RouteFlow: Roteamento Commodity Sobre Redes Programáveis

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

[1]Fundação CPqD – Centro de Pesquisa e Desenvolvimento em Telecomunicações
Campinas, SP – Brazil

[2]Faculdade de Enganharia Elétrica e Computação – Unicamp
Campinas, SP – Brazil

{marcelon,esteve,rdenicol,marcosrs}@cpqd.com.br, 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.*

Resumo. *Os roteadores atuais implementam uma arquitetura composta de uma camada de software fechada rodando em um hardware proprietário. 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 por consequência de baixo custo, no entanto estas soluções apresentam baixo desempenho. Motivados pela disponibilidade de uma API aberta para programar o plano de encaminhamento (i.e., OpenFlow), nós propomos o* ***RouteFlow****, uma arquitetura de roteamento IP focada em commodities que combina o alto desempenho de hardwares de prateleira (commodities) com a flexibilidade de uma pilha de roteamento open source rodando (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 é um novo desenho de soluções de roteamento IP com implicações promissoras. A avaliação experimental inicial do nosso protótipo valida a viabilidade da nova arquitetura.*

1. Introdução

Atualmente, uma infraestrutura de redes de pacotes é composta por equipamentos proprietários, fechados e de alto custo, cuja arquitetura básica é concebida da combinação de um *chip* dedicado ao processamento de pacotes, responsável por garantir o alto desempenho, e uma camada de software de controle, que inclui uma imensa pilha de protocolos suportados, visando atender o universo das redes de forma maximizada [Hamilton 2009]. No entanto, torna-se evidente a necessidade de especialização da lógica de controle de acordo com cada tipo e objetivo de rede, principalmente corporativa. No âmbito da pesquisa, essa exigência cresce com a necessidade de experimentações de novos protocolos [Anwer et al. 2010]. A ausência de flexibilidade e o alto custo da infraestrutura vigente são barreiras que dificultam o avanço das redes e a inovação [Rob Sherwood 2010].

Com o objetivo de suprir tais necessidades, existem alternativas como *software routers* (ex: RouteBricks [Dobrescu and et al. 2009], [GNU Quagga Project], [The XORP Project], [Vyatta]), que são soluções que utilizam "hardware de prateleira" (*commodity*) com grande capacidade de processamento (ex.: x86) e onde todo processamento de pacotes é realizado no nível de software, o que torna a proposta bastante flexível, porém acarreta num grande prejuízo de desempenho [Argyraki et al. 2008]. 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) [Spalink et al. 2001]. Propostas recentes [Han et al. 2010] tem explorado também o uso de *graphics processing units* (GPUs) para acelerar o processamento de pacotes nos *software routers*.

Nossa meta é abordar, simultaneamente, as questões de desempenho, flexibilidade e custo, conforme esboçado em [Nascimento et al. 2010]. A estratégia baseia-se no uso da recente tecnologia de *switches* programáveis [McKeown et al. 2008], o que possibilita mover o plano de controle, antes embarcado no equipamento, para um dispositivo externo [Gude and et al. 2008]. A finalidade, no caso, é permitir a programação remota do plano de encaminhamento. O resultado consiste numa 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 comunicam com os equipamentos do plano de dados através de um elemento chamado de controlador, cuja responsabilidade é transportar para o plano de encaminhamento as decisões tomadas pelo plano de controle.

A arquitetura proposta procura atender os seguintes requisitos: (a) integridade e sincronização das informações de configuração trocadas entre o ambiente físico e virtual; (b) mecanismo eficiente para detecção de atualizações no plano de controle, com o objetivo de minimizar o atraso da respectiva implantação no plano de dados; (c) isolamento

máximo entre as instâncias de roteamento (MVs) de modo que os recursos sejam idealmente compartilhados; (d) gerenciamento dinâmico das conexões entre as MVs e, por último; (e) seleção apropriada do tráfego que deve ser mantido no plano de dados e aquele que necessita subir até 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 consegue atender os requisitos e necessidades de redes corporativas e experimentais evidenciando-se, porém, que a sua aplicação é limitada às redes de campus/corporativas. As redes de alta capacidade, como redes de *backbones* e redes de núcleos, têm requisitos de desempenho e escalabilidade que a solução atual proposta não consegue atender.

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

2. Background

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 programadas pelo fabricante.

É coerente pensar que se a arquitetura hoje é composta por duas camadas auto contidas, elas não precisam estar fechadas num 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 [OpenFlow Switch Consortium].

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

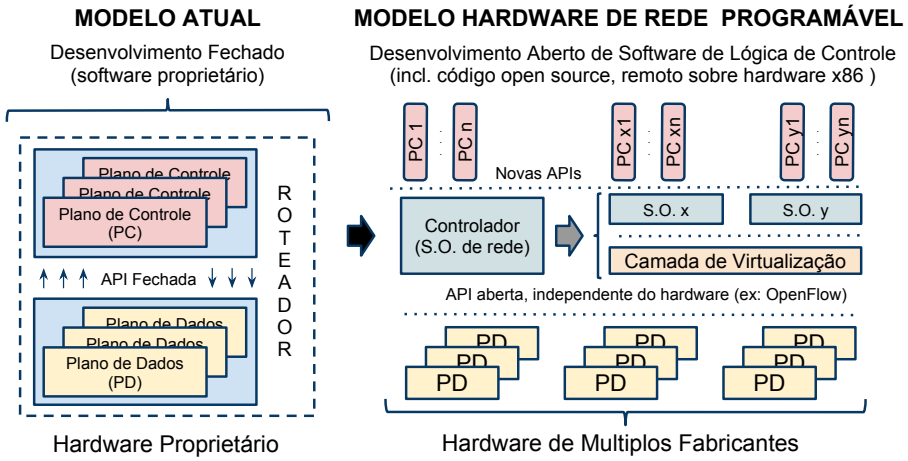


Figura 1. Arquiteturas de roteamento.

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 (e.g., ACL - *Access Control List* em *switches* e roteadores) 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. 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 redes IP mais flexíveis pela facilidade da adição, remoção e especialização de protocolos e algoritmos.

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 proposta RouteFlow, o plano de controle consiste de protocolos de roteamento IP de código aberto. Este plano de roteamento consiste de máquinas virtuais (MVs) conectadas de forma a representar, através de uma topologia lógica de roteamento, a topologia física da rede controlada, conforme 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

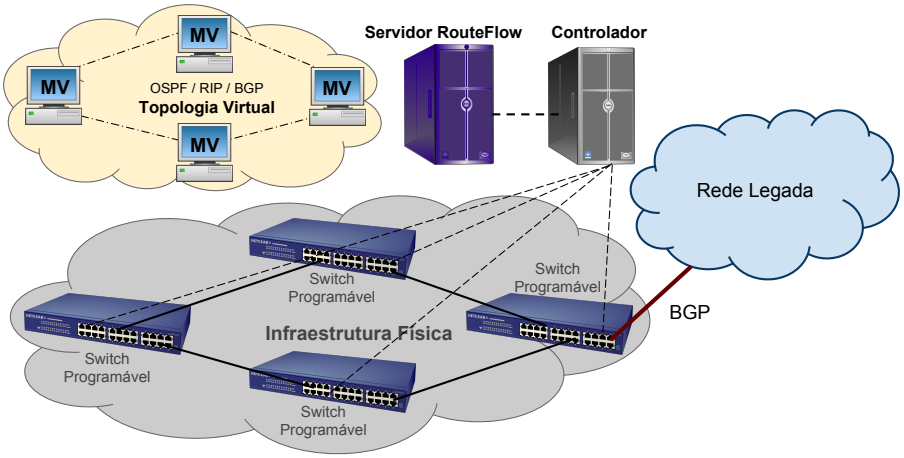


Figura 2. Visão geral do RouteFlow.

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.

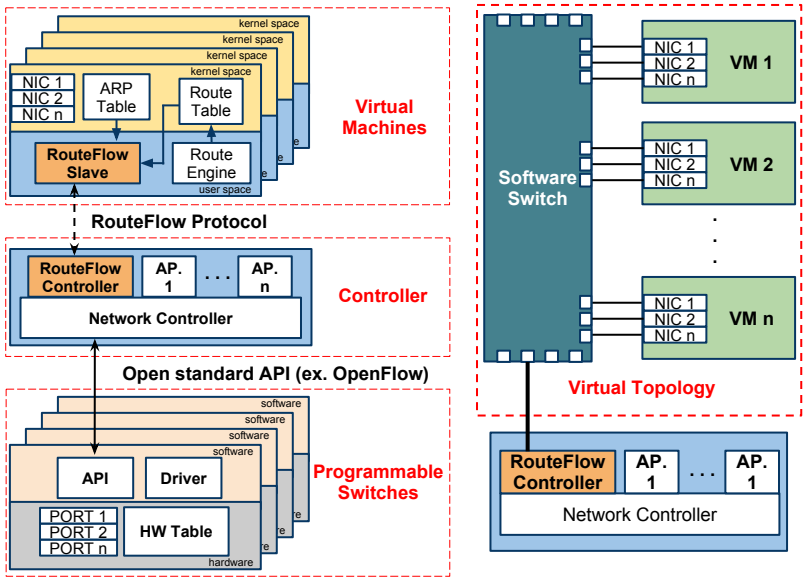
Apesar do controle estar fisicamente centralizado, ele continua distribuído logicamente. Desta forma, não faz-se necessária qualquer alteração dos protocolos de roteamento existentes. Além disso, a solução pode tornar-se mais escalável no futuro com o uso de vários servidores de alto desempenho.

3.1. Componentes

Uma visão global dos componentes do RouteFlow pode ser observada na Figura 3. A seguir apresenta-se a descrição de cada um dos componentes da arquitetura proposta.

RouteFlow-Slave (RF-S): cada MV do plano de controle executa um *daemon* responsável pelas seguintes funções: i) registra a MV no controlador como recurso 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, cujo destino seja o próprio roteador são encaminhados e recebidos pela máquina virtual para processamento (por exemplo, ARP, ICMP, Telnet, SSH, OSPF, RIP, etc.). Estes 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 pacotes de protocolos 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,



(a) Relação entre os componentes. (b) Interconexão das MVs.

Figura 3. Componentes da solução RouteFlow.

eventualmente, para outras MVs dependendo da configuração.

Concomitantemente ao processamento de pacotes da MV, um mecanismo de *polling* 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 fluxos da tabela do elemento de encaminhamento atribuído à MV.

RouteFlow-Controller (RF-C): é o componente central da solução. Como o próprio nome sugere, ele é uma aplicação do controlador, no qual todos os elementos do plano de dados bem como as MVs do plano de controle se conectam. O RF-C é o componente de maior complexidade do RouteFlow pois possui o conhecimento de todo sistema sendo, portanto, o único elemento capaz de gerenciar a amarração do sistema como um todo. As principais responsabilidades do RF-C são: i) registrar-se no controlador de rede para receber eventos de recebimento de pacotes, conexão e desconexão de *switches*; ii) registrar os recursos (MVs) disponíveis na topologia virtual; iii) fazer a configuração do *software switch* para montar a topologia lógica da rede; iv) gerenciar o acoplamento entre *switches programáveis* e máquinas virtuais e; v) instalar/remover fluxos dos equipamentos do plano de dados.

Os pacotes que necessitam subir do plano de dados até o plano de controle são recebidos pelo RF-C através do evento de recebimento de pacotes (*packet-in*). Este pacote chega com as informações do *switch* remetente e da porta de entrada, o que possibilita a entrega na respectiva porta de entrada da MV no plano virtual. O processo reverso ocorre da mesma forma.

A amarração das máquinas virtuais pode ser feita estaticamente, através de um arquivo de configuração, permitindo a configuração de uma topologia independente da 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 (RF-P): protocolo desenvolvido para a comunicação entre os componentes do RouteFlow. Nele, estão definidas as mensagens e os comandos básicos para conexão e configuração das MVs e, também, gerenciamento das entradas de roteamento em hardware. Entre os campos da mensagem padrão estão: identificação do controlador, identificação da MV, tipo da mensagem, comprimento e dado.

4. Protótipo

Com o objetivo de tornar a solução robusta e aplicável em qualquer ambiente, foi utilizado o Quagga [GNU Quagga Project], uma conhecida *engine* de roteamento *open source* com suporte aos principais protocolos de roteamento (RIP, OSPF, BGP). Ainda poderiam ser utilizadas outras pilhas de roteamento como BIRD [The BIRD Project] e XORP [The XORP Project] sem qualquer necessidade de 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 em 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-S, *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 ao encaminhamento para 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

[1]Desta forma conseguimos manter a independência com a pilha de protocolos, apesar do mecanismo não ser muito eficiente. Na versão atual, temos implementado um mecanismo orientado a eventos com a utilização da biblioteca RTNetLink do Linux.

[2]Na versão 1.1 do OpenFlow será possível também a ação de decremento do TTL e o *checksum update*.

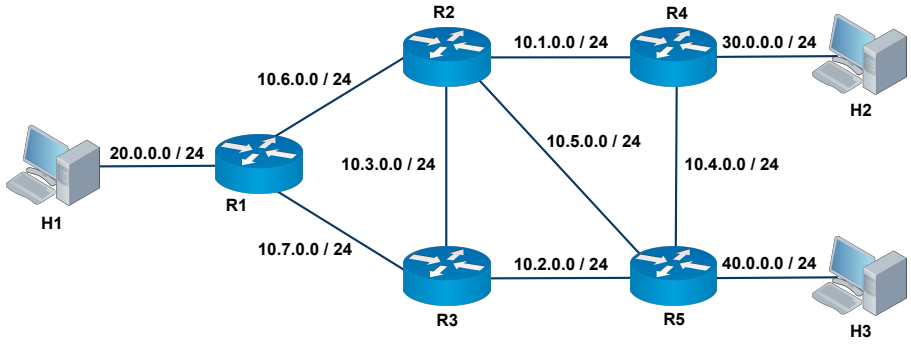


Figura 4. Rede de teste montada com NetFPGAs.

instaladas no plano de dados, pois é através delas que torna-se possível alcançar os nós das redes diretamente conectadas ao roteador. A diferença básica entre uma rota e uma entrada ARP é a máscara, que neste último será sempre 32 (chamamos de *exact match*) e para as rotas irá depender da rede em questão.

O RF-C foi também implementado em C++ como uma aplicação do controlador NOX [NOX]. Este é um controlador OpenFlow *open source* cujo objetivo é prover uma plataforma simples para programação de aplicações de redes OpenFlow. Para o protótipo e com principal objetivo de avaliar a viabilidade da solução, este 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 *link* no plano de encaminhamento a falha não seria reproduzida na topologia virtual. Por essa razão, torna-se necessário o envio do tráfego de controle para o plano de encaminhamento, para que o protocolo detecte a queda de *link* 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.

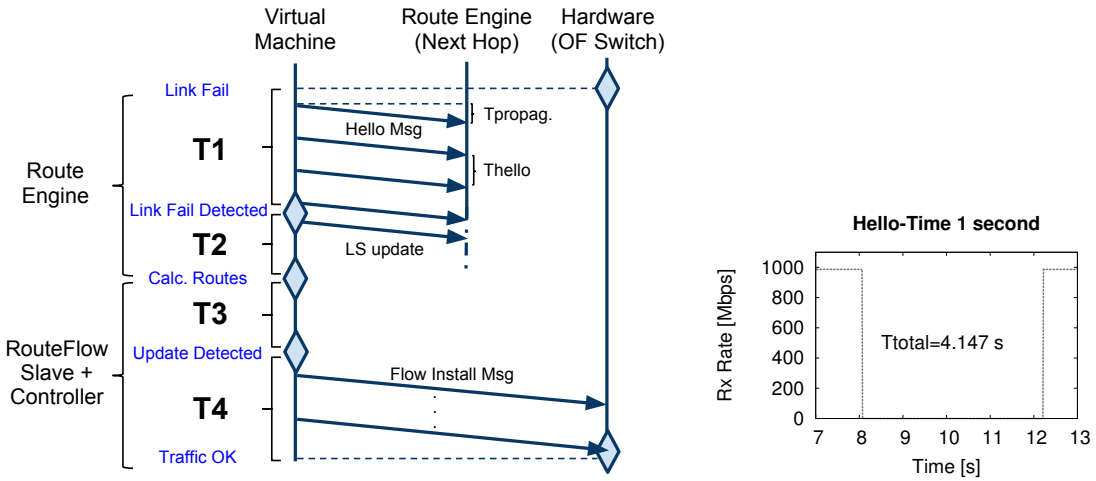
5. Avaliação

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. A configuração de rede utilizada pode ser observada na Figura 4.

Devido ao número de equipamentos disponíveis com suporte ao OpenFlow, por se tratar de um ambiente real e não simulado, a rede testada apresenta um número limitado de nós. Entretanto a quantidade de nós é suficiente para uma avaliação de 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.

5.1. 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 (*full-duplex*) a uma taxa de 1 Gbps. Desta forma garantimos que esta solução de roteamento



(a) Fluxo de eventos durante convergência. (b) Exemplo de tempo de convergência.

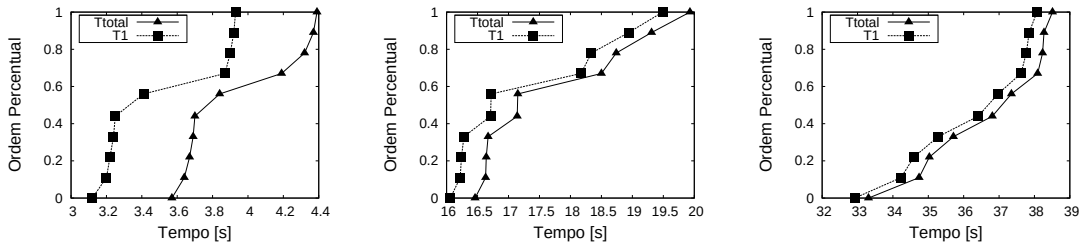
Figura 5. Convergência do OSPF após falha.

remota sobre uma rede programável é capaz de operar com tráfego em *line rate* uma vez que os pacotes são processados em hardware.

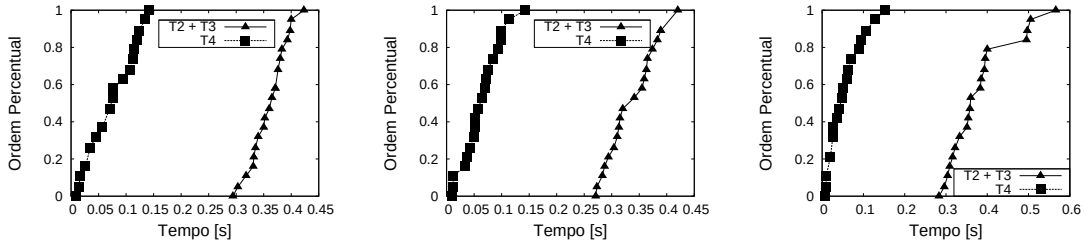
Os tempos de convergência estão ligados ao parâmetro de configuração *hello-time* do OSPF, porém é necessário uma análise mais detalhada para identificarmos o impacto de utilizar uma pilha de protocolos remota ao equipamento. Desta forma podemos fragmentar o tempo de convergência em quatro partes (Figura 5(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 5(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 [Moy 1998]. 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 5(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 6 e na Tabela 1 apresenta-se os tempos conforme a fragmentação



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



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

Figura 6. Fragmentação do tempo de convergência do OSPF.

Tabela 1. *Tempos de convergência após falha.*

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

proposta anteriormente, em que cada um deles possui o valor de tempo abaixo do qual se encontram metade dos resultados e o valor de tempo abaixo do qual se encontram 90% dos testes num total de 20 repetições para cada uma das três configurações de *hello-time*.

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

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

Tabela 2. *Tempo de resposta ICMP.*

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

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

Uma 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, numa transmissão de dados, o *slow path* ocorre, quando necessário, apenas para o primeiro pacote, a partir do qual será realizado o aprendizado e o restante dos pacotes serão processados em *line rate*.

Através deste teste é possível avaliar a diferença de tempo entre um encaminhamento por software e por hardware de um roteador com protocolos embarcado e do 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. Partindo do ponto no qual os *hosts* não têm conhecimento do endereço MAC dos seus *gateways* e o roteador também não contempla ambos *hosts* em suas tabelas. Após o aprendizado dos endereços são obtidos os valores de *fast path*. Foram utilizados os seguintes *switches layer 3*: CISCO 3560-e Catalyst, Extreme x450-e, CPqD Enterprise (protótipo) e o RouteFlow. Os resultados dos testes estão apresentados na Tabela 2.

Foi observado nos testes com o RouteFlow que tanto o primeiro pacote ICMP *request* quanto o *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 é válido pensar em um tempo consideravelmente menor para este teste com uma otimização da forma com que o RF-Slave passa a ter conhecimento das atualizações, como exemplificado na seção 5.1.

Outro ponto a ser destacado sobre o resultado de *slow path*, consideravelmente, superior quando comparado aos dispositivos com software embarcado é o desempenho do controlador OpenFlow, NOX, utilizado nos testes, cuja proposta é a de prover uma plata-

forma 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 e também a 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 nas tabelas de encaminhamento IP.

6. Discussão

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

6.1. 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.

6.2. 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 em 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.

6.3. 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 [Keller and Rexford 2010]. 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 [Fernandes et al. 2010].

7. 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 [Casado et al. 2010].

Com objetivos semelhares ao RouteFlow, o FIBIUM [Sarrar et al. 2010] 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 [Bolla et al. 2009]. O DROP fundamenta-se nas diretrizes do padrão IETF ForCES e tem como foco principal a criação de nós lógicos de rede através da agregação de múltiplos elementos de encaminhamento e uma clara separação entre os elementos de controle e de encaminhamento.

8. 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 [Keller and Rexford 2010].

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. Em trabalhos futuros iremos acrescentar o uso *software switch* na interconexão das MVs, dando suporte à ambientes dinâmicos e fisicamente distribuídos assim como a otimização do ambiente de virtualização [Carlos N. A. Corrêa et al. 2011]. A utilização de técnicas avançadas para a migração e o gerenciamento do estado (ex: *checkpointing*, *rollback*) das VMs também serão exploradas, assim como a agregação e/ou multiplexação dos elementos no plano de controle virtual sobre as contrapartes da infraestrutura física.

Referências

Anwer, M. B., Motiwala, M., Tariq, M. b., and Feamster, N. (2010). Switchblade: a platform for rapid deployment of network protocols on programmable hardware. SIGCOMM '10, pages 183–194, New York, NY, USA. ACM.

Argyraki, K., Baset, S., Chun, B.-G., Fall, K., Iannaccone, G., Knies, A., Kohler, E., Manesh, M., Nedevschi, S., and Ratnasamy, S. (2008). Can software routers scale? PRESTO '08, pages 21–26, New York, NY, USA. ACM.

Bolla, R., Bruschi, R., Lamanna, G., and Ranieri, A. (2009). Drop: An open-source project towards distributed sw router architectures. In *GLOBECOM*, pages 1–6. IEEE.

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

Casado, M., Koponen, T., Ramanathan, R., and Shenker, S. (2010). Virtualizing the network forwarding plane. In *Presto '10*.

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

Fernandes, N., Moreira, M., Moraes, I., Ferraz, L., Couto, R., Carvalho, H., Campista, M., Costa, L., and Duarte, O. (2010). Virtual networks: isolation, performance, and trends. *Annals of Telecommunications*, pages 1–17.

GNU Quagga Project. http://www.quagga.org.

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

Hamilton, J. (2009). Networking: The last bastion of mainframe computing. http://perspectives.mvdirona.com/2009/12/19/ NetworkingTheLastBastionOfMainframeComputing.aspx.

Han, S., Jang, K., Park, K., and Moon, S. (2010). Packetshader: a gpu-accelerated software router. SIGCOMM '10, pages 195–206, New York, NY, USA. ACM.

Keller, E. and Rexford, J. (2010). The 'Platform as a Service' model for networking. In *INM/WREN 10*.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38:69–74.

Moy, J. (1998). OSPF Version 2. RFC 2328.

Nascimento, M. R., Rothenberg, C. E., Salvador, M. R., and Magalhães, M. F. (2010). QuagFlow: partnering Quagga with OpenFlow. *SIGCOMM Comput. Commun. Rev.*, 40:441–442.

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

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

Rob Sherwood, Glen Gibby, K.-K. Y. G. A. M. C. N. M. G. P. (2010). Can the production network be the testbed? OSDI'10, pages 1–14. USENIX Association.

Sarrar, N., Feldmann, A., Uhlig, S., Sherwood, R., and Huang, X. (2010). FIBIUM - towards hardware accelerated software routers. In *EuroView 2010 (poster session)*.

Spalink, T., Karlin, S., Peterson, L., and Gottlieb, Y. (2001). Building a robust software-based router using network processors. *SIGOPS Oper. Syst. Rev.*, 35:216–229.

The BIRD Project. Bird internet routing daemon. http://bird.network.cz.

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

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