Um estudo de qualidade de serviço em ambientes SDN

Diego Martins, Madalena Pereira da Silva, Mario A. R. Dantas
Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD)
Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis, Santa Catarina, Brasil

***Resumo*—O gerenciamento de recursos em *data centers* se torna mais complexo conforme a demanda cresce. O conceito de *Software-Defined Networking* (SDN) facilita o gerenciamento de equipamentos de rede, separando os planos de controle e dados dos dispositivos de rede. E também oferece um ponto centralizado para a aplicação de políticas de rede, como parâmetros de Qualidade de Serviço (QoS). No entanto, o suporte à QoS em *Software-Defined Networking* é limitado e é uma área em que há muito espaço para inovação. Apresentamos o suporte à QoS no controlador SDN Floodlight e as suas limitações para a aplicação de novos métodos de monitoração de parâmetros de QoS.**

***Palavras Chave*— Floodlight, OpenFlow, Qualidade de Serviço (QoS), *Software-Defined Networks*.**

I. Introdução

Redes de computadores presentes em grandes corporações e *data centers* são complexas e difíceis de projetar, implantar e gerenciar. Há vários tipos de equipamentos envolvidos, como NATs (*Network Address Translation*), roteadores, *switches*, *firewalls*, balanceadores de carga e sistemas de detecção de intrusão [1].

Um gerente de redes que precisa aplicar um conjunto de regras em vários dispositivos tem que configurar cada um deles de forma separada, através de interfaces e protocolos diferentes, variando dependendo do fabricante do equipamento. Embora algumas ferramentas de gerência de redes ofereçam um ponto centralizado de configuração, elas ainda operam a nível individual de protocolos, mecanismos e interfaces de configuração.

Diego Martins é bacharel em Ciência da Computação pela Universidade Federal de Santa Catarina (e-mail: diego.martins.d.m@grad.ufsc.br).

Madalena Pereira da Silva é Doutoranda em Engenharia e Gestão do Conhecimento na Universidade Federal de Santa Catarina e Mestre em Ciência da Computação pela Universidade Federal de Santa Catarina (2004). Atualmente é professora da Universidade do Planalto Catarinense no curso de Sistemas de Informação, supervisora de Trabalhos de Conclusão de Curso e orientadora de monografias no Programa de Pós Graduação em Engenharia de Software (e-mail: madalenapereirasilva@gmail.com).

Mario Antonio Ribeiro Dantas é Professor Titular do Departamento de Informática e Estatística (INE), do Programa de Pós-Graduação em Ciência da Computação (PPGCC) e Engenharia e Gestão do Conhecimento (PPGEGC), no Centro Tecnológico (CTC), da Universidade Federal de Santa Catarina (UFSC). Com doutorado em Ciência da Computação pela University of Souhtampton (UK), estágio pós-doutoral e visiting professor na University of Western Ontario (Canada) (e-mail: mario.dantas@ufsc.br).

Um pesquisador que queira testar um novo protocolo de rede em configurações realistas, usando tráfego gerado por usuários e aplicações de produção, acaba enfrentando barreiras

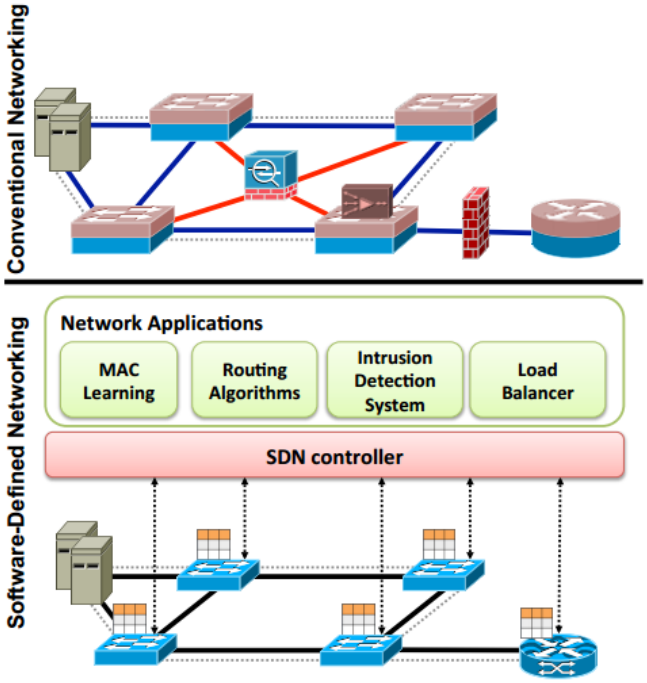


Fig. 1. Comparativo entre redes tradicionais e SDNs.

para testar suas ideias, pois a base instalada de equipamentos e protocolos dá pouco espaço para inovações.

As dificuldades encontradas no projeto, implantação, uso e gerência de arquiteturas tradicionais de redes motivou a concepção de *Software-Defined Networks* (SDNs). Em 2011, seis empresas (Deutsche Telekom, Facebook, Google, Microsoft, Verizon e Yahoo!) criaram uma organização dedicada para o crescimento de SDNs, conhecida como *Open Networking Foundation* (ONF).

A figura 1 [2] mostra um comparativo entre redes tradicionais e redes que usam a abordagem SDN. Em redes tradicionais, *switches* (representados por caixas) e roteadores (representado por um cilindro) possuem planos de controle próprios, demandando uma configuração separada para cada equipamento. Sistemas intermediários como *firewall* (representado como um muro entre um *switch* e o roteador), balanceador de carga (caixa escura sobre o *switch* inferior direito) e detector de intrusão (caixa entre os quatro *switches*)

também precisam ser configurados separadamente. Já em SDNs, os planos de controle não ficam mais nos dispositivos e os sistemas intermediários se tornam aplicações que usam os

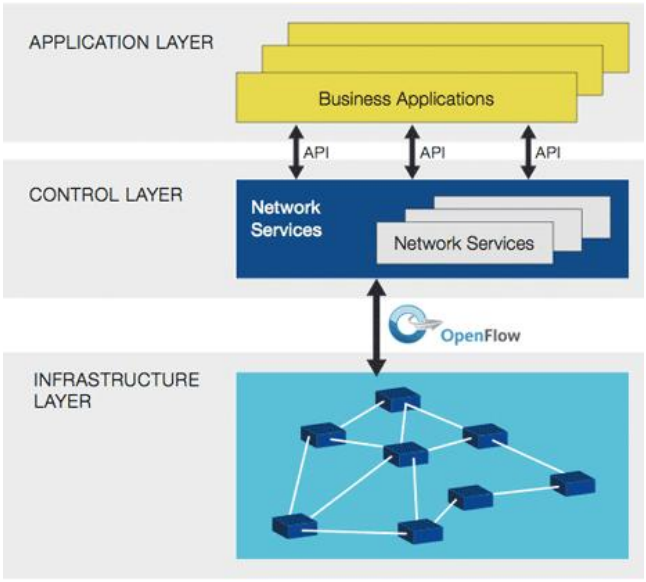


Fig. 2. Arquitetura de uma *Software-Defined Network*.

serviços fornecidos pelo controlador SDN.

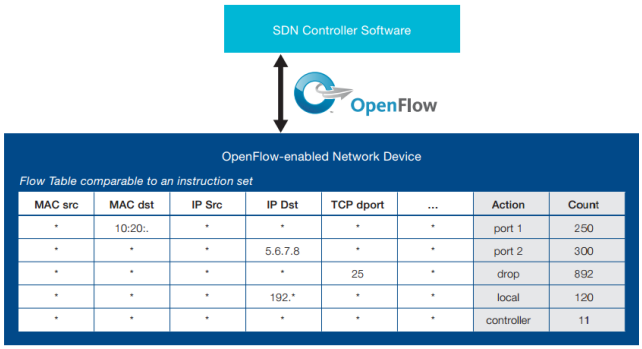
Com a separação dos planos de controle e de dados em SDNs, é possível consolidar o controle de múltiplos elementos dos planos de dados (roteadores, *switches* e outros dispositivos no meio da rede) em apenas uma aplicação que usa uma API bem-definida e padronizada, independente de um fabricante específico. Um dos protocolos que oferece uma API para esse fim de controle é o OpenFlow, abordado na seção 1.2.

A figura 2 [3] apresenta a arquitetura geral de uma SDN. No topo, temos a camada de aplicação, onde ficam as aplicações de negócio que usam recursos da rede através de APIs disponibilizadas pela camada de controle. A camada de controle é responsável em fornecer os serviços que serão usados pelas aplicações e em abstrair os detalhes da rede. Ela também é responsável pela gerência dos dispositivos da camada de infraestrutura, que envolve *switches*, roteadores, *firewalls*, balanceadores de carga, detectores de intrusão, etc.

Este artigo é composto por 4 seções: A seção I apresenta o conceito de *Software-Defined Networks* e o protocolo OpenFlow, um dos alicerces das SDNs. A seção II descreverá as ferramentas utilizadas, que envolvem um simulador de redes chamado Mininet e um controlador SDN feito em Java conhecido como Floodlight. A seção III descreverá como os requisitos de Qualidade de Serviço são associados ao protocolo OpenFlow e ao controlador Floodlight. Na seção IV são apresentadas as conclusões deste trabalho e trabalhos futuros.

A. *Protocolo OpenFlow*

Concebido na Universidade de Stanford e atualmente gerenciado pela *Open Networking Foundation* (ONF), o protocolo OpenFlow é implementado em ambos os lados de uma interface que liga dispositivos de infraestrutura de rede



MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:..	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Fig. 3. Interação entre o controlador SDN e um dispositivo com o protocolo OpenFlow.

com o *software* de controle SDN [4]. A versão mais atual do protocolo é a 1.3, liberada em maio de 2014.

A maioria dos *switches* e roteadores *Ethernet* modernos possui tabelas de fluxo usadas para implementar *firewalls*, qualidade de serviço e coletar estatísticas. Embora a implementação dessas tabelas varie de acordo com cada fabricante, foi possível identificar um conjunto comum de funções que são executadas em vários *switches* e roteadores.

O protocolo usa o conceito de ***fluxos*** (*flows*) para identificar tráfego de rede que corresponda a regras pré-definidas. As regras podem ser programadas pelo *software* de controle SDN dinamicamente ou baseado em estatísticas colhidas a partir do tráfego da rede. Logo, o OpenFlow é baseado em um *switch* Ethernet com uma tabela interna de fluxos e uma interface padronizada para adicionar e remover entradas de fluxos.

Este protocolo permite acessar e manipular diretamente o plano de dados de dispositivos de rede, como *switches* e roteadores, ambos físicos e virtuais (baseadas em *hypervisors*).

A figura 3 [3] mostra a interação entre um controlador SDN e um dispositivo que suporta comandos do OpenFlow. O plano de dados de um *switch* OpenFlow consiste de uma tabela de fluxos e uma ação associada à cada entrada da tabela de fluxos. As ações suportadas pelo protocolo podem ser comparadas ao conjunto de instruções usado por programas em processadores de propósito geral.

As colunas em branco na figura 3 representam o conjunto de regras usadas para classificar os pacotes que chegam ao *switch* OpenFlow, baseadas em dados nos cabeçalhos dos pacotes (MAC, IP, TCP, VLAN ID, Ethernet Type) e da porta do *switch.* A coluna *Action* indica a ação a ser tomada para pacotes de um determinado fluxo e a coluna *Count* é responsável pela coleta de estatísticas.

Entre as ações que podem ser aplicadas sobre pacotes de um fluxo podemos citar o encaminhamento dos pacotes para uma porta específica do *switch*/roteador OpenFlow, descarte dos pacotes ou encapsular e redirecionar pacotes para o controlador SDN.

II. Ferramentas Utilizadas

A. *Mininet*

Mininet é um ambiente que permite prototipação rápida de redes de computadores, em apenas um laptop [5, 6]. O ambiente é disponibilizado em forma de uma máquina virtual com as ferramentas necessárias pré-instaladas, para que outras

pessoas possam baixá-lo, executá-lo, examiná-lo e modificá-lo. Há também a possibilidade de instalar tais ferramentas em

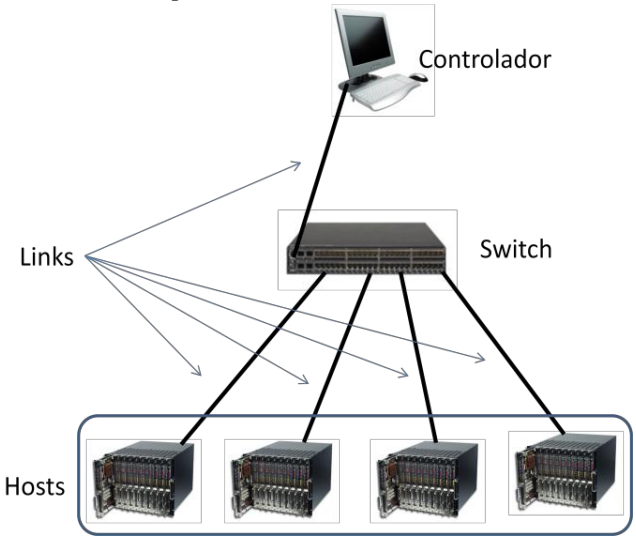


Fig. 4. Topologia de rede criada no Mininet.

computadores que executam Linux nativamente, sem a necessidade de máquinas virtuais.

O código criado neste ambiente pode ser aplicado em redes reais, ao contrário do código gerado em outros simuladores, como ns-2 ou Opnet. E por usar virtualização leve [7], a escalabilidade se torna bem maior. O uso de uma máquina virtual tradicional para cada *host* e *switch* é mais pesado, devido ao *overhead* de memória e de processamento, o que prejudica a escalabilidade da arquitetura ou topologia de rede a ser simulada. Já o uso de virtualização leve resulta em um consumo menor de recursos ao emular *hosts* e *switches*, se comparado com as formas de alocação tradicional de recursos.

Um dos diferenciais do Mininet em relação a outras ferramentas que usam virtualização leve é o suporte a *Software-Defined Networks*. Ao criar uma rede no Mininet, são emulados quatro tipos de componentes: *links*, *switches, hosts* e controladores. A figura 4 mostra uma topologia de rede composta por estes componentes.

Links conectam duas interfaces de rede virtuais, conectando hosts, switches e controladores. *Switches* fornecem a mesma semântica de entrega de pacotes que é empregada por switches físicos.

Hosts agem como computadores na rede, agindo internamente como processos de shell (terminais em linha de comando, tipo bash), conectados via pipe a um processo Mininet pai, mn, que envia comandos e monitora a saída emitida. Cada *host* tem interfaces Ethernet, portas e tabelas de roteamento próprias, graças a *namespaces* de rede [8]. O disco rígido da máquina virtual é compartilhado entre todos os hosts.

Controladores definem critérios de manipulação de pacotes. Podem estar em qualquer lugar da rede, desde que a máquina que está executando os switches tenha conectividade a nível IP com o controlador. Enquanto o Mininet é uma VM, o controlador pode estar dentro da mesma VM, nativamente no hospedeiro da VM ou na nuvem.

Para interagir com a rede, o ambiente de simulação oferece uma interface de linha de comando ciente dos nomes dos nodos (hosts, switches e controladores), permitindo que os nomes dos nodos sejam usados ao invés do endereço IP deles.

O Mininet também disponibiliza um interpretador Python com uma API customizada, facilitando a criação de experimentos, topologias e nodos (hosts, switches, controladores ou outros).

É possível também portar o ambiente para hardware, mas cada componente emulado no Mininet deve agir da mesma forma que o seu equivalente físico, isto é, um componente modelado no Mininet deve ter as mesmas características do componente real, como taxa de transferência máxima, número de portas, etc.

B. Floodlight

O Floodlight [9] é um controlador OpenFlow centralizado, com suporte a *multi-threading*, aberto, implementado em Java, testado e suportado por uma comunidade de desenvolvedores profissionais. Também inclui uma coleção de aplicações construídas sobre o controlador Floodlight. Originalmente é um projeto derivado de outro controlador SDN chamado Beacon. A versão mais recente do Floodlight, denominada Floodlight Plus, usa a versão 1.3 do protocolo OpenFlow.

A figura 5 [10] apresenta a arquitetura do Floodlight e o relacionamento entre o controlador, aplicações construídas como módulos Java compilados com o Floodlight e as aplicações construídas sobre a API REST [11]. O retângulo destacado na parte inferior esquerda da figura representa o núcleo do controlador, que implementa os módulos essenciais para o funcionamento do controlador (Module Manager, Thread Pool, Packet Streamer, REST API Server) gerenciamento da rede (Web UI, Device Manager, Topology Manager / Routing, Link Discovery, Flow Cache), suporte ao protocolo OpenFlow (OpenFlow Services: Switches, Controller Memory, PerfMon, Trace) e serviços de interesse de aplicações SDN (Jython Server, Unit Tests, Storage, Counter Store).

Na parte superior da figura 6, temos aplicações REST, *Circuit Pusher* e *OpenStack Quantum Plugin*. Módulos do Floodlight que oferecem uma API que pode ser usada pelas aplicações REST têm um "R" dentro de um quadrado branco no canto superior direito da representação gráfica do módulo.

E finalmente, ao lado esquerdo da figura 6 temos as Aplicações de Módulo, que possuem maior vazão no canal de comunicação com o controlador, interagindo com ele através de uma API Java, disponibilizada através de numerosas classes de interface. Alguns módulos podem oferecer uma interface que pode ser usada por aplicações REST. A figura apresenta os seguintes aplicações como VNF (Virtual Network Filter – Filtro de Rede Virtual), Static Flow Entry Pusher, Firewall, PortDown Reconciliation, Forwarding, Hub e Learning Switch.

O Floodlight oferece uma API baseada no estilo arquitetural REST (Representational State Transfer), comumente usado em sistemas hipermídia distribuídos. A ideia principal do REST é servir como alicerce para o desenvolvimento de aplicações na Web, mas de forma mais simples que outros mecanismos similares, como CORBA, RPC e SOAP. REST não é um padrão propriamente dito; ele utiliza outros padrões e protocolos já existentes.

Similar aos *web services*, um serviço REST independe de plataforma ou linguagem de programação. Em virtualmente todos os casos, o protocolo HTTP é usado para fornecer as características deste estilo de arquitetura.

III. Qualidade de Serviço em Software-Defined Networking

SDNs devem suportar as funcionalidades oferecidas por equipamentos e protocolos de redes tradicionais. Entre os recursos oferecidos pelos equipamentos tradicionais no que se refere aos planos de controle e dados, temos o suporte aos parâmetros de QoS. [15] apresenta uma sugestão interessante de uso de parâmetros de QoS em *Software-Defined Networks*.

Switches compatíveis com o OpenFlow podem ser classificados como exclusivos ou híbridos [12]. A figura 6 [13] mostra uma visão geral empregada por *switches* híbridos, que suportam ambas as operações de comutação via OpenFlow e via métodos tradicionais, devendo usar um mecanismo de classificação (cuja especificação está fora do escopo do protocolo OpenFlow) que roteie o tráfego ou para o pipeline OpenFlow ou para o pipeline tradicional. O pipeline tradicional já oferece meios de aplicar políticas de QoS. Empresas como Mellanox [13] e Brocade [14] e já lançaram *switches* OpenFlow híbridos.

Já os *switches* exclusivamente OpenFlow são auto-explicativos: todo fluxo de dados é manipulado apenas pelo *pipeline* do protocolo OpenFlow.

A. *QoS no protocolo OpenFlow*

O suporte a políticas de QoS no protocolo OpenFlow ainda está em sua estágios iniciais. Atualmente há apenas uma mecanismo simples de enfileiramento, similar à abordagem *DiffServ*, onde uma ou mais filas podem ser anexadas à uma porta de um switch OpenFlow e serem usadas para mapear fluxos daquela porta. Fluxos mapeados em uma fila específica são tratados de acordo com as propriedades daquela fila.

A especificação do OpenFlow 1.3 [12] mostra apenas vazões mínima e máxima como propriedade de uma fila. É comum em SLAs a existência de cláusulas em que a parte contratada deve garantir uma vazão mínima e/ou uma vazão máxima para a aplicação da parte contratante. Outros parâmetros de QoS normalmente presentes em SLAs, como atraso, taxa de erros ou variação de atraso não são formalizados explicitamente pelo protocolo.

Porém o OpenFlow oferece um campo "Experimenter", o que permite a definição de novas propriedades para uma fila de um switch OpenFlow.

A estrutura básica de uma fila de pacotes que deve ser implementada por um controlador OpenFlow deve possuir um identificador numérico, um indicador para a porta na qual está anexada, dois campos usados para facilitar a interpretação dos bytes da estrutura (um para alinhar a estrutura para ter 64 bits e outro para indicar quantos bytes relativos às propriedades da fila foram usados) e um arranjo com as propriedades da fila.

As propriedades de taxa mínima e taxa máxima de dados requerem um cabeçalho que lista a propriedade representada, a taxa de dados, representada em décimos de percentagem (isto é, se o valor da taxa de dados for 100, um décimo de 100 resulta em 10%), e um campo fixo para alinhamento de 64 bits da estrutura. A estrutura usada para uma propriedade experimental é um pouco diferente: além do header definindo a propriedade e do alinhamento de 64 bits, é necessário identificar o experimentador através de um ID fornecido tanto pela IEEE ou pela ONF. Na parte final da estrutura ficam os dados que são definidos pelo experimentador e que não são analisados pelo protocolo.

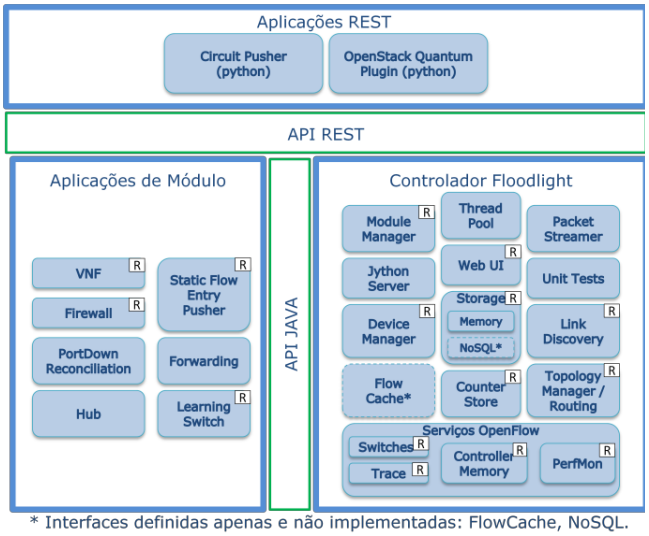


Fig. 5. Arquitetura do controlador SDN Floodlight (adaptado).

Outro suporte oferecido pelo OpenFlow no que se refere à Qualidade de Serviço é a ação *Set Queue*. Essa ação modifica o identificador de fila de um pacote. Quando o pacote é redirecionado para uma porta usando a ação de saída, o identificador da fila determina qual fila anexada à esta porta será usada para escalonamento e redirecionamento do pacote. O comportamento do redirecionamento é ditado pela configuração da fila e é usado para fornecer suporte básico a QoS. No entanto, *Set Queue* é uma ação opcional, isto é, nem todo switch OpenFlow deve suportar esta ação.

Há outra estrutura, chamada *Meter Table*, que mede a taxa de pacotes associados a um fluxo e permite controlar a taxa de pacotes, agindo como um limitador de banda. A *Meter Table*, portanto, pode ser usada para aplicar operações simples de QoS. Essas tabelas são associadas ao fluxos (*flows*), que podem ser associadas com as filas das portas que o switch possui, auxiliando a implementação de frameworks de QoS complexos, como *DiffServ*.

B. *QoS no controlador Floodlight*

Até a versão 1.0 do protocolo OpenFlow, era possível adicionar suporte básico a parâmetros de QoS no controlador Floodlight. O protocolo inclui mecanismos que permitem configurar um Tipo de Serviço em um fluxo, tal como enfileirar pacotes correspondentes ao fluxo para uma fila específica de uma porta específica. No entanto, estas ações de modificação são consideradas opcionais pelo protocolo OpenFlow e nem sempre serão suportadas por todo switch OpenFlow.

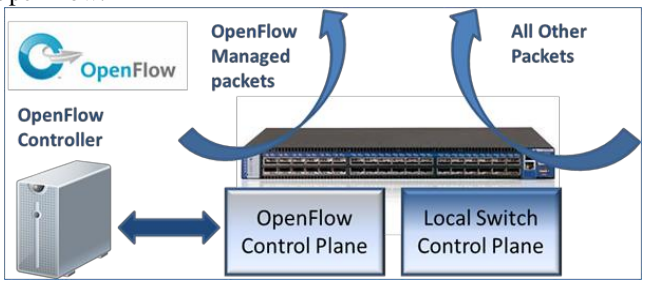


Fig. 6. Visão geral de um switch OpenFlow híbrido.

Na abordagem *DiffServ* para fornecimento de QoS, é possível usar o campo Type of Service (ToS) de um header IPv4 para classificar o tráfego e colocar o pacote em uma fila específica. Métodos iniciais para suportar QoS no Floodlight envolviam a manipulação do campo ToS para selecionar em qual fila o pacote deveria ser colocado. Mas este suporte é limitado: a implementação de alguns switches que usam a versão 1.0 do protocolo OpenFlow não suportam estruturas de fila anexadas à portas específicas e *DiffServ* ou outros métodos de QoS baseados em classes raramente são suportados. E o campo ToS do cabeçalho IP, considerado defasado, foi substituído pelos campos DS (*Differentiated Services*) e ECN (*Explicit Congestion Notification*) para classificação e condicionamento de tráfego de rede.

Com a versão 1.3 do OpenFlow, foi adicionada a ação *Set Queue*, que independe do valor do campo ToS do header IPv4 de um pacote.

O Floodlight Plus oferece suporte ao enfileiramento de pacotes de rede e implementação simples de QoS através das classes do pacote `org.openflow.protocol.queue`. As classes deste pacote implementam as filas (classe `OFPacketQueue`), propriedades de fila (classes `OFQueueProperty` e `OFQueuePropertyType`) e as duas propriedades definidas pelo padrão OpenFlow, que são *Minimum Rate* (classe `OFQueuePropertyMinRate`) e *Maximum Rate* (classe `OFQueuePropertyMaxRate`). O pacote `org.openflow.protocol.meter` implementa os medidores de banda.

IV. Conclusões e Trabalhos Futuros

A separação dos planos de controle e de dados dos dispositivos de rede facilitou o gerenciamento de redes enormes e em crescimento contínuo. Aplicações e clientes demandam cada vez mais recursos da rede. Para garantir que as aplicações tenham suas demandas atendidas pela rede, são feitos Acordos de Nível de Serviço, onde são explicitados os requisitos de Qualidade de Serviço que devem ser garantidos pela prestadora de serviços.

A garantia de parâmetros de Qualidade de Serviço em Software-Defined Networking ainda está em estágios iniciais e é uma área ampla de pesquisa. O suporte oferecido pelo protocolo OpenFlow e por fabricantes de switches ainda é limitado apenas à vazão mínima e máxima de um fluxo e a um parâmetro personalizado que não está sujeito à padronização pelo protocolo e que depende de implementação em controladores SDN diferentes. Soluções tradicionais de gerência e monitoração de QoS ainda são mais robustas para aplicações comerciais.

O conceito de SDN é elegante e escalável, mas a sua adoção ainda é limitada principalmente por causa de equipamentos legados que ainda têm acoplados os planos de controle e de dados e que não foram projetados para suportar a separação desses planos. Também são necessários o amadurecimento dos protocolos envolvidos e a adaptação de aplicações existentes para utilizar os serviços e APIs fornecidos pelos controladores SDN.

Para oferecer uma implementação de parâmetros de Qualidade de Serviço mais robusta em Software-Defined Networking, é necessário garantir que todas as vantagens que SDNs possuem sobre redes tradicionais não sejam perdidas.

Futuramente, como alternativa ao simulador Mininet, utilizar o EstiNet [16] para abordar a simulação de SDNs e avaliar os resultados dos experimentos.

Referências

[1] Nick Feamster, Jennifer Rexford, and Ellen Zegura. 2013. **The Road to SDN**. Queue 11, 12, pages 20 (December 2013), 21 pages. DOI=10.1145/2559899.2560327 http://doi.acm.org/10.1145/2559899.2560327.

[2] Kreutz, D., Ramos, F., Verissimo, P., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2014). **Software-Defined Networking: A Comprehensive Survey**. arXiv preprint arXiv:1406.0440.

[3] **Open Networking Foundation white papers**. Disponível em https://www.opennetworking.org/sdn-resources/sdn-library/whitepapers. Acessado em: 06/2014.

[4] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. **OpenFlow: enabling innovation in campus networks**. SIGCOMM Comput. Commun. Rev. 38, 2 (March 2008), 69-74. DOI=10.1145/1355734.1355746 http://doi.acm.org/10.1145/1355734.1355746.

[5] **Mininet - An Instant Virtual Network on your laptop (or other PC)**. Disponível em http://mininet.org/. Acessado em 05/2014.

[6] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. **A network in a laptop: rapid prototyping for software-defined networks**. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX). ACM, New York, NY, USA, Article 19, 6 pages. DOI=10.1145/1868447.1868466 http://doi.acm.org/10.1145/1868447.1868466.

[7] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. **Large-scale virtualization in the emulab network testbed**. In USENIX 2008 Annual Technical Conference, pages 113–128. USENIX, 2008.

[8] **lxc Linux containers**. Disponível em http://lxc.sf.net. Acessado em 05/2014.

[9] **Project Floodlight**. Disponível em http://www.projectfloodlight.org/floodlight/. Acessado em 06/2014.

[10] **Project Floodlight Architecture**. Disponível em http://docs.projectfloodlight.org/display/floodlightcontroller/Architecture. Acessado em 06/2014.

[11] Roy Thomas Fielding. 2000. **Architectural Styles and the Design of Network-Based Software Architectures**. Ph.D. Dissertation. University of California, Irvine. AAI9980887.

[12] **OpenFlow Specifications**. Disponível em https://www.opennetworking.org/sdn-resources/onf-specifications/openflow. Acessado em 09/2014.

[13] **Mellanox Boosts SDN and Open Source with New Switch Software**. Disponível em http://www.mellanox.com/blog/2014/01/mellanox-boosts-sdn-and-open-source-with-new-switch-software. Acessado em 09/2014.

[14] **Brocade Delivers 100 Gigabit Ethernet Solutions for Software-Defined Networks**. Disponível em http://newsroom.brocade.com/press-releases/brocade-delivers-100-gigabit-ethernet-solutions-fo-nasdaq-brcd-0890051. Acessado em 09/2014.

[15] Madalena P. da Silva, Mario A. R. Dantas, Alexandre L. Gonçalves, Alex R. Pinto: **A Managing QoE Approach for Provisioning User Experience Aware Services Using SDN**. Q2SWinet@MSWiM 2015: 51-58

[16] Shie-Yuan Wang; Chih-Liang Chou; Chun-Ming Yang, "**EstiNet OpenFlow network simulator and emulator**", *Communications Magazine, IEEE* , vol.51, no.9, pp.110,117, September 2013. doi: 10.1109/MCOM.2013.6588659