UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Diego Martins

UM ESTUDO DE QoS EM AMBIENTES DE REDE SDN

Florianópolis

2014

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

UM ESTUDO DE QoS EM AMBIENTES DE REDE SDN

Diego Martins

Trabalho de Conclusão de Curso submetido ao curso de Ciência da Computação para a obtenção do Grau de Bacharel em Ciência da Computação.

Orientador:
Profº. Drº. Mario Antonio Ribeiro Dantas

Florianópolis

2014

AGRADECIMENTOS

Agradeço primeiramente os meus pais, por terem me incentivado e me apoiado durante o decorrer do curso.

Aos meus tios Jair, Ângela e Bigo (*in memorian*) pela ajuda dada durante os meus primeiros dias em Florianópolis.

À Verinha, ex-secretária do Programa de Pós-Graduação em Ciência da Computação, por ter sido uma segunda mãe durante os meus primeiros anos longe de casa.

Aos professores do curso, em especial ao profº Mario Dantas, pela boa convivência dentro e fora das salas de aula.

À Madalena, doutoranda do Programa de Pós-Graduação em Engenharia de Gestão do Conhecimento, pelo imenso apoio na compreensão do assunto e pelas sugestões oferecidas.

Aos amigos obtidos no curso e fora dele pelos bons momentos que passamos juntos.

RESUMO

O rápido avanço tecnológico na área de redes de computadores incentivou o surgimento de aplicações e consumidores cada vez mais exigentes. Para suprir essa necessidade, a infraestrutura de rede teve que crescer também, tornando o gerenciamento dos recursos uma tarefa cada vez mais complicada conforme novos equipamentos são adicionados à rede. O conceito de *Software-Defined Networking* evoluiu com o passar dos anos e tem como objetivo facilitar o gerenciamento desses equipamentos, separando os planos de controle e dados dos dispositivos de rede. Também oferece um ponto centralizado para a aplicação de políticas, como as impostas para garantir o fornecimento de parâmetros de Qualidade de Serviço (QoS). No entanto, o suporte à QoS em *Software-Defined Networking* ainda é limitado e é uma área em que há muito espaço para inovação. Neste trabalho de conclusão de curso apresenta-se o suporte à QoS no controlador Floodlight, um controlador SDN escrito em Java compatível com o protocolo OpenFlow, e as limitações existentes para a aplicação de novos métodos de monitoração de parâmetros de QoS no Floodlight.

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

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CLI	Command Line Interface
CORBA	Common Object Request Broker Architecture
DiffServ	Differentiated Services
DS	Differentiated Services
DSCP	Differentiated Services Code Point
ECN	Explicit Congestion Notification
HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IntServ	Integrated Services
IP	Internet Protocol
LLDP	Link Layer Discovery Protocol
MAC	Media Access Control
NAT	Network Address Translation
NETCONF	Network Configuration Protocol
NoSQL	Not only Structured Query Language
OF-Config	OpenFlow Management and Configuration Protocol
ONF	Open Networking Foundation
OVSDB	OpenVSwitch Database Management
RSVP	Resource Reservation Protocol
QoS	Quality of Service
REST	Representation State Transfer
RFC	Request for Comments
RPC	Remote Procedure Call
SDN	Software-Defined Network
SLA	Service-Level Agreement
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
ToS	Type of Service
URI	Universal Resource Identifier
VLAN	Virtual Local Area Network

VM	Virtual Machine
VoIP	Voice over Internet Protocol

SUMÁRIO

1. INTRODUÇÃO

Com o avanço das tecnologias de rede e da acessibilidade à elas nos últimos anos, o número de usuários cresceu rapidamente e aplicações com requerimentos maiores de rede surgiram, como *streaming* de vídeo, jogos online, videoconferência, VoIP e transferência rápida de arquivos de grande tamanho.

Se a infraestrutura de rede não acompanhar o crescimento da base de usuários e os requisitos necessários para executar aplicações cada vez mais robustas, os usuários da rede ficarão insatisfeitos com o serviço prestado.

Para permitir o crescimento das redes, são adicionados novos equipamentos, como switches, roteadores, NATs, balanceadores de carga, *firewalls* e sistemas de detecção de intrusão. Mas gerenciar cada dispositivo individualmente não é eficiente e as chances de ocorrer algum erro humano durante a configuração da rede são grandes. O conceito de *Software-Defined Networking* ajuda no gerenciamento de equipamentos dos mais variados tipos, oferecendo um ponto centralizado de controle.

Clientes que precisam oferecer e/ou utilizar alguma aplicação sensível à alguma característica da rede, como atraso, taxa de erros ou vazão, podem assinar um Acordo de Nível de Serviço com a fornecedora da infraestrutura de rede para garantir que características específicas (conhecidas como parâmetros de Qualidade de Serviço) da rede sejam garantidas.

1.1 OBJETIVO GERAL

O objetivo geral deste trabalho é compreender políticas existentes para aplicação de parâmetros de Qualidade de Serviço e controle de redes baseadas em Software-Defined Networking e propor uma parametrização para a monitoração de SDNs.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos deste trabalho são:

- Fazer um levantamento bibliográfico sobre métodos de suporte à Qualidade de Serviço em ambientes SDN que lidam com a monitoração de redes;
- Abordar as vantagens e desvantagens desses métodos;
- Verificar como é feita a monitoração dos parâmetros para implementação de controle de QoS diferenciado.

1.3 ORGANIZAÇÃO DO TEXTO

No capítulo 2 são introduzidos conceitos básicos de Software-Defined Networking, o protocolo OpenFlow e seu funcionamento, Qualidade de Serviço e Serviços Integrados e Diferenciados.

O capítulo 3 apresenta o Mininet, uma ferramenta que permite a prototipação rápida de uma grande rede que suporta múltiplos equipamentos através de uma máquina virtual rodando o sistema operacional Ubuntu. Também é apresentado o Floodlight, um controlador OpenFlow desenvolvido em Java e que conta com várias aplicações. Este controlador possui interfaces de programação para o desenvolvimento de novos módulos. Ele também suporta aplicações que usem o estilo arquitetural REST, também detalhado no capítulo 3.

No capítulo 4 é apresentada a forma como Software-Defined Networks oferecem suporte a parâmetros de Qualidade de Serviço, à nível do protocolo OpenFlow. No capítulo 5 é feita uma análise sobre o suporte do controlador Floodlight quanto ao gerenciamento de parâmetros de Qualidade de Serviço. E por fim, no capítulo 6 é apresentada a conclusão do trabalho.

2. CONCEITOS BÁSICOS

Neste capítulo são apresentados os conceitos de Software-Defined Networking (SDN), o protocolo OpenFlow, que é usado em SDNs, e o que é Qualidade de Serviço em redes de computadores. Também são apresentadas as abordagens de Serviços Integrados e Serviços Diferenciados, usadas para fornecimento de políticas de Qualidade de Serviço, normalmente requeridas em Acordos de Nível de Serviço.

2.1 SOFTWARE-DEFINED NETWORKING (SDN)

Redes de computadores são complexas e difíceis de se projetar e de gerenciar. Há vários tipos de equipamentos envolvidos em uma rede, como NATs, roteadores, switches, *firewalls*, balanceadores de carga e sistemas de detecção de intrusão [10].

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, que variam entre fabricantes e até mesmo modelos diferentes de equipamentos de um mesmo fabricante. 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.

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

Havia poucas plataformas de software livre que permitiam a realização de experimentos de rede, mas elas não tinham uma quantidade grande de portas (como ocorre com equipamentos de rede dedicados) ou o desempenho desejados. Por exemplo, pode-se citar o uso de um computador tradicional com múltiplas interfaces de rede. Porém o número de interfaces de rede suportadas é baixo e a vazão de dados é muito inferior, se comparado com equipamentos de rede dedicados.

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) [8].

Dispositivos tradicionais têm os planos de controle (que define as regras sobre como manusear o tráfego de dados que passa pelo dispositivo) e de dados (que

redireciona o tráfego de dados de acordo com as regras definidas no plano de controle) unificados, enquanto em SDNs esses planos são separados [16, 19]. Redes tradicionais com planos de controle e dados unificados ficam mais complexas conforme mais dispositivos de rede são adicionados. Em situações de grande escala, como *data centers*, atualizar políticas manualmente e uniformemente entre milhares de dispositivos se torna uma tarefa demorada e deixa margem para erros de configuração.

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.

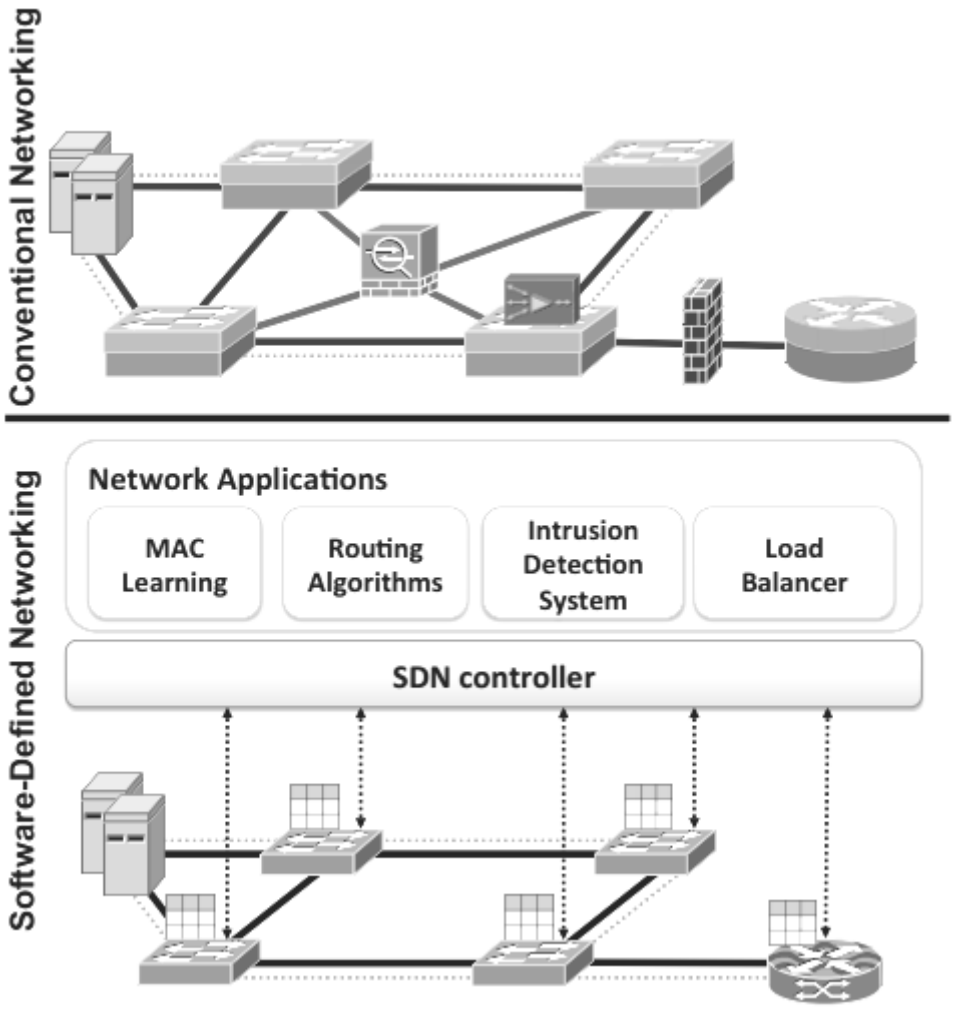

Figura 1: Comparativo entre redes tradicionais e SDN.

A figura 1 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 serviços fornecidos pelo controlador SDN.

É importante notar que esta aplicação é logicamente centralizada, e não fisicamente centralizada como em redes tradicionais. Dispositivos de *forwarding* de dados (switches, por exemplo) em uma SDN podem ter conexão com múltiplos controladores SDN, o que permite redundância em caso de uma falha em um dos controladores, facilitando a escalabilidade da rede conforme mais dispositivos são adicionados.

Há também controladores SDN distribuídos, que além de oferecer APIs para interagir com aplicações e realizar a comunicação com os planos de dados dos switches da rede, devem oferecer uma API que permita a comunicação com outros controladores SDN. A comunicação é usada para os casos em que é necessário importar/exportar dados entre controladores, checar consistência de dados e ter capacidade de monitorar e notificar outros controladores, como em casos de falha ou quando é necessário associar alguns dispositivos de rede a um controlador específico.

Um dos protocolos que oferece uma API para realizar a comunicação entre o controlador SDN e os dispostivios de rede é o OpenFlow, abordado com mais detalhes na seção 2.2. Outros protocolos que podem ser usados para realizar a comunicação dos dispositivos de rede com um controlador são:

- OVSDB (*Open vSwitch Database Management Protocol*): switch virtual usado em ambientes de servidores virtualizados, redirecionando tráfego entre diferentes máquinas virtuais em um mesmo host físico ou entre máquinas virtuais e uma rede física. Especificado pela RFC 7047 [22];
- NETCONF (*Network Configuration Protocol*): mecanismo para instalar, manipular e excluir a configuração de dispositivos de rede. Especificado pela RFC 6241 [23];
- SNMP (*Simple Network Management Protocol*): usado na gerência de redes baseadas em TCP/IP. Especificado pela RFC 1157 [24];

- OF-Config (*OpenFlow Management and Configuration Protocol*): usado para gerenciar switches físicos e virtuais em um ambiente OpenFlow [25].

Os protocolos citados oferecem algumas funcionalidades que complementam o OpenFlow, mas são usados para gerenciar os dispositivos de uma maneira diferente, enquanto o OpenFlow é restrito ao plano de controle desses dispositivos. Por exemplo, não é possível desligar uma porta específica de um switch usando OpenFlow para economia de energia.

Enfim, SDNs permitem que aplicações automatizadas e administradores de rede ajustem dinamicamente as políticas usadas no controle de tráfego para alcançar as necessidades de mudança da rede, como engenharia de tráfego, segurança, controle de acesso, qualidade de serviço, otimização de armazenamento, processamento e consumo de energia, por exemplo.

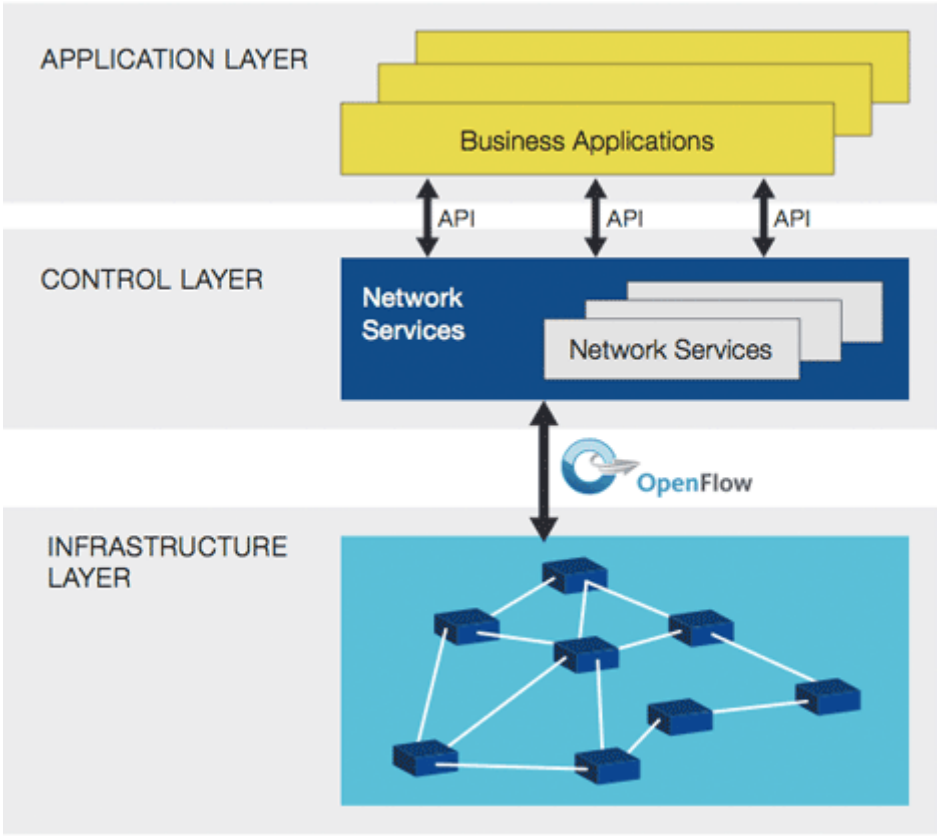

Figura 2: Arquitetura de uma *Software-Defined Network* [1].

A figura 2 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.

2.2 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 com o software de controle SDN [9]. 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.

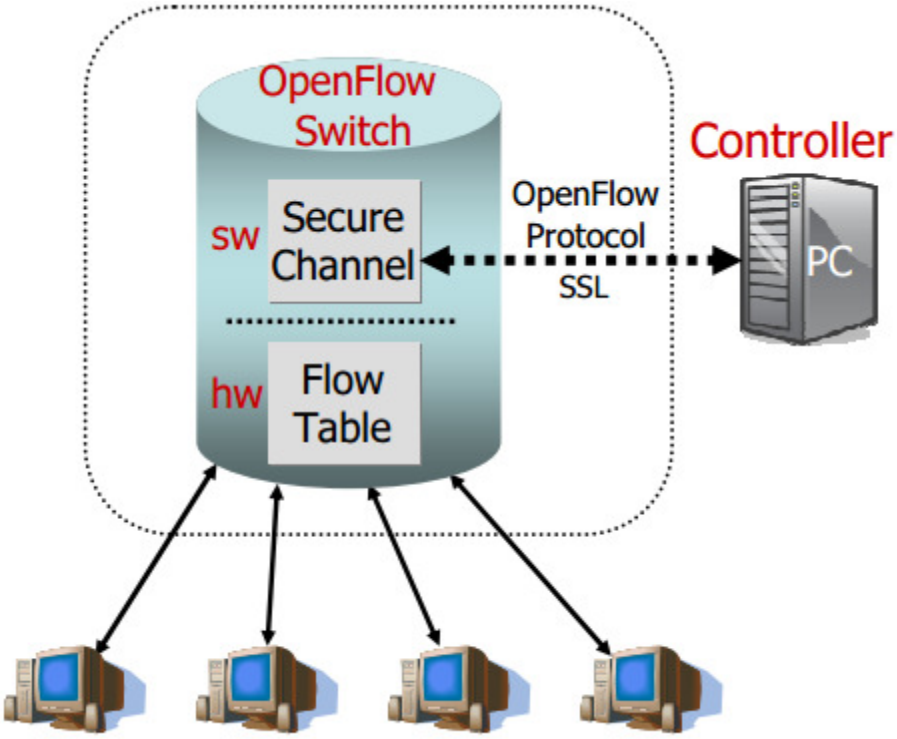

Figura 3: Interação de um switch OpenFlow com o controlador SDN e com os dispositivos de rede [9].

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 mostra uma visão global da interação entre o controlador SDN, o switch compatível com o protocolo OpenFlow e alguns dispositivos de usuário (neste caso, computadores). Um switch OpenFlow usa conexões seguras (neste caso, criptografada, usando SSL) com um ou mais controladores SDN. O controlador SDN consiste de uma ou mais aplicações responsáveis em ditar as regras de *forwarding* (redirecionamento) de pacotes. O switch mantém uma ou mais Tabelas de Fluxos (*Flow Tables*). Quando um pacote não é compatível com nenhuma dessas tabelas, diz-se que houve um *table miss*, isto é, não há uma ação específica para aquele tipo de pacote a ser tomada. Em casos de *table miss*, o switch deve possuir uma ação padrão, que normalmente é encapsular o pacote e enviá-lo ao controlador, via conexão segura.

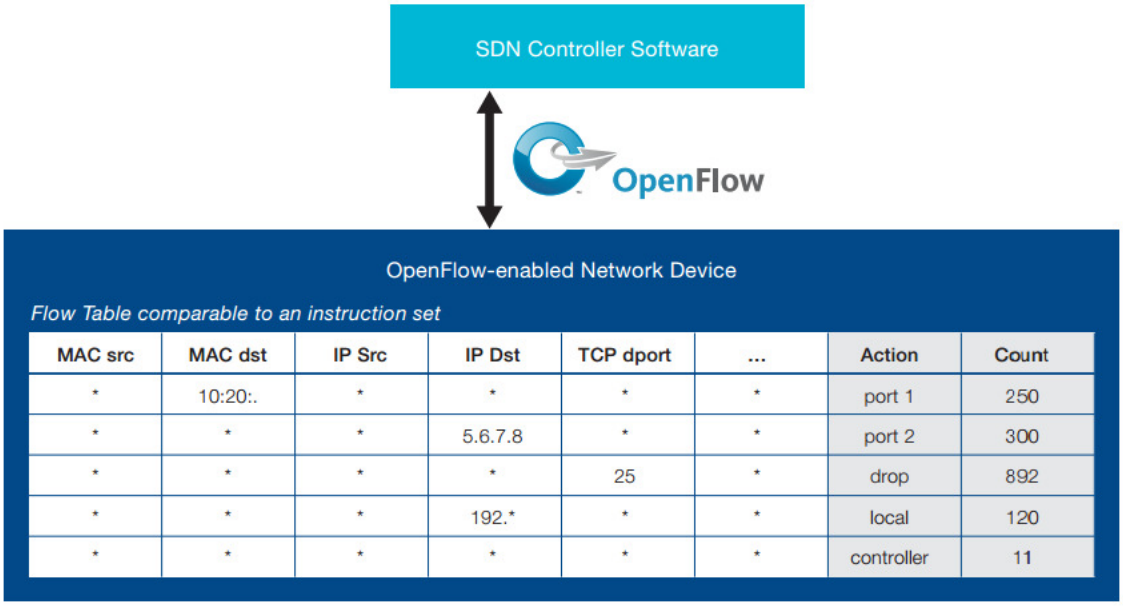

Figura 4. Exemplo de uma interação entre um controlador SDN e um dispositivo físico de rede que suporta o protocolo OpenFlow. [1]

A figura 4 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.

Na figura 4, as colunas em branco representam o conjunto de regras usadas para classificar os pacotes que chegam ao switch OpenFlow: os endereços MAC do remetente e do destinatário (`MAC src` e `MAC dst`, respectivamente), os endereços IP do remetente e do destinatário (`IP src` e `IP dst`, respectivamente), as portas TCP de origem (`TCP sport`, não mostrada na figura) e destino (`TCP dport`), VLAN ID, Ethernet Type e porta do switch (estes três últimos não representados na figura 3). Já as colunas sombreadas indicam a ação a ser tomada (coluna `Action`) para pacotes de um determinado fluxo e a coleta de estatísticas (coluna `Count`).

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.

2.3 QUALIDADE DE SERVIÇO (QoS)

De forma similar a estradas e rodovias de metrópoles em horários de pico, redes de computadores também podem ficar congestionadas, isto é, quando há lentidão no fluxo entre dois dispositivos devido à alta demanda de uma ou mais rotas de acesso à rede.

Com a evolução das tecnologias de redes de comunicação, diversos tipos de aplicação foram desenvolvidos tendo como alicerce a Internet. Navegar na web, ler emails, transferir arquivos, ouvir música, assistir vídeos, VoIP, videoconferência e jogos online são algumas aplicações que as pessoas usam diariamente na Internet.

No entanto, tais aplicações usam fluxos que requerem alguns parâmetros para que suas funcionalidades não sejam prejudicadas: largura de banda (também conhecido como vazão), atraso (ou latência), flutuação (também conhecido como variação de atraso ou *jitter*) e confiabilidade (ou perda). Estes quatro parâmetros determinam a **qualidade de serviço** (**QoS**) que um fluxo exige [11, 12].

A vazão de um fluxo determina quanta informação é necessário transmitir em um intervalo de tempo. Transferência de arquivos, transmissão de vídeos e sessões de videoconferência são exemplos de aplicações onde vazão alta é essencial para o

funcionamento da aplicação sem problemas maiores. Quanto a ler emails ou navegar na web, a necessidade de vazão alta é menor.

Jogos online, videoconferência e outras aplicações interativas são mais sensíveis ao atraso da rede. Se o atraso for muito alto, ações efetuadas em jogos online e palavras ditas durante uma conversa via videoconferência demorarão para serem percebidas, causando frustração nos usuários. Vídeos sob demanda, como os do YouTube, por exemplo, não sofrem tanto com atraso.

O parâmetro de flutuação indica uma variação na taxa de entrega de pacotes. Uma pessoa efetuando um login remoto pode demorar a perceber atualizações na tela caso exista alta flutuação na rede. Outro exemplo é o de um usuário assistindo um vídeo ou ouvindo música via streaming cujo tempo de transmissão varia aleatoriamente. Ora a transmissão flui normalmente, ora é necessário esperar pelos dados para prosseguir com a reprodução da mídia.

E finalmente há a confiabilidade do fluxo, que indica se há perda de informações durante a transmissão. Por exemplo: erros em arquivos baixados ou emails são inaceitáveis; se parte de um arquivo foi corrompido na rede e chegou com erros, aquela parte deve ser retransmitida. Se houver perdas durante uma videoconferência ou chamada VoIP, os usuários podem nem perceber ou ignorar o erro, descartando a necessidade de retransmissão.

Aplicações de rede necessitam de garantias de qualidade de serviço para funcionarem de forma adequada. Há duas grandes abordagens de políticas de fornecimento de QoS, conhecidas como Serviços Integrados (*IntServ*) e Serviços Diferenciados (*DiffServ*).

2.4 SERVIÇOS INTEGRADOS (IntServ) E DIFERENCIADOS (DiffServ)

Roteadores que usam a abordagem *IntServ* garantem que os parâmetros de QoS para um determinado fluxo sejam reservados, isto é, não podem ser usados por outros fluxos. É uma abordagem recomendada para aplicações muito sensíveis ou intolerantes à variação de um ou mais parâmetros de QoS.

IntServ requer que todos os roteadores em uma rota tenham controle de admissão para que a qualidade de serviço seja prestada. Para isso, eles devem suportar alguns protocolos, como o RSVP (*Resource ReSerVation Protocol*), que provê certo nível de controle sobre os dados.

Entre as desvantagens desta abordagem, podemos citar a falta de escalabilidade para atender múltiplos fluxos, o uso ineficiente dos recursos reservados para um fluxo específico e alta dependência dos roteadores usados pelo fluxo. Se alguma característica do fluxo for alterada, todos os roteadores afetados também devem alterar essa característica.

Já os Serviços Diferenciados oferecem mais flexibilidade que os Serviços Integrados para fornecer garantias de QoS. Esta abordagem separa os pacotes das aplicações em filas, de acordo com classes de serviço definidos em um determinado roteador; não há necessidade de reservar recursos previamente, o que garante a escalabilidade.

Os principais problemas desta abordagem são a possiblidade de *DiffServ* não ser suportado por alguns roteadores e a necessidade de um bom gerenciamento do núcleo da rede.

3. FERRAMENTAS

Neste capítulo serão descritas as ferramentas estudadas e usadas no desenvolvimento da proposta deste trabalho. O Mininet é um simulador de SDNs e o Floodlight é um controlador SDN que pode ser conectado à rede simulada pelo Mininet.

3.1 MININET

Mininet é um ambiente que permite prototipação rápida de redes de computadores, em apenas um laptop [2, 3]. 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 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 [4], 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:

- **Links:** age como um fio conectando duas interfaces de rede virtuais, conectando hosts, switches e controladores.
- **Hosts:** máquinas da rede. Internamente, hosts são apenas 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 [5]. O disco rígido da máquina virtual é compartilhado entre todos os hosts.
- **Switches:** fornecem a mesma semântica de entrega de pacotes que é empregada por switches físicos.

- **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.

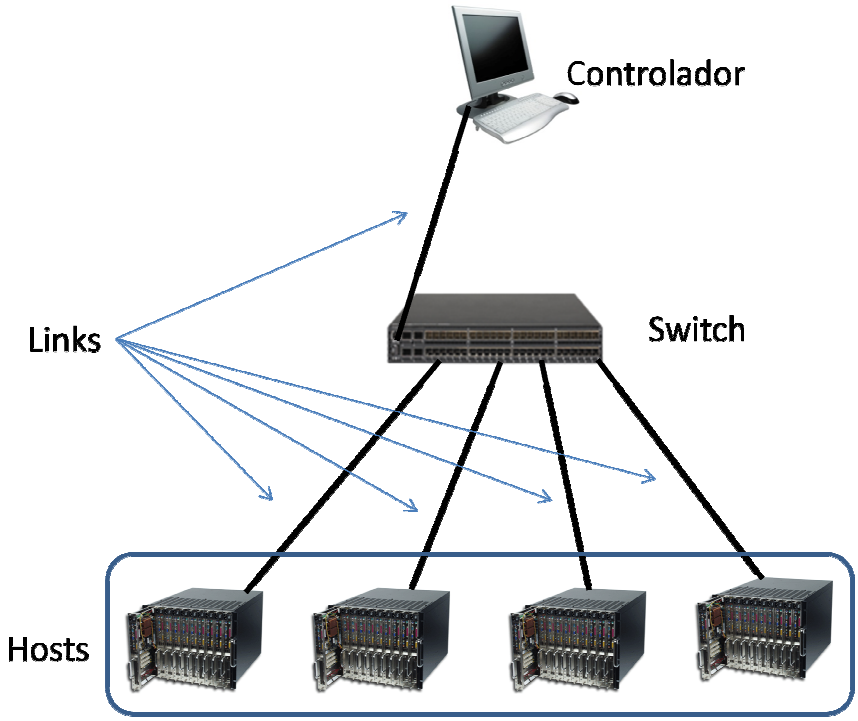

Figura 5: Exemplo de topologia de rede criada no Mininet.

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.

3.2 FLOODLIGHT

O Floodlight [6] 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.

3.2.1 ARQUITETURA DO FLOODLIGHT

A figura 6 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 [7]. O retângulo destacado na parte inferior esquerda da figura representa o núcleo do controlador, que oferece serviços de interesse de aplicações SDN e implementa vários módulos. São eles:

- Module Manager: responsável pelo carregamento de outros módulos, descritos em um arquivo de propriedades que é lido pelo controlador durante a inicialização. O Floodlight pode ser configurado para carregar diferentes módulos para acomodar aplicações variadas.
- Thread Pool: é apenas um *wrapper* para a classe `ScheduledExecutorService` do Java, que serve para fazer com que *threads* sejam executadas em tempos específicos ou periodicamente.
- Packet Streamer: é um serviço de *streaming* de pacotes trocados entre qualquer switch e o controlador para um observador.
- Jython Server: usado para depurar os programas escritos para o controlador usando um interpretador Jython (implementação de Python em Java).
- Web UI: interface gráfica via web para gerenciamento do controlador.
- Unit Tests: testes de unidade que facilitam a busca por problemas no código de um módulo.
- Device Manager: registra dispositivos conforme eles se movem pela rede, isto é, adições e remoções dos dispositivos na rede e define o dispositivo de destino para um novo fluxo.
- Topology Manager / Routing: mantém a informação da topologia da rede para o controlador (Manager) e encontra rotas na rede (Routing). São criadas “ilhas”

OpenFlow que reunem switches OpenFlow fortemente conectados numa mesma instância do Floodlight.

- Link Discovery: é responsável em descobrir e manter o status das ligações de dispositivos na rede OpenFlow. Este serviço usa o protocolo LLDP (Data Link Discovery Protocol) e faz broadcast de pacotes para detectar as ligações.
- Flow Cache: é uma API definida para o manuseio de um conjunto de diferentes tipos de eventos na rede. Aplicações SDN frequentemente necessitam decidir quando e como tais eventos de rede devem ser tratados. Por exemplo, como manusear fluxos quando há uma falha em algum link ou switch.
- Storage: permite que módulos do Floodlight que implementam a interface `IStorageSourceService` possam criar, apagar e modificar dados nessa fonte de armazenamento, que fica em memória, no estilo NoSQL. Todos os dados armazenados nesta área são compartilhados, isto é, não há um mecanismo de proteção de memória que impeça um módulo de sobrescrever dados de outros módulos. E por permanecerem apenas em memória, caso a máquina seja desligada, dados nesta área são perdidos.
- Counter Store: implementa um armazém central para contadores do sistema, como número de pacotes de entrada e de pacotes de saída, por exemplo. Esses contadores centrais não são afetados por outros módulos.
- OpenFlow Services (Switches, Controller Memory, PerfMon, Trace): são os serviços que são oferecidos pelo protocolo OpenFlow e que são usados para se comunicar com os dispositivos compatíveis com o protocolo.

Não incluído na figura 6, há também um *REST API Server*, que é responsável em expor as APIs REST sobre o HTTP.

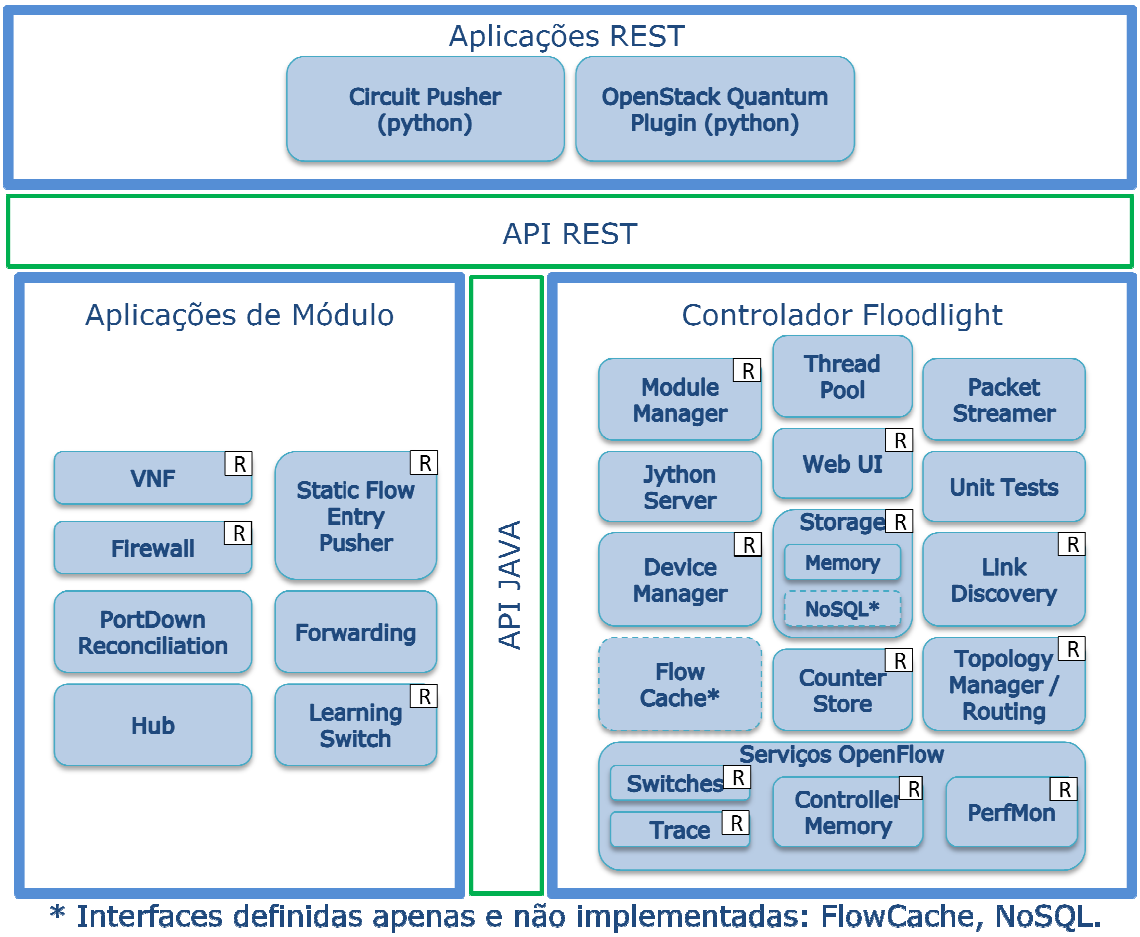

Figura 6: Arquitetura do controlador Floodlight (adaptado de [7]).

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.

O *Circuit Pusher* utiliza as APIs REST para criar um circuito bidirecional, isto é, uma entrada de fluxo permanente em todos os switches na rota entre dois dispositivos baseados em endereços IP com uma prioridade específica. É implementado como um script em Python.

Já o *OpenStack Quantum Plugin* é um módulo de virtualização de rede de camada 2 (baseada em MAC). Assim é possível criar múltiplas redes lógicas na camada 2 (camada de enlace de dados) com apenas um domínio da camada 2. Este módulo pode ser usado para implantar o OpenStack, que é uma plataforma de software de cloud computing.

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.

3.2.2 REST

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 [13, 14].

As principais características do estilo arquitetural REST são:

- Interface uniforme: é a característica principal que distingue o REST de outros estilos arquiteturais baseados em rede. Tornando a interface dos componentes genérica, a arquitetura do sistema é simplificada. A transferência de dados de hipermídia de alta granularidade (longas cadeias de bytes, por exemplo) fica mais eficiente. No entanto, o uso de uma interface genérica degrada a eficiência das aplicações, pois a informação é transferida de forma padronizada e deve ser adaptada para as necessidades da aplicação.
- Cliente-Servidor: a separação de funcionalidades em componentes diferentes (por exemplo, um componente se encarrega da manipulação da interface gráfica no lado do cliente, enquanto outro componente fica responsável pela manipulação de armazenamento de dados do lado do servidor) melhora a portabilidade da aplicação para plataformas variadas e a escalabilidade do sistema.
- Sem estado: a interação entre cliente e servidor não deve depender de informações de contexto armazenadas localmente ou no servidor. Requisições

devem possuir toda a informação necessária para que a mesma seja compreendida.

- Cache: para melhorar a eficiência da rede, os dados dentro de uma resposta de uma requisição devem ser marcados como "cacheáveis". Se uma resposta for "cacheável", então o cliente pode reusá-la posteriormente, quando requisições similares forem emitidas. Isso elimina novas requisições ao servidor, mas pode diminuir a confiabilidade do sistema se os dados da cache diferirem bastante dos dados que teriam sido obtidos caso uma nova requisição ao servidor fosse feita.
- Sistema em camadas: um cliente não deve ser capaz de determinar se ele está conectado diretamente ao servidor final ou a um servidor intermediário. O uso de camadas pode melhorar a escalabilidade do sistema, além de oferecer melhor encpasulamento entre componentes diversos, balanceamento de carga e garantia de políticas de segurança. Mas camadas adicionam *overhead* e latência no processamento de dados, o que pode ser percebido pelo usuário.
- Código sob demanda: permite que a funcionalidade do cliente seja estendida através do *download* e da execução de código em forma de *applets* ou *scripts*, sem necessariamente ter o código pré-implementado antes de disponibilizar o serviço. Embora aumente a extensibilidade do sistema, isto reduz a visibildade. Esta característica é considerada opcional.

Quanto à manipulação de dados, o REST usa o conceito de **recurso** como abstração chave. Um recurso é um mapeamento conceitual para um conjunto de entidades e qualquer informação que possa ser nomeada pode ser um recurso. Por exemplo: documentos, imagens, serviços de previsão do tempo, coleções de recursos, objetos não-virtuais (pessoas reais), etc.

Recursos no REST são identificados através de URIs (Universal Resource Identifiers). E ações realizadas pelos componentes do sistema sobre os recursos são denominadas **representações**. Uma representação consiste de dados, metadados descrevendo os dados e, em algumas ocasiões, metadados descrevendo os metadados (para fins de verificação de integridade da mensagem).

O REST usa vários tipos **conectores** para encapsular as atividades de acesso aos recursos e transferir representações de recursos. Conectores apresentam uma interface abstrata para a comunicação de componentes, melhorando a simplicidade do sistema ao fornecer uma separação clara das funções dos recursos e mecanismos de comunicação.

3.3 CONSIDERAÇÕES DO CAPÍTULO

As ferramentas Mininet e Floodlight foram usadas em conjunto para simular uma SDN de forma rápida e barata, descartando a necessidade de equipamentos de rede físicos para a realização dos experimentos.

O Mininet é disponibilizado como uma máquina virtual Ubuntu e cria de forma eficiente, via linha de comando e scripts escritos em Python, topologias e dispositivos presentes em infraestruturas de rede em ambientes de produção comuns a data centers e grandes organizações. É possível executar aplicações e comandos de rede como ping nos terminais simulados pela ferramenta.

Escrito em Java, o Floodlight é um controlador SDN que pode rodar em qualquer plataforma que suporte a máquina virtual Java. Ele pode ser conectado ao Mininet e substituir o controlador SDN padrão utilizado pelo Mininet.

4. QUALIDADE DE SERVIÇO EM SOFTWARE-DEFINED NETWORKS

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.

Switches compatíveis com o OpenFlow podem ser classificados como exclusivos ou híbridos [15]. Switches híbridos 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 [17] e Brocade [18] e já lançaram switches OpenFlow híbridos.

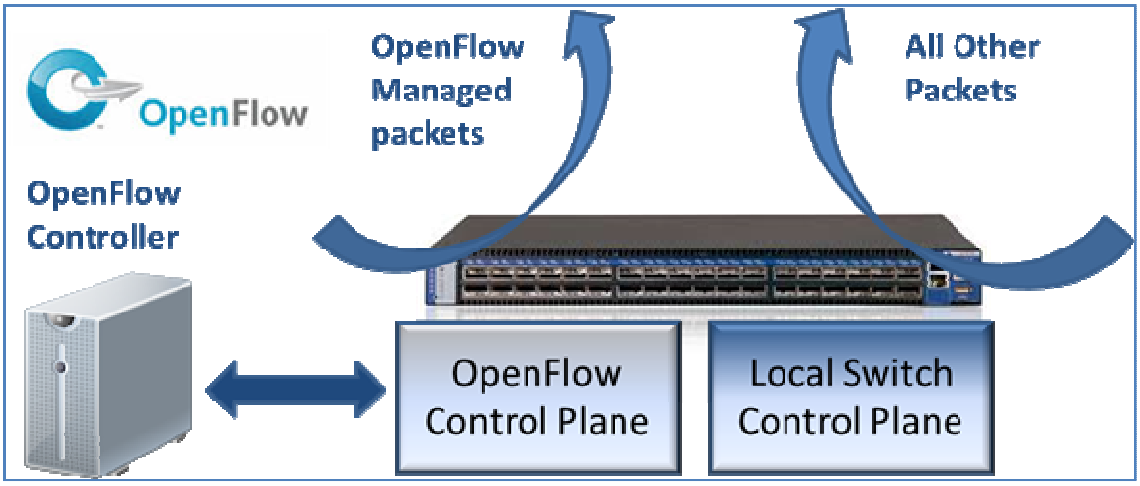

Figura 7: Exemplo de um switch OpenFlow híbrido [17].

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

4.1 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 [15] 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.

```
struct ofp_packet_queue {
  uint32_t queue_id; // Identificador da fila
  uint32_t port; // Porta à qual a fila está anexada
  uint16_t len; // Tamanho em bytes do descritor desta fila
  uint8_t pad[6]; // Alinhamento para 64 bits
  struct ofp_queue_prop_header properties[0]; // lista de propriedades
};
```

Figura 8: Descrição de uma estrutura de fila de pacotes do protocolo OpenFlow.

```
/* Conjunto de tipos de propriedades de uma fila */
enum ofp_queue_properties {
  OFPQT_MIN_RATE = 1, // Taxa de dados mínima garantida
  OFPQT_MAX_RATE = 2, // Taxa de dados máxima garantida
  OFPQT_EXPERIMENTER = 0xffff // Propriedade definida pelo experimentador
}

/* Cabeçalho com a descrição comum para uma fila */
struct ofp_queue_prop_header {
  uint16_t property; // Um dos valores do enum ofp_queue_properties
  uint16_t len; // Tamanho da propriedade, incluindo este cabeçalho
  uint8_t pad[4]; // Alinhamento de 64 bits
};
```

Figura 9: Propriedades de uma fila e cabeçalho comum usado para descrever uma fila de pacotes do protocolo OpenFlow.

A figura 8 apresenta uma estrutura básica de uma fila de pacotes que deve ser implementada por um controlador OpenFlow. Uma fila 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.

Já a figura 9 mostra uma enumeração que lista as propriedades possíveis de uma fila e o cabeçalho de uma estrutura que representará uma propriedade de uma fila. Esse cabeçalho deve indicar uma propriedade da enumeração `ofp_queue_properties`, o tamanho da propriedade e um campo fixo para alinhamento de 64 bits da estrutura.

E na figura 10 temos as descrições específicas de cada propriedade de fila oficialmente suportada pelo protocolo OpenFlow. As estruturas 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 usado 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.

```
/* Descrição da propriedade da fila de taxa mínima */
struct ofp_queue_prop_min_rate {
  struct ofp_queue_prop_header prop_header; // OFPQT_MIN_RATE
  uint16_t rate; // Em 1/10 de porcento; se > 1000, então é desativado
  uint8_t pad[6]; // Alinhamento de 64 bits
};

/* Descrição da propriedade da fila de taxa máxima */
struct ofp_queue_prop_max_rate {
  struct ofp_queue_prop_header prop_header; // OFPQT_MAX_RATE
  uint16_t rate; // Em 1/10 de porcento; se > 1000, então é desativado
  uint8_t pad[6]; // Alinhamento de 64 bits
};

/* Descrição da propriedade da fila do experimentador */
struct ofp_queue_prop_experimenter {
  struct ofp_queue_prop_header prop_header; // OFPQT_EXPERIMENTER
  uint32_t experimenter; // ID do experimentador;
  uint8_t pad[4]; // Alinhamento de 64 bits
  uint8_t data[0]; // Dados definidos pelo experimentador
};
```

Figura 10: Descrição das propriedades das filas usadas para implementar um suporte básico a QoS no protocolo OpenFlow.

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*.

Uma entrada na *Meter Table* (**medidor**) consiste em 3 componentes principais:

- Um **identificador do medidor**, implementado como um número inteiro de 32 bits sem sinal;
- Um ou mais **medidores de banda**, listados de forma não ordenada, usados para especificar a taxa de banda e a forma como os pacotes devem ser tratados.
- **Contadores**, atualizados quando os pacotes são processados por um medidor.

O protocolo indica que os valores válidos para identificadores de medidores devem começar em 1 e alcançar o valor máximo de 4.294.901.760 (0xFFFF0000). Valores acima de 0xFFFF0000 são reservados para medidores virtuais, que não podem ser associados a fluxos.

```
// Enumeração que lista identificadores de medidores
enum ofp_meter {
  OFPM_MAX = 0xffff0000, // Último valor válido para identificadores

/* Medidores Virtuais */
  OFPM_SLOWPATH = 0xfffffffd, // Medidor para datapath lento, se algum
  OFPM_CONTROLLER = 0xfffffffe, // Medidor para conexão com o controlador
  OFPM_ALL = 0xffffffff // Representa todos os medidores, no caso de
};                        requisições de estatísticas
```

Figura 11: Valores reservados de identificadores de medidores de banda no protocolo OpenFlow.

```
/* Enumeração listando os tipos de medidor de banda */
enum ofp_meter_band_type {
  OFPMBT_DROP = 1, // Descartar pacote
  OFPMBT_DSCP_REMARK = 2, // Remarcar campo DSCP no cabeçalho IP
  OFPMBT_EXPERIMENTER = 0xFFFF // Medidor de banda experimental
};

/* Cabeçalho com a descrição comum de um medidor de banda */
struct ofp_meter_band_header {
  uint16_t type; // Um dos tipos descritos em ofp_meter_band_type
  uint16_t len; // Tamanho em bytes deste medidor
  uint32_t rate; // Taxa para esta banda
  uint32_t burst_size; // Tamanho dos bursts
};
```

Figura 12: Tipos de medidor de banda e estrutura de um medidor de banda no protocolo OpenFlow.

Na figura 12, são listados os tipos que um medidor pode ter:

- `OFPMBT_DROP` define um limitador de taxa que descarta pacotes que excedem a taxa de banda indicada;
- `OFPMBT_DSCP_REMARK` indica que o campo DSCP (*Differentiated Services Code Point*, usado para classificar pacotes) no cabeçalho IP de pacotes que excedem a taxa de banda indicada deve ser remarcado;
- `OFPMBT_EXPERIMENTER` é usado para fins experimentais.

E na figura 13 temos as estruturas que definem os tipos de medidor de banda. Nas três estruturas há um campo `burst_size`, usado para aplicar o medidor em pacotes ou rajadas (*bursts*) de bytes que alcancem o valor indicado neste campo (em kilobits ou número de pacotes).

```
/* Descrição do medidor que descarta pacotes  */
struct ofp_meter_band_drop {
  uint16_t type; // Tipo = OFPMBT_DROP
  uint16_t len; // Tamanho em bytes deste medidor de banda
  uint32_t rate; // Taxa para descartar pacotes
  uint32_t burst_size; // Tamanho dos bursts
  uint8_t pad[4]; // Usado para alinhamento de byte
};

/* Descrição do medidor que remarca o campo DSCP no cabeçalho IP */
struct ofp_meter_band_dscp_remark {
  uint16_t type; // Tipo = OFPMBT_DSCP_REMARK
  uint16_t len; // Tamanho em bytes deste medidor de banda
  uint32_t rate; // Taxa para remarcar pacotes
  uint32_t burst_size; // Tamanho dos bursts
  uint8_t prec_level; // Número de nível precedência a ser subtraído
  uint8_t pad[3]; // Usado para alinhamento de byte
};

/* Descrição do medidor definido pelo experimentador */
struct ofp_meter_band_experimenter {
  uint16_t type; // Tipo de banda
  uint16_t len; // Tamanho em bytes desta banda
  uint32_t rate; // Taxa para esta banda
  uint32_t burst_size; // Tamanho dos bursts
  uint32_t experimenter; // Identificador do experimentador
};
```

Figura 13: Definição dos tipos de medidor de banda no protocolo OpenFlow.

5. PARAMETRIZAÇÃO DE MONITORAÇÃO DE REDES

Este capítulo descreverá o desafio de utilização de parâmetros para monitoração de QoS em redes OpenFlow. Desta forma o presente trabalho apresenta um mecanismo de QoS formalizado pelo protocolo OpenFlow que foi implementado Floodlight.

5.1 AMBIENTE UTILIZADO

Para realizar a análise do controlador Floodlight foi utilizado um notebook com um processador Intel Core i5 2450M, com 2,5 GHz de clock, 4 gigabytes de memória e disco rígido de 1 terabyte, rodando o sistema operacional Windows 8.1 de 64 bits.

A versão utilizada do Mininet foi a 2.1.0, em uma máquina virtual no VirtualBox rodando Ubuntu 14.04 de 32 bits. A máquina virtual usa um disco virtual com 8 gigabytes de capacidade e 1,5 gigabytes de memória. O Floodlight foi baixado, compilado e executado na mesma máquina virtual do Mininet para a realização dos experimentos. Foi usada a IDE Eclipse 3.7.2 (Indigo) para manipular o código do controlador Floodlight.

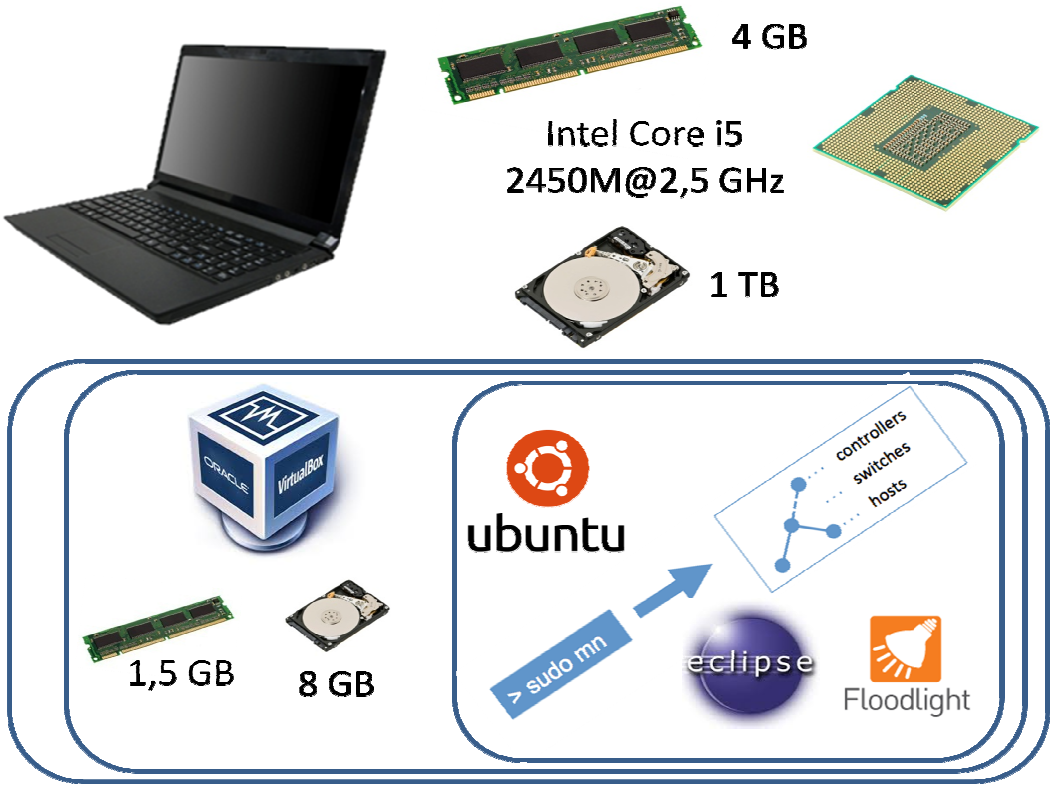

Figura 14: Ambiente utilizado na análise do controlador Floodlight.

5.2 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.

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`, equivalente à estrutura `ofp_packet_queue`), propriedades de fila (`OFQueueProperty` e `OFQueuePropertyType`, equivalentes às estruturas `ofp_queue_prop_header` e `ofp_queue_properties`, respectivamente) e as duas propriedades definidas pelo padrão OpenFlow, que são Minimum Rate (classe `OFQueuePropertyMinRate`, equivalente à estrutura `ofp_prop_min_rate`) e Maximum Rate (classe `OFQueuePropertyMaxRate`, equivalente à estrutura `ofp_prop_max_rate`).

O pacote `org.openflow.protocol.meter` implementa os medidores de banda.

As figuras 15, 16, 17 e 18 exemplificam o fluxo de trabalho de um switch OpenFlow controlado pelo Floodlight. Na parte superior esquerda temos o controlador

Floodlight, que interage com o switch através de um canal de comunicação seguro definido pelo OpenFlow. Dentro do switch, temos uma tabela de fluxos, simplificada apenas para fins de exemplo, com 4 colunas:

- *Match fields*: consolida todos os critérios para classificação de pacotes (conforme mostrado na seção 2.2);
- *Priority*: usado caso um pacote atenda os critérios de classificação de múltiplas entradas da tabela de fluxo. Apenas a entrada com maior prioridade será aplicada sobre o pacote;
- *Counter*: registra o número de pacotes que atenderam os critérios da respectiva entrada na tabela de fluxos;
- *Action*: lista a ação que será efetuada sobre pacotes que atendem os critérios da respectiva entrada na tabela de fluxos.

Na parte inferior das figuras 15, 16, 17 e 18, temos as portas do switch, representadas por setas bidirecionais nomeadas P1, P2 e P3. As elipses acima das setas representam as filas (*Queues*) de pacotes usadas para suportar QoS e que são implementadas pelo Floodlight. Dentro de cada elipse há um quadrado que representa uma Meter Table associada àquela fila.

A tabela de fluxos inicialmente possui duas regras que foram configuradas de forma proativa pelo controlador Floodlight. A primeira regra indica que todo pacote que chegar pela porta P1 (`srcPort == P1`) deve ser colocado na fila Q2 (`SetQueue(Q2)`), que está associada à porta P2. Já a segunda regra indica o que o switch deve fazer quando um pacote não atende os critérios de nenhuma entrada na tabela de fluxos (`miss`). Neste caso, o pacote é encaminhado ao controlador SDN (`SendToController`).

Na figura 15, um pacote (representado por uma elipse “Pkt”) está chegando através da porta P1 do switch. Esse pacote atende os critérios da primeira entrada da tabela, pois ele chegou pela porta P1. Logo, conforme exibido na figura 16, o pacote é colocado na fila Q2, associada à porta P2 do switch e o campo *Counter* da respectiva entrada na tabela de fluxos é incrementado. Algum tempo depois, o pacote é então reencaminhado pela rede.

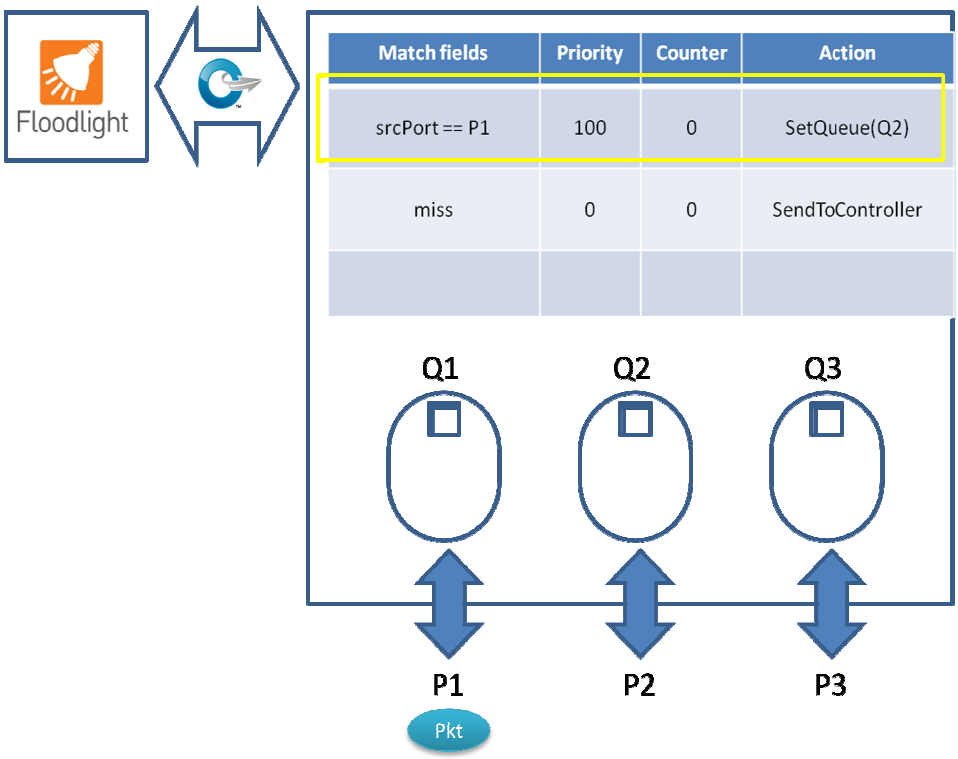

Figura 15: Pacote entrando pela porta P1 em um switch OpenFlow.

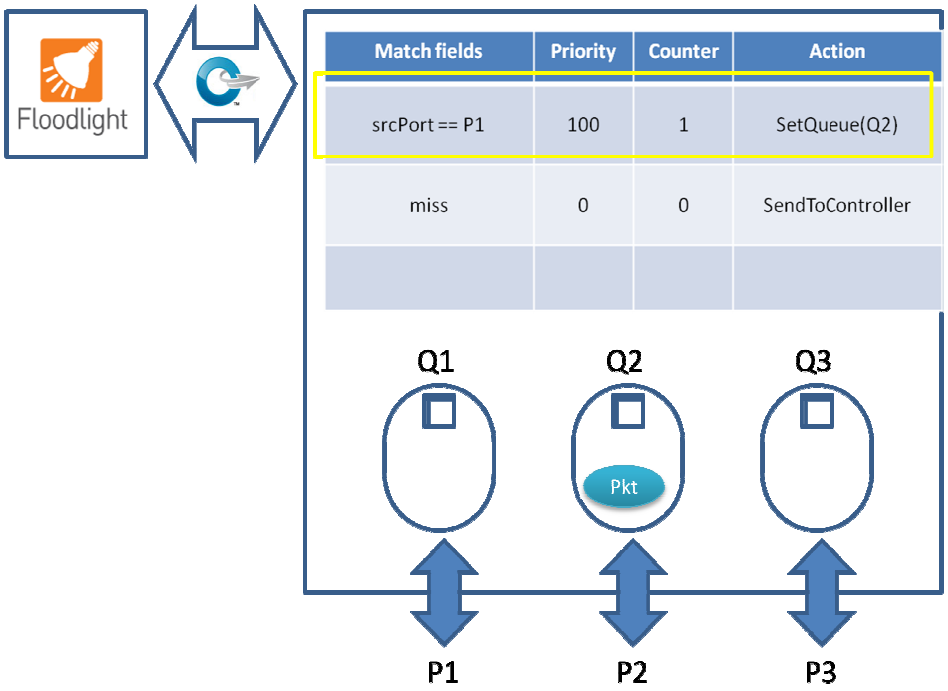

Figura 16: Pacote sendo enfileirado na fila Q2 para encaminhamento através da porta P2 de um switch OpenFlow.

Já nas figuras 17 e 18 é mostrado o que acontece quando um pacote não atende critérios de nenhuma entrada na tabela de fluxos. Na figura 17, outro pacote (elipse "Pkt") chega pela rede através da porta P3 do switch. No entanto há somente uma regra definida para pacotes que chegam através da porta P1.

Quando um pacote não atende critérios de nenhuma entrada na tabela de fluxos, é dito que houve um *table miss*. O controlador SDN pode configurar uma entrada na tabela de fluxos para tratar *table misses*, mas não é obrigatório. Por padrão, o protocolo OpenFlow especifica que, se uma entrada para *table misses* não for definida explicitamente pelo controlador SDN, o pacote que não atender nenhum critério especificado na tabela de fluxos será automaticamente descartado.

Como o controlador SDN configurou uma entrada (`miss`) tratando *table misses*, o pacote deve ser encaminhado ao controlador (`SendToController`).

O pacote é encapsulado e enviado ao Floodlight através do canal de comunicação segura estabelecido pelo OpenFlow. No controlador, o conteúdo do pacote é analisado e então é definida uma nova regra para pacotes que chegarem pela porta P3 do switch.

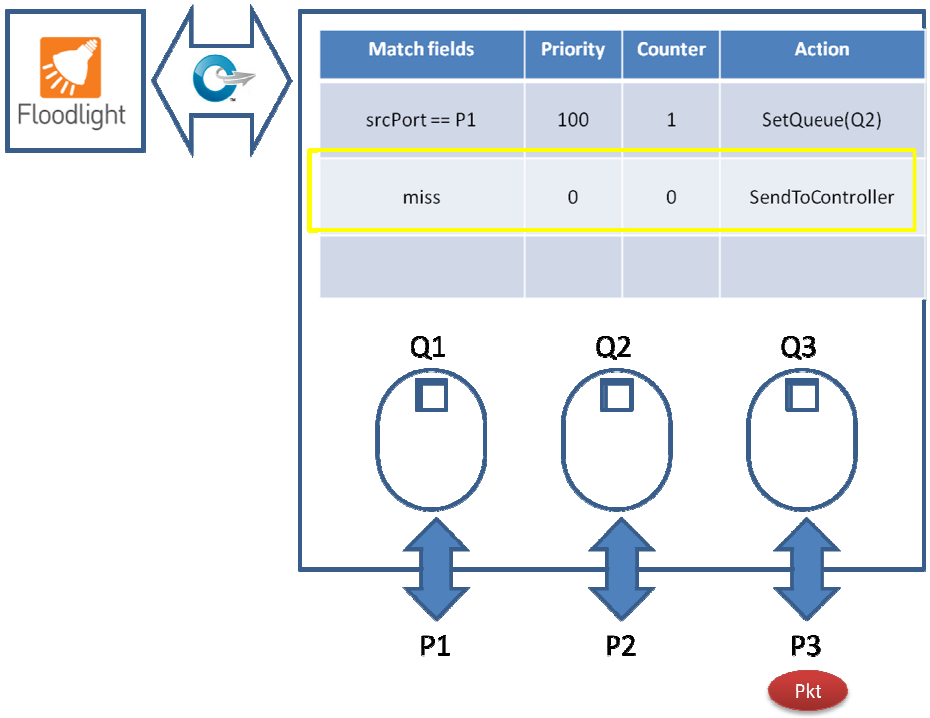

Figura 17: Pacote sem regra definida na tabela de fluxos entrando pela porta P3 em um switch OpenFlow.

A nova regra definida na figura 18 indica que pacotes que chegarem pela porta P3 deve ser encaminhados para a porta P2. O campo *Counter* da linha com a nova regra é então incrementado.

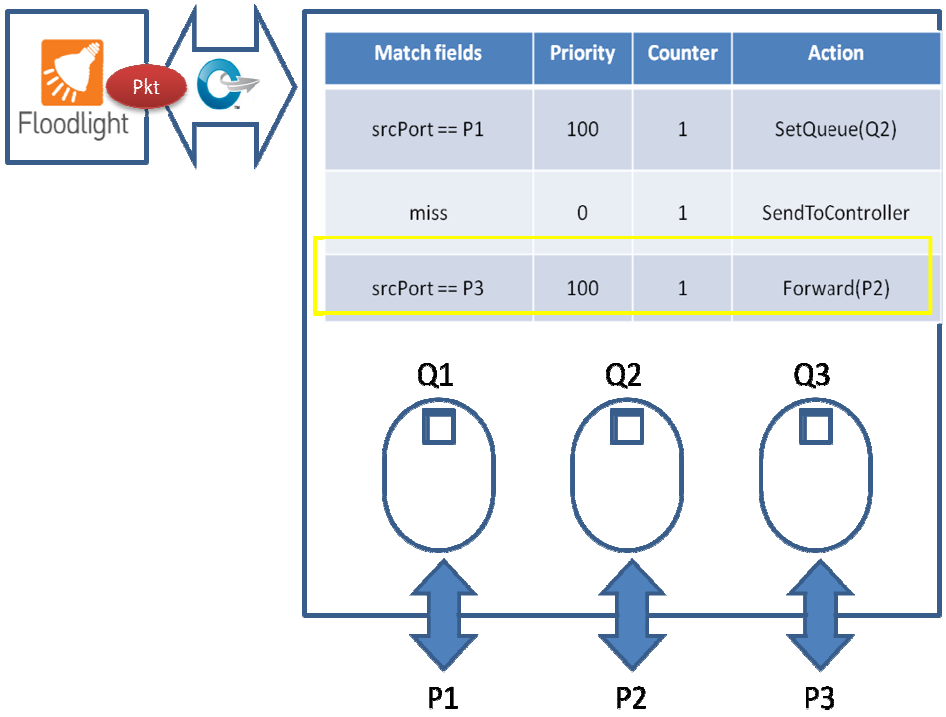

Figura 18: Pacote sem regra definida na tabela de fluxos enviado para o controlador Floodlight.

5.3 CONSIDERAÇÕES SOBRE QoS NO FLOODLIGHT

O controlador Floodlight implementa de forma elegante as propriedades de *Queue* (`OFQueuePropertyMinRate` e `OFQueuePropertyMaxRate`) e de *Meter Band* (`OFMeterBandDrop` e `OFMeterBandDSCPRemark`), indicadas pela especificação do protocolo OpenFlow.

No contexto do pacote que lida com a implementação das *Queues* nas portas de um switch OpenFlow, a propriedade `Experimenter` (renomeada para `Vendor` no contexto do controlador Floodlight) ainda não é suportada oficialmente pelo Floodlight. Na classe `OFQueueProperty`, que define os tipos de propriedades das filas de pacotes, o trecho de código em que é definida a propriedade `Vendor` se encontra dentro de um bloco comentado de código. A classe que implementa o tipo de propriedade de fila `Vendor`, de nome `OFQueuePropertyVendor`, não está presente.

E no contexto da implementação dos medidores de banda, ocorre algo similar ao que acontece com as propriedades de um fila de pacotes: o código que define o tipo `Experimenter` (denominado `Vendor` no contexto do Floodlight) na classe `OFMeterBandType` está num bloco comentado e a respectiva classe que implementa essa propriedade, denominada `OFMeterBandVendor`, também não está presente.

Com a ausência do suporte à propriedade `Experimenter` no controlador Floodlight, a monitoração de parâmetros de QoS fica resumida ao que é oferecido pelo protocolo OpenFlow, baseado em taxas mínima e máxima de vazão, utilizando a política *DiffServ*.

6. 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.

Seria interessante no futuro, como alternativa ao simulador Mininet, usar o EstiNet [20] para abodar a simulação de SDN e avaliar os resultados dos experimentos.

REFERÊNCIAS

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

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

[3] 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

[4] 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.

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

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

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

[8] **Open Networking Foundation overview**. Disponível em https://www.opennetworking.org/about/onf-overview . Acessado em 05/2014.

[9] 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

[10] 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 .

[11] DANTAS, Mario. **Redes de comunicação e computadores: abordagem quantitativa.** Florianópolis: Visual Books, 2010.

[12] TANENBAUM, Andrew S.; WETHERALL, David. **Redes de computadores.** 5. ed. São Paulo: Pearson Prentice Hall, 2011.

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

[14] **Learn REST: A Tutorial**. Disponível em http://rest.elkstein.org/. Acessado em 09/2014.

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

[16] **Technical Papers – Open Networking Foundation**. Disponível em https://www.opennetworking.org/sdn-resources/sdn-library/technical-papers. Acessado em 09/2014.

[17] **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.

[18] **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.

[19] 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*.

[20] 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

[21] Ryan Wallner; Robert Cannistra, "**An SDN Approach: Quality of Service using Big Switch's Floodlight Open-source Controller**," *Proceedings of the Asia-Pacific Advanced Network 2013* v. 35, p. 14-19. http://dx.doi.org/10.7125/APAN.35.2 ISSN 2227-3026

[22] **Request For Comments (RFC) 7047: The Open vSwitch Database Management Protocol**. Disponível em https://tools.ietf.org/html/rfc7047. Acessado em 10/2014.

[23] **Request For Comments (RFC) 6241: Network Configuration Protocol (NETCONF)**. Disponível em https://tools.ietf.org/html/rfc6241. Acessado em 10/2014.

[24] **Request For Comments (RFC) 1157: A Simple Network Configuration Protocol (SNMP)**. Disponível em https://tools.ietf.org/html/rfc1157. Acessado em 10/2014.

[25] **Technical Library – Open Networking Foundation – Project: OF-CONFIG**. Disponível em https://www.opennetworking.org/sdn-resources/technical-library. Acessado em 10/2014.