KeyFlow: Comutação por Chaves Locais de Fluxos Roteados na Borda via Identificadores Globais

Rafael Emerick Zape de Oliveira[1], **Rômulo Vitoi**[2],
Magnos Martinello[2], **Moisés Renato Nunes Ribeiro**[1]

[1]Laboratório de Telecomunicações (LabTel) – Depto. de Engenharia Elétrica
[2]Laboratório de Pesquisa em Redes e Multimídia (LPRM) – Depto. de Informática

Universidade Federal do Espírito Santo (UFES)
CEP 29075-910 - Vitória - ES - Brasil

rafael.emerick@ufes.br,{ravitoi,magnos}@inf.ufes.br, moises@ele.ufes.br

Abstract. *The large bulk of packets/flows in OpenFlow-based networks will require a level of efficiency in the processing of switching elements that do not compare to the classical methods of query tables flows. Simplifying procedures of lookup in core network is a necessary and ongoing effort to enable the transport of data at high rates and low latency. This paper presents and studies the performance in dataplane an approach for modification of OpenFlow switches named KeyFlow. Its goal is to enable simplified identifiers for global operations flows through the rest of the division between the global label and the local key of the switch.*

Resumo. *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 dispomos ainda nos métodos clássicos de consulta a tabelas de fluxos. Simplificar os procedimentos de lookup do núcleo da rede é um esforço necessário e contínuo para viabilizar o transporte de dados em altas taxas e baixa latência. Este artigo apresenta e estuda o desempenho no plano de dados de uma abordagem para modificação dos comutadores OpenFlow nomeada KeyFlow. Seu objetivo é viabilizar identificadores simplificados para fluxos globais via operações de resto da divisão entre o identificador e a chave local do comutador.*

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. Por exemplo, aplicações MapReduce e sistemas de arquivos frequentemente requerem acesso a nodos remotos antes de prosseguir para as operações de entrada e saída (E/S). Neste caso, 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.

Uma 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 estas 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.

Recentemente, o *OpenFlow* [McKeown et al. 2008] tem sido amplamente 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 novo fluxo, de comunicação com o controlador e seu processamento. Soma-se a isto o fato do contínuo crescimento da capacidade de transmissão por enlaces pesar sobre a capacidade de processamento eletrônico dos pacotes para roteamento ou encaminhamento. Desta forma, iniciativas para o desenvolvimento de técnicas sem manutenção de estados (*stateless*), atingindo um bom compromisso entre eficiência e complexidade, são necessárias.

Este trabalho apresenta uma abordagem de modificação da forma de comutar pacotes, analizando a alteração dos comutadores *OpenFlow* para encaminharem pacotes com base em operações mais simples, em contraste ao encaminhamento tradicional baseado em consultas a tabelas de fluxos. O objetivo é apresentar um protótipo de implementação que viabilize um processo simplificado de *lookup* para os identificadores globais de fluxos via operações de resto da divisão entre o rótulo e a chave local, e com isto obter um indicativo de quanto a solução proposta pode beneficiar a criação de circuitos para redes de experimentação e de produção. Sendo assim, busca-se alternativas para criação de conectividade entre nodos de maneira a atender, no plano de dados, demandas de conexões dinâmicas e flexíveis com o mínimo de ocupação de recursos possível. A proposta é implementada e validada num ambiente de prototipação baseado em *Mininet*, tendo o comutador *OpenFlow* padrão 1.0 como referência de desempenho. A avaliação compara os atrasos *Round-Trip Time* (RTT) (média, desvio padrão, melhor e pior caso) variando-se o número de elementos de comutação no núcleo da rede, assim como o tamanho e a ocupação nas tabelas de fluxo dos comutadores.

A seção que segue apresenta alguns trabalhos relacionados. Na seção 3 é apresentada a abordagem de encaminhamento do protótipo do *KeyFlow*. A seção 4 descreve a metodologia adotada, o ambiente de testes como prova de conceito e apresenta os resultados obtidos da avaliação de desempenho comparativa. Por fim, a seção 5 resume o trabalho com as conclusões e os trabalhos futuros.

2. Trabalhos Relacionados

Para melhorar a eficiência na comutação de pacotes em redes *OpenFlow*, uma estratégia é minimizar as interações dos comutadores com o controlador [Mogul et al. 2010]. Embora as limitações não sejam salientes quando se utilizam regras fixas, quando empregado em redes com muitos elementos e sob demandas muito dinâmicas, uma implementação conhecida pode iniciar apenas poucas centenas de fluxos por segundo. Uma estimativa 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 (*statefull*) dos fluxos ativos é um problema para a escalabilidade das redes definidas por *software* (SDN).

O trabalho de [Bianco et al. 2010] apresenta uma análise de desempenho considerando a ocupação das tabelas de manutenção de estado, em diferentes tipos de comutadores implementados em Linux. O trabalho indica que as implementações mais eficientes deverão dispor de memórias mais rápidas, e consequentemente mais caras e com maior consumo de energia.

No que tange ao plano de controle das redes, o atraso de inserção de regras em muitos comutadores *OpenFlow* disponíveis no mercado gira em torno de 10ms para um único fluxo e chega a 1s para 1000 fluxos [Rotsos et al. 2012]. Isto mostra o quanto as redes definidas por *software* (SDN) podem sofrer com a complexidade de informações no plano de controle. Assim, busca-se no presente trabalho viabilizar o encaminhamento por meio de rótulos mais simples de maneira a facilitar o processo de inserção e atualização das regras de identificação dos fluxos.

Um esquema de roteamento na origem com rótulo único para encaminhamento pensado para redes ópticas de comutação de pacotes [Wessing et al. 2002], serve de base para a presente proposta. O artigo original teve por objetivo a não alteração do cabeçalho nos nodos de núcleo em função da dificuldade de tal operação no domínio óptico. O nosso interesse, entretanto, é explorar especificamente sua forma simples de encaminhamento em cada salto com propósito de redução de latência. Ela nos permite uma mudança conceitual importante de mecanismos convencionais de busca em tabelas para operações elementares sobre o identificador de fluxo. Tais operações podem ser realizadas com poucos ciclos de relógio combinando potencialmente redução na latência e processamento demandado por pacote em trânsito pela porta do comutador *OpenFlow*. A nossa proposta, por ser um mecanismo determinístico, também diferencia-se de técnicas baseadas em Bloom Filter, como por exemplo em [Rothenberg et al. 2010] que tem desempenho probabilístico em função da estrutura de dados utilizada.

Em [Casado et al. 2012] é apresenta 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 se alinha fortemente com as propostas de [Casado et al. 2012], porém, como detalhado nas seção 3, no que tange a solução para rede "fabric" entende-se que o KeyFlow pode propiciar uma maior simplificação em relação ao MPLS, o que favorece a sintetização de hardware mais simples e especializado sem a ocupação de recursos de estado em cada nodo do caminho. Além disso, pela utilização do KeyFlow, é possível a definição dos caminhos unicamente pela informação adicionada na borda, o que pro-

piciará o estabelecimento de conectividade para as redes definidas por software de uma maneira ainda mais flexível.

Outra contribuição da presente proposta consiste em estender o conceito de isolamento do plano de dados em relação ao plano de controle no contexto de *OpenFlow 1.0*, pela assimetria na relação comutador(es)-controlador. Os nodos de núcleo não executam operações de consulta no controlador, uma vez que os nodos de borda se responsabilizam por esta tarefa de modo a prover o roteamento na origem. Como consequência, há uma redução no número de consultas, além de agilizar substancialmente o processo de reconfiguração da rede por eliminação do tempo de instalação de regras no núcleo.

3. Abordagem KeyFlow: Rótulos Globais e Chaves Locais

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 de tal modo que o resultado da operação de resto da divisão (*mod*) entre o rótulo transportado no pacote e a chave local, em cada nodo da rede, resulta na porta de saída no comutador, sem utilizar operações de consulta em tabelas e sem a necessidade de sinalização no núcleo da rede após as definições das chaves locais. 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.

Para geração dos rótulos globais, utiliza-se o Teorema Chinês do Resto (TCR). Para cada caminho (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 destes 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 3.1.

O estabelecimento dos caminhos prévios da rede, viabiliza uma importante vantagem do KeyFlow em relação ao MPLS. O estabelecimento de caminhos nestas redes é feitos por protocolos de sinalização como o RSVP e o LDP, em ambos os casos, é necessário a manutenção de estados em cada nodo. 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 significativa na flexibilidade do atendimento à demandas dinâmicas. 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 aplicação tenderá a ser mais flexivel e mais simples, com possibilidade de suporte de tráfego unicast e multicast entre as bordas da rede.

A Fig. 1 ilustra uma visão geral da integração do KeyFlow a SDN. Nela há o controlador principal atendo as demandas dinâmicas da borda, composta por comutares *OpenFlow* extendidos para o suporte aos rótulos que definirão os caminhos utilizados pelos fluxos na rede "fabric". Nesta, verifica-se o controlador KeyFlow especializado em computar rótulos, conforme apresentado na Seção 3.1.

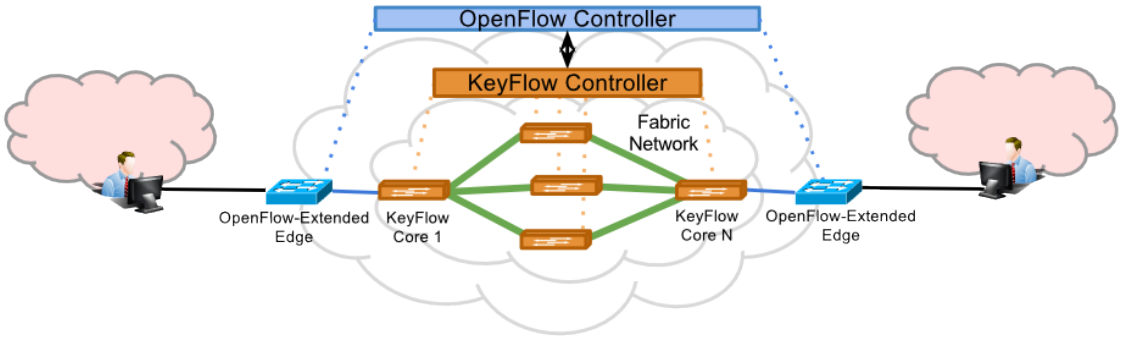


Figura 1. Arquitetura possível para uma rede definida por sofware integrada à solução KeyFlow.

3.1. Teorema Chinês do Resto

Conforme descrito em [Wessing et al. 2002], considere a existência de dois vetores, $\bar{k}$ e $\bar{s}$, respectivamente o conjunto de chaves e o conjunto das portas de saída dos comutadores que compõe um determinado caminho. Se o caminho em questão possui n nodos, tem-se:

(1) $\bar{k} : (k_1, k_2, \ldots, k_n)$ e $\bar{s} : (s_1, s_2, \ldots, s_n)$

O rótulo global r é tal que seja possível restaurar o vetor $\bar{s}$ usando o vetor $\bar{k}$. Para isto, utiliza-se o escalar k,

(2) $k = k_1 \cdot k_2 \cdot k_3 \cdots k_n$

Utilizando-se k e $\bar{k}$, cria-se o vetor $\bar{m}$, 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:

(3) $m_i = \frac{k}{k_i}$, para $i \leqslant n$

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

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

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. 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, verificamos 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:

(5) $r = (s_1 c_1 + s_2 c_2 + s_3 c_3 + \cdots + s_k c_k) \ mod \ k$

Dado o escalar r e o vetor $\bar{k}$, é possível recuperar o vetor $\bar{s}$, por meio da seguinte operação:

(6) $s_i = r \ mod \ k_i$ para $i \leqslant n$

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. Uma análise mais detalhada a respeito do tamanho do rótulo em relação ao tamanho da rede pode ser obtida em [Wessing et al. 2002].

3.2. Implementação do Protótico de Comutador KeyFlow

Pela utilização do protocolo *OpenFlow* modificado no plano de controle da rede KeyFlow pode-se implementar o algoritmo de definição dos rótulos diretamente em uma aplicação no controlador e então distribuir as chaves para o núcleo da rede e os rótulos para as bordas. Com base na visão centralizada oferecida pela arquitetura, pode-se calcular todos os rótulos previamente na fase de inicialização da rede, por exemplo. Em segundo momento, define-se os melhores caminhos a serem repassados aos elementos de borda. Este repasse pode ser de maneira pró-ativa, pela instalação imediata de todas as regras, ou reativa, pela interação do elemento de borda com controlador.

O protótipo foi implementado a partir da implementação de *Stanford* do comutador em software. A diferença entre o protótipo de comutador *KeyFlow* e o padrão de referência da especificação *OpenFlow 1.0* pode ser verificada no fluxograma da Fig. 2.

Os pacotes ao entrarem no comutador *OpenFlow 1.0* têm os cabeçalhos analisados e então inicia-se a busca para verificar se eles pertencem a um determinado fluxo. Isto é feito pela busca sequencial nas tabelas existentes, do menor ao maior índice. Com isto, constata-se que o comutador de referência prioriza as regras mais específicas pela busca inicial na tabela *hash* (*table*=0, máx. 131.070 entradas). Caso não haja uma identificação do fluxo nesta tabela, a consulta é feita na tabela linear (*table*=1, máx. 100 entradas), que possui os registros curingas (com itens caracterizados pelo *, de "não importa"). Se nenhuma regra for encontrada, o pacote é encapsulado e enviado ao controlador da rede, caso contrário, as ações definidas na regra são aplicadas e o pacote é encaminhado para a respectiva porta de saída.

No fluxograma, verifica-se que antes mesmo de qualquer interação com o controlador da rede, de onde já se espera um atraso natural para caracterização do fluxo, há um forte componente aleatório de atraso na pesquisa pela regra nas tabelas de fluxo. Quanto maior for o número destas tabelas, e o respectivo tamanho ocupado, maior será a variação de tempo necessário para se aplicar as ações e enviar o pacote para seu destino. Sendo assim, para um grande número de fluxos ativos no comutador *OpenFlow* é esperada uma maior variação nos tempos de encaminhamento de cada pacote. Vale lembrar que é possível reduzir o tempo de acesso por meio da utilização de memórias mais rápidas, contudo estas memórias têm restrições ligadas ao seu tamanho e ao seu custo, o que implica numa limitação natural para a escalabilidade do encaminhamento.

Ainda na Fig. 2, verifica-se que o comutador *KeyFlow* é um comutador *OpenFlow* instrumentado para comutar pacotes com base em operações de resto da divisão no tratamento de fluxos. Neste comutador, há uma nova variável que controla a chave local do equipamento. Caso ela não esteja instalada (valor padrão zero), o fluxo de tratamento de pacotes segue de maneira idêntica ao padrão *OpenFlow* tradicional. Contudo, se houver a devida definição dessa chave, o fluxo do processamento é deslocado diretamente para a execução da função módulo sobre o campo identificador de vlan em relação à chave local. Uma vez com a chave instalada, o comutador passará a atuar exclusivamente como KeyFlow, não havendo um comportamento híbrido. Com isto, espera-se um tratamento dos pacotes/quadros de uma forma menos aleatória, uma vez que os processos de pesquisa em tabela são eliminados.

O campo identificador de vlan, do quadro Ethernet, foi utilizado no protótipo para

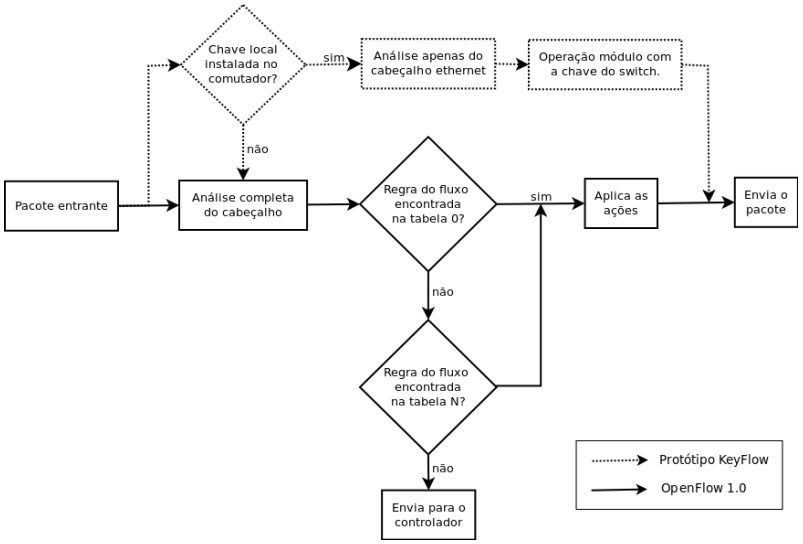


Figura 2. Fluxograma do tratamento de pacotes da especificação *OpenFlow 1.0* [Heller et al. 2009] e do protótipo de comutador KeyFlow.

validação da proposta. Devido ao seu pequeno tamanho (2^{12}, 4096 rótulos) entende-se que este rótulo não seja o ideal para marcação de grande escala. O campo MPLS, conforme apresentado em [Wessing et al. 2002], apresenta boa escalabilidade e possui uma quantidade de bits suficiente para criação de circuitos lógicos com algumas restrições de roteamento.

Alternativamente, pode-se criar um rótulo específico para o encapsulamento dentro do núcleo KeyFlow por meio de alteração mais profunda nos nodos de encaminhamento, dependendo da solução de estabelecimento de caminhos considerada no controlador. Caso mantida a utilização de cabeçalho já existentes, torna-se necessária uma reserva de faixa de rótulos para que circuitos ativos não interfiram em eventuais experimentos de usuários. No tocante a utilização de um novo rótulo, este deve ser totalmente transparente para os pontos finais, sendo utilizado apenas para estabelecimento de caminhos no núcleo da rede. Assim, o núcleo passa a um comportamento similar a um grande comutador "fabric" que interconecta os usuários da borda.

4. Metodologia de Validação

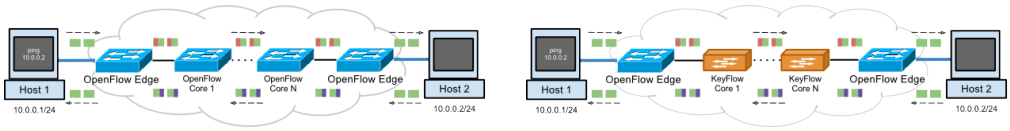
4.1. Cenários Avaliados

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* [1]. Neste ambiente, foi criada uma topologia linear com múltiplos comutadores virtuais conectando duas máquinas reais externas ao ambiente de emulação. 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. Foi realizada uma análise comparativa por meio da realização de testes utilizando os nodos de núcleo com comutadores *OpenFlow*, Fig. 3(a), onde se realizava pesquisa na tabela de fluxos salto a salto, comparativamente com o núcleo composto por

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

comutadores *KeyFlow*, Fig. 3(b), que realizam a operação matemática módulo (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.

Nas extremidades 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.



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

(b) Núcleo com comutação KeyFlow

Figura 3. Topologia utilizada para avaliação do RTT do caminho lógico.

Na topologia da Fig. 3(a), utilizou-se apenas comutadores *OpenFlow* em todo o caminho. A cada salto, fez-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 da vlan, fazendo a definição do circuito virtual, por meio de uma regra devidamente pré-instalada. Na Fig. 3(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*. Para estas sinalizações do plano e 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*.

4.2. Ambiente de Testes

Para realização dos experimentos descritos foram utilizadas três estações. Duas máquinas físicas idênticas realizavam as funções de usuários em rede IP, representados por 'Host 1' e 'Host 2', nas Figs. 3(a) e 3(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 assim, a entrega dos quadros para as máquinas alvos, tornando o serviço de entrega quadros mais próximo de um ambiente real. A Fig. 4 apresenta esquematicamente o ambiente experimental, e a Tabela 1, as configurações de cada máquina utilizada e as suas respectivas versões dos softwares.

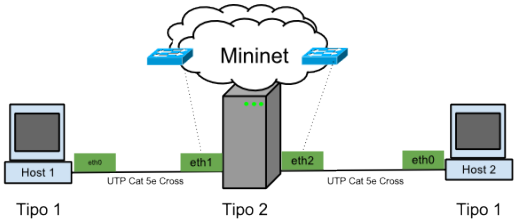


Figura 4. Esquemático do ambiente real de testes.

Tipo	Hardware	Software
1	PC i686 genérico. Interface GigaEthernet.	CentOS 5.0 GNU/Linux 2.6.18-274.el5
2	CPU Intel Xeon 3075 (2Núcleos), 2.66GHz, Cache: L1 32KB, L2 4MB Memória DDR2 800MHz 4GB 2 PCI GigaEthernet 1 FastEthernet (eth0) (acesso remoto não ilustrado) Placa Mae: ProLiant ML110 G5	*Mininet* Ver. 1.0.0 Ubuntu 11.10 GNU/Linux 2.6.38-12-generic i686

Tabela 1. Detalhamento da configuração utiliza no ambiente de testes.

4.3. Avaliação de Desempenho

Os testes foram realizados pela medição do RTT dos pacotes ICMP no caminho entre o 'Host 1' e o 'Host 2', de acordo com os cenários da Fig. 3(a) e Fig. 3(b). Para análise da escalabilidade, variou-se o número de elementos de núcleo de 1 até 15, somando-se os dois elementos da borda, o número total de comutadores no caminho variou de 3 a 17. O tamanho dos pacotes gerados pelo usuário foi de 54 bytes.

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. 5, 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*.

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. Estas regras eram inseridas no final da tabela linear. Além disso, utilizou-se quatro ocupações distintas da tabela *hash*: 0%, 25%, 50%, 75%. Estas porcentagens eram relativas à capacidade máxima da referida tabela.

Os pacotes marcados com o número 5 foram encaminhado para a direita (porta 2, Fig. 5), e com o número 4 eram encaminhados para o sentido contrário. 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*. Apenas os elementos centrais, que receberam a chave 3, determinavam a porta de saída pela função módulo, conforme mostra a Fig. 5.

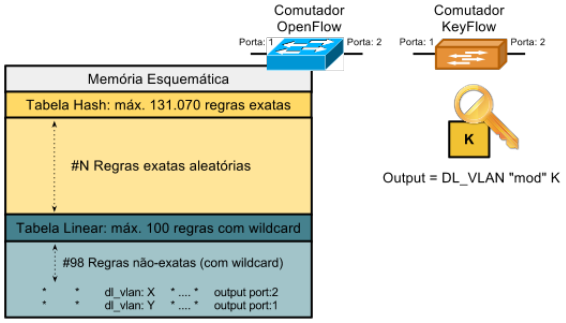


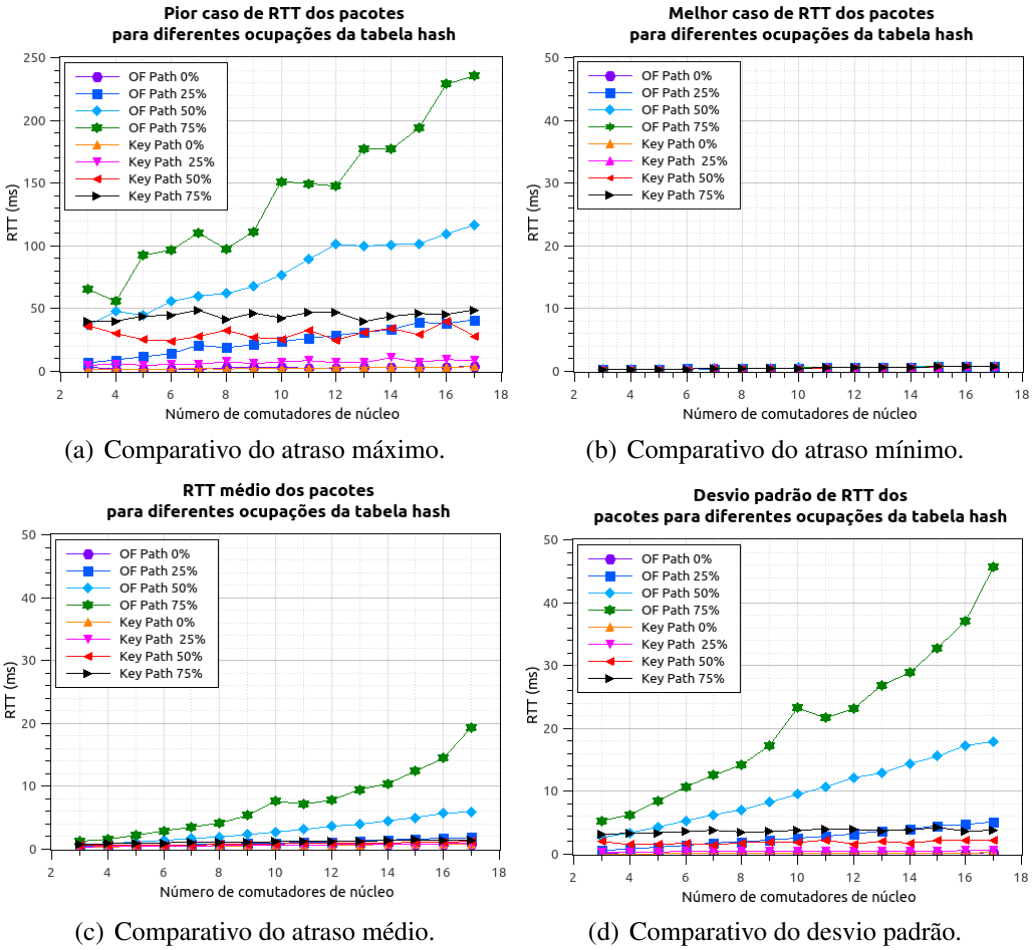
Figura 5. Informações de controle de encaminhamento em cada comutador.

4.3.1. Análise dos atrasos de envio dos pacotes

Para cada teste, foram gerados 5 mil pacotes, 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.

Verifica-se, pelo gráfico da Fig. 6(a), 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 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. 6(b), 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



(a) Comparativo do atraso máximo.

(b) Comparativo do atraso mínimo.

(c) Comparativo do atraso médio.

(d) Comparativo do desvio padrão.

Figura 6. Comparativo entre atrasos.

quanto mais houver memórias rápidas próximas ao sistema responsável pela comutação, mais eficiente será o encaminhamento. Contudo, isto implicará significativamente nos custos e no consumo de energia dos equipamentos.

Para o RTT médio, Fig 6(c), 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.

O desvio padrão, apresentado no gráfico da Fig. 6(d), 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 estas redes se assimilarão às redes de circuitos, e mais uniforme será a comutação dos quadros. Sendo assim, verifica-se que os caminhos baseado 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. 7 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%.

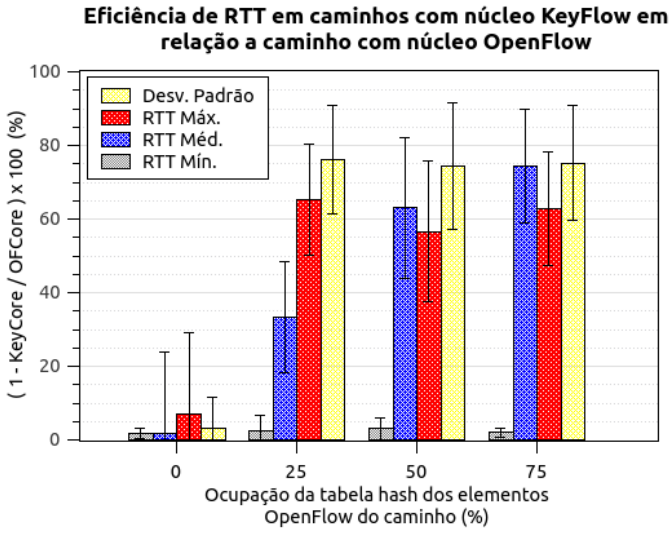


Figura 7. Eficiência na redução do RTT fim-a-fim no caminho com comutadores de núcleo baseados em chaves locais (via *KeyFlow*) em relação a comutadores com busca de tabela de estado na memória (via *OpenFlow*).

4.3.2. Análise do volume de dados nas tabelas de fluxo

Para cada experimento realizado foi criado um arquivo modelo com as regras aleatórias de ocupação da tabela *hash*. Ao se analisar o tamanho destes arquivos, chegou-se aos dados presentes na Tabela 2. A partir dela, verifica-se que uma regra exata, para a especificação *OpenFlow* utilizada possui cerca de 270 bytes (volume de bytes / número de fluxos). A chave implementada utiliza um tipo de dado inteiro de 4 bytes. Sendo assim, verifica-se que o volume total de dados necessários para definição de caminhos baseados no núcleo KeyFlow é praticamente insignificante quando comparado ao tamanho de uma regra exata.

Também pela Tabela 2, constata-se que volume de dados, a serem transferidos no plano de dados, necessário para atualização de tabelas com grande quantidade de regras

é bastante significativo. No caso de reconfigurações da rede, este volume de dados será multiplicado pelo número de comutadores relacionados aos fluxos ativos na rede. Com isso, haverá uma maior exigência sobre os recursos do controlador da rede e representará uma grande sobrecarga no plano de controle. Com a utilização do roteamento nas bordas, apenas estas deverão ser atualizadas em caso de recuperação de falhas, reduzindo assim a demanda de trabalho sobre o plano de controle.

Tipo (máx. de entradas)	Ocupação (%)	Fluxos (un)	Volume (MB)	Volume (Mb)
Linear (100)	100	100	0,004	0,032
Hash (131.070)	0	0	0	0
	25	32.767	8,7	69,6
	50	65.535	18	144
	75	98.302	26	208

Tabela 2. Relação entre ocupação da tabela *hash* e o volume de dados.

5. Conclusões e trabalhos futuros

Este trabalho apresentou uma implementação de uma proposta de comutação baseada em chaves com o objetivo de melhorar a eficiência e a escalabilidade em redes com alto volume de fluxos. Uma prova de conceito é executada, por meio de um protótipo baseado na implementação *OpenFlow 1.0*. Este protótipo utilizou o campo 'vlan id', do quadro Ethernet, para a identificação global dos fluxos e de uma chave local, e instalada em cada comutador de núcleo. Os testes de validação foram criados sobre o ambiente de emulação, com a criação um caminho virtual entre duas estações. Buscou-se, assim, comparar o tempo de encaminhamento para elementos baseados no *KeyFlow*, em relação ao comutador *OpenFlow* convencional.

Os resultados mostraram que o *KeyFlow* apresenta uma significativa redução na variabilidade da entrega dos quadros em aproximadamente 70%. Além disso, a percepção de pior caso de RTT, na visão fim a fim, apresentou uma curva de tendência bem comportada em relação ao aumento do número de saltos e da carga de fluxos ativos na tabela de encaminhamento do comutador.

O *KeyFlow* também mostrou-se como uma alternativa em relação aos comutadores convencionais com memórias rápidas associadas à interface de processamento. É importante ressaltar que estas memórias possuem um alto custo e um elevado consumo de energia. Sendo assim, o *KeyFlow* pode viabilizar a implementação em comutadores com alto desempenho e baixo custo.

Por fim, verificou-se que a identificação dos fluxos pela combinação rótulo global/chave local reduz drasticamente o volume de dados necessários para ativação de fluxos nos comutadores de uma rede *OpenFlow*. Isso facilita também a reconfiguração da rede em caso de falhas, além de reduzir significativamente o tráfego no canal de controle.

Como trabalho futuro, espera-se a implementação do *KeyFlow* em outros comutadores de alto desempenho para realizar os testes de vazão, como em NetFPGA [Naous et al. 2008], e também em soluções embarcadas em dispositivos com menor capacidade de processamento, para validação da solução em ambiente real.

Referências

Bianco, A., Birke, R., Giraudo, L., and Palacin, M. (2010). Openflow switching: Data plane performance. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–5.

Casado, M., Koponen, T., Shenker, S., and Tootoonchian, A. (2012). Fabric: A retrospective on envolving sdn. In *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, Helsink. ACM SIGCOMM.

Heller, B., Pfaff, B., Heller, B., Talayco, D., Erickson, D., Gibb, G., Appenzeller, G., Tourrilhes, J., Pettit, J., Yap, K., Casado, M., Kobayashi, M., McKeown, N., Balland, P., Price, R., Sherwood, R., and Yiakoumis, Y. (2009). In *OpenFlow Switch Specification, Version 1.0.0*.

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(2):69–74.

Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., Curtis, A. R., and Banerjee, S. (2010). Devoflow: cost-effective flow management for high performance enterprise networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 1:1–1:6, New York, NY, USA. ACM.

Naous, J., Erickson, D., Covington, G., A., and Appenzeller, G., M. N. (2008). Implementing an openflow switch on the netfpga platform. In *ANCS '08 Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 1–9.

Rothenberg, C. E., Macapuna, C. A. B., Verdi, F. L., Magalhães, M. F., and Zahemszky, A. (2010). Data center networking with in-packet bloom filters. In *XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*.

Rotsos, C., Sarrar, N., Uhlig, S., Sherwood, R., and Moore, A. W. (2012). Oflops: an open framework for openflow switch evaluation. In *Proceedings of the 13th international conference on Passive and Active Measurement*, PAM'12, pages 85–95, Berlin, Heidelberg. Springer-Verlag.

Wessing, H., Christiansen, H., Fjelde, T., and Dittmann, L. (2002). Novel scheme for packet forwarding without header modifications in optical networks. *Lightwave Technology, Journal of*, 20(8):1277 – 1283.