelas no plano de dados em redes de núcleo definidas por software, variando o número de saltos e o volume de ocupação das tabelas de fluxo.

4.3.1 Escalabilidade do KeyFlow

O Tamanho Ideal do Rótulo para Identificação de Rotas

Inicia-se a avaliação da proposta pela análise do tamanho do rótulo necessário na composição do cabeçalho dos pacotes a serem transportados na rede fabric. Como visto anteriormente, o tamanho deste rótulo por meio do Teorema Chinês do Resto depende: 1) Do comutador com menor número de portas disponíveis; 2) Do tamanho da rede de núcleo; 3) Do número de saltos para um determinado caminho da topologia da rede;

O número mínimo de 24 portas é definido para os comutadores. Baseado em topologias de redes reais como NSFNet, GÉANT e RNP, [6] , uma variação de $15 < n < 60$ nodos é considerada.

Para análise do tamanho do rótulo com relação às redes consideradas, foi utilizado o cenário de pior caso, que ocorre quando o caminho definido possui os nodos com as chaves locais de maiores valores de toda a rede, pois o rótulo, conforme visto em 3.4.1, é um número inteiro cujo maior valor é representado pelo produtório de todas as chaves do caminho.

Por exemplo, uma rede com 15 nodos, as chaves distribuídas serão [24, 25, 29, 31, 37, 41, 43, 47, 49, 53, 59, 61, 67, 71, 73]. Destas, o pior caso para o tamanho do rótulo será definido em um caminho de três saltos pelo produtório das três maiores chaves, no caso $67x71x73 = 347.261$. Para calcular a quantidade de bits, aplicamos a função $log_2(347.261) \approx 18,40$. Assim, o pior caso para uma rede de 15 nodos de 24 portas e com caminhos de três saltos será 19 bits, conforme Fig. 4.2. Estes caminhos normalmente não são realizáveis, principalmente se houver alguma inteligência na distribuição das chaves (por exemplo, os nodos mais centrais possuirem os menores valores de chaves locais), porém esta análise é útil na identificação do limite superior do tamanho da rótulo.

O gráfico da Fig. 4.2 apresenta o tamanho do cabeçalho do pacote para representar um rótulo em função do crescimento do número de saltos no caminho. Esta análise é feita para diferentes tamanhos de redes. Por exemplo, mesmo com uma distribuição de chaves desfavorável em uma rede de núcleo de 60 nodos, uma rota com 11 saltos pode ser implementada utilizando

[5] http://mininet.org
[6] http://www.topology-zoo.org

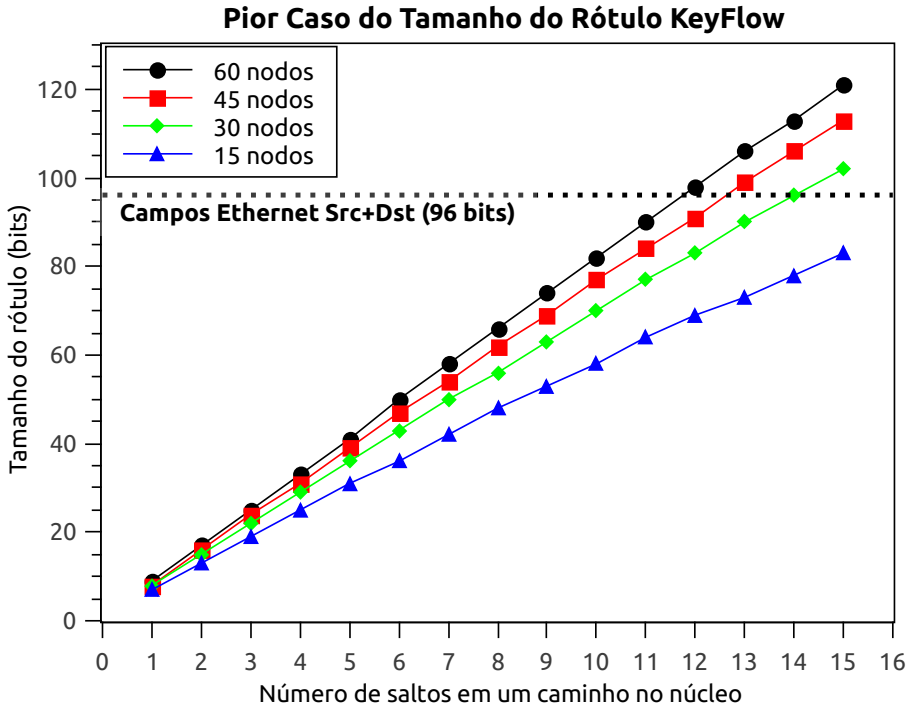


Figura 4.2: Escalabilidade do tamanho do pior(maior) rótulo para um caminho KeyFlow (MARTINELLO et al., 2014)

96 bits. Este volume de bits pode ser realizado pela utilização dos campos de MAC de origem e destino do cabeçalho Ethernet. A utilização desse campo permite que comutadores convencionais OpenFlow sejam utilizados na borda, uma vez que operações de manipulação de MAC são suportados pela especificação de longa data. Nada impede que, neste caso, também sejam utilizados equipamentos legados na borda, desde que haja alguma forma de manipulação dos MACs na entrada e saída da rede *fabric*, que pode ser realizado por ACLs específicas. Contudo, a utilização de comutadores OpenFlow proporciona uma vantagem diferencial: o tratamento de fluxos. Isso possibilita um grande vantagem para estabelecimento de conexões sob demanda para determinados fluxos. Com isto, ações de modificação de campos podem ser utilizadas para inserir um MAC na entrada e restabelecer os originais na saída, provendo para a rede cliente uma rede virtual pelo provimento de adjacência lógica entre dois nodos clientes conectados a rede de serviço OpenFlow/KeyFlow.

O Volume de Dados para Manutenção de Estados na Rede

O estado da rede está associado ao conjunto dados utilizados para representar os fluxos ativos em um determinado momento. Uma rede com manutenção de estados, como o OpenFlow, necessita de certo nível de interações entre o plano de dados e o plano de controle. Este número

de interações é proporcional à ocupação das tabelas de fluxo em cada comutador e ao número de comutadores existentes em um caminho.

Quanto maior o número de fluxos instalados por comutador, maior a carga de tráfego no canal de controle para o gerenciamento (i.e., coleta de estatísticas, troca de mensagens, etc). Nota-se que as regras precisam ser instaladas na tabela de fluxos de cada comutador no caminho definido. Com isso, é esperado um grande volume de dados para manter o estado da rede de núcleo. Estes dados ocupam e retardam a capacidade de transporte, o que não é desejável para redes *fabrics*. A convergência da consistência ao longo de todo o caminho também fica altamente dependente da responsividade de cada elemento.

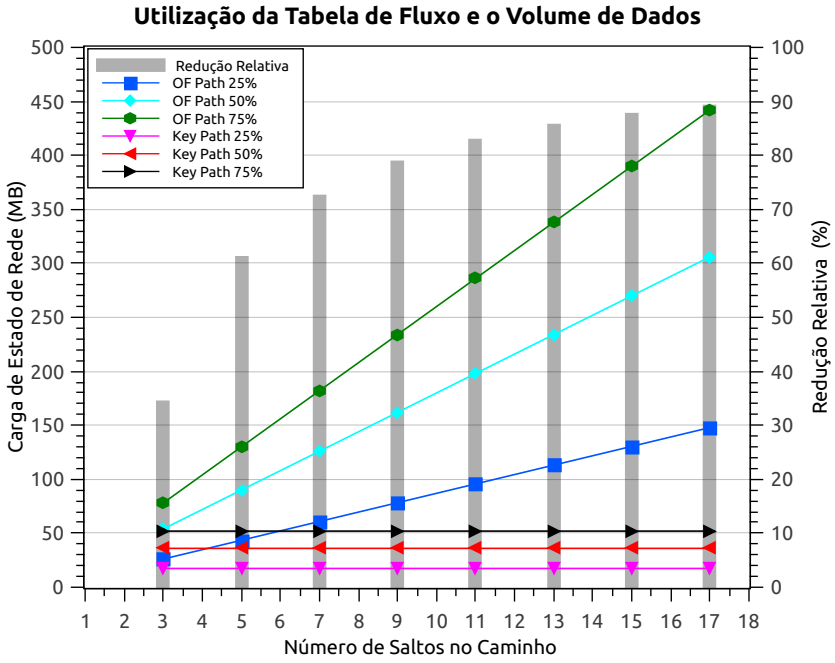


Figura 4.3: Escalabilidade da carga de estados na rede (MARTINELLO et al., 2014)

Para fins de avaliação, supõe-se que cada regra (estado da rede) é uma entrada de fluxo para correspondência exata, com aproximadamente 270 bytes de acordo com a especificação OpenFlow (SPEC, 2009). O comutador de núcleo KeyFlow representa sua chave local por meio de uma inteiro de 4 bytes no referido protótipo. Na implementação de referência do comutador OpenFlow, a maior parte dos fluxos é composto por regras exatas, com máximo de 131.070 entradas, e apenas 100 registros disponíveis para fluxo agregados por entradas com *wildcards*. O gráfico da Fig. 4.3 apresentada os requisitos de volume de dados a serem mantidos numa comparação entre núcleos de rede com OpenFlow e com KeyFlow. Foi utilizado como parâmetros o número de saltos no caminho (incluindo os nodos de borda), variando de 3 até 17, e a ocupação das tabelas existentes no caminho, de 25% a 75%.

Os resultados, apresentados na Fig. 4.3, indicam que em são necessários um montante em torno de 80 Megabytes de dados de controle para manter os caminhos curtos com três saltos, tanto para a abordagem com OpenFlow como para com o KeyFlow. Entretanto, a carga de estado, ou o volume de dados necessários para manter o controle da rede, cresce abruptamente quando as tabelas de fluxos mais carregadas estão presentes nos maiores caminhos de núcleo. Como exemplo, a carga excede 400 Megabytes considerando as tabelas carregadas em 75% da sua capacidade em caminhos com 17 saltos. Por outro lado, o núcleo da rede sem manutenção de estado do KeyFlow necessita de elementos com tabelas apenas nas bordas do caminho e permanecem praticamente inalterado pelo incremento tanto no tamanho do caminho quanto na utilização das tabelas de fluxo. Para sintetizar os resultados, as barras na Fig. 4.3 apresentam a redução potencial, relativa do KeyFlow, na carga de estado da rede. A solução proposta alcança pelo menos 33 % de redução em um circuito de três saltos (i.e., ingresso na borda, um elemento de núcleo e egresso na outra borda). Como esperado, para circuitos com muitos saltos, esta redução tende a ser substancialmente favorável ao KeyFlow, chegando a mais de 90% de redução para caminhos com 17 saltos. Estes resultados mostram que a rede KeyFlow é mais flexível e ágil em se adaptar a padrões de tráfego e a mudanças de topologia. Isto se deve à redução no número de elementos *stateful* e, também, na quantidade e duração das interações entre o plano de dados e o de controle.

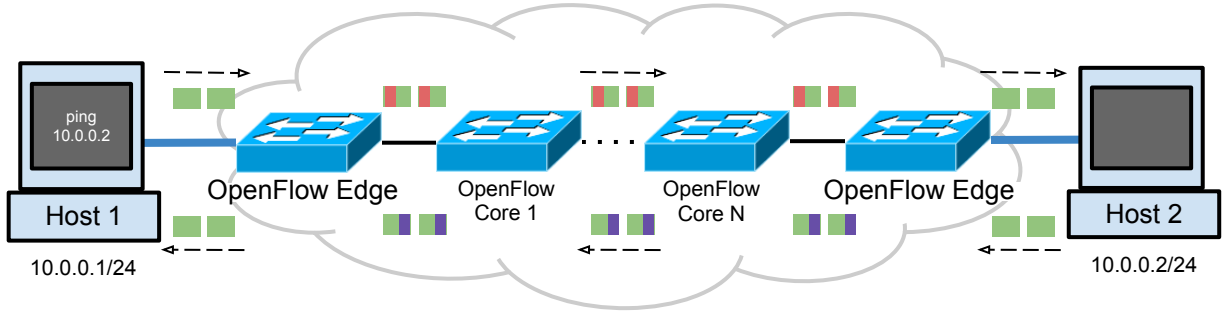
4.3.2 Cenário Experimental: Análise da Latência

Para avaliar a eficiência da comutação baseada em chaves locais, desenvolveu-se um protótipo de comutador em *software*. Para medir a eficiência da comutação, utilizou-se o ambiente de prototipação do *Mininet* [7]. Neste ambiente, foi criada uma topologia linear com múltiplos comutadores virtuais definindo o caminho fim-a-fim. Este circuito proveu conectividade para duas máquinas reais externas ao ambiente de emulação. Esta topologia foi usada para emular redes clientes, formadas pelas máquinas reais, conectando seus nodos remotos via infraestrutura de serviço de conectividade provido pela topologia linear no *Mininet*. O foco da análise foi o monitoramento da latência do comutador como um indicador do gargalo das pesquisas de correspondência de regras nas tabelas de fluxos do plano de dados de uma rede de núcleo definida por software.

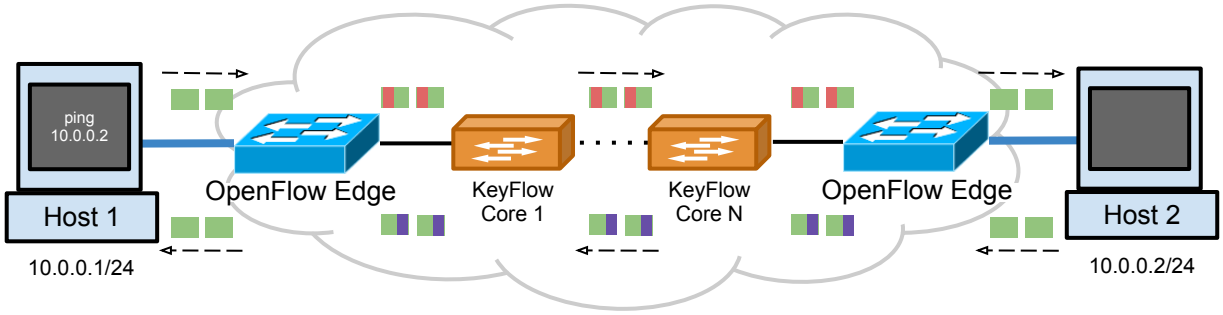
Verificou-se a qualidade da experiência de um usuário conectado a esta rede por meio da medição do RTT dos pacotes até um alvo em outra extremidade do caminho. Para cada experimento foram gerados 5000 envios de pacotes de 64 bytes em uma taxa constante. Foi realizada

[7] http://mininet.github.com

uma análise comparativa por meio da realização de testes utilizando os nodos de núcleo com comutadores *OpenFlow*, Fig. 4.4(a), onde se realizava pesquisa na tabela de fluxos salto a salto, comparativamente com o núcleo composto por comutadores *KeyFlow*, Fig. 4.4(b), que realizam a operação matemática resto da divisão para determinar a porta de saída do pacote. Neste caso, a pesquisa por tabela acontecia apenas nos elementos de borda do caminho.



(a) Núcleo com *lookup* em tabela de fluxo *OpenFlow*



(b) Núcleo com comutação KeyFlow

Figura 4.4: Topologia utilizada para avaliação do RTT do caminho lógico.

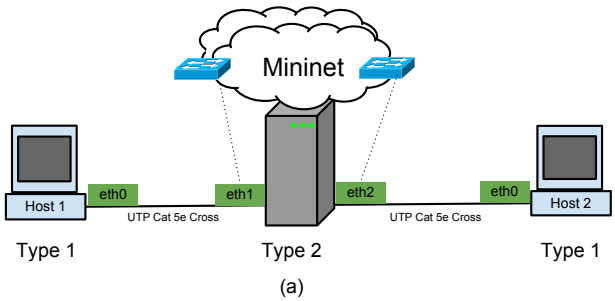
Nas extremidades, a rede cliente, foi configurada uma rede IP, e dela foram disparadas rajadas uniformes de pacotes ICMP para testes de continuidade e avaliação dos tempos de ida e volta dos pacotes. A rede da nuvem entre os nodos fica responsável pelo estabelecimento de conectividade fim-a-fim. Para se verificar a escalabilidade da solução, variou-se o número de comutadores de núcleo. Para avaliação da sensibilidade da transmissão frente a ocupação das tabelas de fluxos, variou-se o volume de regras instaladas nos elementos. Para identificação dos fluxos, utilizou-se diferentes 'id vlan' para se determinar o sentido do encaminhamento dos pacotes. Isto teve por objetivo criar um ambiente de encaminhamento similar às redes baseadas em comutação por rótulos.

Na topologia da Fig. 4.4(a), utilizou-se apenas comutadores *OpenFlow* em todo o caminho. A cada salto, faz-se necessária uma consulta na tabela de fluxo em busca da regra válida para o encaminhamento no sentido correto. Os elementos da borda eram os responsáveis por introduzir

e retirar devidamente o campo do id vlan, fazendo a definição do circuito virtual, por meio de uma regra devidamente pré-instalada. Na Fig. 4.4(b), utilizou-se uma topologia idêntica, mas com comutadores *KeyFlow*. O resultado desta operação definia a porta de saída do pacote, sem a necessidade de ocupação de uma tabela de fluxos.

Toda a instalação das regras e a definição de chaves foram realizadas antes do inicio dos testes ICMP. Isto foi realizado por meio de *scripts* em *shell* do Linux hospedeiro do *Mininet*, disponíveis no Apêndice A.5. Para estas sinalizações do plano de controle foi utilizado o aplicativo 'dpctl' com conexão direta pela *loopback* do sistema em cada comutador virtual, via conexão em diferentes portas TCP, uma para cada elemento. Todos os comutadores virtuais foram executados com privilégio de usuário no sistema. Para instalação da chave foi utilizada uma versão modificada do dpctl capaz de manipular a chave no comutador *KeyFlow*, apresentada na Seção 4.2.

Para realização dos experimentos descritos foram utilizadas três estações. Duas máquinas físicas idênticas realizavam as funções das redes usuárias IP, representados por 'Host 1' e 'Host 2', nas Figs. 4.4(a) e 4.4(b). Neste ambiente, cada algoritmo que representa um comutador é executado em um processo isolado que trata cada pacote recebido de maneira idêntica a um comutador real. A diferença do ambiente emulado para o real está na inexistência de transmissão dos dados, uma vez que cada processo envia e recebe os pacotes, ou mais especificamente quadros Ethernet, de interfaces virtuais do ambiente Linux, sem a necessidade de realização de E/S na comunicação inter-processos. Apenas os processos dos comutadores de borda foram conectados às interfaces físicas da máquina hospedeira, sendo possível a entrega dos quadros para as máquinas alvos, tornando o serviço de entrega de quadros mais próximo de um ambiente real. A Fig. 4.5(a) apresenta esquematicamente o ambiente experimental, e a tabela na Fig. 4.5(b), as configurações de cada máquina utilizada e as suas respectivas versões dos softwares.



(a)

Type	Hardware	Software
1	Generic i686 PC GigaEthernet Interface	CentOS 5.0 GNU/Linux 2.6.18-274.e15
2	CPU Intel Xeon 3075 (2 cores), 2.66GHz, Cache: L1 32KB, L2 4MB Memory DDR2 800MHz 4GB 2 GigaEthernet PCI 1 FastEthernet PCI (eth0) (remote access not shown) Motherboard: ProLiant ML110 G5	*Mininet* Ver. 1.0.0 Ubuntu 11.10 GNU/Linux 2.6.38-12-generic i686

(b)

Figura 4.5: Esquemático do ambiente real de testes.

Para avaliação da sensibilidade do circuito com a ocupação das tabelas de fluxo, variou-se a ocupação das tabelas *hash* de todos os elementos *OpenFlow* pertencentes ao caminho. Testes

preliminares mostraram que a ocupação completa da tabela linear, Fig. 4.6, para o ambiente emulado utilizado, não apresentou comprometimento significativo na comutação dos pacotes, possivelmente pelo baixo número de registros frente à capacidade computacional da máquina hospedeira do *Mininet*.

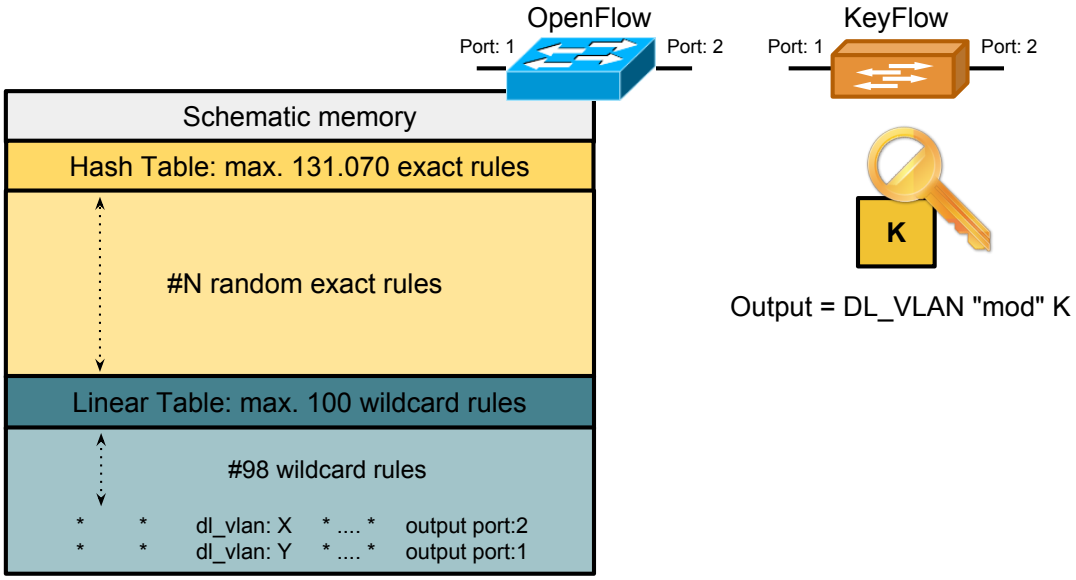


Figura 4.6: Informações de controle para manutenção de estado de encaminhamento em cada comutador.

Sendo assim, para análise da ocupação das tabelas de fluxos, utilizou-se a tabela linear sempre cheia, com 98 regras aleatórias inválidas para a comutação, e 2 regras válidas responsáveis pelo encaminhamento e/ou pela retirada da marcação do pacote, no caso de entrada e saída da rede. Essas regras eram inseridas no final da tabela linear. Além disso, utilizou-se quatro ocupações distintas da tabela *hash*: 0%, 25%, 50%, 75%. Essas porcentagens eram relativas à capacidade máxima da referida tabela (131.070 entradas).

Nos comutadores KeyFlow, para fins prova de conceito, foram instaladas as chaves iguais, com o número 3. Assim, os pacotes marcados com a o número 5, eram encaminhados para porta 2 (para a direita, pois $5 mod 3 = 2$), e os pacotes marcados com o número 4, eram encaminhados para a porta 1(para a esquerda, pois $4 mod 3 = 1$), conforme Fig. 4.6. Os elementos de borda eram responsáveis por adicionar e retirar estes dados do pacote na interface com o usuário IP da rede. No caso dos testes com o *KeyFlow*, os elementos de borda atuavam da mesma maneira e com a mesma ocupação da tabela *hash*.

Análise dos atrasos de envio dos pacotes

Para cada teste, foram gerados 5 mil pacotes ICMP de 64bytes, em intervalo constante, a partir do ‘Host 1’ em direção ao ‘Host 2’, e extraídas as estatísticas do atraso RTT de cada pacote. Foram utilizados os valores de média, de máximo (pior caso), de mínimo (melhor caso), e o desvio padrão. Para todos os testes não foram detectadas perdas de pacotes.

Testes preliminares foram realizados para analisar a interferência da arquitetura do hospedeiro Mininet no processamento das consultas das regras na tabela de fluxo dos comutadores virtuais. As regras na tabela de fluxo de cada comutador ficavam fisicamente inseridas na memória RAM do hospedeiro, uma vez que os comutadores era processos em execução no sistema. O processamento das instruções era feito pela captura de um lote de dados da RAM para a CPU do hospedeiro, assim um conjunto de regras da memória RAM eram armazenadas simultaneamente na memória cache da CPU. Por esta memória ser muito rápida, o tempo de consulta acabava sendo mascarado em relação ao processamento da função "módulo". Como a máquina possuia uma grande quantidade de memória cache em sua CPU, a busca em RAM era feita com menor frequência. Para minimizar a interferência da arquitetura da CPU da máquina do Mininet nos resultados, manteve-se um processo em execução com maior nível de privilégio que os processos dos comutadores virtuais, que realizava constantemente atualização de valores de um grande vetor, de maneira a ocupar uma significativa área da memória cache da CPU, tornando os experimentos mais próximos em termos da realidade de disponibilidade de memórias rápidas em comutadores comerciais.

Verifica-se, pelo gráfico da Fig. 4.7, que há um crescimento aproximadamente linear para o RTT máximo no caminho com comutadores de núcleo com consulta em tabela. Para os caminhos com comutadores KeyFlow, não há variação do RTT máximo com o crescimento do número de elementos. O atraso de pior caso é um bom indicador da eficiência da implementação em ambiente emulado, pois ele representa o atraso sofrido pelos pacotes que sofreram a pior oferta de poder computacional do sistema de emulação. Ou seja, o pior caso, ocorreu para os pacotes que foram processados sem a utilização de cache da CPU. Nota-se que para carga de ocupação da tabela igual, ou superior, a 50%, devido à sobrecarga na memória cache L2 interna à CPU, os resultados mostram-se mais expressivos, com um crescimento acentuado no atraso máximo para os caminhos totalmente baseado em consulta em tabelas. Em contrapartida, o atraso máximo para os caminhos com o *KeyFlow* se mantém constante a partir do valor imposto pela busca nos elementos de borda. Para valores de carga inferiores a 50%, é visível um crescimento mais suave do RTT máximo para o caminho com lookup de tabela. Para tabela vazia, os cenários utilizados não apresentam diferença significativa. Isto se deve à baixa ocupação da

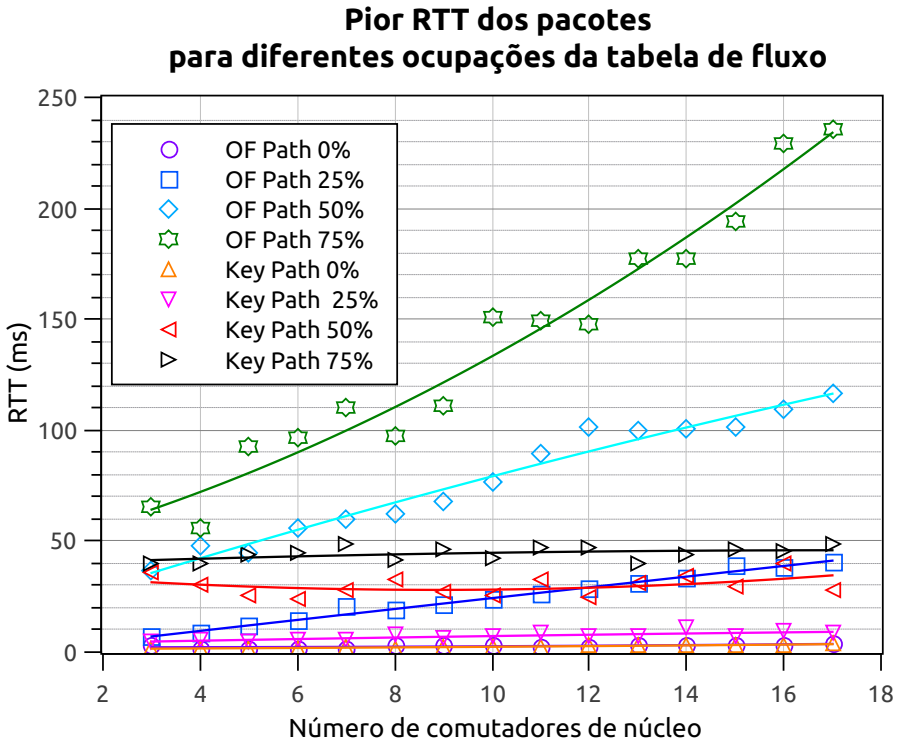


Figura 4.7: Comparativo do atraso máximo. (MARTINELLO et al., 2014; OLIVEIRA et al., 2013)

memória cache da CPU, o que foi verificado em testes preliminares. Mesmo assim, para uma ocupação de 25%, para o caminho mais longo, o RTT máximo chega a ser 80% menor para o circuito que utiliza o núcleo com o comutador *KeyFlow*.

No gráfico da Fig. 4.8, verifica-se que o melhor caso ocorreu para todos os pacotes que se beneficiaram da memória rápida da CPU, a cache L2 de 4MB. Isto indica que quanto mais houver memórias rápidas próximas ao sistema responsável pela comutação, mais eficiente será o encaminhamento. Contudo, isto implicará nos custos e no consumo de energia dos equipamentos.

Para o RTT médio, Fig 4.9, nota-se um crescimento mais discreto para cargas iguais ou superiores a 50% e uma diferença pouco expressiva na média para cargas inferiores a 25%. Novamente, isso ocorreu devido à capacidade de armazenamento intrínseco da CPU do sistema de emulação. Até que esta memória interna fosse saturada, a maioria dos pacotes sofriam atraso mínimo devido ao cache do processador. A partir do momento da saturação deste cache, a maior parte dos pacotes teve maior atraso devido a necessidade de busca na tabela de fluxo localizada na memória RAM. Fica evidente que a proposta baseada em chaves é eficiente e escalável, pois manteve a média em valores inferiores a 3ms independentemente do tamanho do caminho e da ocupação das tabelas das bordas.

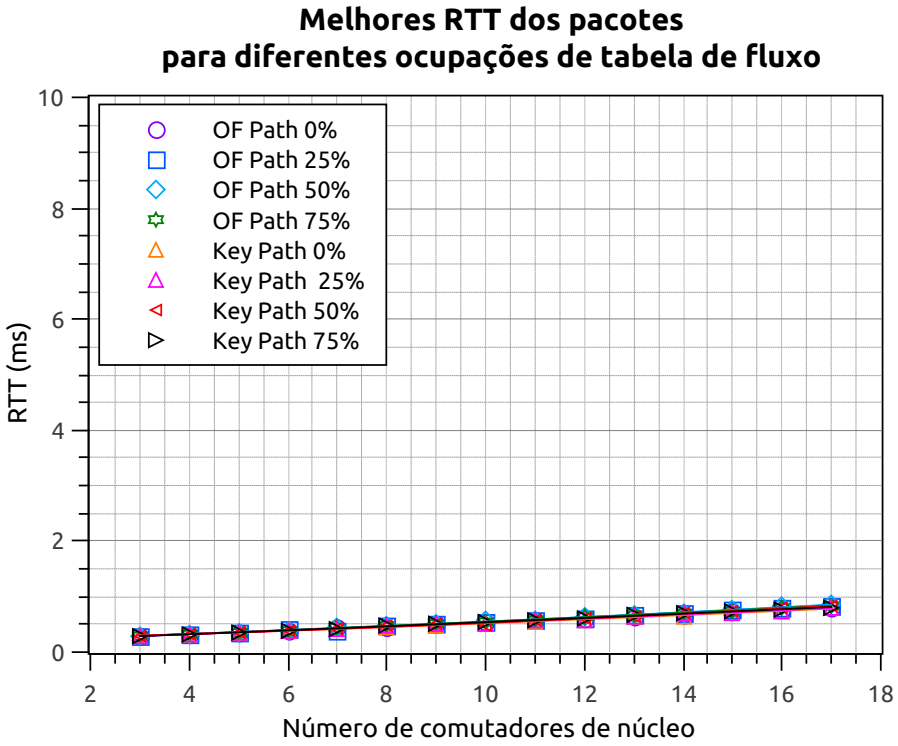


Figura 4.8: Comparativo do atraso mínimo. (OLIVEIRA et al., 2013)

O desvio padrão, apresentado no gráfico da Fig. 4.10, representa a variabilidade do RTT de cada circuito, ou seja, temos o comparativo do desvio padrão percebido em cada experimento. Quanto menor a variabilidade da rede de pacotes, mais essas redes se assimilarão às redes de circuitos, e mais uniforme será a comutação dos quadros. Sendo assim, verifica-se que os caminhos baseados em comutadores *KeyFlow* apresentaram uma significativa redução de desvio padrão no tempo de ida e volta dos pacotes enviados. No pior caso, com ocupação de 75% da tabela *hash*, obteve-se o desvio padrão constante inferior a 5 ms, independentemente do número de saltos do caminho. Já para o caminho baseado em busca em tabela, o pior caso apresenta uma variabilidade média superior em 40ms.

O gráfico da Fig. 4.11 pode ser utilizado para analisar a eficiência na redução dos atrasos dos caminhos baseados em comutadores *KeyFlow* em relação aos tempos obtidos em caminhos com os comutadores *OpenFlow* no núcleo da rede. Para o atraso mínimo, devido a capacidade computacional do emulador, a melhora notada é inferior a 5%. Pela mesma razão, percebe-se que a eficiência na redução da média do RTT cresce com o aumento da ocupação sobre a tabela *hash*, chegando próximo a 60% e a 75% no dois casos de maior ocupação da tabela. O desvio padrão apresentou melhora de 70% para os três patamares de ocupação significativa da tabela de fluxos. O RTT máximo também foi reduzido para estes casos em aproximadamente 60%.

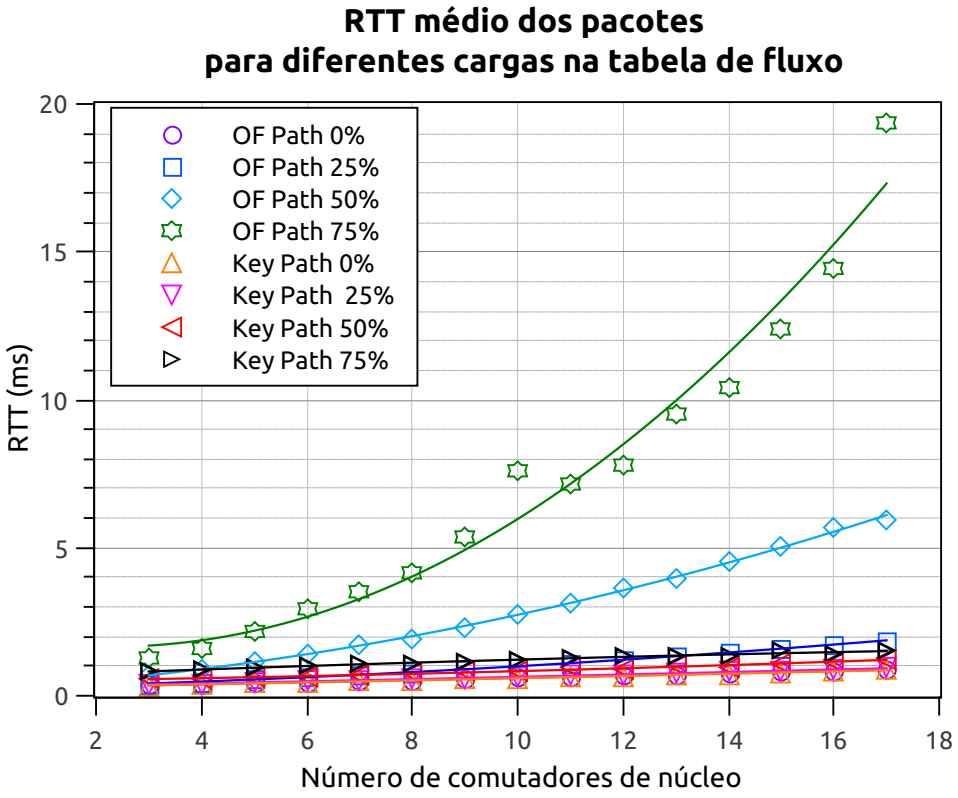


Figura 4.9: Comparativo do atraso médio. (MARTINELLO et al., 2014; OLIVEIRA et al., 2013)

4.4 Conclusão

Neste capítulo foi apresentada a estrutura de implementação do protótipo do KeyFlow e, também, os resultados obtidos na avaliação da escalabilidade da proposta em duas análises: no plano de controle, pela definição do tamanho do rótulo e pela análise do volume de dados necessário para manutenção de estados na rede de núcleo; e no plano de dados, pela avaliação da latência de comutação no estabelecimento de conexões lógicas por meio de caminhos formados por diferentes números de elementos de núcleo.

Os resultados mostraram que o rótulo identificador dos caminhos para a proposta KeyFlow, apesar da grande sensibilidade no crescimento do número inteiro utilizado na sua representação - a partir do teorema chinês do resto - pode ser representado para caminhos até 11 saltos em uma rede de até 60 nodos, para uma análise em pior caso.

A estrutura de comutação proposta apresentou significava redução no jitter das conexões lógicas estabelecidas, alcançando ganhos em relação a abordagem com manutenção de estados com OpenFlow superiores a 60% para modestas ocupação de fluxos ativos na ordem de 25 % da capacidade dos comutadores com tabelas de fluxo.

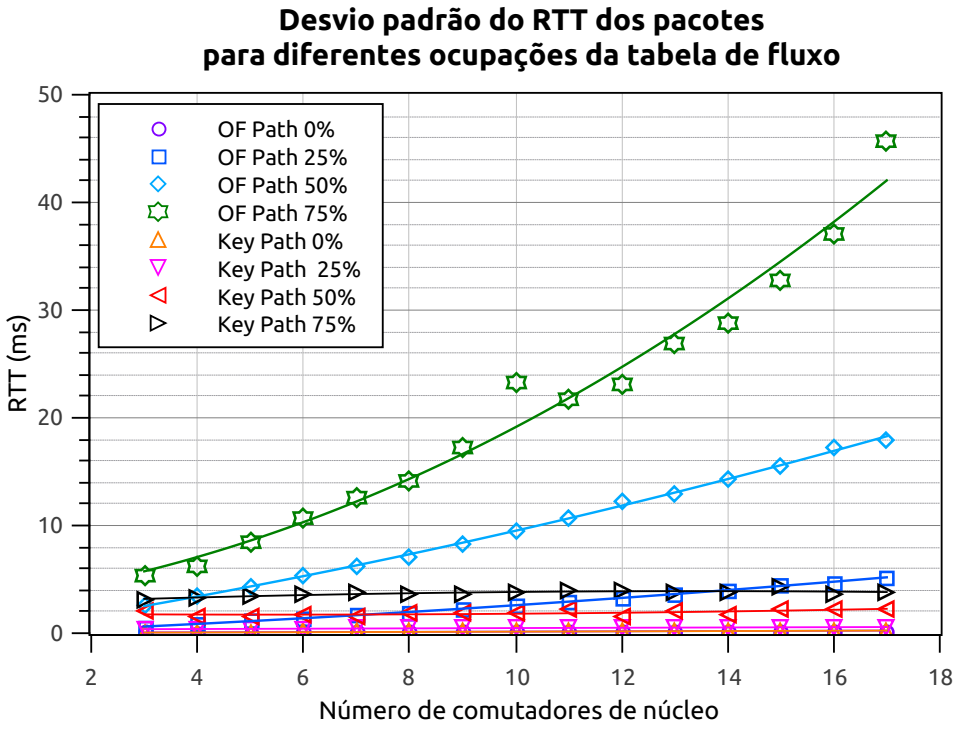


Figura 4.10: Comparativo do desvio padrão. (OLIVEIRA et al., 2013)

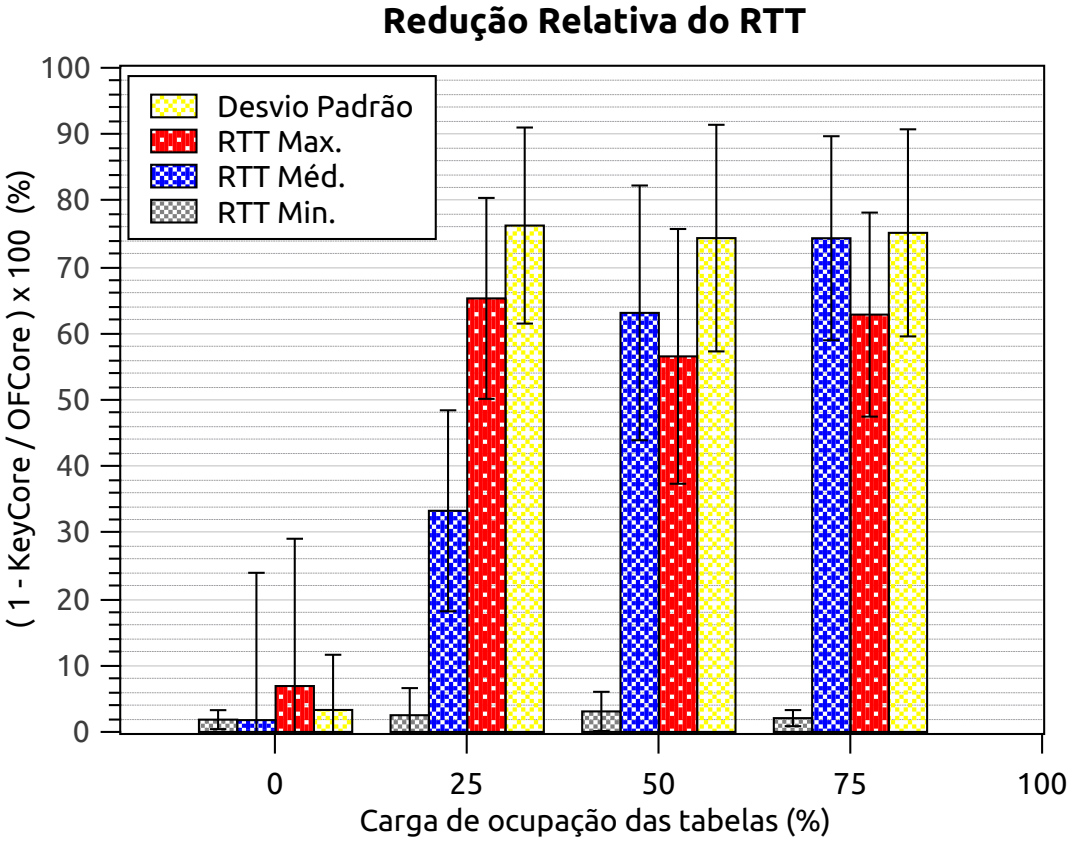


Figura 4.11: Comparativo do desempenho médio do RTT entre o núcleo KeyFlow vs. OpenFlow. (MARTINELLO et al., 2014; OLIVEIRA et al., 2013)

5 *Considerações Finais e Trabalhos Futuros*

Este trabalho apresentou uma alternativa de comutação para extensão da especificação do OpenFlow/SDN para sua aplicação no núcleo das redes. Por meio de uma implementação de uma rede *fabric* simples baseada na comutação de rótulos por meio de um sistema numérico de restos, via Teorema Chinês dos Restos. Este método propicia: latência reduzida na rede de núcleo; e requisitos simples para o *hardware* comutador, pela substituição de buscas em tabelas no mecanismo de encaminhamento. No plano de controle, a proposta conseguiu estender o desacoplamento do plano de controle do plano de dados por meio da redução do excessivo número de eventos necessário para cada novo fluxo junto ao controlador da rede, com KeyFlow este processo é necessário apenas para as bordas da rede.

A necessidade reduzida de interações entre o plano de controle em conjunto com a comutação *fabric* no núcleo da rede sem a manutenção do estado, viabiliza o provimento de conectividade flexível de acordo com as características do fluxo passante pela rede. As bordas podem definir diretamente o melhor caminho, e restringir os requisitos de qualidade de serviço, conforme visão global do controlador da rede sobre a capacidade do núcleo *fabric*. Com isto, a proposta viabiliza o tratamento integralizado para diferentes típos de fluxo em tempo real no núcleo da rede.

O protótipo, desenvolvido baseado na implementação de referência do comutador OpenFlow versão 1.0, foi testado experimentalmente no ambiente de prototipação virtualizado no Mininet. Os resultados obtidos, pela análise do RTT dos caminhos estabelecidos, apresentaram uma redução acima de 50 % no atraso de ida e volta dos pacotes utilizando o núcleo *fabric* KeyFlow em relação ao núcleo com apneas comutadores *OpenFlow*. Redes com alta densidade de fluxos serão as mais beneficiadas pela proposta. Os requisitos de manutenção de estado nas tabelas dos comutadores foram aliviados em mais de 70% considerando um núcleo de rede com pelo menos 7 saltos.

Com relação escalabilidade referente ao número de saltos em função do crescimento rápido

ocasionado pelas restrições do Teorema Chinês dos Restos, os resultados obtidos por uma análise de pior caso apontaram que, para redes de até 60 nodos, é possível a criação de caminhos *fabric* de até 11 saltos com rótulos de até 96 bits. Com isto, uma opção vantajosa está na utilização do próprio cabeçalho Ethernet, por meio dos campos de endereço de origem e destino, para abrigar o rótulo identificador do caminho dentro da núvem *fabric*. Com isto, a implementação do KeyFlow seria mais fácil e imediata, necessitando apenas que o controlador da rede realize as devidas sinalizações por meio do protocolo OpenFlow nos elementos de borda para recomposição dos endereços MACs originais dos pacotes. As bordas necessitariam armazenar os estados originais dos fluxos para apenas aqueles que são destinados a redes diretamente conectadas, sem a necessidade de armazenamento global de todos os fluxos existentes na rede. Assim, além de servir para conectividade de redes OpenFlow, a proposta de KeyFlow também se aplica para outros protocolos, como o IP e Ethernet, viabilizando o criação de caminhos lógicos tanto para reduzir o número de saltos na rede IP, como para estabelecer caminhos sem a existência de *loops* em redes MetroEthernet, o que melhoria a ocupação dos enlaces disponíveis na rede.

Como uma alternativa para os atuais comutadores de alto desempenho, baseados em memórias rápidas com alto consumo de energia, o KeyFlow tem o potencial de viabilizar a implementação de comutadores mais baratos e ecologicamente viáveis. Como trabalho futuro, é planejado a implementação de comutadores KeyFlow em NetFPGA (NAOUS et al., 2008), para análise comparativa de consumo de energia, como também embarcar a solução em comutadores de prateleira, tanto cabeados como sem fio, para avaliar os benefícios da proposta para ambientes com baixa disponibilidade de poder computacional.

É esperado também a aplicação de esforços para investigação de formas para estabelecimento de caminhos de proteção que podem ser pré-computados no controlador, e tão logo o controlador receba a notificação de falhas, esse caminho de proteção possa ser usado de maneira rápida pela atribuição do novo rótulo diretamente no nodo de borda para desviar o tráfego da rota dos elementos em falha na rede.

Referências Bibliográficas

BARONI, S. et al. Analysis and design of backbone architecture alternatives for ip optical networking. *Selected Areas in Communications, IEEE Journal on*, IEEE, v. 18, n. 10, p. 1980–1994, 2000.

CAESAR, M. et al. Design and implementation of a routing control platform. In: USENIX ASSOCIATION. *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. [S.l.], 2005. p. 15–28.

CASADO, M. et al. Ethane: Taking control of the enterprise. *ACM SIGCOMM Computer Communication Review*, ACM, v. 37, n. 4, p. 1–12, 2007.

CASADO, M. et al. Fabric: A retrospective on envolving sdn. In: *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. [S.l.: s.n.], 2012.

CHOWDHURY, N. M. K.; BOUTABA, R. Network virtualization: state of the art and research challenges. *Communications Magazine, IEEE*, IEEE, v. 47, n. 7, p. 20–26, 2009.

CURTIS, A. R. et al. Devoflow: Scaling flow management for high-performance networks. In: ACM. *ACM SIGCOMM Computer Communication Review*. [S.l.], 2011. v. 41, n. 4, p. 254–265.

ENNS, R. et al. *RFC 6241, Network Configuration Protocol (NETCONF)*. [S.l.]: June, 2011.

EWG, O. *Extensibility Working Group*. [S.l.], 4 2013. Disponível em: <https://www.opennetworking.org/images/stories/downloads/working-groups/charter-extensibility.pdf>.

GREENBERG, A. et al. A clean slate 4d approach to network control and management. *ACM SIGCOMM Computer Communication Review*, ACM, v. 35, n. 5, p. 41–54, 2005.

HUANG, D. Y.; YOCUM, K.; SNOEREN, A. C. High-fidelity switch models for software-defined network emulation. In: ACM. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. [S.l.], 2013. p. 43–48.

KOPONEN, T. et al. Onix: A distributed control platform for large-scale production networks. In: *OSDI*. [S.l.: s.n.], 2010. v. 10, p. 1–6.

LEVIN, D. et al. Logically centralized?: state distribution trade-offs in software defined networks. In: ACM. *Proceedings of the first workshop on Hot topics in software defined networks*. [S.l.], 2012. p. 1–6.

LU, G. et al. Using cpu as a traffic co-processing unit in commodity switches. In: ACM. *Proceedings of the first workshop on Hot topics in software defined networks*. [S.l.], 2012. p. 31–36.

MACHADO, I.; STANTON, M.; SALMITO, T. Rnp evaluation of alternative implementations for dynamic circuits. *Proceedings of the Asia-Pacific Advanced Network*, v. 31, p. 23–27, 2011.

MARTINELLO, M. et al. Keyflow: a prototype for evolving sdn toward core network fabrics. *Network, IEEE*, IEEE, v. 28, n. 2, p. 12–19, 2014.

MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>.

MENDONCA, M. et al. A survey of software-defined networking: Past, present, and future of programmable networks. 2013.

MERWE, J. E. Van der et al. The tempest-a practical framework for network programmability. *Network, IEEE*, IEEE, v. 12, n. 3, p. 20–28, 1998.

MOGUL, J. C.; CONGDON, P. Hey, you darned counters!: get off my asic! In: ACM. *Proceedings of the first workshop on Hot topics in software defined networks*. [S.l.], 2012. p. 25–30.

MYSORE, R. N. et al. Portland: a scalable fault-tolerant layer 2 data center network fabric. In: ACM. *ACM SIGCOMM Computer Communication Review*. [S.l.], 2009. v. 39, n. 4, p. 39–50.

NAOUS, J. et al. Implementing an openflow switch on the netfpga platform. In: ACM. *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. [S.l.], 2008. p. 1–9.

OLIVEIRA, R. E. Z. de et al. Keyflow: Comutaçao por chaves locais de fluxos roteados na borda via identificadores globais. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos-SBRC*, v. 13, n. 9, 2013.

ONF. *Software-Defined Networking: The New Norm for Networks*. [S.l.], 04 2013. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>.

PHEMIUS, K.; THALES, M. B. Openflow: Why latency does matter. In: IEEE. *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. [S.l.], 2013. p. 680–683.

PROTOCOL, N. C. Rfc 4741. *Dezembro de*, 2006.

RAGHAVAN, B. et al. Software-defined internet architecture: decoupling architecture from infrastructure. In: *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. [S.l.: s.n.], 2012. p. 43–48. ISBN 978-1-4503-1776-4.

ROTSOS, C. et al. Oflops: an open framework for openflow switch evaluation. In: *Proceedings of the 13th international conference on Passive and Active Measurement*. [S.l.: s.n.], 2012. (PAM'12), p. 85–95. ISBN 978-3-642-28536-3.

SALLENT, S. et al. Fibre project: Brazil and europe unite forces and testbeds for the internet of the future. In: *Testbeds and Research Infrastructure. Development of Networks and Communities*. [S.l.]: Springer, 2012. p. 372–372.

SPEC, O. *OpenFlow Switch Specification Ver. 1.0.0*. [S.l.], 12 2009. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf>.

SPEC, O. *OpenFlow Switch Specification Ver. 1.1.0*. [S.l.], 02 2011. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.1.0.pdf>.

SPEC, O. *OpenFlow Switch Specification Ver. 1.2.0*. [S.l.], 12 2011. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf>.

SPEC, O. *OpenFlow Switch Specification Ver. 1.3.0*. [S.l.], 06 2012. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf>.

SPEC, O. *OpenFlow Switch Specification Ver. 1.4.0*. [S.l.], 10 2013. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>.

STALLINGS, W. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1998.

TAKEDA, T.; KOJIMA, H.; INOUE, I. Layer 1 vpn architecture and its evaluation. In: IEEE. *Communications, 2004 and the 5th International Symposium on Multi-Dimensional Mobile Communications Proceedings. The 2004 Joint Conference of the 10th Asia-Pacific Conference on*. [S.l.], 2004. v. 2, p. 612–616.

TOOTOONCHIAN, A.; GANJALI, Y. Hyperflow: a distributed control plane for openflow. In: USENIX ASSOCIATION. *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. [S.l.], 2010. p. 3–3.

TOWARD, A. Network-wide decision making: Toward a wafer-thin control plane. 2004.

WESSING, H. et al. Novel scheme for packet forwarding without header modifications in optical networks. *Lightwave Technology, Journal of*, v. 20, n. 8, p. 1277 – 1283, aug 2002. ISSN 0733-8724.

YEGANEH, S. H.; GANJALI, Y. Kandoo: a framework for efficient and scalable offloading of control applications. In: ACM. *Proceedings of the first workshop on Hot topics in software defined networks*. [S.l.], 2012. p. 19–24.

YU, M.; WUNDSAM, A.; RAJU, M. Nosix: A lightweight portability layer for the sdn os. *ACM SIGCOMM Computer Communication Review*, ACM, v. 44, n. 2, p. 28–35, 2014.

Anexos

A.1 Código Python para Computação do Rótulo Global Key-Flow

Uma explicação matemática sobre a lógica de cálculo do rótulo encontra-se na Sec. 3.4.1.

```
#!/usr/bin/python
import sys; import math;import random; from random import randint

def extended_euclid(m,k):
        if k == 0:      return m,1,0
        else:
                dl,xl,yl = extended_euclid(k, m%k)
                d,x,y = dl,yl,xl-(m/k)*yl
                return d,x,y

def multiplicative_inverse(m,k):
        d,x,y = extended_euclid(m,k)
        if x < 0: x = x + (-1*x)/k + k
        return x

def genKeyflowLabel(ls_nodos, ls_ports):
        n = len(ls_nodos); k = 1; M = []; Ml = []; C = [];auxSC=0
        for i in range(n): k = k * ls_nodos[i]
        for i in range(n):
                M.append(k / ls_nodos[i])
                Ml.append(multiplicative_inverse(M[i],ls_nodos[i]))
                C.append(M[i]*(Ml[i]%ls_nodos[i]))
        for i in range(n): auxSC = auxSC + S[i] * C[i]
        r = auxSC % k
        return r
```

A.2 Alterações na biblioteca OpenFlow.h para suporte a comutação KeyFlow

```
#> diff openflow/include/openflow/openflow.h keyflow-coreswitch/include
136c136
<     OFPT_QUEUE_GET_CONFIG_REPLY     /* Controller/switch message */
---
>     OFPT_QUEUE_GET_CONFIG_REPLY,    /* Controller/switch message */
137a138,139
>     /* Key messages. */
>     OFPT_KEY_MOD                /* Controller/switch message */
728a731,735
>     /* Switch Key
>      * The request body is empty.
>      * The reply body is struct ofp_key. */
>     OFPST_KEY,
>
968a976,986
>
> struct ofp_key {
>         uint32_t key;
> };
> OFP_ASSERT(sizeof(struct ofp_key) == 4);
>
> struct ofp_key_mod {
>         struct ofp_header header;
>         uint32_t key;
> };
> OFP_ASSERT(sizeof(struct ofp_key_mod) == 12);
```

A.3 Alterações na Plano de Dados do comutador OpenFlow para suporte a comutação KeyFlow

```
1219,1221c1219,1232
<     if (run_flow_through_tables(dp, buffer, p)) {
<         dp_output_control(dp, buffer, p->port_no,
```

```
<                               dp->miss_send_len, OFPR_NO_MATCH);
---
>     if(!dp->key) {
>         if (run_flow_through_tables(dp, buffer, p)) {
>             dp_output_control(dp, buffer, p->port_no,
>                               dp->miss_send_len, OFPR_NO_MATCH);
>         }
>     } else {
> /* MAC */
>         struct eth_header *eh = buffer->data;
>               output_packet(dp, buffer, (*(uint32_t*)(eh->eth_dst)) %
>
> /* VLAN
>         struct vlan_eth_header *veh = buffer->data;
>         output_packet(dp, buffer, veh->veth_tci % dp->key, 0);
> */
#>diff openflow/udatapath/datapath.c keyflow-coreswitch/udatapath/datap

1340a1352,1362
> recv_key_mod(struct datapath *dp, const struct sender *sender UNUSED,
>              const void *msg)
> {
>     const struct ofp_key_mod *opm = msg;
>
>     dp->key = opm->key;
>
>     return 0;
> }
>
> static int
1548a1571,1581
> static int
> key_stats_dump(struct datapath *dp UNUSED, void *state UNUSED,
>                struct ofpbuf *buffer)
> {
```

```
>     struct ofp_key *ods = ofpbuf_put_uninit(buffer, sizeof *ods);
>
>     ods->key = dp->key;
>
>     return 0;
> }
>
2035a2069,2076
>     {
>         OFPST_KEY,
>         0,
>         0,
>         NULL,
>         key_stats_dump,
>         NULL
>     },
2295a2337,2340
>         break;
>     case OFPT_KEY_MOD:
>         min_size = sizeof(struct ofp_key_mod);
>         handler = recv_key_mod;
```

A.4 Alterações do utilitário *dpctl* de controle referência do comutador OpenFlow de referência

```
# diff dpctl-of.c dpctl-kf.c
227a228,229
>             "  dump-key SWITCH              print switch key\n"
>             "  key-mod SWITCH KEY           modify switch key\n"
729a732,758
> do_dump_key(const struct settings *s UNUSED, int argc UNUSED, char *a
> {
>     dump_trivial_stats_transaction(argv[1], OFPST_KEY);
> }
>
> static void
```

```
> do_mod_key(const struct settings *s UNUSED, int argc UNUSED, char *arg
> {
>     struct vconn *vconn;
>     struct ofpbuf *buffer;
>     struct ofp_key_mod *ofm;
>
>     buffer = ofpbuf_new(sizeof(*ofm));
>     ofpbuf_put_uninit(buffer, sizeof(*ofm));
>
>     ofm = ofpbuf_at_assert(buffer, 0, sizeof(*ofm));
>     ofm->header.version = OFP_VERSION;
>     ofm->header.type = OFPT_KEY_MOD;
>     ofm->header.length = htons(sizeof(*ofm));
>     ofm->key = atoi(argv[2]);
>
>     open_vconn(argv[1], &vconn);
>     send_openflow_buffer(vconn, buffer);
>     vconn_close(vconn);
> }
>
> static void
1770a1800,1803
>
>     { "dump-key", 1, 1, do_dump_key },
>     { "key-mod", 2, 2, do_mod_key },
>
```

A.5 Scripts utilizados para manipulação do ambiente de prototipação do Mininet

```
#!/bin/bash

create_topo=/home/rezo/mininet/examples/linehw.py

intf1="eth1"    #interface fisica da maquina
intf2="eth2"
```

```
mode=$1 #modo key instala chave nos switches
num_switches=$2
ofpsw_first="6634"  #porta do primeiro switch para conexao do plano de
ofpsw_last=$(( ofpsw_first + num_switches -1 ))

if ! [[ "$num_switches" =~ ^[0-9]+$ || $# < 3 ]] ; then
 echo "***# Invalid command.Try again."
 echo " $0 <mode> <number of switches>"
 echo " $0 <mode [default/key]> <number of switches [int>0] >"
else
 if [[ "$mode" == "default" ]]; then
   echo "Initializing NORMAL mode openflow switch experiment..."
   echo "Creating topology and connect to interfaces..."
   $create_topo $num_switches $intf1 $intf2
   ERRO=0
   echo "Installing experiment flows in switches..."
   if [[ $ofpsw_first != $ofpsw_last ]]; then
     for (( i=$ofpsw_first; i<=$ofpsw_last; i++))
       do
         sleep 1
         if [[ $i == $ofpsw_first ]];  then
            vlanToRight=1
            echo "vlanToRight = $vlanToRight"
            echo "vlanToLeft = $vlanToLeft"
            echo "in_port=1,idle_timeout=0,actions=mod_vlan_vid:$vlanTo
            echo "in_port=2,idle_timeout=0,actions=strip_vlan ,output:1"
            echo "Installing flows in switch $i"
            dpctl add-flows tcp:127.0.0.1:$i /tmp/sw-flows-$i
            echo "== FLOWS IN SWITCH LISTEN PORT $i ==================
            dpctl dump-flows tcp:127.0.0.1:$i
            chk=$(dpctl dump-flows tcp:127.0.0.1:$i | wc -l)
            if ! [[ "$chk" == 3 ]]; then
               echo "ERROR! Switch listen port $i have incorret number
               ERRO=1
```

```
    else
       echo "Ok! Flows installed in switch $i"
    fi
elif [[ $i == $ofpsw_last ]]; then
    vlanToLeft=$(( 2*$i - 2*ofpsw_first ))
    echo "vlanToRight = $vlanToRight"
    echo "vlanToLeft = $vlanToLeft"
  # The last switch, unfortunately, only put phy interface int
  # So, the rule must invert in_port number
    echo "in_port=2,idle_timeout=0,actions=strip_vlan,output:1"
    echo "in_port=1,idle_timeout=0,actions=mod_vlan_vid:$vlanTo
    echo "Flows installed in switch $i"
    dpctl add-flows tcp:127.0.0.1:$i /tmp/sw-flows-$i
    echo "== FLOWS IN SWITCH LISTEN PORT $i ==================
    dpctl dump-flows tcp:127.0.0.1:$i
    chk=$(dpctl dump-flows tcp:127.0.0.1:$i | wc -l)
    if ! [[ "$chk" == 3 ]]; then
        echo "ERROR! Switch listen port $i have incorret number
        ERRO=1
    else
        echo "Ok! Flows installed in switch $i"
    fi
else
    vlanToRight=$(( $i - $ofpsw_first + 1 ))
    vlanToLeft=$(( 2*$i - 2*$ofpsw_first ))
    echo "vlanToRight = $vlanToRight"
    echo "vlanToLeft = $vlanToLeft"
    echo "dl_vlan=5,idle_timeout=0,actions=output:2" > /tmp/sw-
    echo "dl_vlan=4,idle_timeout=0,actions=output:1" >> /tmp/sw
    dpctl add-flows tcp:127.0.0.1:$i /tmp/sw-flows-$i
    echo "== FLOWS IN SWITCH LISTEN PORT $i ==================
    dpctl dump-flows tcp:127.0.0.1:$i

    chk=$(dpctl dump-flows tcp:127.0.0.1:$i | wc -l)
    if ! [[ "$chk" == 3 ]]; then
```

```
                echo "ERROR! Switch listen port $i have incorret number
                ERRO=1
            else
                echo "Ok! Flows installed in switch $i"
            fi
        fi
      done
    else
      echo "in_port=1,idle_timeout=0,actions=output:2" > /tmp/sw-flows-$
      echo "in_port=2,idle_timeout=0,actions=output:1" >> /tmp/sw-flows-
      dpctl add-flows tcp:127.0.0.1:$ofpsw_first /tmp/sw-flows-$ofpsw_fi
      echo "== FLOWS IN SWITCH LISTEN PORT $ofpsw_first ==============
      dpctl dump-flows tcp:127.0.0.1:$ofpsw_first
      chk=$(dpctl dump-flows tcp:127.0.0.1:$ofpsw_first | wc -l)
      if ! [[ "$chk" == 3 ]]; then
         echo "ERROR! Switch listen port $ofpsw_first have incorret numb
         ERRO=1
      else
         echo "Ok! Flows installed in switch $i"
      fi
    fi
    if [[ "$ERRO" == 1 ]]; then
      echo "Error in flow install process"
    fi

elif [[ "$mode" == "key" ]]; then
    echo "Initializing KEY mode openflow switch experiment..."
    echo "Creating topology and connect to interfaces..."
    $create_topo $num_switches $intf1 $intf2
    ERRO=0
    echo "Installing experiment flows in switches..."
    if [[ $ofpsw_first != $ofpsw_last ]]; then
       for (( i=$ofpsw_first; i<=$ofpsw_last; i++))
       do
          sleep 1
```

```
if [[ $i == $ofpsw_first ]];  then
  vlanToRight="5"
  echo "in_port=1,idle_timeout=0,actions=mod_vlan_vid:$vlanToR
  echo "in_port=2,idle_timeout=0,actions=strip_vlan ,output:1"
  echo "Installing flows in switch $i"
  dpctl add-flows tcp:127.0.0.1:$i /tmp/sw-flows-$i
  echo "== FLOWS IN SWITCH LISTEN PORT $i ==================
  dpctl dump-flows tcp:127.0.0.1:$i
  chk=$(dpctl dump-flows tcp:127.0.0.1:$i | wc -l)
  if ! [[ "$chk" == 3 ]]; then
    echo "ERROR! Switch listen port $i have incorret number of
    ERRO=1
  else
    echo "Ok! Flows installed in switch $i"
  fi
elif [[ $i == $ofpsw_last ]]; then
  vlanToLeft="4"
  echo "vlanToRight = $vlanToRight"
  echo "vlanToLeft = $vlanToLeft"
  # The last switch, unfortunately, only put phy interface int
  # So, the rule must invert in_port number
  echo "in_port=2,idle_timeout=0,actions=strip_vlan ,output:1" :
  echo "in_port=1,idle_timeout=0,actions=mod_vlan_vid:$vlanToL
  echo "Flows installed in switch $i"
  dpctl add-flows tcp:127.0.0.1:$i /tmp/sw-flows-$i
  echo "== FLOWS IN SWITCH LISTEN PORT $i ==================
  dpctl dump-flows tcp:127.0.0.1:$i
  chk=$(dpctl dump-flows tcp:127.0.0.1:$i | wc -l)
  if ! [[ "$chk" == 3 ]]; then
    echo "ERROR! Switch listen port $i have incorret number of
    ERRO=1
  else
   echo "Ok! Flows installed in switch $i"
  fi
else
```

```
            dpctl key-mod tcp:127.0.0.1:$i 3
        fi
      done
    else
       echo "Error! Number of switche must be greater then 3"
    fi
    if [[ "$ERRO" == 1 ]]; then
       echo "Error in flow install process"
    fi
  elif [[ "$mode" == "stop" ]]; then
    killall ofprotocol
    killall ofdatapath
    killall controller

  else
    echo "Invalid mode to operate.Try again."
    echo "$0 <mode [default/key]> <number of switches [int>0] >"
  fi
fi
```