Tá na Hora: analisando a latência de modificação de tabelas de fluxo em arquiteturas de switches SDN

Fabrício Mazzola[1], Daniel Marcon[1,2], Miguel Neves[1], Marinho Barcellos[1]
[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
[2]Universidade do Vale do Rio dos Sinos (UNISINOS)

Abstract. *Existing knowledge about control plane latency in Software Defined Network (SDN) enabled switches is limited to a narrow set of devices. Recent work either focuses only on instances equipped with TCAM or performs a shallow comparison between a few different designs. However, there is a great diversity of SDN switches nowadays, including distinct implementation designs, use of memories and match-action structures. In this paper, we perform a much richer evaluation, highlighting differences among several distinct switch architectures. Specifically, we measure their performance when executing each key flow table operation (namely insertion, modification and deletion). Unlike previous work, we show that different switch architectures and OpenFlow parameters can significantly influence flow table updating latency. In particular,* (i) *the contrast in latency can scale up to three orders of magnitude; and* (ii) *improper parameterization may increase flow setup times by a factor of 12x for modification and up to 6x for deletion within the same switch.*

Resumo. *O conhecimento existente sobre latência do plano de controle em switches de Redes Definidas por Software (SDN) é limitado a um pequeno conjunto de dispositivos. Trabalhos recentes focam somente em instâncias equipadas com memória TCAM ou realizam uma comparação simplista entre um conjunto limitado de dispositivos. Entretanto, atualmente há uma grande diversidade de switches SDN, incluindo diferentes implementações, uso de memórias e estruturas de casamento (match) de fluxos. Essa heterogeneidade torna desafiador prever o desempenho dos switches ao lidar com requisições do plano de controle. Nesse artigo, apresentamos uma avaliação abrangente que considera diversas arquiteturas de switches SDN e avalia o seu respectivo desempenho ao executar as operações-chave de atualização da tabela de fluxos (inserção, modificação e remoção). Nós mostramos que arquiteturas distintas, juntamente com parâmetros do OpenFlow, podem influenciar significativamente no tempo de atualização das tabelas de fluxos. Em particular,* (i) *a diferença de latência pode chegar a até três ordens de magnitude; e* (ii) *parâmetros do OpenFlow podem aumentar o tempo de configuração dos fluxos em até 12x para modificação e 6x para remoção de regras em um mesmo switch.*

1. Introdução

O encaminhamento generalizado introduzido por Redes Definidas por Software (SDN) [McKeown et al. 2008] permite a gerência de fluxos com granularidade fina. Em cada dispositivo, as tabelas de fluxo são populadas por regras descrevendo diferentes políticas de rede. Tais regras definem como os fluxos serão tratados pelo switch. Uma vez que há poucas entradas nas tabelas de encaminhamento, somente uma pequena quantidade de regras pode ser armazenada ao mesmo tempo. Dada essa limitação, é necessário que regras sejam instaladas, modificadas e removidas das tabelas a taxas potencialmente altas. Por exemplo, em redes de datacenter de maior porte, a taxa de chegada de

novos fluxos é da ordem de milhões por segundo [Roy et al. 2015]). O atraso adicional introduzido pela configuração das tabelas de encaminhamento pode afetar negativamente aplicações de rede sensíveis à latência, cuja adição de 1 ms pode já ser intolerável [Alizadeh et al. 2010].

Switches atuais não estão mais limitados a dispositivos de hardware utilizando memória TCAM. Diferentes tendências (como virtualização de redes [Koponen et al. 2014] e plano de dados programável [Bosshart et al. 2014]) introduziram novas técnicas para desenvolver switches SDN com diferentes implementações (software vs. hardware), uso de memórias (TCAM vs. RAM) e estruturas de casamento (match) de fluxos (tabela hash vs. árvore binária). Adicionalmente, cada arquitetura pode ter peculiaridades intrínsecas à sua implementação como suporte para tipos específicos de casamento de fluxos (máscara curinga vs. casamento exato) ou falta de suporte para prioridade de regras.

Considerando essa heterogeneidade, é desafiador prever o desempenho dos switches ao lidar com requisições do plano de controle. Trabalhos anteriores [He et al. 2015, Kuźniar et al. 2015, Sieber et al. 2017, Lazaris et al. 2014] mostram variabilidade de desempenho em diferentes dispositivos ao processarem e manipularem mensagens do plano de controle. Entretanto, eles focam somente em instâncias equipadas com memória TCAM (Ternary Content-Addressable Memory) ou realizam uma comparação superficial entre poucos dispositivos diferentes. Além disso, como é mostrado nesse artigo, as diferenças de latência entre as diferentes arquiteturas não são compreendidas de forma adequada.

Neste artigo, busca-se fazer uma avaliação compreensiva da latência de atualização das tabelas de fluxo, usando projetos distintos de switches: switch utilizando memória TCAM, switch de software utilizando memória RAM e dispositivo *whitebox* com tabelas de fluxo implementadas ou em TCAM ou tabelas hash em RAM. Para avaliar esses dispositivos, definiu-se uma metodologia robusta e sistemática. Utilizamos uma instância similar de cada projeto para avaliar o tempo adicional introduzido para executar as três operações fundamentais (inserção, modificação e remoção de regras) nas tabelas de fluxo dos dispositivos. Essa métrica reflete exclusivamente o tempo para modificar a tabela do switch, desconsiderando atrasos de propagação e latências de chamadas do sistema. Além disso, diversos fatores são considerados na avaliação, como diferentes tipos de casamento, prioridade de regras e parâmetros do OpenFlow.

Os resultados mostram que diferenças na arquitetura dos dispositivos podem afetar significativamente o desempenho. As principais conclusões são descritas a seguir. Primeiro, a disparidade da latência de inserção de regras entre diferentes arquiteturas pode chegar a até duas ordens de magnitude, dependendo dos padrões de prioridade utilizados. Segundo, a latência de modificação e remoção de regras não é influenciada pelo tipo de casamento e pelo padrão de prioridades empregado. Contudo, ela é profundamente afetada pelo uso do parâmetro *strict* que, quando ausente, pode aumentar o tempo de configuração de fluxos em 10x e 6x para a modificação e remoção, respectivamente. Por fim, o switch de software (Open vSwitch, ou OVS) apresenta comportamentos anômalos ao remover regras, com alta variabilidade e padrões bimodais.

O restante do artigo está organizado da seguinte maneira. A Seção 2 discute os trabalhos relacionados e ressalta a novidade de nosso trabalho. A Seção 3 oferece uma contextualização das arquiteturas de switch existentes, detalhando os dispositivos que empregados nesse trabalho e suas características. Na Seção 4, descrevemos a metodologia utilizada para avaliar os switches de maneira precisa, incluindo ambiente, métricas e configurações gerais das medições. Na Seção 5, por sua vez, apresentamos os experimentos de forma mais específica, os resultados obtidos com os mesmos, e analisamos os porquês.

Finalmente, a Seção 6 mostra considerações finais e perspectivas para trabalhos futuros.

2. Trabalhos relacionados

Trabalhos investigando o desempenho da modificação de tabelas de fluxo podem ser classificados em duas categorias: medições baseadas em switches somente com TCAM e medições de dispositivos de diversas arquiteturas.

Medições somente de TCAM. Chen e Benson [Chen and Benson 2017] avaliam o impacto de switches com TCAM em ações do plano de controle, considerando traços de redes reais e diferentes tipos de aplicações SDN. Entretanto, os dispositivos são avaliados via simulações, utilizando modelos empíricos, o que pode não representar precisamente o comportamento de switches reais. Kuzniar et al. [Kuźniar et al. 2015] não apresentam uma avaliação em profundidade da latência de atualização das tabelas de fluxo. Especificamente, as medições não consideram o desempenho da inserção e remoção de regras individualmente, tornando difícil de compreender o impacto de cada operação no desempenho do plano de controle. He et al. [He et al. 2015] apresentam uma avaliação extensiva do desempenho de switches. Entretanto, a metodologia empregada é incapaz de isolar a latência do plano de controle de potenciais ruídos introduzidos por outros componentes (por exemplo, do gerador de pacotes).

Medições de múltiplas arquiteturas. Huang et al. [Huang et al. 2013] medem o tempo de conclusão de consulta para uma aplicação específica em diferentes switches. Apesar de conclusões interessantes, os autores não exploram a latência de atualização da tabela de fluxos para as arquiteturas apresentadas no artigo. Sieber et al. [Sieber et al. 2017] avaliam somente a latência da operação de inserção de regras para dispositivos com TCAM e RAM, utilizando cenários simplistas de avaliação (como inserção de uma única regra em uma tabela vazia). Lazaris et al. [Lazaris et al. 2014] também realizam medições para duas arquiteturas (switches de TCAM e RAM). No entanto, os autores não consideram a operação de remoção de regras individualmente e não estendem os cenários avaliados para todos os dispositivos mencionados no artigo.

Diferente dos estudos acima, o presente trabalho apresenta uma extensiva avaliação de dispositivos com diferentes arquiteturas. Mais especificamente, (i) é o primeiro a identificar e medir o impacto de parâmetros do OpenFlow no tempo de atualização de tabelas; (ii) se baseia em uma metodologia robusta, que leva em consideração uma gama maior de cenários e fatores; e (iii) considera um conjunto mais abrangente de arquiteturas de switch (hardware utilizando TCAM, software utilizando RAM e híbrido utilizando ou RAM ou TCAM). Nós detalhamos as arquiteturas avaliadas e a metodologia nas seções seguintes.

3. Arquiteturas de Switch

Atualmente, há uma maior diversidade de tecnologias de switches SDN, incluindo projetos de implementação (software vs. hardware), uso de memórias (TCAM vs. RAM) e estruturas de casamento de fluxos (tabela hash vs. árvore binária). A seguir, detalhamos as diferentes arquiteturas existentes e os dispositivos utilizados na avaliação.

3.1. Arquiteturas existentes

Nesta seção, descrevemos três projetos de switches SDN que representam um conjunto abrangente de dispositivos, a saber: switches de hardware utilizando TCAM, de software utilizando RAM e dispositivos híbridos.

Switch de hardware utilizando TCAM. Essa tecnologia realiza uma busca paralela de todas as entradas da tabela de fluxos, fornecendo rápida classificação de pacotes.

Modelo	CPU	RAM	Tamanho da tabela
TCAM	500MHz	512MB	2,5K
HTCAM	533MHz	2GB	2K
HRAM			32K
RAM	3.10GHz	4GB	1-2M

Tabela 1. Especificação dos dispositivos

A busca eficiente contudo demanda de um alto consumo energético, uma vez que todas as células de memória são ativadas ao mesmo tempo para uma única busca na tabela do dispositivo. Além do consumo de energia, o alto custo por bit impede que switches com TCAM possuam tabelas de fluxo muito grandes. Nesse contexto, políticas de rede precisam ser agregadas utilizando regras com máscaras curinga, potencialmente dificultando (ou, em alguns casos, impedindo) o gerenciamento de fluxos com granularidade fina.

Switch de software utilizando RAM. Esses dispositivos são executados, geralmente, em processadores de propósito geral e com grandes quantidades de memória disponíveis. Por um lado, a quantidade de memória permite a especificação de políticas de rede e gerenciamento de fluxos com granularidade fina. Por outro, a falta de hardware dedicado para o encaminhamento de pacotes está associada à menor taxa de transmissão de pacotes, baixa densidade de portas e alto uso de CPU [Honda et al. 2015]. Sua vazão, limitada, e a baixa densidade de portas impedem o uso desses switches em diversos contextos (por exemplo, redes de alto desempenho).

Switch de hardware híbrido. Switches com memória TCAM são mais eficientes dos que os baseados em RAM, porém são limitados em tamanho e podem inviabilizar a aplicação de políticas de rede com granularidade fina. Já os switches de RAM não possuem limite no tamanho de tabela, mas em princípio com desempenho inferior. Dispositivos híbridos combinam o uso de tabelas TCAM e RAM e permitem o uso de tabelas reconfiguráveis. Por exemplo, em um cenário que requer gerenciamento fino dos fluxos, é possível armazenar regras na tabela de RAM caso exceda a capacidade das tabela TCAM. Pela combinação de tecnologias, o desempenho de switches híbridos é menos previsível do que o de demais dispositivos.

3.2. Dispositivos utilizados na avaliação

Esta seção descreve as arquiteturas de switches consideradas, e após trata das diferenças e aspectos peculiares de cada uma.

A principal contribuição deste trabalho é medir a latência do plano de controle em diferentes arquiteturas de switches ao atualizar tabelas de fluxo. Para isso, foram avaliados as seguintes arquiteturas: (i) TCAM (switch de hardware com memória TCAM); (ii) RAM (switch de software com memória RAM); (iii) HTCAM (switch de hardware híbrido utilizando somente TCAM); e (iv) HRAM (switch de hardware híbrido utilizando somente tabela de casamento exato na RAM). Devido à uma limitação do dispositivo híbrido, não foi possível avaliar o modo de operação combinando simultaneamente TCAM e RAM. A Tabela 1 resume as informações sobre os dispositivos considerados.

Uma das arquiteturas avaliadas é um switch TCAM puro (ou seja, buscou-se medir o desempenho da memória). O dispositivo TCAM possui um processador de núcleo único e uma tabela de hardware implementada usando memória TCAM. Essa configuração encaminha todos os pacotes que casem com entradas instaladas na tabela com máxima velocidade. Para avaliar o impacto da tabela TCAM, todas as tabelas virtuais do switch foram desabilitadas durante os experimentos, ou seja, o switch não armazena tabelas em RAM. Essas tabelas de RAM são implementadas por fabricantes, em alguns dispositivos, para aumentar o número de regras armazenadas no switch. Entretanto, o baixo desempe-

nho delas afeta diretamente o tempo de configuração dos fluxos.

Para avaliarmos o dispositivo RAM, foi utilizada uma instância do OVS 2.5.4 LTS em um hospedeiro com processador multi-núcleo, pois é a alternativa mais aceita atualmente. Essa solução é de propósito geral, ou seja, opera em hardware genérico (de prateleira), sem a necessidade de hardware dedicado. Para alcançar alto desempenho utilizando componentes gerais, o OVS faz uso extensivo de cache de regras, mantendo tabelas tanto em espaço de kernel como em espaço de usuário do sistema [Pfaff et al. 2015].

As tabelas OpenFlow são armazenadas em espaço de usuário e servem para popular as tabelas de cache presentes em espaço de kernel. Já as tabelas em kernel servem para encaminhar pacotes de forma mais rápida, evitando o processamento de todo o fluxo em espaço de usuário. A grande desvantagem dessa estratégia é a necessidade de revalidação da cache sempre que uma tabela OpenFlow é alterada. Caso não aconteça essa verificação, mudanças nas tabelas em espaço de usuário não são refletidas na cache e levam a um estado incorreto das políticas de rede. Dependendo das alterações feitas, esse processo pode ser complexo e pode afetar diretamente o tempo de configuração dos fluxos.

O dispositivo híbrido (arquiteturas HTCAM e HRAM) também possui um processador de núcleo único e permite o uso de tabelas de fluxo reconfiguráveis, oferecendo diferentes modos. Cada modo possui tamanhos de tabela diferentes de acordo com a generalidade das entradas utilizadas, incluindo tabelas com entradas para regras com máscaras curinga em TCAM, entradas com casamento exato em tabelas hash em RAM ou até modos projetados para aplicações específicas, como NAT e tráfego IPv6. Entretanto, não é possível combinar dois ou mais modos simultaneamente. Na nossa avaliação, foram analisados dois modos de tabelas reconfiguráveis diferentes: uma única tabela utilizando TCAM e uma única tabela hash utilizando RAM. Estas opções foram adotadas pois representam os extremos (modos mais genéricos) do switch de arquitetura híbrida. Além disso, ao escolher estas alternativas, foi possível comparar o desempenho de dois tipos distintos de memórias e estruturas de casamento.

4. Metodologia

Esta seção descreve a metodologia empregada: ambiente de avaliação (Seção 4.1) e informações sobre o conjunto de fluxos, carga de trabalho e métricas (Seção 4.2).

4.1. Ambiente de avaliação

A avaliação pode ser resumida através de quatro componentes: um *controlador* da rede, que envia mensagens de controle ao switch alvo; um *gerador* de pacotes, que faz transmissões ao switch; o *switch* sendo avaliado; e por fim, um *receptor*, que recebe os pacotes repassados pelo switch.

Topologia. A topologia utilizada é simples porém suficiente para os experimentos em questão (Figura 1). Uma vez que informações tanto do controlador como do receptor são usadas na medição da latência, nós executamos esses dois componentes, juntamente com o gerador de tráfego, no mesmo hospedeiro para evitar o problemas de sincronização fina de relógios. O hospedeiro roda Ubuntu 14.04.3 LTS e executa uma instância do controlador RYU 3.23.2. Monitoramos a carga no hospedeiro de maneira a assegurar que não haveria nem contenção de recursos nem interferências entre os três módulos. O hospedeiro está fisicamente conectado ao switch usando três interfaces de 1 Gbps. O controlador se conecta à porta de controle do switch através da interface `eth0`, enviando requisições para inserir/modificar/remover entradas da tabela de fluxo do switch.

O hospedeiro utiliza as interfaces `eth1` e `eth2` respectivamente para enviar receber e pacotes. Ambas interfaces estão conectadas a portas físicas do switch. Nas avaliações com dispositivo RAM, o switch físico foi substituído pelo hospedeiro com OVS.

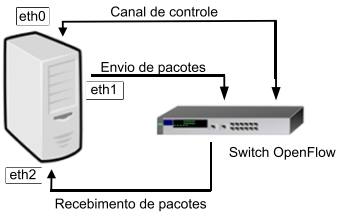


Figura 1. Topologia dos experimentos.

O mesmo possui três interfaces de rede que simulam portas de dados e controle. Ressaltamos que o OVS não está sendo executado em um cenário virtualizado ou emulado, e sim sobre o Linux com seu módulo de kernel habilitado. Para facilitar a reprodução dos resultados, os códigos-fonte encontram-se disponíveis online[1].

4.2. Configuração geral dos experimentos

Nesta seção, são descritas as configurações comuns a todo conjunto de experimentos, como tipos de casamento de regras, carga de trabalho e métricas. A configuração específica utilizada em cada experimento é descrita na Seção 5, antes de apresentar os resultados correspondentes, para facilitar o entendimento.

Tipo de casamento. Dadas as características e limitações dos dispositivos analisados, foram considerados dois conjuntos distintos de entradas de fluxo. O primeiro é identificado por haver regras com casamento exato. Ou seja, os valores dos campos devem ser exatamente iguais, sem curingas: IP origem/destino, número de porta origem/destino, ToS, porta de entrada do switch, EtherType e protocolo de transporte. Nesse caso, prioridades não são necessárias para distinguir diferentes entradas na tabela, porque não haverá casamento com mais de uma regra. Geralmente, regras com valores exatos são armazenadas em RAM (por exemplo, tabelas hash ou árvores de prefixos), pois comportam tabelas de fluxos com maior capacidade e possuem baixo custo [Yu et al. 2016].

O segundo conjunto contém regras cuja maioria dos campos de casamento possuem máscaras curinga. Nos cenários avaliados, regras possuem valores exatos somente para IP origem/destino e EtherType, a fim de permitir a instalação de múltiplas entradas que casem com uma busca. Nesse caso, o uso de prioridades é estritamente necessário, haja visto que o casamento de diferentes regras podem se sobrepor, potencialmente ocasionando inconsistência na aplicação de políticas de rede. Não foi possível avaliar regras com curingas, devido à limitação do dispositivo HRAM, que exige regras com casamento exato. Além disso, a tabela desse dispositivo possui suporte a uma única prioridade, o que impossibilita a avaliação dos cenários com variação de prioridades.

Carga de trabalho. A carga de trabalho é dividida em duas partes: plano de controle e plano de dados. A primeira é utilizada para alterar a tabela de fluxos do switch, e a segunda, para detectar quando as alterações foram de fato realizadas no dispositivo. A carga do plano de controle consiste em uma rajada de requisições (*flow_mod*) emitidas pelo controlador para inserir/modificar/remover regras nas tabelas. As rajadas são enviadas a uma taxa de aproximadamente 6000 requisições/s e são compostas por um único tipo de casamento por vez, de acordo com o experimento sendo executado. São utilizadas rajadas com tamanho variando entre 50 e 1950 requisições, pois esse valor é inferior à capacidade da menor tabela entre todos os switches (Tabela 1).

No plano de dados, por sua vez, a carga de trabalho consiste de um único fluxo partindo da interface `eth1` do hospedeiro, passando pelo switch avaliado, e retornando

[1]https://github.com/fmmazz/SDN-table-update

ao hospedeiro via interface `eth2`. Esse fluxo será ativado antes do envio da rajada de *flow_mods* e permanecerá ativo até que as atualizações da tabela sejam concluídas. Os pacotes do fluxo casam com a regra instalada/modificada mais recentemente. Essa entrada não possui necessariamente a maior prioridade entre as demais regras presentes na tabela. Ao utilizar essa carga de trabalho, o número de mensagens de controle emitidas pelo controlador é moderado e previne saturações tanto no plano de dados como de controle do switch.

O gerador envia segmentos UDP com tempo entre chegadas constante ($\simeq$10 μs) em `eth1` a uma taxa de 50 Mbps. A baixa vazão é proposital, uma vez que desejamos isolar o tempo que leva para o switch processar atualizações nas tabelas de fluxos (nós não avaliamos a capacidade/desempenho do plano de dados dos dispositivos). A confirmação de que a tabela de fluxo foi atualizada é obtida via plano de dados: um analisador de pacotes em software examina a interface `eth2` do hospedeiro. Esta estratégia é necessária para evitar resultados imprecisos [Rotsos et al. 2012, Kuźniar et al. 2015]. Além disso, utilizamos um analisador com precisão de microssegundos e asseguramos via medições adicionais que o mesmo não introduz sobrecarga.

Métricas. A métrica considerada em nossos cenários de avaliação é a latência de atualização da tabela de fluxo do switch, similar a métricas encontradas na literatura [He et al. 2015, Sieber et al. 2017, Lazaris et al. 2014]. Entretanto, tais trabalhos incluem em suas medições os atrasos de transmissão de pacotes. No nosso entender, a métrica deve refletir *apenas* o tempo de modificação de tabela, não incluindo atrasos de propagação e latências de chamadas do sistema (por menores que sejam). Com esse objetivo, medimos o tempo de RTT *(Round Trip Time)* entre o controlador e o switch ($\simeq$0.2 ms) e subtraímos um valor proporcional ao número de *flow_mod*s emitidos pelo controlador dos resultados finais.

Os gráficos na próxima seção mostram o tempo combinado para execução de um lote de operações de inserção, modificação ou remoção de regras. O eixo x representa sempre o tamanho do lote, ou seja, número de operações, ao passo que y, o tempo necessário para efetuar todas operações. Foram executadas pelo menos 5 repetições de cada ponto no eixo x, variando de 50 em 50, mostrando todos os pontos ao invés da média.

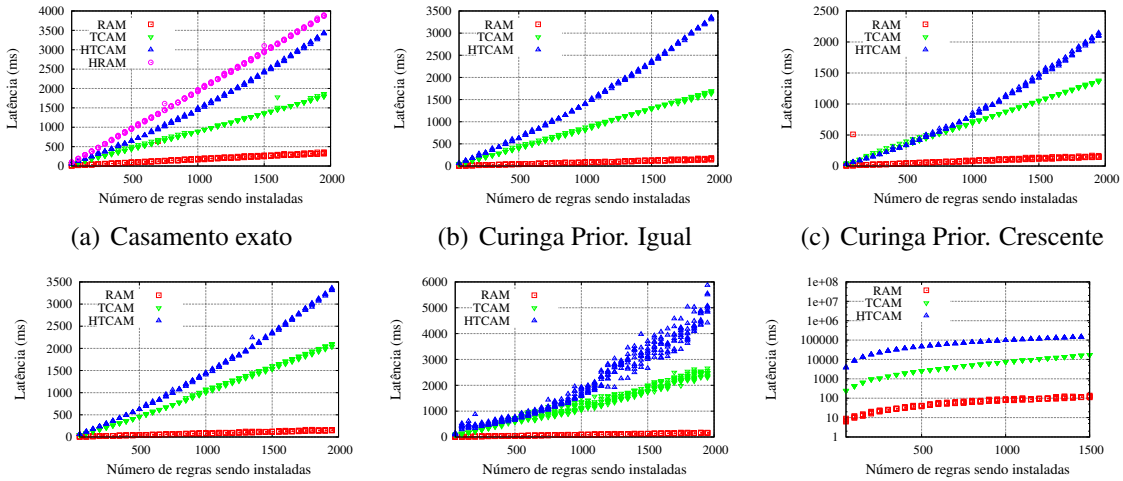
5. Resultados

Nesta seção, descrevemos os experimentos realizados e analisamos os resultados obtidos. Os mesmos estão organizados de acordo com as três operações de atualização avaliadas: inserção (Seção 5.1), modificação (Seção 5.2) e remoção de regras (Seção 5.3).

5.1. Inserção

São investigados dois fatores que intuitivamente influenciam no tempo necessário para inserção de regras nas tabelas de fluxo: tipo de casamento e prioridade das regras. Entendemos que o casamento, exato ou com máscaras curinga, deve afetar o custo de processamento e, consequentemente, o desempenho do switch ao inserir regras. As conclusões são obtidas ao se comparar resultados de um gráfico com casamento exato com resultados de gráficos com variações de casamento curinga. Por sua vez, a prioridade da regra sendo inserida (em relação às demais na tabela) pode exigir o deslocamento de outras entradas previamente instaladas, o que pode influenciar diretamente no desempenho do dispositivo ao configurar novos fluxos.

A seguir descrevemos a metodologia usada no experimento de inserção. Primeiro, a tabela do dispositivo é esvaziada. Em seguida, é instalada uma regra de baixa prioridade, que casa com qualquer pacote e tem como ação o descarte; seu papel é eliminar pacotes que não casam com nenhuma entrada da tabela. Depois, é instalado um lote de R regras



(a) Casamento exato (b) Curinga Prior. Igual (c) Curinga Prior. Crescente

(d) Curinga Prior. Decrescente (e) Curinga Prior. Aleatória (f) Curinga Intercalação Prior.

Figura 2. Inserção de regras.

com IP destino sequencial. A ação associada a essas regras é encaminhar o pacote ao hospedeiro. A latência de inserção é definida como T_d - T_c, onde T_c é a emissão da primeira requisição *flow_mod* da rajada de instalação e T_d o primeiro pacote observado via plano de dados na interface do hospedeiro.

Nos valemos do conjunto de regras de casamento exato para compreender a influência do tipo de associação na inserção de regras nas *quatro* arquiteturas. Todas as regras possuem a mesma prioridade. A Fig. 2(a) apresenta os resultados obtidos. Em resumo, pode-se perceber que: (i) o switch RAM possui o melhor desempenho em operações de inserção; (ii) há uma grande disparidade entre dispositivos com estruturas de correspondência semelhantes (por exemplo, switches HRAM e RAM); (iii) há um contraste significativo na latência de inserção em um switch híbrido dependendo do modo usado (HRAM ou HTCAM); e (iv) a latência de atualização de tabela cresce de maneira linear de acordo com o número de regras sendo inseridas. A seguir, analisamos melhor cada conclusão.

A menor latência em operações de inserção foi observada no switch RAM. Explicamos isso por conta do uso extensivo de cache de regras e ao alto número de otimizações implementadas nas últimas versões do OVS [Pfaff et al. 2015]. O uso de múltiplas threads e a configuração de fluxos em lote aproveita processadores com múltiplos núcleos para balancear a carga de novas requisições de fluxos e diminuir o número de acessos à memória. Haja visto que a maioria dos switches tradicionais possuem processadores específicos com um único núcleo, essas otimizações do OVS não são exploradas. Explicamos as diferenças significativas de latência entre dispositivos com características semelhantes (HRAM e RAM) através da disparidade de poder computacional de cada dispositivo.

O segundo ponto a destacar na Fig. 2(a) é a diferença em latências de inserção no switch híbrido (comparando HRAM e HTCAM). Intuitivamente, HRAM (que possui estrutura de correspondência e memória semelhante ao RAM) teria desempenho superior ao HTCAM. Para nossa surpresa, observamos que o HRAM tem um custo sempre maior do que HTCAM (em torno de 500 ms para 750 entradas ou mais). Ao investigar as causas, descobrimos que ao longo da instalação do lote de regras o HRAM processava algumas requisições, durante um curto período de tempo, enfileirando os *flow_mods* subsequentes. Entendemos que a causa para tal é a implementação não otimizada de tabelas hash, que precisa ser constantemente redimensionada, introduzindo uma sobrecarga adicional para a atualização da tabela do switch.

Prioridade de regras. No segundo conjunto de experimentos, buscamos avaliar o impacto de prioridades de regras nas operações de inserção. Foram usados máscara curinga e quatro padrões de prioridade: igual, crescente, decrescente e aleatória. O primeiro instala uma rajada de regras com prioridades iguais. Os padrões crescente/decrescente empregam um lote de regras com prioridades únicas, seguindo a ordem do respectivo padrão. O padrão de prioridade aleatória utiliza uma rajada de regras com prioridades não-exclusivas variando de 1k a 15k. Esse intervalo de prioridades evita que várias regras possuam prioridades iguais e permite a representação de um cenário mais realístico, em que as regras instaladas não seguem uma ordenação predeterminada (como prioridades crescentes e decrescentes).

As Figuras 2(b), 2(c), 2(d) e 2(e) apresentam os resultados para as diferentes prioridades. Lembramos que a avaliação nesse caso se restringe às três arquiteturas que permitem múltiplas prioridades. Em resumo, observamos que (i) a inserção de regras com prioridades aleatórias (Fig. 2(e)) teve latências comparativamente mais altas (notar que a escala do eixo y no gráfico é mais ampla) e maior variabilidade; (ii) o padrão de prioridade decrescente (Fig. 2(d)) teve latências mais altas do que crescente (Fig. 2(c)); e (iii) a latência de inserção de regras cresce de maneira linear de acordo com o número de regras sendo instaladas para todos os dispositivos independente de prioridade. A seguir, aprofundamos cada um desses pontos.

A inserção de regras com prioridades aleatórias apresentou latência mais alta, contrariando os resultados encontrados por [Lazaris et al. 2014]. Acreditamos que os resultados apresentados naquele estudo não são corretos em função da metodologia simplista adotada pelos autores. Observamos, além disso, alta variabilidade nos resultados para os dispositivos HTCAM e TCAM. Ao investigar[2], detectamos que o aumento de latência média e da sua variabilidade estão associadas à aleatoriedade na prioridade das regras sendo inseridas (entre 1k e 15k). Essa variação gera sobrecarga expressiva para o processo de busca, reordenação e instalação de regras: o reordenamento pode levar ao deslocamento de um número considerável de entradas a cada nova instalação de regra.

A inserção em padrão de prioridade crescente (Fig. 2(c))) teve latências menores do que prioridade decrescente (Fig. 2(d)), em linha com experimento similar em [Lazaris et al. 2014]. Ao comparar os gráficos, percebemos que o desempenho de HTCAM e TCAM diminui consideravelmente ao instalar regras com prioridade decrescente. A latência de inserção para o caso decrescente aumenta substancialmente: para uma rajada de 1950 requisições, em torno de 1500 ms para HTCAM e 700 ms para TCAM. O cenário de prioridade igual (Fig. 2(b)) apresentou resultados semelhantes ao de prioridade decrescente (Fig. 2(d)). Constatamos que as inserções são mais lentas porque as regras precisam ser deslocadas na tabela; conclui-se que a memória de dispositivos pode não estar preparada para instalar regras com padrão decrescente de prioridade. Em contraste, notamos que o dispositivo RAM implementa suas tabelas fluxo como estruturas hash (que não ordenam as entradas por prioridade), portanto suas regras não precisam ser deslocadas e seu desempenho não é afetado.

Intercalação de prioridades. Para melhor compreender o efeito de prioridades e ocupação de tabela na latência, realizamos um terceiro conjunto de experimentos. Neste caso, as regras a serem instaladas possuem prioridades distintas em relação às regras presentes na tabela (novamente, lembramos que o uso de prioridades limita a comparação a três arquiteturas).

Neste conjunto, fizemos dois experimentos. A seguir, descrevemos a metodologia

[2]incluindo interações com engenheiros do fabricante dos switches, que não podemos identificar devido à assinatura de um acordo de confidencialidade.

usada no primeiro. Inicialmente, a tabela do switch é esvaziada e são inseridas 500 regras com prioridade 1000. Depois, o switch recebe uma rajada de R requisições de inserção, contendo regras de 'alta' prioridade, 32000. Avaliamos o tempo necessário considerando rajadas com 50 a 1450 requisições. A ação associada a essas regras é encaminhar o tráfego ao hospedeiro. O segundo experimento é similar, porém com prioridades invertidas: regras de baixa prioridade são instaladas em uma tabela já populada por regras de alta prioridade.

A Fig. 2(f) exibe os resultados do primeiro experimento, utilizando escala logarítmica para o eixo y e linear para o eixo x. Observamos que a diferença de latência de atualização da tabela entre os dispositivos é de pelo menos uma ordem de magnitude, podendo chegar a três, com um lote de 1000 requisições (HTCAM e RAM). A latência para o dispositivo HTCAM ultrapassou 10 segundos mesmo com uma rajada de pequeno tamanho (150 requisições). A causa das latências aumentadas parece ser a sobrecarga excessiva ao processar, reordenar e inserir as atualizações nas tabelas em taxa de linha. Percebemos que a gerência de memória nesses dispositivos é incapaz de lidar com a diversidade de prioridades e instalar as regras de maneira eficiente.

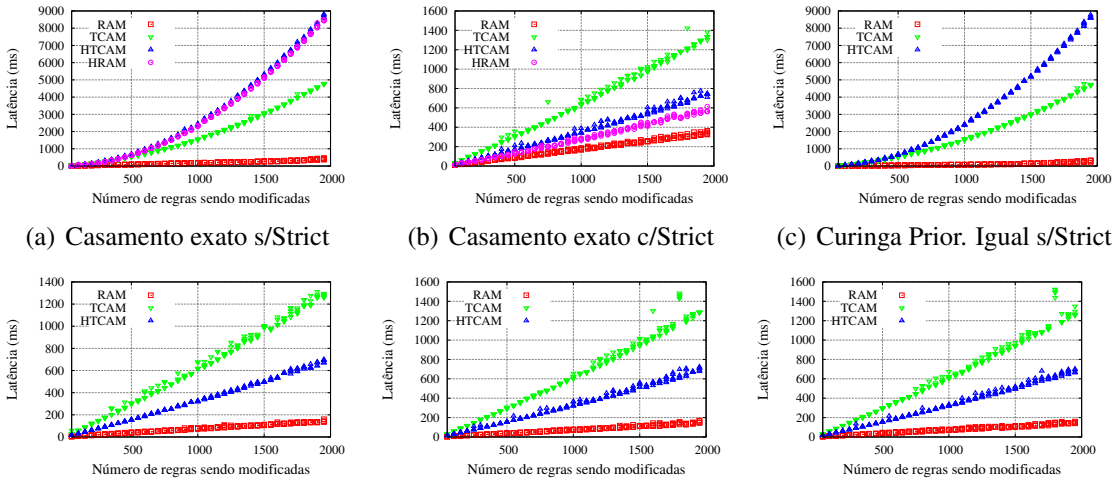
Os resultados do segundo experimento com intercalação de prioridades foram muito similares à inserção com prioridades crescentes, da Fig. 2(c), e por isso, omitidos. A latência de atualização de tabela cresce de maneira linear de acordo com a quantidade de regras sendo instaladas. Os valores obtidos para TCAM e HTCAM mostram que os dispositivos tendem a instalar regras de baixa prioridade nos últimos endereços de memória, em linha com experimento similar em [He et al. 2015]. Essa ordenação é feita para evitar o deslocamento de regras ao inserir entradas de baixa prioridade em uma tabela populada com regras de alta prioridade.

5.2. Modificação

Assim como na inserção, primeiro usamos casamento exato e prioridade fixa para avaliar as operações de modificação nas quatro arquiteturas. Após, avaliamos três arquiteturas considerando o impacto de diferentes prioridades de regras. Diferentemente de trabalhos anteriores, em nossa análise consideramos o impacto do parâmetro *strict* do OpenFlow. Conforme o nome indica, ele força que o casamento seja 'estrito', ou exato, considerando todos os campos e prioridade. Assim, quando *strict* é usado, no máximo uma entrada pode ser modificada por uma requisição, e múltiplas caso contrário.

A seguir, descrevemos a metodologia nos experimentos com operações de modificação. Primeiramente, a tabela é esvaziada e, em seguida, R regras iniciais são instaladas. A ação associada a essas regras é encaminhar o tráfego para uma porta inutilizada do switch, o que irá descartar o tráfego. Depois, é enviada uma rajada com R requisições de modificação, alterando a ação das regras previamente instaladas para encaminhar o tráfego à porta de saída do switch conectada ao hospedeiro. A latência de modificação é definida como T_d - T_c, onde T_c é a emissão da primeira requisição *flow_mod* da rajada de modificação e T_d o primeiro pacote observado via plano de dados na interface do hospedeiro.

O primeiro experimento se baseia em operações com regras de casamento exato e com mesma prioridade. As Figuras 3(a) e 3(b) apresentam os resultados para as quatro arquiteturas, com e sem *strict*. Em resumo, observamos que: (i) comparando os dois gráficos (notar que as escalas em y são bem distintas), o uso do *strict* permite uma diminuição significativa na latência, chegando a 15x para HRAM e até 11x para HTCAM; (ii) ao não utilizar o parâmetro *strict* (Fig. 3(a)), a latência cresce de maneira exponencial de acordo com o número de regras sendo modificadas para os dispositivos TCAM, HTCAM e HRAM e de maneira linear para o dispositivo RAM. Explicamos os resultados a seguir.



(a) Casamento exato s/Strict (b) Casamento exato c/Strict (c) Curinga Prior. Igual s/Strict

(d) Curinga Prior. Igual c/Strict (e) Curinga Prior. Cresc. c/Strict (f) Curinga Prior. Decr. c/Strict

Figura 3. Modificação de regras.

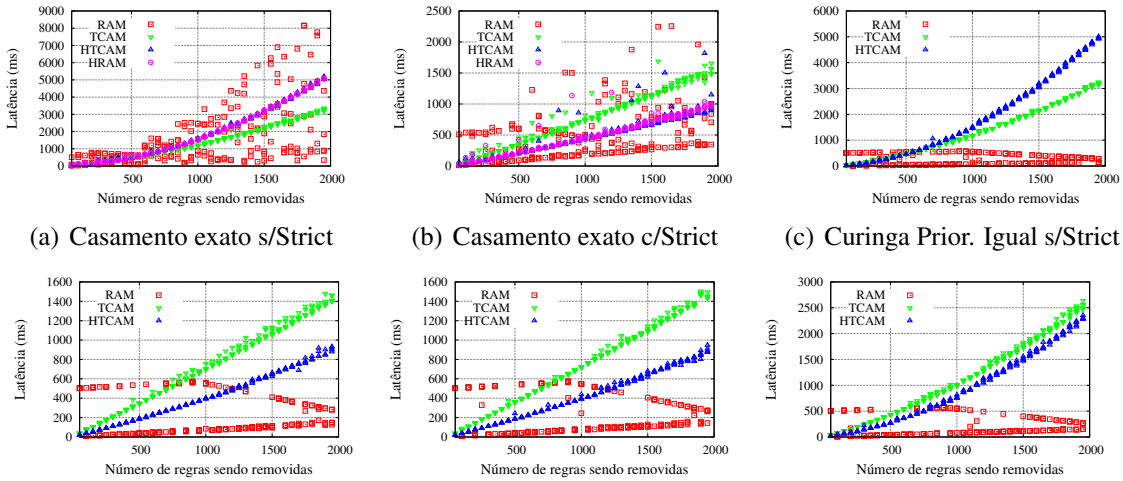
Ao comparar os gráficos das Figuras 3(a) e 3(b), é evidente o impacto do parâmetro *strict* (notar o eixo y). A seguir, explicamos o porquê dessa diferença. Sem ele, uma requisição de modificação demanda uma busca completa em toda a tabela, pois múltiplas entradas podem casar com informações contidas no *flow_mod*. Mesmo para estruturas utilizando hash (HRAM e RAM), há uma busca completa da tabela. A grande disparidade de latência entre os dispositivos HRAM e RAM se deve à diferença de poder computacional, em linha com o que fora comentado anteriormente. O uso de processadores com um único núcleo influencia diretamente no custo de uma busca completa. Em contraste, com *strict*, a operação é encerrada assim que um casamento (estrito) ocorre, o que diminui consideravelmente a latência de atualização da tabela.

Prioridade de regras. Assim como na inserção, usamos um conjunto de experimentos para avaliar o desempenho de operações de modificação mediante diferentes esquemas de prioridade (e regras com máscara curinga). Foram considerados três diferentes padrões de prioridade de regras: iguais, crescente e decrescente. Esses padrões foram utilizados para a instalação das regras iniciais na tabela de fluxos e para o lote de modificações emitido pelo controlador.

As Figuras 3(c), 3(d), 3(e) e 3(f) mostram os resultados. Em resumo, observamos que: (i) o parâmetro *strict* teve o maior impacto ao modificar regras com prioridades distintas; (ii) o padrão de prioridade não afeta no tempo para processar requisições de modificação; e (iii) o tipo de casamento não influencia no tempo de modificação de regras.

O switch RAM apresentou um crescimento linear de acordo com o número de regras sendo modificadas, independentemente do padrão de prioridade e do uso do *strict*. Nos dispositivos HTCAM e TCAM, por sua vez, a não utilização do *strict* implica um crescimento exponencial da latência, de acordo com o número de regras sendo modificadas. Em contraste, com o parâmetro, a latência cresce linearmente de acordo com o número de regras sendo modificadas.

Percebemos que a modificação de regras não é afetada pelo padrão de prioridade e pelo tipo de casamento. Os resultados mostram que para qualquer variação desses fatores, os valores obtidos permaneceram similares e apresentaram comportamentos idênticos. Os autores em [He et al. 2015] indicam que a modificação de regras não é afetada pela sua prioridade. Entretanto, os resultados naquele estudo podem ser questionados, pois aquela avaliação desconsiderou o parâmetro *strict*. Sem o *strict*, a prioridade das regras nunca é examinada e, evidentemente, os resultados serão semelhantes para qualquer padrão de



(a) Casamento exato s/Strict (b) Casamento exato c/Strict (c) Curinga Prior. Igual s/Strict

(d) Curinga Prior. Igual c/Strict (e) Curinga Prior. Incr. c/Strict (f) Curinga Prior. Decr. c/Strict

Figura 4. Remoção de regras.

prioridade utilizado.

5.3. Remoção

Assim como na inserção e modificação, na avaliação das operações de remoção primeiro usamos casamento exato e prioridade fixa para avaliar as operações de remoção nas quatro arquiteturas. Após, avaliamos remoções em três arquiteturas considerando o impacto de diferentes prioridades de regras, com e sem *strict*. Este é o primeiro trabalho a avaliar a influência do uso de *strict* e do tipo de casamento na latência de remoção de regras, bem como o primeiro a analisar o desempenho de um switch de RAM ao tratar dessa operação.

A seguir descrevemos a metodologia utilizada no primeiro conjunto de experimentos, com remoção para conjunto de regras com casamento exato e prioridade única. Primeiramente, a tabela de fluxos é esvaziada e R regras iniciais são inseridas. A ação associada a essas regras é encaminhar o tráfego ao hospedeiro. Em seguida, é feita a remoção das regras iniciais com uma rajada de R requisições de remoção. A latência de remoção é representada por T_d - T_c, onde T_c é a emissão da primeira requisição *flow_mod* da rajada de remoção, e T_d o último pacote observado via plano de dados na interface do hospedeiro. Essa metodologia foi utilizada considerando a limitação do dispositivo HRAM, que impede o uso de uma regra de baixa prioridade responsável por processar pacotes que não casam com nenhuma entrada da tabela.

As Figuras 4(a) e 4(b) apresentam os resultados. Observamos, em resumo, que: (i) o parâmetro *strict* é o aspecto com maior impacto no tempo de remoção de regras, causando uma diminuição na latência de 2x para TCAM e até 5x para HRAM e HTCAM (Fig. 4(b)); e (ii) a latência cresce de maneira linear de acordo com o número de regras sendo removidas para os dispositivos TCAM, HTCAM e HRAM.

Surpreendentemente, houve grande variação nos resultados do dispositivo RAM. Investigamos a questão e discutimos as possíveis causas na lista de desenvolvimento do OVS. Concluímos que o comportamento se deve à arquitetura do OVS: um volume substancial de threads é criado para realizar a tarefa de remoção de um grande número de regras, aumentado o grau de concorrência com as threads responsáveis pela obtenção de estatísticas e revalidação de cache. O comportamento variável na latência foi observado mesmo com 30 repetições redundantes.

Prioridade de regras. Nesse segundo conjunto de experimentos, foram utilizado casamento curinga e três padrões distintos de prioridades nos experimentos: igual, cres-

cente e decrescente. A seguir descrevemos a metodologia utilizada para a remoção de regras com diferentes prioridades. Primeiro, a tabela do switch é esvaziada. Em seguida, é instalada uma regra de baixa prioridade, que casa com qualquer pacote e tem como ação encaminhar o tráfego para o hospedeiro. A seguir, são inseridas R regras iniciais, utilizando um dos padrões de prioridade, com ações para descartar qualquer tráfego correspondente. O gerador de pacotes é acionado e o tráfego casa com uma das regras iniciais, que descarta esses pacotes. Depois, uma rajada de R remoções é emitida pelo controlador, usando o mesmo padrão de prioridades das regras iniciais. É possível observar o momento que as regras são removidas da tabela pois os pacotes param de casar com a última regra inserida e passam a casar com a regra geral de baixa prioridade. A latência é definida como T_d - T_c, onde T_c é a emissão da primeira requisição *flow_mod* da rajada de remoção e T_d o primeiro pacote observado via plano de dados na interface do hospedeiro.

As Figuras 4(c), 4(d), 4(e) e 4(f) apresentam os resultados. Novamente, é clara a importância do parâmetro *strict*, comparando-se os gráficos nas Figuras 4(c) e 4(d) (notar diferenças na escala do eixo y). Independente do padrão de prioridade utilizado para remover regras e o tipo de casamento, o comportamento para TCAM e HTCAM foi bastante similar, aumentando a latência linearmente de acordo com o número de regras sendo removidas. Notamos também que, para esses dispositivos, casamento exato ou curinga não influencia na latência. Além disso, constatamos que nesses switches remover regras com prioridade *decrescente* leva mais tempo quando comparada com os outros dois padrões, até mesmo com *strict*. Estimamos que o comportamento se deve ao deslocamento de regras nas tabelas de TCAM. Como comentado anteriormente, percebemos que a memória TCAM dos dispositivos pode não estar preparada para buscar regras com padrão decrescente de prioridade.

Em relação ao dispositivo RAM, observamos um comportamento bimodal bastante incomum em todos os padrões de prioridade e independentemente do *strict* (vide o que parecem ser duas curvas para o dispositivo RAM nas Figuras 4(c), 4(e), 4(e) e 4(f)). Esse comportamento foi observado mesmo em 60 repetições redundantes. Nós contatamos os desenvolvedores do OVS e as respostas indicam que a variabilidade se deve ao processo de revalidação de cache realizado pelo switch. Sempre que as tabelas OpenFlow são modificadas, o OVS precisa iterar sobre todas as entradas nas tabelas de cache em espaço de kernel para verificar se elas precisam sofrer alguma alteração em suas ações ou se permanecem corretas. Para realizar uma execução otimizada, o OVS agrupa requisições *flow_mod* sucessivas para realizar as operações de uma vez só. Além disso, ele atrasa a revalidação de cache por breves momentos quando duas alterações de tabela ocorrem em rápida sucessão, gerando assim o comportamento bimodal. Em contraste, as operações de remoção com casamento curinga não registrou a alta variabilidade apresentada no casamento exato (comparando RAM nas Figuras 4(a) e 4(b) com o das Figuras 4(c), 4(d), 4(e) e 4(f))). Isso pode ser explicado pela substancial redução de carga do switch quando o mesmo descarta um pacote, ao invés de enquadrar e encaminhar o mesmo por uma porta de saída. Essas conclusões contribuem para o entendimento do desempenho de switches baseados no OVS, e habilitam melhorias em sua implementação.

6. Considerações Finais

Neste trabalho, realizamos uma avaliação bastante completa da latência de atualização de tabelas de fluxos, comparando o desempenho de quatro arquiteturas de switches OpenFlow. Os resultados mostraram que a latência pode variar em até duas ordens de magnitude para inserção de regras e a configuração de parâmetros do OpenFlow pode aumentar o tempo de atualização da tabela de fluxos em até 10x para modificação e 6x para remoção

de regras. Além disso, revelamos comportamentos incomuns (e com alta variabilidade) para a remoção de regras no OVS.

Em trabalhos futuros, pretendemos (i) explorar novas arquiteturas de switches, incluindo dispositivos com plano de dados programáveis; (ii) investigar o desempenho de switches usando cargas de trabalho mesclando diferentes tipos de operações, combinando operações no plano de dados (buscas) e no de controle (alterações em tabelas).

Referências

Alizadeh, M. et al. (2010). Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 63–74, New York, NY, USA. ACM.

Bosshart, P. et al. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

Chen, H. and Benson, T. (2017). The case for making tight control plane latency guarantees in sdn switches. In *Symposium on SDN Research*, SOSR, pages 150–156.

He, K. et al. (2015). Measuring control plane latency in sdn-enabled switches. In *ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR)*.

Honda, M. et al. (2015). mswitch: A highly-scalable, modular software switch. In *ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR.

Huang, D. Y. et al. (2013). High-fidelity switch models for software-defined network emulation. In *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN, pages 43–48, New York, NY, USA. ACM.

Koponen, T. et al. (2014). Network virtualization in multi-tenant datacenters. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX.

Kuźniar, M. et al. (2015). *What You Need to Know About SDN Flow Tables*. Springer International Publishing.

Lazaris, A. et al. (2014). Tango: Simplifying sdn control with automatic switch property inference, abstraction, and optimization. In *ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT.

McKeown, N. et al. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

Pfaff, B. et al. (2015). The design and implementation of open vswitch. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX.

Rotsos, C. et al. (2012). Oflops: An open framework for openflow switch evaluation. In *International Conference on Passive and Active Measurement*, PAM. Springer-Verlag.

Roy, A. et al. (2015). Inside the social network's (datacenter) network. *SIGCOMM Comput. Commun. Rev.*, 45(4):123–137.

Sieber, C. et al. (2017). How fast can you reconfigure your partially deployed SDN network? In *Networking*, pages 1–9. IEEE.

Yu, C. et al. (2016). Characterizing rule compression mechanisms in software-defined networks. In *International Conference on Passive and Active Network Measurement*, pages 302–315. Springer.