RouteFlow: Roteamento Commodity Sobre Redes Programáveis

Marcelo Nascimento[1], **Christian Rothenberg**[1],
Rodrigo Denicol[1], **Marcos Salvador**[1], **Maurício Magalhães**[2]

[1]CPqD - Centro de Pesquisa e Desenvolvimento em Telecomunicações
marcelon@cpqd.com.br

[2]Unicamp - Universidade Estadual de Campinas
mauricio@dca.fee.unicamp.br

Abstract

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

Keywords: OpenFlow, RouteFlow, routing, architecture

Resumo

Os roteadores atuais implementam uma arquitetura composta de uma camada de software e um hardware proprietários. Este modelo resulta em soluções de alto custo e inviabiliza a experimentação de novas ideias. Em contrapartida, existem alternativas de alta flexibilidade baseadas em software e, consequentemente, de baixo custo. Entretanto, essas soluções apresentam baixo desempenho. Motivado pela disponibilidade de uma API aberta para programação do plano de encaminhamento (i.e., OpenFlow), este trabalho apresenta o projeto RouteFlow. Trata-se de uma arquitetura de roteamento IP que procura combinar o alto desempenho de hardwares de prateleira (commodities) com a flexibilidade de uma pilha de roteamento executada remotamente em computadores de uso geral. O grande desafio é garantir confiabilidade, escalabilidade e desempenho à rede, a partir de um controle remoto e centralizado, cuja arquitetura permita maior flexibilidade no mapeamento entre os elementos de controle e encaminhamento. O resultado corresponde a uma nova solução de roteamento IP com perspectivas promissoras do ponto de vista do custo e da flexibilidade. A avaliação do protótipo apresentada no artigo comprova a viabilidade da arquitetura proposta.

Palavras-chave: OpenFlow, RouteFlow, roteamento, arquitetura

1. INTRODUÇÃO

A infraestrutura das redes de pacotes é normalmente composta por equipamentos proprietários, fechados e de alto custo, cujas arquiteturas básicas são concebidas a partir da combinação de um processador dedicado ao processamento de pacotes e responsável por garantir o alto desempenho. A arquitetura é implementada por uma camada de software de controle constituída por uma com-

plexa pilha de protocolos com o objetivo de atender adequadamente os requisitos das redes de computadores [10]. No entanto, torna-se evidente a necessidade de especialização da lógica de controle de acordo com cada tipo e objetivo de rede, em especial, no caso das redes corporativas. No âmbito da pesquisa, essa exigência cresce com a necessidade de experimentações de novos protocolos [1]. A ausência de flexibilidade e o alto custo da infraestrutura vigente são barreiras que dificultam o avanço das redes e a inovação [18].

Com o objetivo de suprir tais necessidades, existem alternativas como *software routers* (ex: RouteBricks [6], [22]), que são soluções que utilizam "hardware de prateleira" com grande capacidade de processamento (ex.: x86) e onde todo processamento de pacotes é realizado no nível de software. Essa opção torna a proposta bastante flexível mas acarreta, por outro lado, um grande prejuízo de desempenho [2]. Alternativas baseadas em FPGAs apresentam bom desempenho, entretanto, são de alto custo e o tempo de desenvolvimento é bastante elevado. Este último fator é ainda mais crítico nas alternativas baseadas em *network processors* (NP) [20]. Propostas recentes [11] têm explorado também o uso de *graphics processing units* (GPUs) para acelerar o processamento de pacotes nos *software routers*.

A meta desse trabalho é abordar, simultaneamente, as questões de desempenho, flexibilidade e custo, conforme apresentado em [15]. A estratégia baseia-se no uso da recente tecnologia de *switches* programáveis [13], o que possibilita mover o plano de controle, antes embarcado no equipamento, para um dispositivo externo [9]. A finalidade, no caso, é permitir a programação remota do plano de encaminhamento. O resultado consiste em uma solução flexível de alto desempenho e comercialmente competitiva, denominada **RouteFlow**, a partir da combinação de recursos disponíveis, tais como: (a) *switches* programáveis de baixo custo e software embarcado reduzido; (b) pilha de protocolos de roteamento *open source*; e (c) servidor de prateleira de alto poder de processamento e também de baixo custo.

O RouteFlow armazena a lógica de controle dos *switches* programáveis utilizados na infraestrutura de rede através de uma rede virtual composta por máquinas virtuais (MVs), cada uma executando um código (*engine*) de roteamento de domínio público (*open source*). Essas MVs podem ser interconectadas de maneira a formar uma topologia lógica espelhando a topologia de uma rede física correspondente. O ambiente virtual é armazenado em um servidor externo, ou um conjunto deles, que se comunica com os equipamentos do plano de dados através de um elemento chamado de controlador, cuja responsabilidade é transportar para o plano de encaminhamento as decisões tomadas pelo plano de controle.

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

Deve ser destacado que a arquitetura do RouteFlow promove a efetiva separação entre os planos de controle e de encaminhamento. Essa separação traz inúmeras vantagens com relação ao isolamento de falhas e, também, às questões relacionadas à segurança. Outros destaques da arquitetura proposta são a de flexibilidade de configuração e implementação das funções de controle (roteamento) nas máquinas virtuais e o desempenho, uma vez que o tráfego de dados é processado em hardware na taxa de transferência das interfaces (*line rate*). Podemos ressaltar ainda a utilização de equipamentos comerciais de prateleira, o que resulta em uma implementação de baixo custo. Esta abordagem privilegia, inicialmente, os requisitos das redes corporativas/campus. Extensões como, por exemplo, a hierarquização do plano de controle, poderá permitir, futuramente, a aplicação de modelos similares no caso das redes de núcleo *backbones*.

O restante deste artigo está organizado da seguinte forma: a Seção 2 discute as tecnologias e arquiteturas relacionadas à proposta apresentada neste artigo; a Seção 3 trata dos trabalhos relacionados; a Seção 4 apresenta a arquitetura e os componentes do RouteFlow; a Seção 5 descreve a elaboração e implementação de um protótipo aderente à arquitetura RouteFlow; a Seção 6 apresenta o ambiente de testes e a avaliação do protótipo implementado; a Seção 7 apresenta uma discussão dos resultados obtidos; a Seção 8 trata dos trabalhos futuros e, por último, a Seção 9 apresenta as conclusões.

2. TECNOLOGIAS E ARQUITETURAS RELACIONADAS

Uma análise da arquitetura de roteamento atual (lado esquerdo da Figura 1) permite observar que se trata de um modelo formado basicamente por duas camadas bem distintas: o software de controle e o hardware dedicado ao encaminhamento de pacotes. O primeiro, encarregado de tomar as decisões de roteamento, transfere essas decisões para o plano de encaminhamento através de uma API proprietária. A única interação da gerência com o dispositivo é através de interfaces de configuração (e.g., Web, SNMP, CLI), limitando o uso dos dispositivos às funcionalidades

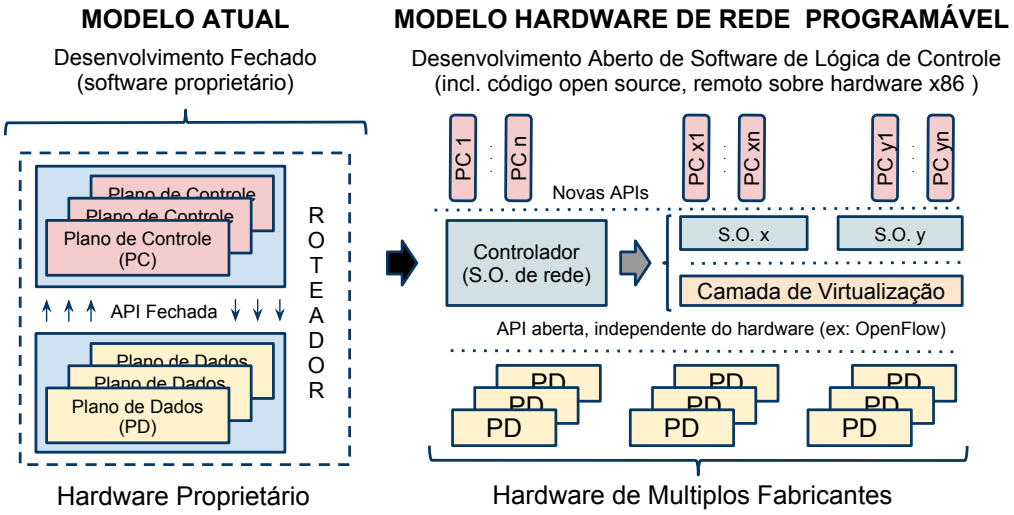


Figura 1. Arquiteturas de roteamento.

programadas pelo fabricante.

A conclusão de que a arquitetura atual é composta por duas camadas auto-contidas sugere que elas não precisariam encontrar-se acopladas em um mesmo equipamento. Para isso, basta que exista uma forma padrão de programar o dispositivo de rede remotamente, permitindo a camada de controle ser movida para um servidor dedicado e com alta capacidade de processamento. Deste modo, mantém-se o alto desempenho no encaminhamento de pacotes aliado à flexibilidade de inserir, remover e especializar aplicações utilizando uma API aberta para programação do roteador (lado direito da Figura 1). Com este propósito, surgiu o consórcio OpenFlow [17].

O OpenFlow define um protocolo padrão para determinar as ações de encaminhamento de pacotes em dispositivos de rede tais como *switches*, roteadores e pontos de acesso sem fio. A principal abstração utilizada na especificação do OpenFlow é o conceito de fluxo. Um fluxo é constituído pela combinação de campos do cabeçalho do pacote a ser processado pelo dispositivo. As tuplas podem ser formadas por campos das camadas de enlace, de rede ou de transporte, segundo o modelo TCP/IP. Deve ser enfatizado que a abstração de tabela de fluxos ainda está sujeita a refinamentos com o objetivo de expor melhor os recursos do hardware.

Pragmaticamente, a especificação OpenFlow procura reutilizar as funcionalidades dos hardwares existentes (ACL - *Access Control List*) através da definição de um conjunto simples de regras e das ações associadas (ex., encaminhar, descartar, enviar para o controlador, reescrever campos do cabeçalho do pacote, etc.). Deste modo, o OpenFlow pode ser suportado por dispositivos existentes através de uma atualização do *firmware*. Deve ser destacado que o OpenFlow tem atraído a atenção da indústria o que tem se traduzido na disponibilidade de equipamentos com suporte ao OpenFlow e protótipos de grandes empresas como Cisco, HP, NEC, Extreme e Juniper.

3. Trabalhos Relacionados

O RouteFlow alinha-se com a tendência de logicamente centralizar o controle da rede, unificando a informação do estado da rede e desacoplando-a (encaminhamento e configuração) dos elementos de hardware [5].

Com objetivos similares ao RouteFlow, o FIBIUM [19] é uma proposta que combina um roteamento em software, rodando em um computador *commodity*, com um hardware OpenFlow responsável pela maior parte do encaminhamento de pacotes. O foco do trabalho é a otimização do número de entradas instaladas em hardware baseada na observação da popularidade das mesmas. A nossa proposta diferencia-se, principalmente, pelo plano de controle virtualizado em servidores remotos cuja arquitetura permite o mapeamento flexível da infraestrutura de rede programável.

Outro trabalho que também visa paradigmas avançados e flexíveis sobre roteadores baseados em software é o DROP [3]. O DROP fundamenta-se nas diretrizes do padrão IETF ForCES e tem como foco principal a criação de nós lógicos de rede através da separação entre os elementos de controle e de encaminhamento.

4. Arquitetura RouteFlow

O RouteFlow é uma proposta de roteamento remoto centralizado que visa o desacoplamento entre o plano de encaminhamento e o plano de controle, tornando as re-

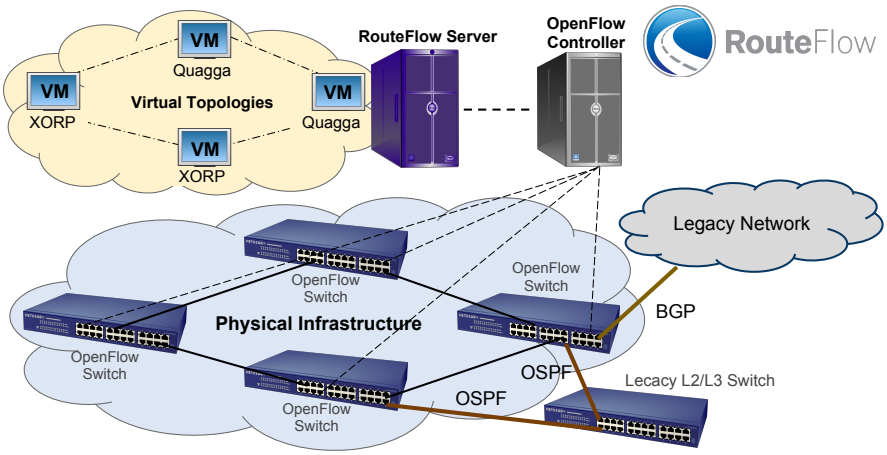


Figura 2. Visão geral do RouteFlow.

des IP mais flexíveis pela facilidade da adição, remoção e especialização de protocolos e algoritmos. No caso, o RouteFlow move a lógica de controle dos equipamentos de rede, até então embarcada, para um controlador de rede remoto cuja responsabilidade é tomar decisões de encaminhamento e programar os elementos do plano de encaminhamento para aplicar essas decisões. Portanto, a base da proposta é a existência de elementos de hardware do plano de encaminhamento que ofereçam interfaces de programação de aplicação (APIs), como por exemplo a proposta na especificação OpenFlow, que permitam tal programação.

Na arquitetura RouteFlow, o plano de controle é formado por protocolos de roteamento IP de código aberto. O plano de roteamento é constituído de máquinas virtuais (MVs) conectadas de forma a representar, através de uma topologia lógica de roteamento, a topologia física da rede controlada conforme mostrado na Figura 2.

Na topologia lógica virtual, cada MV roda uma *engine* de roteamento padrão, ou seja, o mesmo software de controle (protocolos) que estaria embarcado em um roteador. As interfaces das MVs representam as portas do roteador e se conectam a um software *switch* através do qual é possível configurar fluxos e promover as devidas conexões entre as MVs. Ainda por meio dos fluxos pode-se configurar quais pacotes devem permanecer na rede virtual e quais devem ir para o plano de encaminhamento.

Manter os pacotes de protocolos confinados na topologia virtual torna-se interessante para efetuar a separação entre o plano de controle e plano de dados. As decisões tomadas pelas *engines* de rotamento dentro das MVs são enviadas para o controlador, que as instala nos respectivos elementos físicos da rede. Além da instalação de rotas, o controlador faz o gerenciamento das máquinas virtuais bem como a amarração entre elas, isto é, a configuração dos fluxos do *switch* no qual as MVs estão conectadas também é de responsabilidade do controlador.

Apesar de o controle estar fisicamente centralizado, ele continua distribuído logicamente. Desta forma, não há necessidade de qualquer alteração dos protocolos de roteamento existentes. Além disso, a solução pode tornar-se mais escalável no futuro com o uso de vários servidores de alto desempenho.

4.1. Modos de Operação

A arquitetura RouteFlow promove a efetiva separação entre os planos de controle e encaminhamento. Esta característica combinada com os recursos da tecnologia de virtualização utilizada no plano de controle RouteFlow agregam interessantes funcionalidades ao sistema. Dentre elas pode-se destacar a flexibilidade no mapeamento entre os equipamentos físicos e as instâncias de controle virtuais (MVs), que resulta em três modos de operação com características bem definidas, como ilustrado na Figura 3.

A **separação lógica** reflete o mapeamento (1:1), que representa o espelhamento entre os planos de controle e dados. O modo de **multiplexação** (1:n) permite que múltiplas instâncias de roteamento operem sobre o mesmo equipamento físico, promovendo melhor aproveitamento de recursos (virtualização da rede). Por fim, no modo de **agregação** (m:1) é possível simplificar a engenharia de protocolos de rede através do agrupamento de *switches*, resultando em um único elemento lógico.

4.2. Componentes

Uma visão global dos componentes do RouteFlow pode ser observada na Figura 4. A seguir apresenta-se a descrição de cada um dos componentes.

RouteFlow-Slave: cada MV do plano de controle executa um processo (*daemon*) responsável pelas seguintes funções: i) registro da MV no RF-Server como recurso

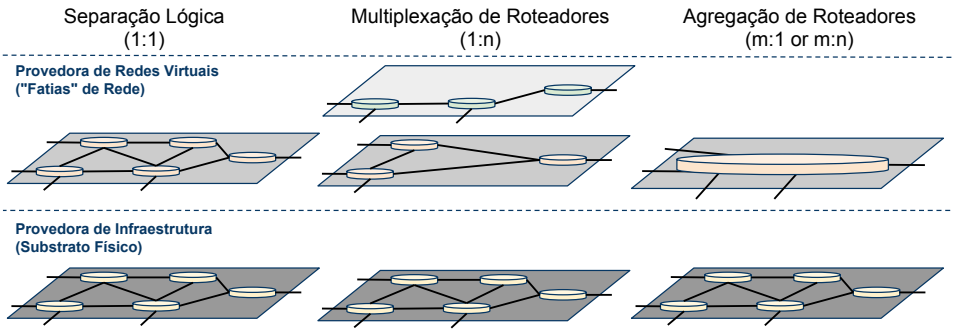


Figura 3. Diferentes variantes de virtualização de redes.

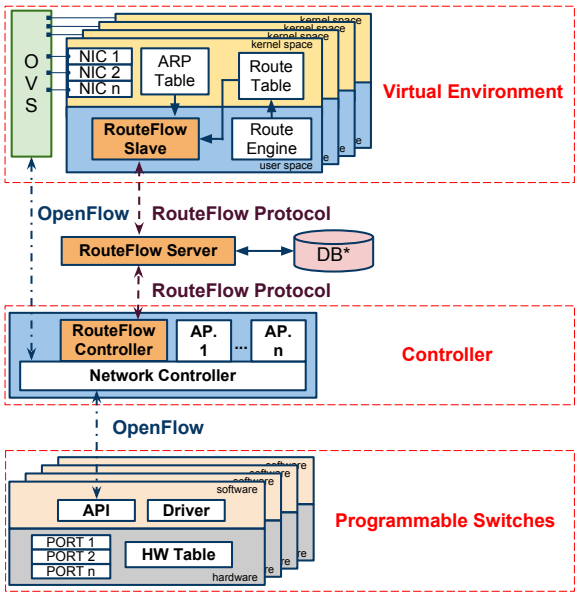


Figura 4. Componentes da solução RouteFlow.

da topologia virtual; ii) gerenciamento das interfaces de rede do sistema (portas do roteador); iii) detecção das atualizações das tabelas ARP e de roteamento do sistema e, iv) conversão de rotas em fluxos a serem instalados no plano de dados por meio da API do OpenFlow.

Pacotes de protocolos e outros, cujos destinos sejam o próprio roteador são encaminhados e recebidos pela MV para processamento (por exemplo, ARP, ICMP, Telnet, SSH, OSPF, RIP, etc.). Esses pacotes chegam às MVs pelas interfaces do sistema para que sejam devidamente tratados. Pacotes como ARP, ICMP são tratados pela pilha TCP/IP do Linux, enquanto os pacotes dos protocolos de controle são utilizados pela *engine* de roteamento para cálculo de rotas. O caminho inverso ocorre do mesmo modo, ou seja, os pacotes injetados pelo Linux, ou pela *engine* de roteamento, nas interfaces são recebidos pelo *software switch* responsável por encaminhar os pacotes para o controlador ou, dependendo da configuração, para outras MVs.

Concomitantemente ao processamento de pacotes da MV, um mecanismo executa a tarefa de verificar as tabelas ARP e ROUTE do sistema em busca de atualizações que devem ser reproduzidas no plano de dados. Quando atualizações são detectadas, as modificações correspondentes são convertidas na instalação ou remoção de entradas da tabela de fluxos atribuída à MV.

RouteFlow-Controller: aplicação que executa sobre o controlador de rede. Ele é a interface entre o componente central do sistema (RF-Server) e os elementos do plano de dados, atuando como um *proxy*. O objetivo principal é isolar o sistema do controlador de rede facilitando a adaptação do RouteFlow para operar com múltiplos controladores. As principais responsabilidades deste componente são: i) registro no controlador de rede para recebimento eventos de entrada de pacotes, conexão/desconexão de *switches* e estado dos enlaces; ii) encaminhamento dos eventos de rede ao RF-Server; iii) instalação/remoção de fluxos nos equipamentos do plano de dados e, iv) envio de pacotes aos planos de dados e de controle, este último através do *software switch*.

Os pacotes que necessitam subir do plano de dados até o plano de controle são recebidos pelo RF-Controller através do evento de entrada de pacotes (*packet-in*). Esse pacote chega com as informações do *switch* remetente e da porta de entrada. O pacote é então armazenado em um *buffer* e um evento de entrada de pacote é enviado ao RF-Server, cuja responsabilidade é resolver o mapeamento entre os elementos dos planos de dados e de controle e devolver ao RF-Controller a informação da porta do *software switch* na qual o pacote deve ser enviado para ser entregue à MV e porta correspondentes. O processo reverso ocorre da mesma forma.

RouteFlow-Server: componente central e de maior complexidade do RouteFlow, pois possui o conhecimento de todo o sistema sendo, portanto, o único elemento capaz de gerenciar a amarração entre o plano de dados e a topologia virtual. As principais responsabilidades do RF-Server são: i) conexão com a aplicação RF-Controller;

ii) processamento de eventos de rede recebidos do RF-Controller; iii) registro de recursos (MVs) disponíveis na topologia virtual, através da conexão com os componentes RF-Slave de cada MV; iv) configuração do *software switch* para construção da topologia lógica da rede; v) gerenciamento do acoplamento entre *switches programáveis* e MVs e, vi) processamento de mensagens de comandos recebidas do RF-Slave.

A amarração das MVs pode ser feita estaticamente através de um arquivo de configuração permitindo, dessa forma, a configuração de uma topologia virtual independente da topologia física. Alternativamente, a configuração pode ser dinâmica com base em um mecanismo de descoberta de topologia governado pelo controlador de rede. Neste último caso, a rede virtual será uma réplica da topologia física. Outra possibilidade consiste em agregar um conjunto de nós físicos como, por exemplo, *switch stacking/trunking*, em uma única MV no plano virtual, ou ter várias *engines* de roteamento atuando sobre um mesmo elemento físico, o que remete ao conceito de roteadores virtuais.

RouteFlow-Protocol: protocolo desenvolvido para a comunicação entre os componentes do RouteFlow, que opera com mensagens no modelo de TLV. Nele estão definidas as mensagens e os comandos básicos para conexão e configuração das MVs, gerenciamento das entradas de roteamento em hardware e, também, troca de informações de eventos de rede.

5. PROTÓTIPO

Com o objetivo de tornar a solução robusta e aplicável em qualquer ambiente, utilizou-se na implementação do protótipo o Quagga [8], uma conhecida *engine* de roteamento *open source* com suporte aos principais protocolos de roteamento (RIP, OSPF, BGP). Poderia-se também utilizar outras pilhas de roteamento como o XORP [21] sem alteração de código.

Existe uma variedade de virtualizadores que podem seguir paradigmas diferenciados. Nesta implementação foi utilizado o QEMU, que é um software livre e permite uma virtualização completa. Neste modelo de virtualização temos a vantagem de um isolamento forte, no entanto o consumo de recursos é mais elevado e apresenta um desempenho inferior quando comparado com um virtualizador no nível do sistema operacional.

Como plataforma para programabilidade remota dos equipamentos de rede utilizou-se a tecnologia OpenFlow na versão 1.0. Uma grande vantagem desse protocolo é a abordagem *multi-vendor*. Isto significa que independentemente do fabricante e do modelo do equipamento que compuser a rede, desde que haja suporte ao protocolo OpenFlow, não serão necessárias mudanças no controlador nem nas aplicações de rede que executam sobre ele.

A nossa infraestrutura do plano de dados é formada por NetFPGAs, que são hardware programáveis com quatro interfaces de rede Gigabit e com módulo OpenFlow. A escolha da NetFPGA se deu pela flexibilidade de programação do plano de encaminhamento e pela taxa de processamento de pacotes em Gigabits.

O componente RF-Slave, *daemon* que roda em cada máquina virtual, foi implementado em C++ e se utiliza de *bash scripts* e chamadas de sistemas para monitorar as atualizações das tabelas ARP e ROUTE do Linux através de um mecanismo de *polling* com intervalo de verificação ajustado em 100 milissegundos.[1] As rotas aprendidas pelo Quagga são convertidas em entradas de fluxos OpenFlow, onde o MAC da interface local e o IP da rede destino se traduzem na regra cuja ação correspondente é a reescrita do SRC_MAC (endereço MAC da interface local), do DST_MAC (endereço MAC do *nexthop*) e, por fim, no encaminhamento para a porta do roteador que se encontra conectada ao próximo nó.[2] Além da tabela de rotas IP, as entradas da tabela ARP também devem ser instaladas no plano de dados, pois é através delas que se torna possível alcançar os nós das redes diretamente conectadas ao roteador. A diferença básica entre uma rota e uma entrada ARP é a máscara, que nesta última será sempre 32 bits (denominado de *exact match*) e, no caso das rotas, dependerá da rede em questão.

A componente RF-Server foi integrado ao RF-Controller e também implementado em C++ como uma aplicação do controlador NOX [16]. Trata-se de um controlador OpenFlow *open source* cujo objetivo é prover uma plataforma simples para programação de aplicações de redes OpenFlow. Com o objetivo principal de avaliar a viabilidade da solução, esse módulo foi desenvolvido apenas com a funcionalidade de criação estática da rede virtual como uma réplica da topologia física. Nesta implementação, onde as conexões entre as MVs não ocorrem de forma dinâmica, em caso de quebra de enlace no plano de encaminhamento, a falha não será reproduzida na topologia virtual. Por essa razão, é necessário o envio do tráfego de controle para o plano de encaminhamento para que o protocolo detecte a queda do enlace e possa recalcular as rotas e se recuperar do problema. Portanto, o tráfego de controle dos protocolos de roteamento (ex. OSPF Hello, LSA) é sempre direcionado às NetFPGAs de modo que os protocolos detectem falhas automaticamente.

[1] Na versão atual, temos implementado um mecanismo orientado a eventos com a utilização da biblioteca RTNetLink do Linux.

[2] No OpenFlow 1.1 existem as ações de decremento do TTL e *checksum update*.

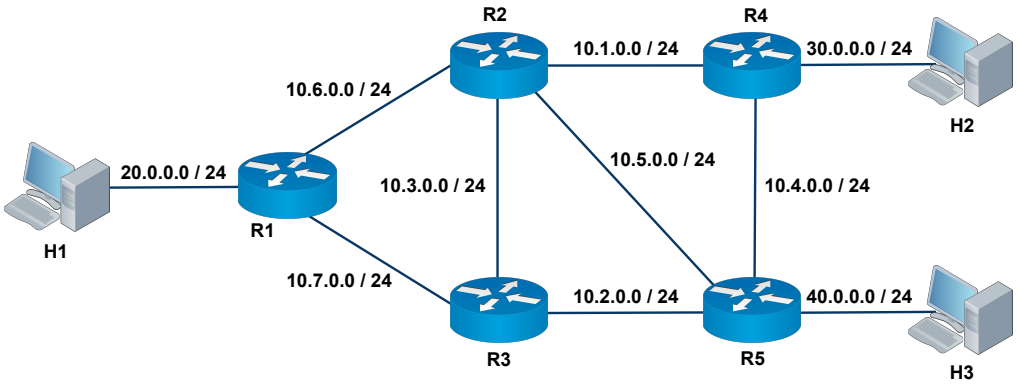


Figura 5. Rede de teste montada com NetFPGAs.

6. AVALIAÇÃO

A avaliação foi elaborada em cima de um ambiente de rede com cinco NetFPGAs como nós de encaminhamento com suporte ao OpenFlow, um servidor QuadCore para virtualização das cinco máquinas virtuais e ainda um servidor para o controlador NOX, conforme a configuração de rede na Figura 5.

Por se tratar de um ambiente real e não simulado e, consequentemente, devido à limitação do número de equipamentos disponíveis com suporte ao OpenFlow, a rede testada apresenta um número reduzido de nós. Entretanto, a quantidade de nós é suficiente para uma avaliação da viabilidade da proposta através da análise detalhada das trocas de mensagens e dos tempos relacionados à convergência da rede em caso de falha como, por exemplo, o tempo de processamento das mensagens RouteFlow e OpenFlow, que não existe em uma arquitetura clássica de roteador com pilha de protocolos embarcada.

6.1. TEMPO DE CONVERGÊNCIA COM TRÁFEGO *line rate*

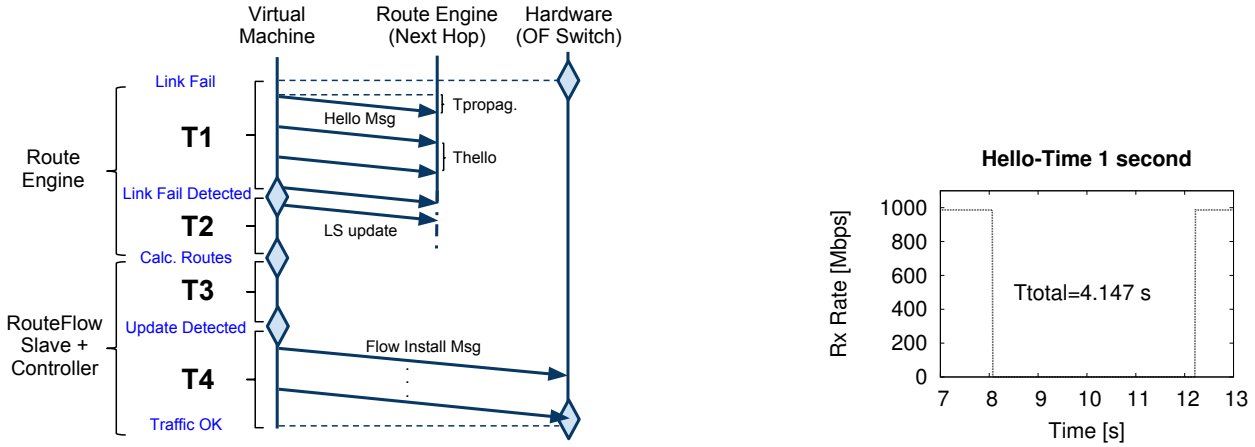
Para os testes de convergência foi utilizado um gerador e analisador de tráfego configurado para transmitir pacotes IP de 1500 bytes de tamanho em ambos os sentidos (*fullduplex*) a uma taxa de 1 Gbps. Desta forma garante-se que esta solução de roteamento remota é capaz de operar com tráfego na taxa de linha uma vez que os pacotes são processados em hardware.

Os tempos de convergência estão ligados ao parâmetro de configuração *hello-time* do OSPF, porém é necessário uma análise mais detalhada para identificarmos o impacto de utilizar uma pilha de protocolos remota ao equipamento. Desta forma podemos fragmentar o tempo de convergência em quatro partes (Figura 6 (a)), a saber: (T1) tempo que o protocolo leva para identificar a falha de *link*; (T2) tempo gasto pelo OSPF com as trocas de mensagens e o cálculo de novas rotas; (T3) tempo necessário para o RouteFlow-Slave detectar as modificações na tabela de roteamento no Linux; e (T4) tempo requerido para o RouteFlow-Slave encaminhar as mensagens de instalação de fluxos para o RouteFlow-Controller e este, através do controlador NOX, instalar os fluxos em hardware. De fato, os dois primeiros tempos (T1 e T2) são inerentes do protocolo de roteamento, portanto o impacto do RouteFlow está claramente presente nos tempos T3 e T4.

O gráfico da Figura 6 (b) mostra o tempo de convergência da rede, utilizando o protocolo OSPFv3 configurado com o *hello-interval* em 1s, após falha de um *link*. Este é o principal parâmetro do OSPF diretamente relacionado ao tempo de detecção de falhas. O *hello-interval* foi configurado para 1 segundo, 5 segundos e 10 segundos, correspondendo aos valores típicos da indústria para enlaces Ethernet [14]. O *dead-interval* usado foi o recomendado de quatro vezes o *hello-interval*.

Com o intuito de fragmentar os tempos mostrados no gráfico da Figura 6 (b), foram feitas análises das mensagens enviadas e recebidas pelo hardware OpenFlow para o controlador NOX. Nos resultados está incluso o tempo gasto pelas mensagens para transitar entre o dispositivo de encaminhamento, o controlador e a MV. O tempo T1 pode ser obtido a partir do intervalo entre o instante em que ocorre a interrupção do tráfego até a primeira mensagem de LS-Update gerada pela *engine* de roteamento ao detectar a falha, em seguida leva T2 + T3 para que o RF-Slave inicie o envio da primeira mensagem de alteração de fluxo e por último o T4 finaliza o processo com o restabelecimento do tráfego. Nos gráficos da Figura 7 e na Tabela 1 apresenta-se os tempos conforme a fragmentação proposta anteriormente, em que cada um deles possui o valor de tempo abaixo do qual se encontra metade dos resultados e o valor de tempo abaixo do qual se encontram 90% dos testes num total de 20 repetições para cada uma das três configurações de *hello-time*.

Conforme os dados apresentados na Tabela 1, o tempo total de convergência é bem próximo do tempo T1, ou seja, o tempo de detecção da falha. Portanto, o tempo



(a) Fluxo de eventos durante convergência.

(b) Tempo total de convergência.

Figura 6. Convergência do OSPF após falha.

Tabela 1. *Tempos de convergência após falha.*

Hello Time	T1 [s]		T2+T3 [s]		T4 [s]		Ttotal [s]	
OSPF	Tmed.	T90%	Tmed.	T90%	Tmed.	T90%	Tmed.	T90%
1 sec.	3.249	3.923	0.360	0.398	0.070	0.123	3.700	4.373
5 sec.	16.713	18.937	0.320	0.389	0.057	0.099	17.135	19.308
10 sec.	36.406	37.846	0.358	0.497	0.042	0.106	36.807	38.266

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

Dada a baixa representatividade dos tempos T3 e T4 sobre o tempo total, consideramos viável a solução RouteFlow dentro do contexto proposto.

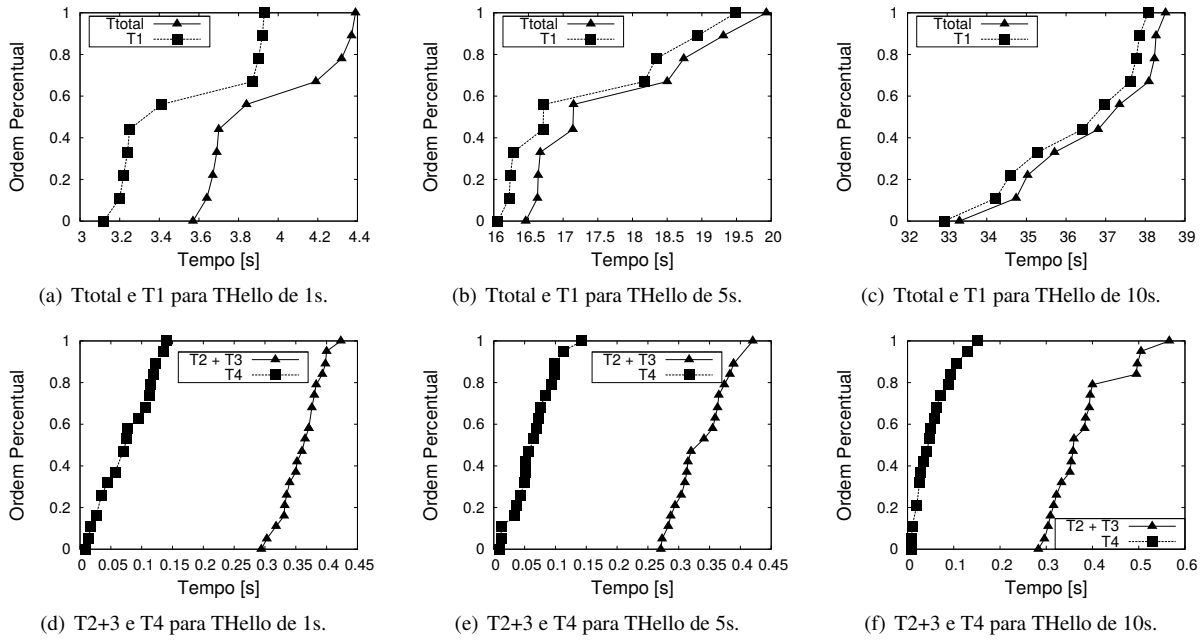
6.2. Encaminhamento em *Slow Path* e *Fast Path*

Outra forma de avaliar o impacto de uma pilha remota de protocolos de roteamento é comparar os tempos para os modos de encaminhamento por *slow path* e *fast path*. *Slow path* refere-se ao encaminhamento lento que ocorre no plano de controle, em software, por exemplo, quando o roteador precisa se comunicar com um nó de uma rede diretamente conectada a ele e ainda não ocorreu o aprendizado do endereço MAC do destino, que é necessário para o encaminhamento em hardware. Este cenário tipicamente ocorre para os nós que entraram na rede ou que ficaram sem comunicação por um longo período, por isso a relevância desta operação nos roteadores é baixa, embora necessária. O *fast path* refere-se ao encaminhamento rápido, em hardware, que ocorre após o aprendizado dos endereços dos nós vizinhos e o preenchimentos das tabelas em hardware. Portanto, o *slow path* ocorre, quando necessário, apenas para o primeiro pacote, a partir do qual será realizado o aprendizado e o restante dos pacotes serão processados na velocidade de linha.

Através deste teste é possível avaliar a diferença de tempo entre um encaminhamento por software e por hardware no caso de um roteador com protocolos embarcados e o RouteFlow, cujo plano de controle é remoto.

Com este objetivo foram realizados testes de PING (ICMP) entre dois *hosts* diretamente conectados a portas distintas de um mesmo roteador, configuradas em redes diferentes. O experimento parte do ponto no qual os *hosts* não têm conhecimento do endereço MAC dos seus *gateways* e o roteador também não contempla ambos *hosts* em suas tabelas. Após o aprendizado dos endereços, obtém-se os valores de *fast path*. Foram utilizados os seguintes equipamentos de camada 3 (*Layer 3 Switches*): CISCO 3560-e Catalyst, Extreme x450-e, CPqD Enterprise (protótipo) e o RouteFlow. Os resultados dos testes estão apresentados na Tabela 2.

Foi observado nos testes com o RouteFlow que ambos



(a) Ttotal e T1 para THello de 1s. (b) Ttotal e T1 para THello de 5s. (c) Ttotal e T1 para THello de 10s.

(d) T2+3 e T4 para THello de 1s. (e) T2+3 e T4 para THello de 5s. (f) T2+3 e T4 para THello de 10s.

Figura 7. Fragmentação do tempo de convergência do OSPF.

Tabela 2. *Tempo de resposta ICMP.*

Equipamento	Slow Path [ms]		Fast Path [ms]	
	Tmed.	T90%	Tmed.	T90%
CISCO 3560-e Catalyst	5.46	7.75	0.100	0.130
Extreme x450-e	11.30	14.00	0.106	0.141
CPqD Enterprise	14.20	17.30	0.101	0.147
RouteFlow	116.00	138.00	0.082	0.119

os pacotes ICMP *request* e ICMP *reply* são encaminhados por software. O motivo deste comportamento está relacionado principalmente ao tempo de *polling* de 100ms para que as atualizações nas tabelas do Linux sejam detectadas. No momento da resposta ICMP os fluxos ainda não estão instalados e o pacote precisa novamente ser direcionado ao controlador para ser encaminhado por software. Portanto, como exemplificado na Seção 6.1, é válido pensar em um tempo consideravelmente menor para este teste como uma otimização da forma com que o RFSlave passa a ter conhecimento das atualizações.

Outro ponto a ser destacado sobre o resultado do *slow path*, consideravelmente, superior quando comparado aos dispositivos com software embarcado, é o desempenho do controlador OpenFlow, NOX, utilizado nos testes, cuja proposta é a de prover uma plataforma simples de desenvolvimento de aplicações de rede sem o compromisso de manter o foco em desempenho. Neste sentido, já foram identificadas possíveis otimizações que devem trazer ganhos significativos no tempo de processamento das mensagens no controlador como, por exemplo, tornar o controlador multi-tarefa de forma a realizar o processamento paralelo das mensagens; outra opção consiste na implementação de mensagens que agreguem mais informação resultando em um menor número de chaveamentos de contextos da aplicação para processar cada pacote recebido.

Em contrapartida, pode-se observar o melhor desempenho de *fast path* do RouteFlow, que é a forma de encaminhamento mais relevante. Este fato pode ser explicado pelo rápido encaminhamento da tabela de fluxos quando comparado ao *longest prefix match*, que é utilizado no encaminhamento IP.

7. DISCUSSÃO

Nesta seção serão discutidos brevemente algumas considerações adicionais.

7.1. ISOLAMENTO

Durante os testes de avaliação foi possível notar variações de desempenho que de modo geral podem estar relacionadas à questão do isolamento entre as instâncias de roteamento virtualizadas. Problemas de isolamento entre MVs já foram identificados na literatura, principalmente na análise do processamento do sistema em qualquer solução montada em cima de ambientes virtuais. No entanto, tal fato pode ser compensado com a utilização de (um *cluster* de) servidores com grande poder de processamento.

7.2. Escalabilidade

O *testbed* esteve limitado ao número de dispositivos disponíveis, o que inviabilizou uma avaliação de escalabilidade do RouteFlow. É clara a necessidade de uma infraestrutura virtual com servidores distribuídos e alta capacidade de processamento para suportar o aumento do tamanho da rede sem qualquer prejuízo à solução. Porém, sabe-se que a capacidade de processamento disponível em soluções comerciais aumenta rapidamente com o passar dos anos (lei de Moore), além de uma redução progressiva dos custos envolvidos, contribuindo assim com a relação custo-eficiência e a escalabilidade do RouteFlow.

Outra questão pertinente é o gargalo observado nas implementações disponíveis do OpenFlow quanto ao tamanho da tabela de fluxos (entre 2 e 4 mil) e a capacidade de instalação de novas entradas por segundo (valor em torno de 100 fluxos/seg.). Entendemos que estas limitações são transitórias e serão contornadas com a maturidade da tecnologia, incluindo o acesso às tabelas L2/L3 de hardware a partir da versão 1.1 do OpenFlow.

7.3. Virtualização de redes

A partir de uma arquitetura baseada na separação entre os planos de controle e dados e no isolamento, tanto do tráfego na infraestrutura física quanto nas instâncias de controle, torna-se possível explorar o conceito de roteamento como serviço [12]. Desta forma podemos ter topologias lógicas independentes sobre uma mesma rede e que rodem protocolos distintos, resultando numa abordagem de virtualização com um aproveitamento melhor dos recursos da infraestrutura que agora pode ser compartilhada com diferentes propósitos, em alinhamento com as propostas de arquiteturas pluralistas [7].

8. Trabalhos Futuros

No site do projeto RouteFlow[3] encontra-se uma lista atualizada dos trabalhos em andamento. Recentemente foi integrado o uso do *software switch* na interconexão das MVs, dando suporte a ambientes dinâmicos e fisicamente distribuídos. Também foi implementado o mecanismo de detecção de atualizações da tabela de roteamento destacado na secão 6.1.

Dentre os próximos passos identificados daremos foco nos testes com o protocolo BGP, que tem sido um dos grandes desafios e causa de problemas nas arquiteturas de roteamento IP. Em paralelo iremos investir nos recursos da tecnologia de virtualização com foco na otimização do ambiente virtual [4]. A utilização de técnicas avançadas para a migração e o gerenciamento do estado (ex: *checkpointing*, *rollback*) das MVs também serão exploradas, assim como os modos de operação de agregação e multiplexação discutidos na Seção 4.1.

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

9. Conclusões

A proposta deste trabalho representa uma abordagem *"scale-out"* para arquiteturas de redes de pacotes. Com o plano de controle externo ao dispositivo de rede, passa a existir maior independência entre este plano e o de encaminhamento, de forma que ambos escalem e evoluam separadamente. Neste sentido o RouteFlow possibilita o surgimento de redes mais baratas e flexíveis mantendo compatibilidade com redes legadas, apoiando uma evolução das redes onde a conectividade IP pode se tornar um *commodity* ofertado em um modelo de plataforma como serviço [12].

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

Referências

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

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

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

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

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

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

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

[8] GNU Quagga Project. http://www.quagga.org.

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

[10] James Hamilton. Networking: The last bastion of mainframe computing. http://perspectives.mvdirona.com/2009/12/19/NetworkingTheLastBastionOfMainframeComputing.aspx, Dec 2009.

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

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

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

[14] J. Moy. OSPF Version 2. RFC 2328, April 1998.

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

[16] NOX. An open-source openflow controller. http://noxrepo.org.

[17] OpenFlow Switch Consortium. Official website. http://www.openflowswitch.org.

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

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

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

[21] The XORP Project. extensible open router platform. http://www.xorp.org.

[22] Vyatta. Series 2500. http://vyatta.com/downloads/datasheets/vyatta_2500_datasheet.pdf.