UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

AIRTON NOBUMASA ISHIMORI

QOSFLOW: CONTROLE AUTOMÁTICO DE QUALIDADE DE SERVIÇO EM REDES OPENFLOW

BELÉM-PA

Fevereiro / 2013

AIRTON NOBUMASA ISHIMORI

QOSFLOW: CONTROLE AUTOMÁTICO DE QUALIDADE DE SERVIÇO EM REDES OPENFLOW

Dissertação submetida à banca julgadora na Universidade Federal do Pará como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Antônio J. G. Abelém

BELÉM-PA

Fevereiro / 2013

AIRTON NOBUMASA ISHIMORI

QOSFLOW: CONTROLE AUTOMÁTICO DE QUALIDADE DE SERVIÇO EM REDES OPENFLOW

Dissertação submetida à banca julgadora na Universidade Federal do Pará como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. Antônio Jorge Gomes Abelém
Universidade Federal do Pará
Orientador

Prof. Dr. Agostinho Luiz da Silva Castro
Universidade Federal do Pará
Membro Externo

Prof. Dr. Luciano Paschoal Gaspary
Universidade Federal do Rio Grande do Sul
Membro Externo

A mim e aos meus familiares.

Agradecimentos

Agradeço toda minha família pelos momentos de lazer, confiança, incentivos e todo o apoio recebido, de várias formas e em todos os momentos

Agradeço aos amigos por todos os momentos de diversão e pelas sugestões recebidas ao longo de toda essa jornada.

Agradeço, em especial, aos amigos do laboratório GERCOM por todas as conversas informais ou formais, as quais foram todas com certeza de grande valia para o amadurecimento e a finalização deste trabalho de dissertação.

Agradeço ao órgão de fomento à pesquisa CNPQ pelos recursos financeiros recebidos, que permitiu realizar o curso do mestrado com bolsa no Programa de Pós-Graduação em Ciência da Computação (PPGCC) - UFPA durante esses longos anos de mestrado, 2011 a 2013.

Agradeço ao PROCAD que possibilitou a realização do mestrado sanduíche na UFRGS, onde pude conviver com novas pessoas e criar novos laços de amizade, além de aprimorar meus conhecimentos da área. Agradeço especialmente aos professores Luciano Gaspary e Lisandro Granville por todo apoio recebidos.

Gostaria de agradecer especialmente ao Fernando Farias, atual gerente do subgrupo de pesquisas em Internet do Futuro/GERCOM por toda ajuda e, também agradeço especialmente ao meu orientador, professor Antônio Jorge Gomes Abelém, por ter me aceitado no programa de mestrado, pela confiança e por todas as orientações e apoio recebidos nessa jornada.

Nas últimas palavras, gostaria de agradecer (embora não estejam citados aqui, pois a lista é enorme) a todas as pessoas que me incentivaram, apoiaram e contribuiram de forma, direta ou indireta, para a realização deste trabalho e para minha formação acadêmica. Muito obrigado!! :)

Resumo

Resumo da Dissertação apresentada à UFPA como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

QoSFlow: Controle Automático de Qualidade de Serviço em Redes OpenFlow

Orientador: Prof. Dr. Antônio J. G. Abelém
Palavras-chave: OpenFlow, QoS, Escalonadores de Pacotes, QoSFlow

O OpenFlow é um protocolo que habilita o uso do modelo das Redes Definidas por Sofware e possibilita que aplicações de rede possam orquestrar os equipamentos de comutação. Este protocolo torna a rede dinâmica e flexível para estabelecer rotas, criar circuitos e coletar informações de estado dos comutadores. Entretanto, controle de Qualidade de Serviço (QoS) é primitiva, pois o administrador da rede fica na dependência da ferramenta de linha comando do OpenFlow chamada `dpctl`. A falta de automação para realizar o controle de QoS não reduz a sobrecarga do custo operacional por parte do administrador, aumenta a instabilidade da rede devido a possíveis erros do operador, além de não permitir a configuração e atualização de parâmetros de QoS rapidamente. Além disso, o OpenFlow suporta somente escalonador de pacotes FIFO (*First-In First-Out*). Portanto, este trabalho apresenta a proposta QoSFlow, um arcabouço de *software* capaz de diferenciar tráfegos em múltiplas filas e controlar diferentes tipos de escalonadores de pacotes do *kernel* do Linux em alto nível de abstração, além de oferecer um conjunto de interfaces de programação de QoS. A avaliação dos resultados obtidos nos experimentos realizados com QoSFlow apresentaram-se satisfatórios, pois a proposta garantiu isolamento de largura de banda para fluxos TCP e UDP. Além disso, a utilização de diferentes tipos de escalonadores de pacotes, além do tradicional FIFO, apresentaram melhoria substancial no desempenho das aplicações, que se refletiram em ganho na Qualidade de Experiência (QoE).

Abstract

Abstract of the Dissertation presented to UFPA as a partial fulfillment of the requirements for the degree of Master in Computer Science.

QoSFlow: Automatic Quality of Service Control on OpenFlow Networks

Advisor: PhD. Prof. Antônio Jorge Gomes Abelém
Keywords: OpenFlow, QoS, Escalonadores de Pacotes, QoSFlow

OpenFlow is a Software-Defined Networking protocol to allows network applications to orchestrate network devices. The protocol enables management facilities and it allows to collect network information state from OpenFlow devices. But, the OpenFlow proposal does not offer support for dynamic Quality of Service (QoS) control. Because, the network administrator must use a Command Line Interface (CLI) called `dpctl` from OpenFlow utility. The lacks of automation to control QoS do not reduce the operational costs, increases the network instability, and the network operator is not able to configure QoS faster. Furthermore, OpenFlow only supports First-In First-Out (FIFO) sheduling. Therefore, this work presents the QoSFlow proposal. It is a framework to enable Linux kernel packet schedulers control. Moreover, it offers a set of QoS programming interfaces at the control plane. The evaluation shows good performance results and the proposal is able to give network isolation for TCP and UDP flows. In addition, SFQ scheduling shows improvement on Quality of Experience (QoE).

Sumário

Lista de Abreviaturas

VoIP	*Voice Over IP*
P2P	*Peer-to-Peer*
RSVP	*Resource reSerVation Protocol*
SDN	*Software-Defined Networks*
API	*Application Interface Programming*
FIFO	*First-In First-Out*
ARP	*Address Resolution Protocol*
IANA	*Internet Assigned Numbers Authority*
OXM	*OpenFlow Extensible Match*
L2	*Layer 2*
L3	*Layer 3*
L4	*Layer 4*
SFQ	*Stochastic Fairness Queuing*

RED	*Random Early Detection*
HTB	*Hierarchical Token Bucket*
PFIFO	*Packet First-In First-Out*
BFIFO	*Bytes First-In First-Out*
PID	*Process ID*
QoE	*Quality of Experience*
PSNR	*Peak Signal-to-Noise Ratio*
MPEG-4	*Moving Picture Experts Group - 4*

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

1.1 A Internet e as Redes Definidas por Software

A Internet, ao longo dos anos, vem sofrendo muitas adaptações em sua arquitetura para suprir as necessidades administrativas e dos usuários. Essa rede de escala global integra várias redes de domínios autoritativos espalhados no mundo todo. No entanto, antes disso, as suas primeiras formas de comunicação ocorriam entre algumas dezenas de computadores interconectados para fins puramente acadêmicos e experimentais. Atualmente a Internet é um meio de comunicação importante para todos os tipos de usuários os quais utilizam para diversas finalidades. Nesse contexto, tem-se observado um surgimento de diversos tipos de serviços juntamente com o aumento da demanda de usuários na Internet.

Os serviços multimídia, como serviços de vídeo sob demanda (*Video on Demand*, VoD), serviço de voz sobre IP (*Voice over IP*, VoIP), serviços de vídeos 3D e de alta definição, contribuem fortemente para o processo de crescimento da demanda de acesso à Internet, sendo um dos responsáveis pela grande quantidade de tempo dispensada na Internet. Embora os serviços não-multimídia sejam essencias no cotidiano de muitas pessoas, como serviços de email, serviços par-a-par (*Peer-to-Peer*, P2P), serviços de transferência de arquivos e serviços para interação social, a exemplo do Facebook. Todos esses serviços possivelmente coexistirão nas próximas gerações da Internet, juntamente com outros tipos de serviços que poderão surgir pela frente.

Ao longo dos anos, muitos trabalhos [3, 4, 5, 6] vieram sendo propostos na tentativa de criar mecanismos para tornar o provisionamento de QoS dinâmico nas tradicionais redes de computadores. Além desses, ainda houve trabalhos mais antigos que focaram no contexto de roteamento e arquiteturas para suporte a QoS, como o IntServ [7] e Diff-

Serv [8] da IETF, ambos garantindo um nível de QoS de modo a atender aos requisitos das aplicações. A primeira utiliza o protocolo RSVP (*Resource reSerVation Protocol*) para os comutadores sinalizarem reservas de recurso, dessa forma, alocando sob-demanda os recursos para cada fluxo nos comutadores da rede, enquanto que na segunda forma, os recursos são alocados e os fluxos são agregados em classes definidas pela arquitetura. No entanto, ainda com relação ao IntServ e DiffServ, a configuração e a manutenção de parâmetros de QoS nos dispositivos eram relativamente difíceis e manuais, sujeitos a numerosos erros administrativos [9, 10].

As pesquisas em Internet do Futuro (IF) incluem, entre seus principais objetivos, a formulação de propostas que ofereçam suporte a QoS. As redes tornaram-se parte da infraestrutura crítica das empresas, das cidades e dos campi [11] e, com o aumento dos serviços que oferecem conteúdo multimídia, novas adaptações na arquitetura da Internet são necessárias, a fim de melhorar o provisionamento de QoS. A garantia da qualidade de serviço para aplicações de vídeo, voz e dados em geral são necessários para atender aos requisitos mínimos de atraso, variação do atraso (*jiiter*), largura de banda e taxas de perdas de pacote, de maneira diferenciada para cada tipo de aplicação.

Algumas frentes de pesquisas em IF afirmam que o *backbone* da Internet chegou a um estado de "engessamento" [12], o que torna a execução de adaptações no núcleo da rede nociva ao desempenho da mesma. Diante deste desafio, desenvolvimento de ambientes para experimentação, como o GENI [13], vem sendo alvo de pesquisas. Tais ambientes são utilizados por pesquisadores de rede para validação de novas propostas de protocolos ou testes de novas aplicações. No entanto, esses ambientes necessitam de arcabouços de controle, como ProtoGENI [14], para gerenciar os recursos que são compartilhados entre os pesquisadores do mundo todo. Como uma das propostas de trabalho para o GENI, surgiu a proposta do protocolo OpenFlow [11], que oferece uma outra alternativa ao experimentador, pois ele pode utilizar uma rede de produção (ex., campi) para fins de experimentação sem prejudicar o tráfego da mesma.

O OpenFlow pertence a um paradigma de rede onde o comportamento dos comutadores são reprogramáveis dinamicamente, independente do tipo de fabricante, através de aplicações desenvolvidas e executadas em um servidor remoto, modelo este conhecido pelo nome de Redes Definidas por *Software* (*Sofware-Defined Networks*, SDN) [2]. Nesse paradigma, o plano de controle, responsável pela lógica da tomada de decisão sobre os fluxos de pacotes são desacoplados do plano de dados, responsáveis pelo processamento das funções de envio e recepção dos pacotes. Por exemplo, a decisão do próximo salto de um pacote é tomada no servidor remoto, que por sua vez, insere uma ação no comutador para encaminhar aquele pacote por uma determinada porta.

No jargão do OpenFlow, o servidor remoto e o substrato da rede são conhecidas por controlador OpenFlow e *datapath* OpenFlow, respectivamente. Existem diversos controladores OpenFlow implementados atualmente, citam-se como exemplos o NOX [15], baseado em C++, o Beacon [16] e Floodlight [17], baseados em Java e o POX [18], baseado em Python, que oferecem interfaces de programação (*Application Interface Programming*, API) OpenFlow para que os programadores de rede possam desenvolver suas aplicações

de controle e monitoramento da rede. Essas aplicações são executadas sobre os controladores, sendo que elas podem possuir conhecimento a respeito do estado da rede (ex., saturação, interfaces ativas) e/ou sobre informações de endereço MAC, endereço IP, tipo de serviço (TCP/UDP), VLAN ID e a porta da aplicação, para então definir uma ação de encaminhamento ou descarte dos pacotes.

No entanto, apesar do OpenFlow ser emergente tanto no meio acadêmico quanto nas indústrias, a abordagem utilizada para o controle de recursos é primitiva, visto que para sua configuração é necessário o uso da ferramenta que executam instruções por meio de linhas de comando chamada `dpctl`. Neste mecanismo é alto o nível de intervenção do operador, implicando em configurações de natureza estática, sujeitas a maiores chances de falhas. Em ambientes com diferentes tipos de serviços ofertados na rede, ou seja, em redes altamente dinâmicas, é vital a adoção de mecanismos automatizados de QoS para atender aos requisitos das aplicações, dos usuários e da administração.

Neste trabalho é apresentado o QoSFlow, uma proposta de controle de QoS em redes OpenFlow. Ele permite um controle sobre escalonadores de pacotes (disciplinas de filas) do *kernel* do Linux, os quais controlam a transmissão dos pacotes nas filas da porta de saída. Além disso, o QoSFlow também oferece API de QoS, de modo que as aplicações de QoS possam ser desenvolvidas para realizar reprogramação dinâmica, por exemplo, de largura de banda nas portas dos dispositivos de comutação.

1.2 Motivação e Justificativas

O OpenFlow é a primeira interface de comunicação aberta definido entre a camada de controle e dados do modelo SDN [2]. Ele abstrai as configurações de baixo nível, possibilitando que as configurações que antes eram manuais, realizadas por ferramentas específicas dos fabricantes de equipamentos (*switches*/roteadores), tornem-se agora, independente do tipo de fabricante. No entanto, apesar de ser um protocolo emergente, tanto no meio acadêmico quanto nas indústrias, as configurações de parâmetros de QoS não se encontra adequado as necessidades administrativas das redes atuais.

Atualmente, para controlar a largura de banda para um tipo de serviço, para uma aplicação ou para um usuário, é requerido por parte do administrador, uma habilidade de manuseio da ferramenta que acompanha o OpenFlow chamada `dpctl`. Através dessa ferramenta, é possível criar fatias de largura de banda (filas) na rede, o que garante um isolamento de tráfego, todavia não garante automação do controle de forma dinâmica e eficiente. Além disso, o OpenFlow oferece suporte a um único tipo de disciplina de fila na transmissão dos pacotes, conhecida como FIFO (*First-In First-Out*), a qual possui comprimento estático. No entanto, esse tipo de escalonador de pacotes geralmente não é suficiente para algumas situações da rede, como o congestionamento, por exemplo.

A falta de automação para realizar o controle de banda não reduz a sobrecarga do custo operacional por parte do administrador, aumenta a instabilidade na rede devido a possíveis erros do operador, além de não permitir a configuração e atualização

de parâmetros de QoS rapidamente. Em resumo, é possível notar um certo nível de complexidade e tempo de operação longa para controlar recursos disponíveis no domínio OpenFlow.

Além desses problemas, desde as suas primeiras versões, muitos trabalhos foram propostos para melhorar outros problemas do OpenFlow, sendo que algumas dessas propostas abrangem melhorias no plano de controle do OpenFlow, na tentativa de diminuir a sobrecarga ocasionada pela excessiva troca de mensagens de controle entre o dois planos (dados e controle) [19, 20, 21], além da proposta que integra MPLS no OpenFlow [22]. Em outras visões, o OpenFlow é um protocolo emergente que está passando por uma fase de adaptação na sua especificação. Portanto, este trabalho é mais uma contribuição na tentativa de agregar melhoria para o OpenFlow no contexto de QoS.

1.3 Objetivos

O objetivo geral da proposta é melhorar o mecanismo de controle de QoS em redes OpenFlow, oferecendo suporte ao controle de múltiplos escalonadores de pacotes do *kernel* do Linux. Isto inclui:

- **Automatização do controle de QoS.** Automatizar as configurações de QoS em ambientes físicos e virtuais. A implementação atual do OpenFlow necessita do comando `dpctl` para garantir recursos de rede.
- **Monitoramento de QoS.** A proposta permite que o administrador colete informações de QoS através de uma aplicação no controlador. Essas estatísticas podem ser informações relacionadas tanto às filas associadas ou às portas da interface do roteador/*switch*.
- **Controle de escalonadores de pacotes.** Possibilita ao administrador da rede associar um escalonador de pacotes nas classes de filas. Esses escalonadores garantem um comportamento diferenciado na transmissão dos pacotes.
- **API de QoS.** Possibilita o desenvolvimento de aplicações que lidam com o controle de banda de rede de forma automática, ou seja, sem a necessidade ou com pouca intervenção do operador da rede.

1.4 Organização do Trabalho

O Capítulo 2 tem o propóstito de revisar o OpenFlow. O capítulo apresenta os principais conceitos teóricos sobre OpenFlow, seu funcionamento, alguns tipos de mensagens e as especificações do protocolo.

O Capítulo 3 apresenta um estudo do estado da arte de QoS no contexto do OpenFlow. O capítulo aborda, em linhas gerais, as propostas de QoS para OpenFlow existentes na literatura, diferenças entre eles e um quadro de resumo das propostas apresentadas.

O Capítulo 4 apresenta o QoSFlow, um arcabouço de *software* que visa suprir as deficiências de controle de QoS que o protocolo OpenFlow apresenta em seu estado atual. Isto é, o capítulo apresenta uma proposta para remover a necessidade das configurações manuais de QoS na rede OpenFlow.

O Capítulo 5 apresenta todos os equipamentos dos testes que foram utilizados para avaliar o QoSFlow e os resultados que foram obtidos.

O Capítulo 6 apresenta as principais conclusões a respeito da proposta QoSFlow. Pontos positivos e negativos, além de apresentar trabalhos futuros.

CAPÍTULO 2

Referencial Teórico

Este capítulo tem o propóstito de revisar o OpenFlow. O capítulo apresenta os principais conceitos teóricos sobre OpenFlow, seu funcionamento, alguns tipos de mensagens e as especificações do protocolo.

2.1 OpenFlow

O OpenFlow é uma proposta de SDN que possibilita o desenvolvimento de aplicações para orquestrar os equipamentos de comutação da rede. Sua arquitetura é composta por três partes conhecidas por (1) *switch* OpenFlow, um equipamento de comutação aberto; (2) controlador OpenFlow, um servidor centralizado para o controle da rede e (3) protocolo de comunicação OpenFlow que faz a comunicação entre as duas entidades (*switch* e controlador OpenFlow). A Figura 1 apresenta a arquitetura do OpenFlow.

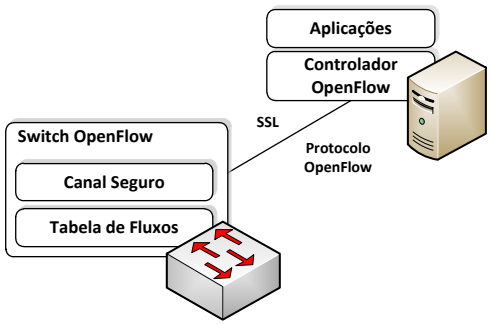


Figura 1: Entidades do OpenFlow. Fonte adaptada da especificação 1.0 do OpenFlow [1].

O OpenFlow desacopla a função de envio e recepção dos pacotes da função lógica que define o encaminhamento das mesmas. Essa responsabilidade é inserido em um servidor remoto, que atua como um "orquestrador" da rede. Logo, sua arquitetura fica dividida em dois planos conhecidas por plano de controle e plano de dados.

No plano de dados é projetado uma **tabela de fluxos** para ser manipulada pela entidade controladora. Nessa entidade (plano de controle) a computação lógica é centralizada e os comandos de controle são transmitidos por mensagens padrões do protocolo para os dispositivos, por onde se pode realizar as funções básicas do modo tradicional da computação distribuída, tais como receber e encaminhar pacotes, atualizar as entradas da tabela e consultar o estado dos dispositivos.

Um dos objetivos do OpenFlow é possibilitar aos pesquisadores executar seus experimentos sobre uma rede de produção para obtenção de resultados mais realísticos, se comparados com ambientes de simulação. O sucesso para um bom desempenho das redes depende dos pesquisadores que propõem inovações. No entanto, segundo os desenvolvedores da proposta OpenFlow [11], apesar dos trabalhos feitos por pesquisadores de rede serem relevantes, as chances de os mesmos causarem impactos no mundo real, são mais remotos. A redução desse impacto, se deve a presença de uma quantidade enorme de equipamentos e protocolos instalados nos equipamentos, além da presença de tráfegos de produção que dificultam a experimentação em ambientes reais.

O OpenFlow pode ser comparado a um conjunto de instruções de um processador. Tal protocolo especifica primitivas básicas (mensagens do protocolo) que são usadas na programação dos dispositivos da rede, assim como um conjunto de instruções de um processador que programa um sistema de computação [2].

O *switch* OpenFlow é projetado para ser simples, de padrão aberto e independente de fabricantes. Diferente das "caixas pretas" de rede produzidas pelos fabricantes, que possuem suas próprias especificações e sua implementações mantidas em sigilo. Além disso, os detalhes da implementação interna dos equipamentos são diferentes entre os fabricantes. A Figura 2 ilustra a comunicação do plano de controle com o plano de dados do OpenFlow.

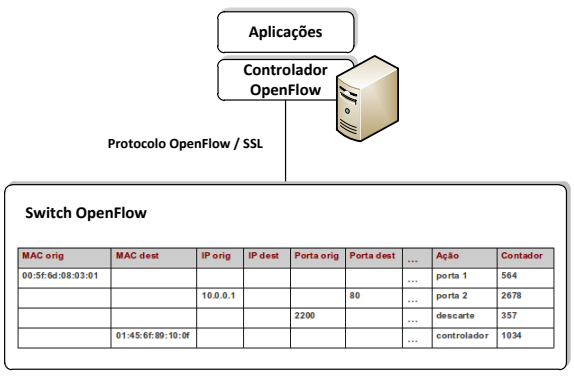


Figura 2: Tabela de fluxos do *switch* OpenFlow. Fonte adaptada de ONF [2].

A tabela de fluxos contém um conjunto de entradas de fluxos que são **campos do cabeçalho** de um pacote. Esses campos serão usados para fazer a comparação com os pacotes que atravessam os dispositivos. Além dos campos de cabeçalho, a tabela de fluxos possui **contadores** de atividades e uma lista **ações** associadas para serem aplicadas nos pacotes.Todos os pacotes processados pelo *switch* são comparados com a tabela de fluxos e basicamente, duas situações podem ocorrer:

- Caso uma comparação com a entrada seja encontrada, as ações são aplicadas no pacote (ex.: ação de encaminhar o pacote para uma porta de saída);
- Caso contrário, se nenhuma comparação com a entrada seja encontrada, o pacote é encaminhado para o controlador sobre o canal seguro. O controlador é responsável em determinar como tratar os pacotes.

As Figuras 3 e 4 apresentam, em maiores detalhes, as etapas lógicas que os pacotes enfrentam ao entrarem nos *switches* OpenFlow. A Figura 3, representa um pacote sendo processado dentro do *switch* para fazer a comparação com as entradas da tabela de fluxos. A Figura 4, representa a fase de "Tratamento do cabeçalho" da Figura 3. Nessa fase, os campos de cabeçalho do pacote entrante são copiados para uma estrutura de dados para possuir todos ou parcialmente os cabeçalhos do pacote entrante. Os campos de cabeçalho que deverão ser considerados dependem de algumas condições que deverão ser satifeitas. Essa informação obtida do pacote é utilizada pelo *switch* para fazer a comparação com a tabela de fluxos (*matching process*).

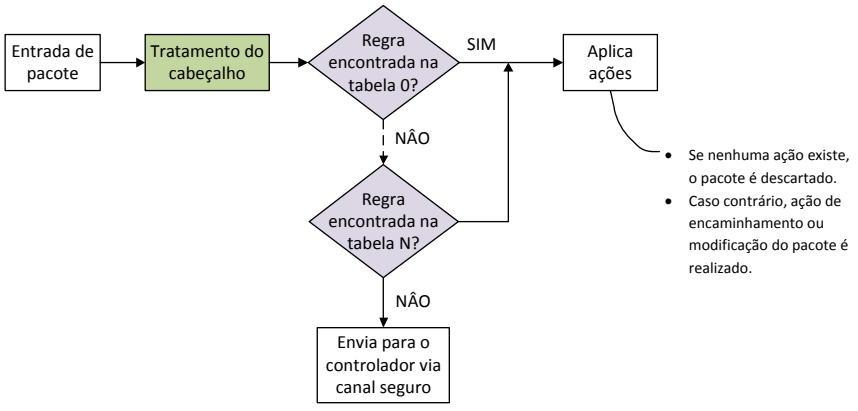


Figura 3: Percurso do um fluxo de pacote em um switch OpenFlow. Fonte adaptada da especificação 1.0 do OpenFlow [1].

Após a fase de comparação com a tabela de fluxos, se nenhuma regra estiver presente, o pacote é encapsulado e enviado ao controlador por meio do canal seguro. Isto possibilita ao controlador definir regras para esse pacote. Um vez definida, o controlador insere uma regra na tabela de fluxo do comutador. Dessa forma, o controlador define uma regra na tabela de fluxos e todos os pacotes entrantes que satisfazem essa regra, são encaminhados por uma porta de saída ou descartados, caso nenhuma ação seja exista.

Segundo a especificação 1.0 do OpenFlow, além da ação de trasmissão de pacotes por uma porta e descarte das mesmas, outras ações são possíveis de serem executadas. São as ações para encaminhar os pacotes para a pilha de protocolos locais do *switch*, modificar campos do cabeçalho do pacote, por exemplo, endereço IP de origem ou VLAN ID e ação de enfileiramento, para uma fila associada a uma porta. Nesse último caso, como descrito no capítulo anterior, apresenta problemas de automação.

A Figura 4, como foi mencionado nos parágrafos acima, representa a fase de "Tratamento do cabeçalho". Os campos de cabeçalho desse pacote que serão utilizados na comparação com a tabela dependem do tipo da informação contida nos cabeçalhos do pacote.

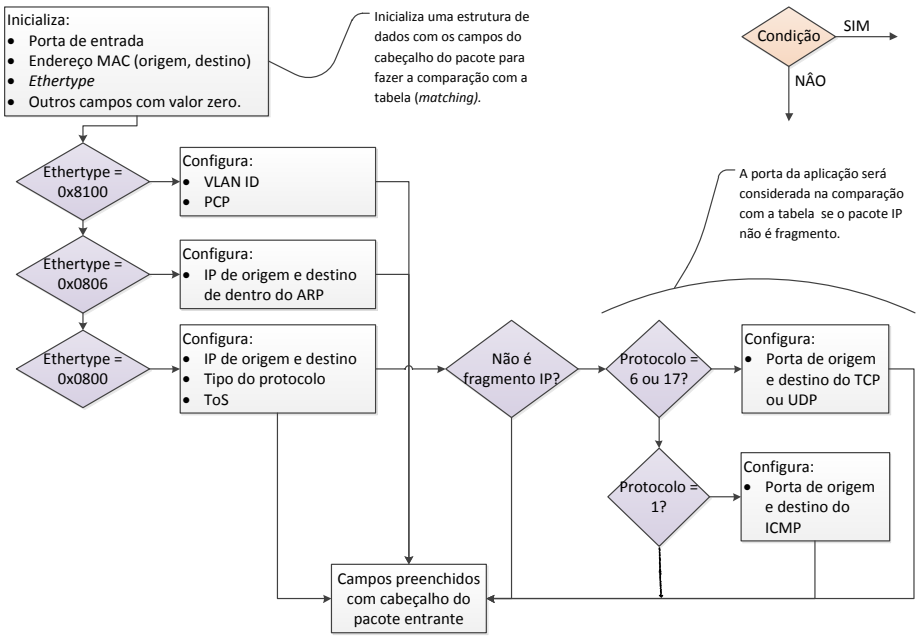


Figura 4: Tratamento dos campos de cabeçalho do pacote que entra no switch OpenFlow. Fonte adaptada da especificação 1.0 do OpenFlow [1].

As condições do processo de tratamento do cabeçalho (*packet parser*) estão relacionado ao Ethertype (L2), ao campo de fragmento (*Fragment Offset*), *Flags* e ao campo Protocolo que está sobre a camada IP (L3), ou seja, campos do cabeçalhos referentes a camada de enlace e a camada IP. O Ethertype [23] é um campo de 8 *bits* do cabeçalho, que pertence ao quadro Ethernet, pelo qual é usado para indicar o tipo de dados (*payload*) que está encapsulado no quadro Ethernet. O *switch* OpenFlow lida com três tipos de Ethertype para serem considerados na comparação com as regras da tabela de fluxos.

- Ethertype com valor `0x8100` para um pacote com *tag* de VLAN do padrão IEEE 802.1Q. Esse padrão define um mecanismo de etiquetagem (*tagging*) para os quadros Ethernet que entram e saem pelas portas por uma placa de rede Ethernet;
- Ethertype com valor `0x0806` para pacotes ARP (*Address Resolution Protocol*) que

são trocados entre as entidades que possuem endereços IP, caso não exista uma entrada que mapeie um endereço MAC com um determinado endereço IP na tabela ARP;

- Ethertype com valor `0x0800` para os pacotes baseados em IPv4.

Após passar por uma das três condições de avaliação do Ethertype do pacote, o processo de tratamento do cabeçalho pode continuar ou terminar. Isto depende do tipo do Ethertype. Para os casos em que os valores são `0x8100` ou `0x0806`, o processo termina. Logo, a fase de comparação com a tabela pode ser iniciada, como foi apresentada na Figura 3. No entanto, caso o valor seja `0x0800`, uma próxima condição deverá ser avaliada nesse processo.

A próxima condição a ser avaliada é verficar se o pacote IPv4 [24] não é um fragmento. Para isso, o *switch* OpenFlow verfica no cabeçalho IPv4 do pacote se, o campo de fragmento não é zero ou se o campo *More Fragments* está ativado. Em caso negativo, as portas das aplicações definidas não serão considerados e o processo de tratamento do cabeçalho termina. Para outro caso, se a condição for verdadeira, isto signifca que o pacote não é um fragmento IPv4. Logo, o campo de *bits* Protocolo será avaliado.

O campo Protocolo de 16 *bits* do cabeçalho do IPv4 define o tipo do protocolo que está encapsulado no corpo do datagrama IP, o qual contém um dos valores definidos pela IANA (*Internet Assigned Numbers Authority*) em [25]. O *switch* OpenFlow avalia e aceita três tipos de protocolo no processo de comparação com a tabela de fluxo.

- Protocolo com valor `6` para o caso de ser TCP;
- Protocolo com valor `17` para o caso de ser UDP;
- Protocolo com valor `1` para o caso de ICMP.

Para os casos de serem ou TCP ou UDP, os campos de porta de origem e destino desses protocolos serão considerados. Caso seja ICMP, os campos Tipo e Código serão levados em conta no processo de comparação com a tabela de fluxos.

O protocolo OpenFlow suporta três tipos de mensagens: controlador para *switch*, assíncrona e simétrica. As mensagens do controlador para *switch* são iniciados pelo controlador e usadas diretamente para gerenciar ou inspecionar o estado do *switch*. As mensagens assíncronas são iniciadas pelo *swicth* e utilizadas para atualizar o controlador sobre eventos na rede (ex.: novo fluxo na rede) ou mudanças no estado do *switch* (ex.: uma entrada removida da tabela de fluxo). Enquanto que as mensagens simétricas são iniciadas, tanto pelo *switch* quanto pelo controlador [1]. Os tipos dessas mensagens são baseadas na especificação 1.0 e resumidas nos parágrafos seguintes.

Mensagem controlador para *switch*: especificação (*features*), configuração (*configuration*), modificação de estado (*modify-state*), leitura de estado (*read-state*), envio

de pacote (*send-packet*). Esses são tipos de mensagens que são iniciados pelo controlador e podem requerer uma resposta do *switch* ou não.

Tabela 1: Tipos de mensagem controlador para *switch*

Tipos de Mensagem	**Comentários**
Especificação	Em reposta ao controlador, o *switch* com as capacidades suportadas (ex.: suporte a estatística de fluxo, número de portas, ...)
Configuração	Controlador pode configurar ou consultar parâmetros de configuração no *switch* para tratar fragmentos IP
Modificação de Estado	Utilizada para modificar estado do *switch* como, por exemplo, as entradas da tabela de fluxo através da mensagem `ofp_flow_mod`
Leitura de Estado	Utilizada para coletar informações de estados do *switch*, por exemplo, o número de pacotes transmitidos/recebidos por uma porta
Envio de Pacote	O controlador pode usá-lo para instruir um *switch* a enviar um pacote que está no *buffer*

Mensagem assíncrona: pacote de entrada (*packet-in*), fluxo removido (*flow-removed*), estatus da porta (*port-status*), erros (*errors*). São iniciados pelo *switch* sem a solicitação do controlador. Esses tipos de mensagens são usados para denotar entrada de um novo fluxo na rede, mudanças de estado e para notificação sobre erros.

Tabela 2: Tipos de mensagem assíncrona

Tipos de Mensagem	**Comentários**
Pacote de Entrada	Usada para notificar o controlador sobre a entrada de um novo fluxo de pacotes no *switch* através da mensagem `ofp_packet_in`
Fluxo Removido	O *switch* pode notificar o controlador sobre as regras de fluxo da tabela que foram removidas após um tempo de expiração (*timeout*)
Estatus da Porta	O *switch* notifica o controlador sobre o estatus das portas, por exemplo, se uma porta foi administrativamente desligado
Erros	Notifica o controlador sobre possíveis erros que podem ocorrer (ex.: erro ao tentar adicionar uma entrada na tabela de fluxos)

Mensagem simétrica: pacote hello (*hello*), pacote eco (*echo*), pacote fabricante (*vendor*). São mensagens enviadas sem solicitação e podendo ser iniciadas tanto pelo controlador ou pelo *switch*.

Tabela 3: Tipos de mensagem simétrica.

Tipos de Mensagem	Comentários
Pacote Hello	São trocados por ambas as partes quando o controlador e o *switch* são conectados
Pacote Eco	Pode ser utilizado para checar a conexão do canal de controle do OpenFlow
Pacote Fabricante	O OpenFlow oferece um modo para adicionar novas mensagens ao protocolo, estampando essa mesagem com o nome `OFPT_VENDOR`.

2.2 Versões do OpenFlow

Esta seção apresenta as principais mudanças nas funcionalidades, ocorridas no protocolo OpenFlow, a partir da versão 1.0. Muitos incrementos foram realizados ao protocolo desde a sua primeira versão 0.2 lançada oficialmente em Março de 2008. A versão mais atual do protocolo, atualmente, é a versão 1.3 lançada oficialmente em Abril de 2012. Abaixo, descreve-se em tópicos, as principais mudanças sofridas no protocolo ao longo desses anos [26].

- Versão 1.0 do OpenFlow (Dezembro de 2009)
  - ***Slicing***

    Introdução de mecanismo de fila através do uso das linhas de comandos `dpctl` e `tc`. Esse mecanismo suporta a habilidade de prover grantias de largura de banda mínima. A largura de banda alocada para as filas são configuráveis.
  - ***Flow cookies***

    Os fluxos foram extendidos para incluir um identificador opaco chamado *cookie*. O valor do *cookie* é especificado pelo contrador quando o fluxo for inserido no *switch*. Os fluxos podem ser removidos ou modificados baseando-se no valor de *cookie*.
  - **Descrição do plano de dados especificado pelo usuário**

    O retorno da mensagem para coletar a informação de descrição do *switch*, inclui um campo *string* para especificar um nome que possa descrever o *switch*.
  - **Operar sobre campo IP no pacote ARP**

    A implementação pode associar campos de endereço de origem e destino IP do pacote ARP na comparação com a tabela de fluxos.

  - **Consultar estatísticas de porta individualmente**

    A mensagem de estatística para consultar informações de portas, incluem um campo de 16 *bits* para especificar a porta a ser consultada.

- Versão 1.1 do OpenFlow (Fevereiro de 2011)

  - **Múltiplas tabelas**

    Nas versões anteriores da especificação do protocolo as múltiplas tabelas eram expostos ao controlador com a visão de uma única tabela. Nessa nova especificação, o processo de *pipeline* do pacote entre as múltiplas tabelas podem ser controlados.

  - **Grupos**

    A abtração em grupos possibilita ao OpenFlow representar um conjunto de portas como se fossem uma única entidade. Vários tipos de grupos são fornecidos para representar diferentes abstrações tais como `All` para *multicast* ou inundação e `Select` para múltiplos caminhos. Além disso, um grupo pode encaminhar para outros grupos.

  - ***Tags*: MPLS e VLAN**

    Suporte a múltiplos níveis de VLAN. Versões anterioriores do OpenFlow tem um suporte limitado a VLAN, podendo representar um único nível de etiquetagem (*tagging*). Além disso, ações explícitas para adicionar, remover e atualizar *tags* de VLAN e identificadores MPLS foram inseridos.

  - **Portas virtuais**

    Suporte a porta virtual no OpenFlow. Versões anteriores assumiam que todas as portas do OpenFlow eram físicas. Na nova versão, o OpenFlow é capaz de distinguir o tipo das portas, se é virtual ou física, para isso, a mensagem `packet_in` foi extendida.

- Versão 1.2 do OpenFlow (Dezembro de 2011)

  - **Suporte a comparação extensível (*Extensible match support*)**

    A estrutura de dados `ofp_match` possui tamanho extensível nesta nova versão. Nas versões anteriores, seu tamanho é fixo. A nova estrutura é conhecida como OXM (*OpenFlow Extensible Match*).

  - **Suporte a IPv6**

    Adicionado suporte a IPv6. Os campos do cabeçalho do IPv6 como endereço de origem e destino podem ser usados na comparação com as entradas da tabela de fluxos.

  - **Mecanismo de mudança de papel do controlador**

    Os *switches* OpenFlow podem se concetar com multiplos controladores em paralelo como medida de seguraça à falhas. Nesse mecanismo, basicamente cada controlador possui um papel na rede como mestre ou escravo, podendo ao mesmo, trocar de papel.

  - **Propriedade de taxa máxima das filas**

    Inserido um mecanismo para garantir largura de banda máxima através de linhas de comandos `dpctl` em conjunto com `tc`.

- Versão 1.3 do OpenFlow (Abril de 2012)

  - **Métricas por fluxo**

    Adicionado suporte a métricas por fluxo. Isto permite um maior controle sobre os fluxos para medição ou até mesmo limitar a taxa pacotes que é enviado ao controlador.

  - **Filtragem de eventos por conexão**

    O controlador possui a habilidade de filtrar ou controlar os tipos de eventos provenientes da rede.

  Além dessas mudanças, existem outras mudanças importantes na versão 1.3. Porém, as principais mudanças da versão 1.3 incluem adaptações estruturais (implementação) para tornar o OpenFlow mais flexível. Essa forma de adaptação, inclusive está presente nas versões anteriores a versão 1.3.

2.3 Conclusões do Capítulo

Este capítulo apresentou a arquitetura, funcionamento do protocolo OpenFlow e as especificações do protocolo, aprsentando as principais mudanças no protocolo que ocorreram desde a versão 1.0.

A solução primitiva de filas, no OpenFlow, começou na sua especificação 1.0. Apesar de na sua versão 1.2 ter ocorrido um incremento na funcionalidade, a forma de lidar com o QoS peramaneceu-se na mesma situação. De outra forma, o uso de ferramenta externa ao OpenFlow e a configuração manual, ainda prevalece até a sua última especificação.

CAPÍTULO 3

Trabalhos Relacionados

Este capítulo apresenta propostas de trabalhos com QoS no contexto do OpenFlow. O capítulo aborda cada proposta, diferenças entre eles e um quadro de resumo das propostas apresentadas.

3.1 QoS no OpenFlow

Convergência de rede é a coexistência de comunicações de voz, fluxos de vídeos e dados em geral, de forma eficiente, sobre uma mesma infraestrutura de rede. Segundo o trabalho de Kim *et al* [10] a convergência de rede oferece redução de custo e aumento na flexibilidade no gerenciamento. A partir de uma infraestrutura integrada, os administradores de redes não necessitam instalar e gerenciar múltiplos equipamentos de diferentes fabricantes utilizando protocolos e configurações diferentes. De acordo com [10], uma das formas comumente adotada atualmente para prover QoS é através de isolamento físico da rede. No entanto, isto aumeta custo da instalação, de gerenciamento e de operação.

Sendo assim, seu trabalho propôs um arcabouço de controle de QoS a partir da extensão das interfaces de programação do protocolo OpenFlow. Esta solução permite a programação dos dispositivos segundo alguns parâmetros de QoS. Nesta arquitetura, o controlador é capaz de criar fatias (*slices*) e associar diferentes tráfegos dentro delas de forma individual ou agregado. Além de manter as informações do estado da rede atualizada no controlador e oferecer APIs de QoS a partir da extensão do OpenFlow. No entanto, apesar de ser um trabalho interessante, o seu trabalho não lida com a utilização de múltiplos escalonadores de pacotes.

Civanlar *et al* [9] descreveu uma formulação de roteamento com QoS para mel-

horar o desempenho dos fluxos de vídeos codificados com SVC (*Scalable Video Coding*) que fluem em um domínio OpenFlow. Sua arquitetura consiste, basicamente, na operação cliente-servidor. O servidor de vídeo transmite alguns parâmetros de QoS para o componente de roteamento (controlador) gerar uma tabela de fluxos à serem configurados nos encaminhadores da rede. No controlador dessa proposta, é decidido que os pacotes que pertencem à base da codificação do vídeo devem ser tratados com qualidade de serviço. Enquanto que os demais, são tratados por melhor esforço.

Uma evolução do trabalho de Cinvalar *et al* apresentado em [9], Egilmez *et al* [27] propôs o OpenQoS. Essa proposta insere uma camada de serviço, acima da camada de controle da proposta, formulada em [9]. O controlador provê uma interface aberta e segura para os provedores de serviços configurarem definições de fluxos para que o módulo de cálculo de rotas com QoS possa determinar um caminho. Porém, nos trabalhos apresentados em [9] e em [27], apesar de parâmetros de QoS como atraso e perdas de pacotes terem sido formulados para oferecer garantias de QoS, a proposta deles não permitem realizar a classificação dos fluxos, isto é, fazer o controle de tráfego fatiando a banda.

Um sistema de gerenciamento de QoS nomeada QFlow foi proposta por Mattos *et al* [28]. Este sistema atua sobre a arquitetura XenFlow [29], um esquema de virtualização de topologias que agrega as entidades Xen e OpenFlow. Em linhas gerais, o QFlow se propõem fazer o monitoramento da rede e dividir o processamento, a memória e a largura de banda para todos os roteadores virtuais presentes na sua arquitetura. Dessa forma, o sistema provê tais recursos para os provedores de serviços da Internet. No entanto, apesar de automatizar o mecanismo de controle de tráfego no OpenFlow, seu arcabouço não integra o QoS totalmente no plano de dados, isto é, utiliza-se a ferramenta `tc` do Linux. Além disso, não permite que o usuário possa instanciar outras formas de escalonadores de pacotes.

Sonkoly *et al* propõem em [30] uma idealização de extensões ao arcabouço de controle do Ofelia[1] para ofercer suporte ao gerenciamento de QoS, após identificarem uma série de limitações no atual estado do protocolo OpenFlow e incluindo, seu próprio arcabouço de controle. Nesse trabalho proposto, foi descrito uma possível extensão ao Ofelia de modo a torná-lo capaz de executar experimentos relacionados com QoS. Ainda, segundo esse trabalho, é mencionado a falta de um suporte integrado de QoS nos atuais *testbeds* de experimentação com OpenFlow, incluindo Ofelia.

Segundo Sonkoly *et al*, o provisionamento de QoS requer inúmeras configurações sobre as filas e mapeamento de fluxos. A proposta do Ofelia aponta que o esse mapeamento deve ser suportado pela especificação do protocolo OpenFlow. Além disso, o trabalho afirma que os *switches* devem garantir suporte a QoS sem que ocorra uma degradação siginificante no desempenho durante o encaminhamento dos fluxos.

Nas adaptações sobre a arquitetura atual do aracabouço de controle do Ofelia, incluem extender o componente Expedient, Opt-In Manager, FlowVisor e *switches* OpenFlow. Nessa proposta da nova arquitetura, inclui o desenvolvimento de uma camada de

[1] http://www.fp7-ofelia.eu/

abstração para os diferentes tipos de fabricantes de equipamentos, sendo que um usuário poderá especificar parâmetros de QoS e o módulo de abstração seria capaz de direcionar esses parâmetros para os fabricantes específicos. Em seu trabalho se explanou a situação da ilha Ofelia que compõem diferentes tipos de fabricantes de *swicthes* OpenFlow como a HP e a NEC. Além de alguns equipamentos habilitados com OpenFlow *software switch*. Ainda, segundo Sonkoly *et al*, as diferentes soluções de *switches* OpenFlow, apresentam comandos específicos para manipular o QoS através de diferentes tipos de interfaces, como SNMP e Netconf.

3.2 Conclusões do Capítulo

Este capítulo apresentou o estado da arte de trabalhos que abordam QoS no domínio OpenFlow. Os trabalhos apresentados mostraram-se interessantes e promissores. A maioria desses trabalhos, propõem melhorias na arquitetura ou no protocolo OpenFlow para lidar com as deficiências de QoS existente atualmente. Entretanto, os trabalhos em geral, propõem soluções para particionar a largura de banda disponível na rede.

Todos os trabalhos relacionados apresentaram proposta de QoS para melhorar a forma de lidar com o controle de QoS atual do OpenFlow. A Tabela 4 apresenta um resumo dos pontos mais globais de cada proposta apresentada e comparada com a proposta do trabalho, QoSFlow.

Tabela 4: Tabela comparativa entre os trabalhos relacionados e a proposta QoSFlow

Autores	**Nome da Proposta**	**API**	**CB**	**CE**	**M**	**P**
Kim *et al*, 2010	-	x	x		x	x
Civanlar *et al*, 2010	-				x	x
Mattos *et al*, 2011	QFlow	x	x		x	
Egilmez *et al*, 2012	OpenQoS				x	x
Sonkoly *et al*, 2012	OCF		x		x	
Dissertação	QoSFlow	x	x	x	x	x

- Legenda:
  - **API -** Inteface de programação de QoS
  - **CB -** Controle de largura de banda
  - **CE -** Controle dos escalonadores de pacotes
  - **M -** Monitoramento
  - **P -** Políticas definidas em alto nível de abstração

A proposta QoSFlow, além de possuir a característica de controlar a largura de banda, também permite o controle de diferentes tipos de escalonadores de pacotes, o monitoramento das estatísticas de filas e habilita interfaces de programação desses escalonadores de pacotes no nível do controlador.

CAPÍTULO 4

QoSFlow

Este capítulo apresenta o QoSFlow, um arcabouço de *software* que visa suprir as deficiências de controle de QoS que o protocolo OpenFlow apresenta em seu estado atual. Isto é, o capítulo apresenta uma proposta para remover a necessidade das configurações manuais de QoS na rede OpenFlow.

4.1 Apresentação Geral

QoSFlow é uma proposta de um arcabouço de *software* que implementa o modelo SDN/OpenFlow. Visto que o OpenFlow apresenta problemas de automatização no contexto de QoS, foi necessário criar extensões ao mesmo para permitir um melhor suporte às atividades administrativas de QoS. O modelo da arquitetura da proposta é representado em três níveis, conforme apresentado pela Figura 5.

No nível mais baixo está o **plano de dados**. Esse nível é composto por elementos de comutação habilitados com OpenFlow (*OpenFlow software switch*) modificados para suportar a configuração de alguns escalonadores de pacotes do Linux. Esses dispositivos são responsáveis pela execução da ação de encaminhamento, ou enfileiramento, dos pacotes e pela configuração de baixo nível de QoS, sendo que tal configuração inclui a criação de filas e da associação dos escalonadores de pacotes sobre essas filas.

No nível intermediário está o **plano de controle**. Esse plano é composto por um controlador OpenFlow, baseado no NOX, e pelos componentes que interpretam comandos definidos no plano da administração. No entanto, da mesma forma como o *switch* OpenFlow (ou *datapath*) foi adaptado para suportar a configuração dinâmica de QoS, uma adaptação semelhante foi realizada no controlador. Nesse plano, implementa-se dois

tipos de conjuntos de API conhecidas como API, de Ponte Norte (*Northbound API*) e API de Ponte Sul (*Southbound API*) [31].

Na arquitetura do QoSFlow, ambos os tipos de interfaces de programação possibilitam controlar QoS na rede OpenFlow. As interfaces de Ponte Norte, são compostas pelas mensagens ou esquemas definidas em JSON e são utilizadas pela camada de administração. As interfaces de Ponte Sul são utilizadas pelas aplicações, que executam no controlador, que reprogramam o plano de dados. Dessa forma, a arquitetura da proposta torna mais modular o que possibilita que futuras manutenções na arquitetura possam ser de forma independente, contanto que as interfaces de comunicação sejam mantidas.

No nível mais alto está o **plano administrativo**. Nessa camada são definidas as filas com as configurações de banda mínima e máxima garantida (*traffic shaping*) para os fluxos, uma política de escalonamento de pacotes para ser utilizada nessas filas, além de definições de políticas para encaminhamento dos fluxos que entram na rede. Essas políticas de encaminhamento são do tipo evento-condição-ação, isto é, para cada evento de novo fluxo na rede (`packet_in`), se uma condição for satisfeita, as políticas que foram definidas nesse plano são mapeadas para o plano de dados, onde ações de encaminhamento para as filas serão aplicadas aos fluxos.

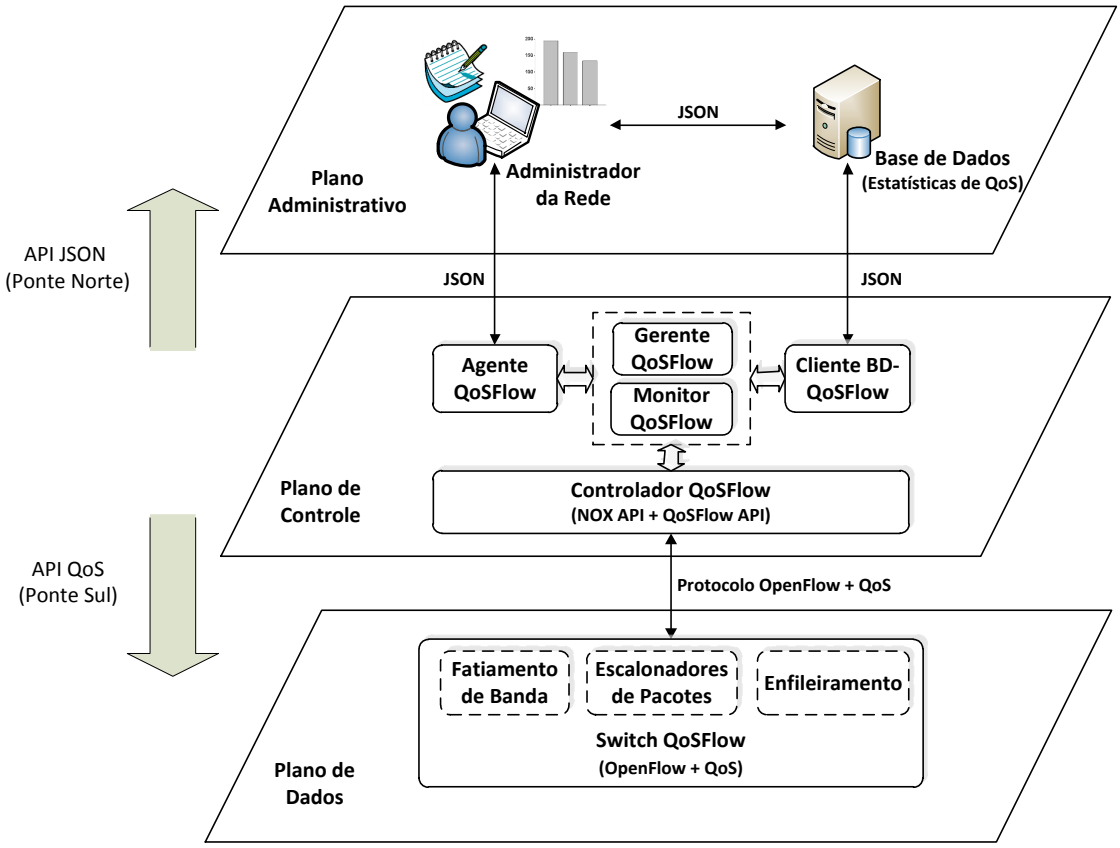
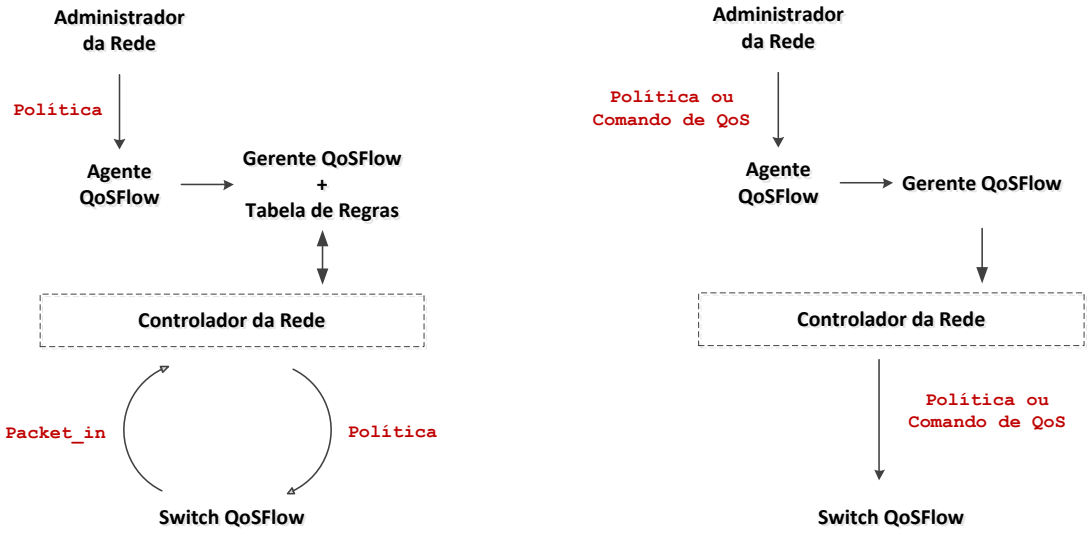


Figura 5: Arquitetura do QoSFlow

As condições das políticas de encaminhamento são baseadas nas informações da

camada de enlace (*Layer 2*, L2), rede (*Layer 3*, L3) e/ou transporte (*Layer 4*, L4) do modelo TCP/IP. Além disso, no QoSFlow as ações de encaminhamento envolvem uma fila e uma porta de saída do comutador. A cada fila que pertence às portas do *switch*, tem-se uma largura de banda associada a elas.

O QoSFlow possibilita ao administrador executar um controle de QoS de forma **proativa** ou **reativa**. A Figura 6(a) ilustra um processo reativo, onde as políticas de encaminhamento definidas no plano administrativo são armazenados no plano de controle. Dessa forma, o plano de controle automaticamente reage para cada novo fluxo que entra na rede. Por conseguinte, essas políticas são mapeadas no plano de dados para que os *switches* possam fazer o encaminhamento dos pacotes.



(a) Abordagem reativa (b) Abordagem proativa

Figura 6: Controle de QoS proativo ou reativo.

Na abordagem reativa, as políticas de encaminhamento são armazenadas na tabela de regras do Gerente QoSFlow, enquanto que na abordagem proativa as políticas ou comandos de QoS são transmitidos imediatamente para os *switches*.

Além da abordagem reativa, o QoSFlow possibilita realizar um controle proativo, conforme é apresentado na Figura 6(b). Nesse tipo de controle o usuário administrador define e envia uma política de encaminhamento diretamente para o plano de dados, através do plano de controle. A diferença está no fato de que essa regra de encaminhamento não é armazenada no plano de controle, o que faz com que os *switches* não consultem o plano intermediário, pois as regras de encaminhamento já estarão preenchidas nas suas tabela de fluxos.

Os comandos de QoS que envolvem fatiar a largura de banda (criar filas), associar um escalonador de pacotes nas filas e consultar as estatísticas de QoS são realizados proa-

tivamente. Em ambos os modos (proativo e reativo), uma execução de adição, remoção ou atualização podem ser realizadas no plano de dados. A Tabela 5 resume o que pode ser executado em ambos os modos.

Tabela 5: Abordagem proativa e reativa

Funcionalidade	**Proativa**	**Reativa**
Monitoramento de estatísticas	x	
Controle de escalonadores de pacotes	x	
Controle de banda	x	
Política de encaminhamento	x	x

O monitoramento das estatísticas de QoS sempre será de modo proativo, pois esse mecanismo implementado na proposta não tem a capacidade de reagir a eventos, como por exemplo um evento de entrada de um novo fluxo na rede, já que o componente Monitor QoSFlow não executará monitoramento desse novo fluxo. Além disso, o monitoramento não ocorre de maneira individual aos fluxos, pois essas consultas periódicas ocorrem sobre as filas associadas nelas, sendo possível que coexista mais de um fluxo em uma mesma fila, dependendo da regra de encaminhamento aplicada pelo administrador.

Os comandos de QoS, por exemplo, para associar ou desassociar um escalonador de pacotes é definida proativamente. O arcabouço, até o presente trabalho, não oferece suporte à configuração de escalonadores de pacotes reagir, por exemplo, ao evento de novo fluxo na rede. Atualmente, a política de encaminhamento é a única forma que o arcabouço possibilita o uso da abordagem proativa ou reativa.

4.2 Arquitetura em Detalhes

Esta subseção compromete-se em apresentar as funcionalidades da proposta em maiores detalhes. No entanto, o trabalho atual não foca no plano administrativo o qual envolve uma interface de administração do usuário. O foco do presente trabalho concentra-se no plano de controle e principalmente no plano de dados, onde as funcionalidades de QoS, no nível dos dispositivos, foram incorporadas ao OpenFlow.

Conforme a Figura 5, além do papel do administrador, a arquitetura do QoSFlow é composta por mais duas entidades principais, chamadas de **Switch** e **Controlador QoSFlow**. O primeiro, além da responsabilidade de fazer o encaminhamento, realiza as configurações de QoS no nível do dispositivo, através do módulo **OpenFlowQoS**. O segundo, por sua vez, habilita a administração dos recursos da rede através de dois componentes que executam sobre o controlador, chamados de **Monitor** e **Gerente QoSFlow**.

O **Agente QoSFlow** possui a responsabilidade de criar um meio de comunicação entre uma ferramenta de administração e os outros dois componentes do controlador, o Gerente e Monitor QoSFlow. Essa interface de comunicação do Agente é no formato

JSON[1], onde estão as definições de políticas ou comandos de QoS para serem executadas na rede OpenFlow.

O administrador pode realizar consultas, inserção, remoção ou atualização de informações dos recursos registrados na base de dados. Essas informações são povoadas pelo componente **Cliente BD-QoSFlow**, cuja importância na arquitetura é realizar consultas periódicas no plano de encaminhamento para coletar estatísticas de portas e filas, como por exemplo, número de *bytes* transmitidos e recebidos.

O controlador da proposta foi baseada na versão 0.9 do NOX (zaku) e todos os componentes do controlador QoSFlow foram desenvolvidas em C++, assim como as interfaces de programação de QoS que foram inseridas no NOX original. Quanto ao *switch* QoSFlow, as adaptações foram realizadas sobre a versão 1.0 do *OpenFlow software switch*, implementada em C.

As subseções sequentes apresentam as responsabilidades de todos os componentes que participam da arquitetura do QoSFlow, acompanhada por exemplos e por pseudocódigos.

4.2.1 A Interface de Administração e o Esquema em JSON

O administrador da rede coordena os fluxos de pacotes e as fatias de largura de banda para as aplicações dos usuários, além da escolha de uma política de escalonamento de pacotes (disciplinas de fila) para essas fatias. Esses comandos são operados por meio de uma ferramenta de administração, que pode ser uma interface baseada em Web, uma interface gráfica *desktop* ou até mesmo uma interface de linha de comando. Ressalta-se que o usuário responsável pela administração da rede e a sua interface de controle pertencem à camada de administração.

Apesar da arquitetura do QoSFlow possibilitar a utilização de diferentes tipos de interfaces para controlar o QoS, essas mesmas interfaces devem obrigatoriamente converter os comandos de controle em um modelo de mensagem desenvolvido em JSON. Nesse esquema são descritas as políticas de encaminhamento ou os comandos de QoS (ex.: instanciar uma fila ou configurar um escalonador de pacotes) definidos na interface. São vários os modelos que foram criados, cada um com objetivos específicos, na arquitetura QoSFlow. Suas definições podem ser visualizadas em maiores detalhes no Apêndice A.

Os esquemas em JSON possibilitam controlar os escalonadores de pacotes, associando ou desassociando a uma fila, habilitar a criação de fatias de banda e definir políticas de encaminhamento baseadas em filas. Além das funcionalidades de controle de QoS, o arcabouço permite o monitoramento das estatísticas de portas e filas dos comutadores da rede e o armazenamento em uma base de dados no formato JSON. Através dessa base de dados o usuário consulta informações referentes aos dispositivos.

[1]http://www.json.org/

A Mensagem JSON 4.1 apresenta um exemplo de uma mensagem definida em JSON que deve ser produzida pela interface para armazenar uma política de encaminhamento no plano de controle, determinada pelo par ‘‘controller’’ : ‘‘save’’.

No momento em que a condição definida pela chave ‘‘condition’’ é satisfeita pelo packet_in, os valores das chaves ‘‘condition’’ e ‘‘action’’ são mapeadas na tabela de fluxo do *switch* que possui o identificador 1.

A partir de então, no nível do dispositivo, os fluxos que satisfazem a **condição** são manipulados de acordo com a **ação** estabelecida. No exemplo da Mensagem JSON 4.1, a ação consiste em enfileirar um fluxo para a fila 2 (ou classe 2) da porta de saída eth0 quando a condição for satisfeita, que a mesma consiste em: um fluxo deve entrar no *switch* 1 pela porta eth1, com IP de origem 192.168.1.1 e o número da porta da aplicação de destino ser 80.

Mensagem JSON 4.1: Exemplo de uma política de encaminhamento descrita em JSON

```
{
   ‘‘command’’ : ‘‘management’’,
   ‘‘dp’’ : 1,
   ‘‘controller’’ : ‘‘save’’,
   ‘‘table_entry’’ : ‘‘add_policy’’,
   ‘‘action’’ : {‘‘cid’’ : 2, ‘‘out_port’’ : ‘‘eth0’’},
   ‘‘condition’’ : {‘‘in_port’’ : ‘‘eth1’’,
                   ‘‘nw_src’’ : ‘‘192.168.1.1’’,
                   ‘‘tp_dst’’ : 80}
}
```

O exemplo da Mensagem JSON 4.2 apresenta um formato JSON para adicionar uma classe (ou fila) com identificador 3 na porta eth0 do *switch* 1. Para essa fila são alocados 6 *Mbps* de capacidade de largura de banda mínima (‘‘rate’’ : 6000000). A unidadade da taxa de *bits* representada nesse tipo de mensagem deve estar sempre no formato *bits/s*.

Mensagem JSON 4.2: Exemplo de um comando de QoS descrita em JSON

```
{
   ‘‘command’’ : ‘‘management’’,
   ‘‘dp’’ : 1,
   ‘‘class’’ : ‘‘add’’,
   ‘‘out_port’’ : ‘‘eth0’’,
   ‘‘cid’’ : 3,
   ‘‘htb’’ : {
       ‘‘rate’’ : 6000000
   }
}
```

As estatísticas de QoS, persistidas na base de dados e obtidas dos switches através da aplicação QoSFlow Monitor, são monitoradas pela arquitetura do QoSFlow. Tal aplicação é reprogramada para consultar periodicamente o plano de dados e armazenar as estatísticas na base de dados da arquitetura. A Mensagem JSON 4.3 exemplifica um formato JSON enviado pelo adminstrador para ordenar o componente de monitoramento a consultar os dispositivos em intervalos de 2 segundos.

Mensagem JSON 4.3: Exemplo de comando descrita em JSON para monitorar QoS

```
{
   ``command'' : ``monitoring'',
   ``dp'' : 1,
   ``interval:'' : 2
}
```

A Mensagem JSON 4.4 exemplifica um formato de JSON produzido no plano de administração para consultar informações de QoS persistidas na base de dados. Nessa mensagem informações referentes a fila `3` da porta `eth0` que pertence ao *switch* `1`.

Mensagem JSON 4.4: Exemplo de uma consulta de estatística de QoS descrita em JSON

```
{
   ``command'' : ``monitoring'',
   ``dp'' : 1,
   ``out_port'' : ``eth0'',
   ``cid'' : 3
}
```

O conteúdo da Mensagem JSON 4.5 informa para o administrador as estatísticas de QoS referentes ao exemplo da Mensagem JSON 4.4. A mensagem mostra que foram transmitidos 10 pacotes (`‘‘tx_packets’’ : 10,`), 1280 *bytes* de dados (`‘‘tx_bytes’’ : 1280`) e 2 pacotes não transmitidos por alguma falha (`‘‘tx_errors’’ : 2`). Essas informação são referentes a fila `3` da porta `eth0` do *switch* `1`.

Mensagem JSON 4.5: Exemplo de estatística de QoS descrita em JSON

```
[
   {
       ``dp'' : 1
       ``devices'' :
       [
          {
              ``port_name'' : ``eth0'',
              ``queue_id'' : 3,
              ``tx_packets'' : 10,
              ``tx_bytes'' : 1280,
              ``tx_errors'' : 2
          }
       ]
   }
]
```

A utilização do esquema em JSON permite que a interface do administrador da rede permaneça desacoplada do plano de controle da arquitetura do QoSFlow. Essa característica modular, torna sua arquitetura mais flexível. Tal flexibilidade se manifesta pela simplicidade para o desenvolvimento de uma interface de administração do usuário devido a (1) independência do tipo da linguagem de programação, (2) não necessidade de manipular as mensagens do protocolo OpenFlow e (3) alterações estruturais no plano de controle não requerer modificações na interface de administração.

4.2.2 Controlador QoSFlow e seus Componentes

Os componentes administrativos do controlador, Gerente e Monitor QoSFlow, desempenham papeis importantes na arquitetura da proposta para orquestrar a rede junto ao administrador. As Tabelas 6 e 7 apresentam uma síntese das funcionalidades dos dois componentes (Gerente e Monitor) no plano de controle.

Tabela 6: Funcionalidades do Gerente QoSFlow

Componente	**Funcionalidades**	**Comentários**
Gerente QoSFlow	Manipular mensagem JSON	Executa a tradução do JSON para OpenFlow e vice-versa
	Processar sobre a tabela de regras	Adiciona, remove ou atualiza uma entrada na tabela de regras
	Enviar mensagem de política de encaminhamento	Envia uma mensagem `ofp_flow_mod` para o *switch*
	Enviar mensagem SFQ	Envia uma mensagem de QoS para configurar o escalonador de pacotes SFQ
	Enviar mensagem RED	Envia uma mensagem de QoS para configurar o escalonador de pacotes RED
	Enviar mensagem PFIFO	Envia uma mensagem de QoS para configurar o escalonador de pacotes PFIFO
	Enviar mensagem BFIFO	Envia uma mensagem de QoS para configurar o escalonador de pacotes BFIFO
	Enviar mensagem HTB	Envia uma mensagem de QoS para configurar fatias de banda (filas) com garantias mínima e/ou máxima (*traffic shaping*)
	Enviar resposta de configuração	Retorna uma confirmação de sucesso ou erro durante a configuração de QoS no *swicth*

Tabela 7: Funcionalidades do Monitor QoSFlow

Componente	**Funcionalidades**	**Comentários**
Monitor QoSFlow	Manipular mensagem JSON	Executa a tradução do JSON para OpenFlow e vice-versa
	Enviar mensagem de estatística de porta	Envia uma mensagem `ofp_port_stats` para coletar estatística de QoS de porta
	Enviar mensagem de estatística de fila	Envia uma mensagem `ofp_queue_stats` para coletar estatística de QoS de fila
	Enviar estatística de QoS coletada	As informações de QoS coletadas são enviadas para a interface do administrador

As funcionalidades dos componentes Gerente e Monitor são somente executadas após a decisão do Agente QoSFlow. Em outras palavras, o Agente seleciona o componente adequado que deverá ser chamado para execução. Essa seleção, depende do tipo de instrução estampada na mensagem JSON enviada pelo administrador, como pode ser observado pela chave `‘‘command’’` em um dos exemplos de mensagem JSON apresentados na subseção 4.2.1. Se o par for `‘‘command’’ : ‘‘management’’`, o Gerente entrará em ação, caso contrário, se o par for `‘‘command’’ : ‘‘monitoring’’`, o Monitor será selecionada para executar suas funções na arquitetura.

O componente Gerente QoSFlow se responsabiliza em fazer o controle de QoS no domínio da rede e instruir os *switches* para encaminhar os fluxos. No entanto, uma primeira etapa deve ser executada antes de transmitir mensagens aos dispositivos, a qual consiste em fazer a tradução da mensagem JSON para mensagem OpenFlow. De modo análogo, ocorre para o Monitor QoSFlow.

O resultado da tradução é utilizado para duas finalidades, (1) para fazer a comparação com o `packet_in` e (2) para enviar a mensagem traduzida ao *switch*. O Trecho de Código 4.6, apresenta o exemplo simplificado da Mensagem JSON 4.1 traduzido.

Trecho de Código 4.6: Exemplo de uma política de encaminhamento traduzido

```
...
ofm->header.type = OFPT_FLOW_MOD;
...
ofm->match.in_port = in_port;          // eth1
ofm->match.nw_src = nw_src;            // 192.168.1.1
ofm->match.tp_dst = tp_dst;            // 80
ofm->command = htons(OFPFC_ADD);
...
action.type = htons(OFPAT_ENQUEUE);
action.port = outport;                 // eth0
action.queue_id = queue_id;            // 3
...
```

Para a política de encaminhamento, a mensagem JSON traduzida é armazenada da **tabela de regras** do componente Gerente. Essa tabela é semelhante à tabela de fluxos presente nos dispositivos da rede. A principal diferença é que a tabela de regras é utilizada para fazer a comparação com o `packet_in`, enquanto que a tabela de fluxos é utilizada no processo de encaminhamento dos pacotes. Se o processo de comparação do `packet_in` com a tabela de regras for verdadeiro para alguma entrada ("linha de condição"), a política de encaminhamento associada é transmitida para a tabela de fluxos do *switch*.

Durante o processo diversas funções de manipulação de mensagens são executadas no Gerente ou no Monitor. O Apêndice B apresenta um diagrama de classe simplificado dos dois componentes, com o objetivo de apenas ilustrar algumas funções existentes dentro dos componentes administrativos.

O Algoritmo 1 apresenta o funcionamento do controlador QoSFlow junto aos seus componentes. O controle da rede entra em funcionamento após o início da execução do componente Agente QoSFlow, o qual inicializa *buffers* e abre uma conexão TCP na porta 6600 para escutar a ferramenta de administração, por onde as mensagens JSON são enviados e recebidos entre o plano administrativo e de controle.

Algoritmo 1: Agente QoSFlow

```
Inicializa buffers e abre uma conexão TCP para escutar a ferramenta do administrador.
AlocarBufferRx (TAM_MAX_BUF_RX);
AlocarBufferTx (TAM_MAX_BUF_TX);
AbrirConexao (NUM_PORTA_ESCUTA);
while true do
    AceitarConexao ();
    ReceberDados ();
    Comando recebido no formato JSON deve ser convertido para uma
    estrutura de dados do controlador antes de executar operações no switch.
    Err ← ConverterJsonParaQosflow ();
    if Err then
        ReportarErroParaAdministrador ();
    Após a conversão, ocorre a programação da rede.
    NetworkProgramming ();
    Depois de recebido uma resposta do switch, essa resposta deve ser
    convertida para um formato JSON antes de ser enviado para
    ferramenta do administrador ou para a base de dados.
    PrepararJsonParaResponder ();
    EnviarDados ();
end
FecharConexao ();
```

Após o Administrador receber a mensagem no formato JSON, a mesma é convertida para uma estrutura de dados do controlador antes de executar operações no *switch*. Essa conversão é necessária para que os componentes Gerente ou Monitor QoSFlow possam transmitir as políticas de encaminhamento para o *switch* ou mensagens de QoS, por exemplo, para configurar largura de banda no *switch*. No entanto, erros podem ocorrer durante esse processamento, como por exemplo o caso de o Administrador, equivocadamente, tentar realizar operações sobre um *switch*, uma porta ou fila inexistente. Tais erros são reportados ao administrador no formato JSON. Para detecção destes tipos de erros, as aplicações Gerente e Monitor QoSFlow mantém um estado de informação dos dispositivos da rede.

Depois de recebida uma resposta do *switch*, a mesma deve ser convertida para um formato JSON antes de ser enviada para ferramenta do administrador ou para a base de dados. Tal direção de envio depende do componente que enviou mensagens ao *switch*. No caso de ser o Gerente QoSFlow, ele pode receber uma confirmação se um recurso foi alocado. Caso seja o Monitor QoSFlow, o mesmo pode receber mensagem de erro ou receber dados de estatísticas para serem persistidos na base de dados.

O QoSFlow, além de ser um arcabouço de *software* para lidar com a deficiência de QoS existente atualmente no OpenFlow, também se propõe em oferecer um conjunto de primitivas de QoS ou APIs de QoS. Essas primitivas podem ser utilizadas para que os desenvolvedores de redes possam criar suar próprias aplicações de controle de QoS no domínio OpenFlow. Os componentes Gerente e Monitor QoSFlow foram, em parte, desenvolvido utilizando essas primitivas de QoS, incorporados ao tradicional NOX. O Trecho de Código 4.7 apresenta um exemplo de utilização de uma dessas primitivas.

Trecho de Código 4.7: Exemplo de uma política de encaminhamento traduzido

```
Htb htb;
struct ofp_qos_msg *msg;

/* cabecalho da mensagem qos */
htb.set_object_type(OFP_QOS_CLASS);
htb.set_action_type(OFP_QOS_ACTION_ADD);

/* corpo da mensagem qos */
htb.set_port_class(2);          // porta de saida
htb.set_class_id_class(1);      // class id = queue id
htb.set_rate_class(5000000);    // limite para 5mpbs

/* mensagem pronta para ser enviada */
msg = qfo.make_htb_class(htb);
```

No exemplo de código acima, uma mensagem de QoS é criada para adicionar uma fila com identificador 1, limitado a 5 *Mbps*. Essa mensagem aloca recurso na porta 2 e, basicamente, para criar uma mensagem de QoS, constrói-se o cabeçalho e o corpo dessa mensagem, passando-se os parâmetros desejados

4.2.3 Switch QoSFlow

A disciplinas de filas, ou escalonadores de pacotes, habilitam os dispositivos a controlarem os dados que são enviados. Enfileirar possibilita determinar a forma como os dados são transmitidos pelas interfaces de rede. Por exemplo, através do controle de tráfego do Linux é possivel controlar a transmissão dos pacotes limitando sua taxa de transmissão, retardando a transmissão ou determinando a ordem em que os pacotes são enviados [32]. O *switch* QoSFlow utiliza alguns escalonadores de pacotes do Linux para fazer o controle de QoS, habilitando o controle da taxa de transmissão dos pacotes, por exemplo.

O controle sobre a taxa de transmissão possibilita garantir recurso de largura de banda da rede para um usuário que assina um contrato com um provedor de serviço, por exemplo. Os provedores de serviços garantem aos seus clientes um limite de banda pela qual um usuário se propõem a pagar. Em outros tipos de cenários, como por exemplo, em uma rede local, o controle de tráfego poderia ser empregado para restringir a taxa com que os pacotes entram e saem pela rede local. Para essa forma de controle, as interfaces do *gateway* teriam que ser controladas.

Os escalonadores de pacotes são algoritmos capazes de gerenciar as filas associadas à uma interface de rede. O Linux contém um sistema robusto para controlar os fluxos de pacotes e distribuí-los de acordo com algumas regras definidas [32]. O conjunto das disciplinas de fila disponíveis no *kernel* do Linux estão classificadas em disciplinas de fila com classes (*classfull*) e sem classes (*classless).*

As disciplinas de fila sem classes não possuem subdivisões em classes, isto é, são disciplinas que não executam a classificação de pacotes, escolhendo apenas o próximo pacote da fila que deve ser transmitido pela porta de saída. A política para definir o

próximo pacote a ser transmitido, depende do tipo da disciplina de fila instanciada no *kernel* como o FIFO (*First In First Out*) e SFQ (*Stochastic Fair Queueing*).

O controle da trasmissão dos pacotes através das disciplinas de fila com classes, contemplam múltiplas classes e filtros. Uma disciplina de fila com classe precisa determinar a classe ou fila que o pacote pertence e, para isto, os filtros são utilizados para fazer a classificação dos pacotes que chegam na porta de entrada. Define-se um filtro como um conjunto de condições que deverão ser satisfeitas para ocorrer a classificação dos pacotes. Sendo assim, um dispositivo atuando como um roteador TCP/IP tradicional, baseado em Linux, utilizaria esses filtros para classificar os pacotes a fim de que sejam direcionados a uma fila [32].

No *switch* da proposta QoSFlow existem funções capazes de classificar ou enfileirar os pacotes entrantes nas filas de uma determinada porta para saída dos pacotes. A tabela de fluxos do *switch* determinará as ações de enfileiramento a serem aplicadas nesses pacotes. Esse conjunto de funções é abrangido no módulo OpenFlowQoS, como apresentado na Figura 7.

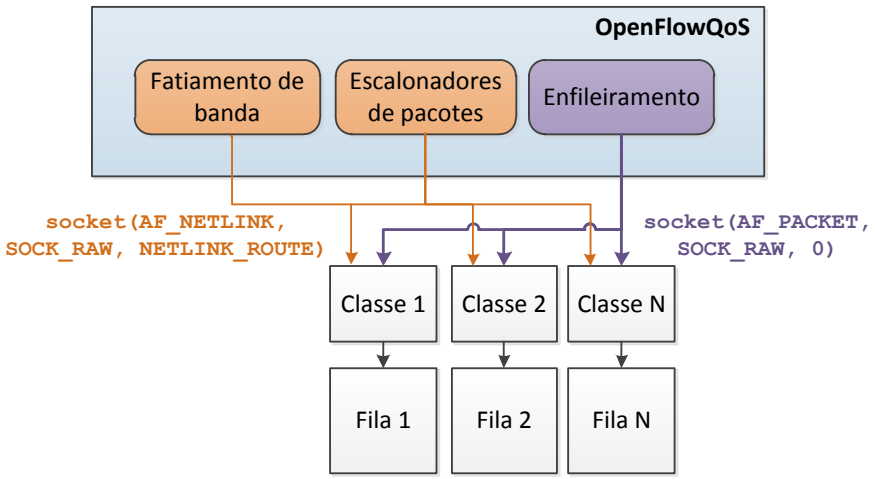


Figura 7: Idealização das funções do módulo OpenFlowQoS.

O módulo executa no espaço do usuário e utiliza *sockets* de comunicação com o espaço do kernel para controlar o enfileiramento, o fatiamento de largura de banda e os escalonadores de pacotes.

Um *switch* QoSFlow, assim como a especificação padrão de um *switch* OpenFlow, consiste de uma tabela de fluxo, onde as regras de encaminhamento são adicionadas, removidas ou atualizadas, e de um canal seguro para comunicação com o controlador, por onde as aplicações de controle que executam sobre o controlador QoSFlow são capazes de controlar o QoS e coordenar o comportamento da rede.

O módulo OpenFlowQoS possibilita ao *switch* adicionar, remover ou atualizar

parâmetros dos escalonadores de pacotes e regras de enfileiramento através de mensagens de QoS e OpenFlow. Essas mensagens são recebidas e interpretadas pelo módulo OpenFlowQoS, cuja as funcionalidades são:

- **Enfileiramento.** Políticas de encaminhamento definidas pelo administrador são recebidos e processados para que os pacotes possam ser enfileirados adequadamente. É uma mensagem OpenFlow e seu cabeçalho é identificado pelo tipo da mensagem, chamada `OFPT_FLOW_MOD`.

- **Fatiamento de banda.** Essa funcionalidade utiliza funções de baixo nível para criar, excluir ou atualizar limite de largura de banda estabelecida na rede. É uma mensagem de QoS que tem estampada em seu cabeçalho um novo tipo da mensagem, chamada `OFPT_QOS_QUEUEING_DISCIPLINE`.

- **Escalonamento de pacotes.** Possibilita a configuração de uma disciplina de fila para dar suporte a um tratamento diferenciado, além do tradicional FIFO para o envio dos pacotes. As operações a serem feitas sobre esses escalonadores, são comandadas pelas mensagens com o tipo `OFPT_QOS_QUEUEING_DISCIPLINE` estampada em seu cabeçalho.

Essas novas funcionalidades executam no espaço do usuário do sistema operacional Linux. Portanto, para que um processo que executa nesse espaço pudesse se comunicar com o sistema de controle de tráfego do Linux, situado no *kernel*, foram utilizados chamadas de sistema envolvendo `socket` da família `AF_PACKET` e `AF_NETLINK`.

O *swicth* OpenFlow, desde a versão 1.0 do protocolo OpenFlow, suporta o direcionamento de pacotes para as filas. Para isto, utiliza-se da família de *sockets*, chamada `AF_PACKET`. Entretanto, essas filas são previamente criadas a partir da linha de comando `dpctl` e `tc`. O OpenFlowQoS utiliza essa funcionalidade existente para dar suporte às regras de enfileiramento definidas no nível acima do plano de dados. Porém, para que a proposta QoSFlow pudesse oferecer condições para adição ou remoção dinâmica de filas e escalonadores de pacotes, utilizou-se a família de *socket* `AF_NETLINK`, que possibilita a um processo do usuário comunicar-se com o sistema de controle de tráfego do Linux.

Os escalonadores de pacotes suportados pela proposta QoSFlow são: SFQ (*Stochastic Fairness Queuing*), RED (*Random Early Detection*), HTB (*Hierarchical Token Bucket*) e o tradicional FIFO, tendo seu nome variando em PFIFO (*Packet First-In First-Out*) ou BFIFO (*Bytes First-In First-Out*). A única diferença entre PFIFO e BFIFO é na forma de como configurar o tamanho da fila. Na primeira forma o comprimento da fila é instanciado baseando-se no número de pacotes, enquanto que na segunda forma baseia-se no número de *bytes* que podem ser suportados na fila.

- **SFQ.** É uma disciplina de fila sem classe. Ela pertence à família dos algoritmos de enfileiramento justo, pois objetiva garantir que todos os fluxos de dados tenham acesso à largura de banda disponível, distribuindo-se o recurso de forma mais

igualitária possível. O algoritmo instancia várias filas do tipo FIFO e, através de funções *hash*, cada pacote é enfileirado nessas filas, baseando-se no endereço IP de origem-destino e na porta de origem. Porém, podem ocorrer colisões entre os índices (estocasticamente) que são gerados pela função *hash*. Além disso, na transmissão de pacotes cada fila é servida em turno (*round-robin*) [33].

- **RED.** É uma disciplina de fila sem classe que gerencia o tamanho da fila de maneira mais robusta. As filas baseadas nas tradicionais políticas de FIFO, em caso de congestionamento, simplesmente os pacotes são descartados pela cauda da fila, o que pode não ser o melhor comportamento. O RED, também realiza o descarte pela cauda, mas de uma forma mais gradual e eficiente. Em linhas gerais, um limite para marcação aleatória dos pacotes na fila estabelecida e, após o comprimento da fila ultrapassar esse limite, os pacotes marcados começam a ser descartados [34].

- **HTB.** É uma disciplina de fila com classes. Essa disciplina permite controlar a largura de banda na transmissão dos pacotes por um enlace, permitindo particionar a largura de banda total do enlace. Desta forma, é possível que um usuário utilize um enlace físico para simular outros enlaces mais lentos e assim transmitir diferentes tipos de dados sobre este enlace simulado [35]. Quando uma classe é criada, o *kernel* do Linux automaticamente associa uma política de FIFO a ela[36].

O Algoritmo 2 apresenta o o módulo OpenFlowQoS, totalmente itegrado ao *datapath*, para fazer o processamento das mensagens com o cabeçalho estampada para o tipo da mensagem `OFPT_QOS_QUEUEING_DISCIPLINE`, ou seja, apto a fazer o tratamento das mensagens de QoS. Por outro ponto de vista, o módulo OpenFlowQoS possibilita aos usuários implementarem suas aplicações e e controlarem os recursos de QoS da rede, de forma dinâmica.

Algoritmo 2: Manipulação de Mensagem de QoS no Switch

```
CorpoMsg ← RemoverCorpo (MensagemQoS);
TipoMsg ← RemoverTipo (MensagemQoS);
switch TipoMsg do
    case OFP_QOS_SCHED_HTB
        TipoErro ← ProcessarMsgQosHtb (DatapathId, CorpoMsg);
    case OFP_QOS_SCHED_SFQ
        TipoErro ← ProcessarMsgQosSfq (DatapathId, CorpoMsg);
    case OFP_QOS_SCHED_RED
        TipoErro ← ProcessarMsgQosRed (DatapathId, CorpoMsg);
    case OFP_QOS_SCHED_PFIFO
        TipoErro ← ProcessarMsgQosPfifo (DatapathId, CorpoMsg);
    case OFP_QOS_SCHED_BFIFO
        TipoErro ← ProcessarMsgQosBfifo (DatapathId, CorpoMsg);
    otherwise  TipoErro ← OFP_QOS_ERROR_UNKNOWN_MSG;
endsw
if TipoErro != OFP_QOS_ERROR_NONE then
    ReportarErroParaControlador (DatapathId, Controlador, TipoErro);
```

No corpo da mensagem de QoS, diferentes subtipos de mensagens podem ser "carregadas". Esses subtipos indicados nas declarações `case` do agoritmo acima, são

usados para indicar o tipo do escalonador de pacotes (SFQ, RED, HTB ou FIFO) a serem instanciados ou removidos das portas dos dispositivos. Para cada escalonador existe uma função que se comunica com o *kernel* do Linux através da família de *socket* `AF_NETLINK` para fazer as configurações de baixo nível.

No entanto, durante o processamento das novas mensagens de QoS erros podem ocorrer, como por exemplo, o *switch* poderia receber uma mensagem para configurar um escalonador de pacotes na qual não é suportado pelo *datapath*. Em casos como esses, uma mensagem de notificação do erro é transmitida ao controlador. Dessa forma, o administrador poderia estar ciente da ocorrência de erros que podem acontecer durante uma fase de configuração.

Netlink é uma das formas de comunicação entre processos que o Linux oferece ao espaço do usuário, sendo também uma família é uma família de *socket* oriunda do sistema BSD para suportar o envio de mensagens e requisição de informações do espaço do *kernel* a partir do espaço do usuário. Entretanto, diferentemente de outras famílias de *socket*, como por exemplo, *socket* da família `AF_INET` para modelos comunicação remota cliente-servidor, o *socket* Netlink é utilizado somente para comunicação do *kernel* com o espaço do usuário e vice-versa, isto porque na comunicação Netlink os processos são identificados pelos seus endereços de processo (*Process ID*, PID). A Figura 8 apresenta um formato básico de uma mensagem Netlink para comunicação com o subsistema de controle de tráfego do *kernel* do Linux.

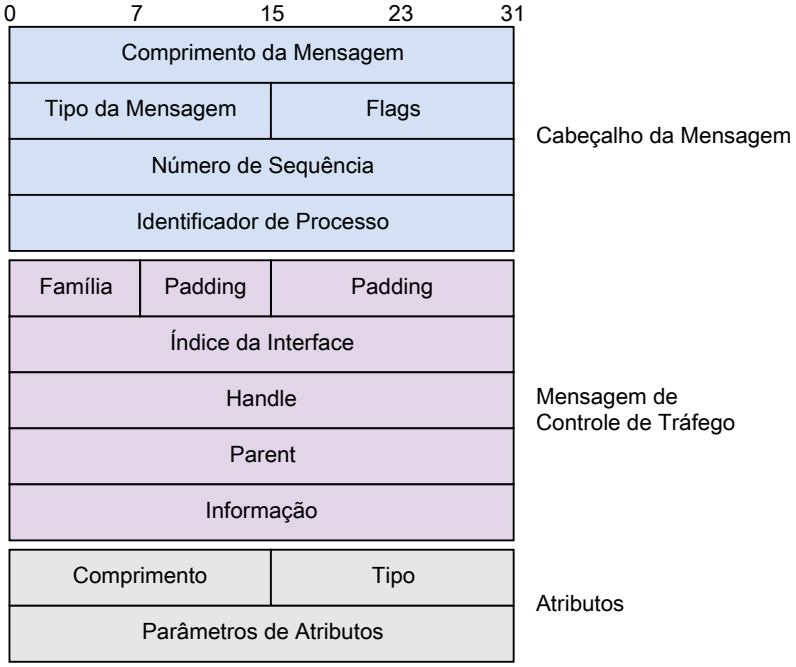


Figura 8: Formato básico da mensagem Netlink para comunicação com o subsistema de controle de tráfego do *kernel* do Linux

Todas as mensagens de QoS recebidos pelo *switch* são mapeadas para uma mensagem Netlink. Na comunicação com *kernel*, um canal de comunicação com *socket* da

família Netlink é aberto para a transmissão da mensagem Netlink e, desta forma, o módulo OpenFlowQoS pode adicionar, remover ou atualizar configurações de banda ou número de filas. No Apêndice C as mensagens de QoS inseridas no OpenFlow são apresentadas.

O formato da mensagem Netlink apresentado na Figura 8 é usado pelo módulo OpenFlowQoS para instanciar os escalonadores FIFO, RED, SFQ e HTB no *kernel* do sistema operacional do *switch*. Essa mensagem consiste de (1) um cabeçalho, comum a todas os tipos de mensagens; (2) um corpo, onde os valores dos campos variam com o tipo de ação desejada (adição, remoção ou atualização) de um escalonador de pacotes e (3) parâmetros de atributos, sendo que seus valores variam de acordo com o tipo do escalonador, por exemplo, para FIFO, o atributo será o tamanho da fila, enquanto que para o escalonador HTB poderia ser o identificador da classe.

Os campos da mensagem Netlink são descritos nos tópicos seguintes:

- Cabeçalho da Mensagem
  - **Comprimento da Mensagem -** tamanho total em *bytes* da mensagem Netlink incluindo incluindo o cabeçalho.
  - **Tipo da Mensagem -** define o formato dos dados que segue no corpo da mensagem Netlink.
  - **Flags -** *bits* de controle para requisitar uma configuração no *kernel*.
  - **Número de Sequência -** correlacionar requisição e resposta
  - **Identificador de Processo -** idenficador do originador da mensagem Netlink
- Mensagem de Controle de Tráfego
  - **Família -** família do *socket* a ser usado na comunicação com *kernel*
  - **Padding -** campos de *bits* não utilizados
  - **Índice da Interface -** identificador único de interface
  - **Handle** e **Parent -** campos usados para "posicionar" um escalonador de pacotes no *kernel*
  - **Informação -** informações descritivas da mensagem Netlink
- Parâmetros de Atributos
  - **Comprimento -** tamanho em *bytes* dos atributos
  - **Tipo -** tipo do escalonador de pacotes
  - **Parâmetros de Atributos -** parâmetros dos escalonadores de pacotes

4.3 Conclusões do Capítulo

Este capítulo apresentou a proposta QoSFlow, uma proposta para habilitar controle dos escalonadores de pacotes para provisionar QoS na rede definida por *software*, baseada no OpenFlow. Sua arquitetura é composta por três níveis, chamados de plano administrativo, plano de controle e plano de encaminhamento, onde suas responsabilidades foram apresentadas neste capítulo com auxílio de pseudocódigos e figuras.

O QoSFlow se utiliza de alguns recursos que o *kernel* do sistema operacional Linux oferece para prover QoS de forma dinâmica e abstraída de configurações de baixo nível. Nesse quesito, os escalonadores HTB, SFQ e RED foram agregados ao OpenFlow.

Futuramente, se pretende realizar incrementos no QoSFlow para: suportar novos tipos de escalonadores de pacotes bem como a configuração sob demanda dos mesmos. Além de dar suporte a monitoramento e controle de banda sob demanda.

CAPÍTULO 5

Avaliação da Proposta

Este capítulo apresenta cenários, testes e a análise dos resultados obtidos com a proposta QoSFlow.

5.1 Cenário com TP-Link 1043ND

5.1.1 Configuração de Teste

O TP-Link 1043ND[1] é um *switch* de fábrica desenvolvido para prover acesso a redes locais. O equipamento possui 400 Mhz de processamento, 32 MB de memória RAM, *chipset* Atheros, 4 portas LAN de 1 Gigabit Ethernet, 2 interfaces sem fio IEEE 802.11b/g/n e uma porta WAN de 1 Gigabit Ethernet. Além de uma interface USB 2.0 e um *firmware* proprietário que provê serviços DHCP, DNS, controle de banda, controle de acesso, *firewall* e comutação de pacotes. No entanto, para que o dispositivo pudesse operar como *switch* OpenFlow, seu *firmware* original foi substituido por uma distribuição Linux chamada OpenWrt Backfire[2].

O sistema operacional OpenWrt é descrito como uma distribuição Linux para dispositivos embarcados. Ele oferece gerenciamento de pacotes Linux e sistema de arquivos com permissão de escrita para que os desenvolvedores possam desenvolver suas próprias aplicações e instalá-los dentro do dispositivo. Portanto, o usuário matém-se livre de aplicações dos fabricantes e *firmware* estaticamente fornecida pelo proprietário [37].

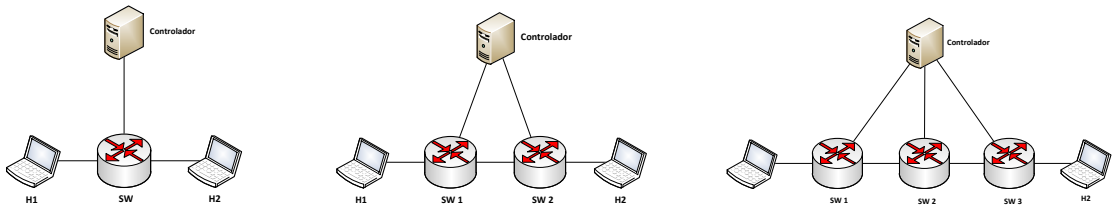
O OpenFlow, mais especificamente, o *software switch* é implementado e executado

[1] http://www.tp-link.com/
[2] https://openwrt.org/

sobre o sistema operacional OpenWrt embarcado no TP-Link. Os procedimentos da instalação foram seguidas de acordo o tutorial disponível na página oficial da comunidade do OpenFlow[3]. O procedimento para habilitar o OpenFlow no equipamento TP-Link, consiste em três etapas: (1) baixar o código-fonte do *kernel* do OpenWrt Backfire, (2) baixar a versão do OpenFlow padrão fornecida pela equipe da Universidade de Stanford e (3) compilar o *kernel* do sistema operacional em conjunto com o OpenFlow *software switch* e instalá-lo no TP-Link.

Um procedimento semelhante, para habilitar o OpenFlow padrão, é realizada para habilitar a proposta QoSFlow no TP-Link. A principal diferença no processo de instalação é o uso de um *framework* de desenvolvimento chamado OpenWrt SDK. Esse SDK é utilizado para gerar um arquivo que possui uma extensão com o nome `.ipk`. Sendo este, um pacote que possui um formato específico do sistema OpenWrt. Por meio desse SDK, um pacote QoSFlow instalável é produzido.

Nos testes, são utilizados dois hospedeiros e *switches* TP-Link 1043ND com *firmware* modificados para OpenWrt, conforme apresenta a Figura 9. Os hospedeiros consistem em um *notebook* Acer (processador Intel Core i5 2.4Ghz, memória RAM de 6 GB, 1 porta Gigabit Ethernet e 1 porta sem fio IEEE 802.11b/g/n) com sistema operacional Fedora 17 i686 (versão do *kernel* 3.6.8-2) e um *netbook* Asus (processador AMD Dual Core 800 Mhz, memória RAM de 2 GB, 1 porta Gigabit Ethernet, 1 porta sem fio IEEE 802.11b/g) com sistema operacional Debian 6.0 x86_64 (versão do *kernel* 3.2.0-3). Esses hospedeiros são utilizados para atuarem como gerador e receptor de tráfegos de teste.



(a) Topologia com 1 switch e 2 hospedeiros (b) Topologia em linha com 2 switches e 2 hospedeiros (c) Topologia em linha com 3 switches e 2 hospedeiros

Figura 9: Topologias de testes utilizadas com TP-Link 1043ND

O controlador da rede, é situado em um *Notebook* HP DM4-1055br com processador Core i5 2.0 Ghz e memória RAM de 3 GB com sistema operacional Ubuntu 11.10 i686 (versão do *kernel* 3.0.0-12).

5.1.2 Procedimento

No processo dos testes, tráfegos sobre TCP e UDP são inseridas na rede a partir do gerador de tráfego chamado `iperf`[4]. Essa ferramenta permite avaliar o desempenho da

[3]http://www.openflow.org/
[4]http://iperf.sourceforge.net/

rede a partir das métricas de largura de banda, atraso, *jiiter* e perdas de pacotes. Além do gerador de tráfego, uma ferramenta do Linux chamada `top`[5] foi usada para medir o consumo de processamento e memória dos dispositivos.

Os resultados dos testes são coletados e analisados graficamente. Com exceção do resultado obtido e apresentado na Figura 12(a), todos os testes são executados durante 120 segundos e uma amostra é coletado a cada intervalo de 1 segundo. Esses testes consistem em 3 fases.

A primeira fase de testes, tem como objetivos avaliar o controle da largura de banda, onde os resultados representados estão expressas na Figura 10. Nessa fase, em cada passo, é gerado 2 fluxos: (i) TCP x TCP, (ii) UDP x UDP e (iii) TCP x UDP para verificar o isolamento, e (iv) dois tráfegos TCP são inseridos para avaliar o controle dinâmico de recursos para esses fluxos. Esses testes são realizados na topologia da Figura 9(a), onde os fluxos são enviados de H1 para H2 e a taxa de *bits* recebidos são coletados e analisados no H2.

A segunda fase de testes, tem objetivos em avaliar a taxa de utilização dos recursos de processamento e memória dos dispositivos de rede, cujo os resultados estão ilustradas na Figura 11. Dessa forma, avaliar a viabilidade do uso da proposta. Nesses testes, as topologias das Figuras 9(a), 9(b) e 9(c) são utilizadas, sendo que um fluxo TCP é transmitido de H1 para H2.

Na terceira fase de testes, uma avaliação do tempo de resposta das funções que executam as operações de adição, remoção e atualização dos escalonadores de pacotes, nos *switches*, são analisados. Além de avaliar o desempenho dos fluxos com o SFQ. Nessa fase, a topologia da Figura 9(a) é utilizada, onde 40 repetições são realizadas para as três operações. O resultado se encontra expressa na Figura 12(a). Por fim, a topologia de rede da Figura 9(c) é considerada para avaliar o desempenho no escalonador SFQ, sendo que, um fluxo TCP é transmitida para a fila 1 que possui limite de 6 $Mbps$ e 3 fluxos TCP são transmitidos para a fila 2 que possui limite de 4 $Mbps$, de H1 para H2. Os resultados são apresentados nas Figuras 12(b), 12(c) e 12(d).

5.1.3 Resultados

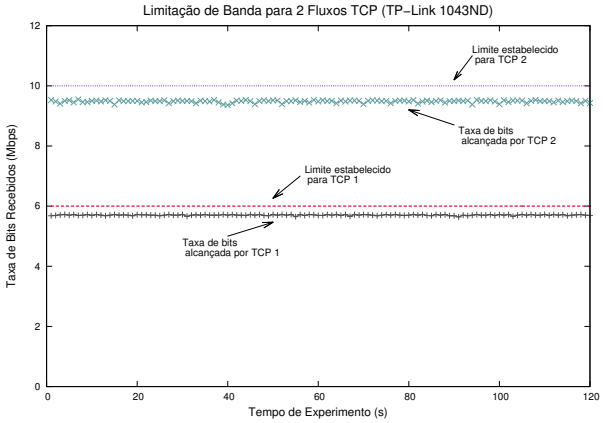
5.1.4 Avaliação do Isolamento de Tráfego

Os gráficos das Figuras 10(a), 10(b), e 10(c) apresentam que, os limites de largura de banda configurados para 6 $Mbps$ e 10 $Mbps$, são respeitados durante um intervalo de 120 segundos. No entanto, os valores exatos de 6 $Mbps$ e 10 $Mbps$ não são alcançados, possuindo diferenças mínimas em torno de 0.2 $Mbps$ a 0.5 $Mbps$, aproximadamente. Isto se deve as configurações do próprio *hardware* dos dispositivos.

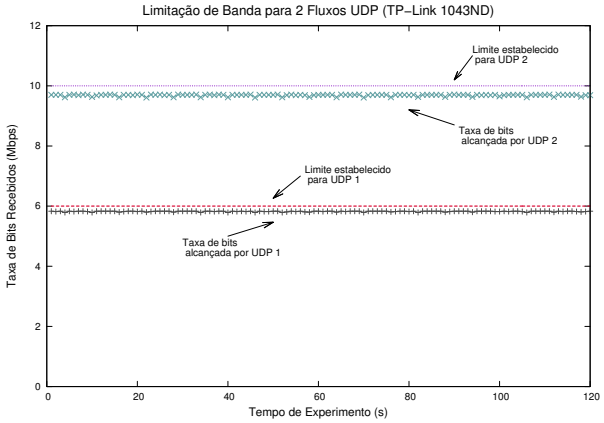
O gráfico da Figura 10(d) ilustra o controle dinâmico da largura de banda para

[5]http://procps.sourceforge.net/index.html

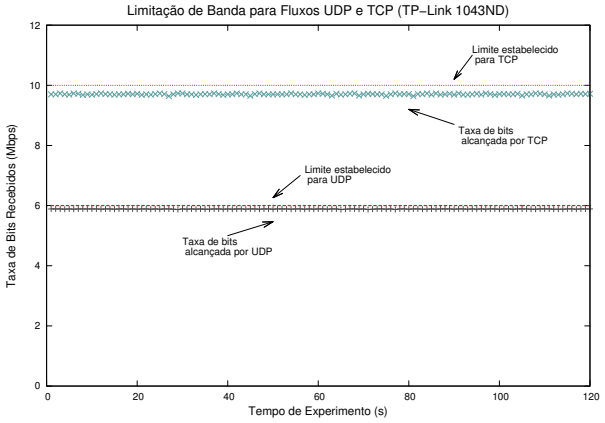
dois fluxos TCP durante um intervalo de 120 segundos. Nesse intervalo, ajustes dos limites de banda são realizados nos instantes 30 e 60 segundos. Sendo assim, pela Figura 10(d), nos instantes 30 e 60 segundos, ocorrem mudanças na taxa de *bits* recebidos dos fluxos TCP.



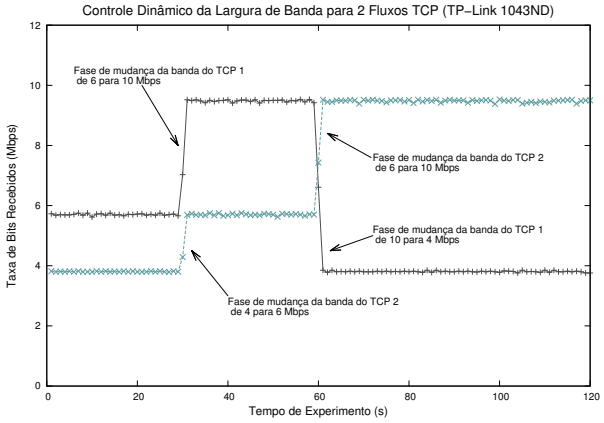
(a) Garantia de banda para TCP x TCP



(b) Garantia de banda para UDP x UDP



(c) Garantia de banda para UDP x TCP



(d) Controle de banda dinâmica

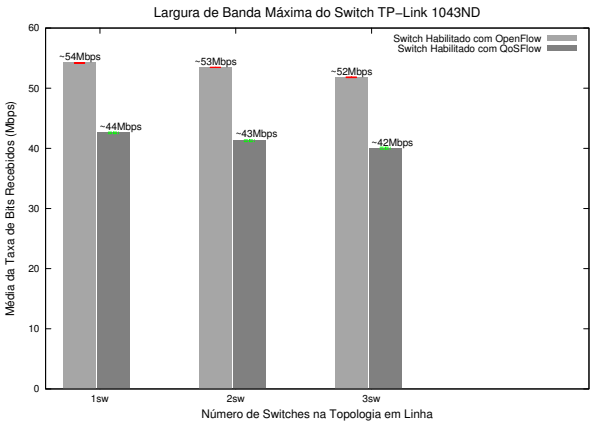
Figura 10: Controle da largura de banda no TP-Link 1043ND habilitado com QoSFlow

Inicialmente, os limites de 6 *mpbs* e 4 $Mbps$ são estabelecidos para os fluxos TCP 1 e TCP 2, onde os mesmos são mantidos até o instante 30 s. Nesse momento, tais limites são ajustados para ambos os fluxos. Para o fluxo TCP 1, um novo limite é estabelecido para 10 $Mbps$, enquanto que para o fluxo TCP 2, o limite é ajustado para 6 $Mbps$. Esses valores são mantidos até o instante 60 s, onde novamente, novos limites são estabelecidos. Nesse instante, os limites de 4 $Mbps$ e 10 $Mbps$ são estabelecidos, respectivamente, para os fluxos TCP 1 e TCP 2 até o instante de 120 s, momento em que termina o tempo de experimento.

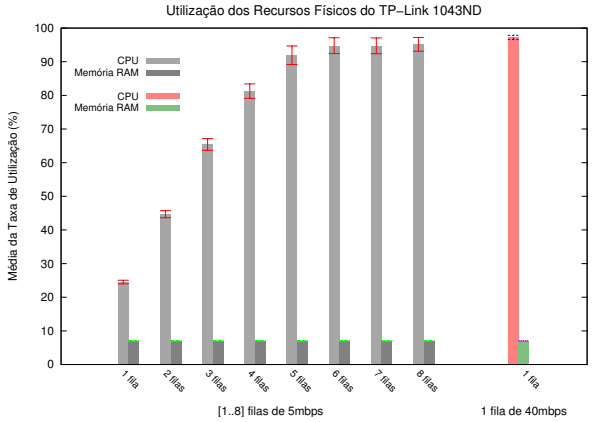
5.1.5 Avaliação da Utilização de Recursos Físicos

Os resultados da Figura 11 apresentam a taxa de utilização dos recursos de processamento e memória RAM e a capacidade máxima do enlace alcançados. Na avaliação

desses resultados, um nível de confiança de 95 % foi considerada no cálculo do intervalo de confiança para a média das amostras obtidas.



(a) Capacidade máxima do enlace



(b) Sobrecarga do processador e da memória RAM

Figura 11: Taxa de utilização do processamento e da memória RAM e a capacidade máxima do enlace alcançada com TP-Link 1043ND habilitado com QoSFlow

Na avaliação da capacidade máxima do enlace do dispositivo, as topologias da Figura 9 são utilizadas, cujo os resultados observados estão representados na Figura 11(a). Esses teste, teve o objetivo de verficar a capacidade máxima que pode ser alcançada pelo dispositivo atuando como, *switch* OpenFlow e como *switch* QoSFlow.

A capacidade máxima do enlace alcançada com um, dois e três *switches* OpenFlow, respectivamente, são 54 *Mbps*, 53 *Mbps* e 52 *Mbps*, aproximadamente. Enquanto que, a capacidade máxima atingida pelos *switches* QoSFlow, respectivamente, são 44 *Mbps*, 43 *Mbps* e 42 *Mbps*, aproximadamente. Logo, é perceptível que o desempenho do *switch* QoSFlow é menor que o OpenFlow.

A diferença de desempenho, de aproximandamente 10 *Mbps*, é justificado pelo fato de ocorrer um atraso de enfileiramento nas filas virtuais (no nível do sistema operacional) do *switch* QoSFlow. No entanto, apesar da existência dessa queda de desempenho, o *switch* QoSFlow consegue manter o limite estabelecido e permitir o controle de banda dinâmico da rede, como foi apresentado anteriormente pela Figura 10.

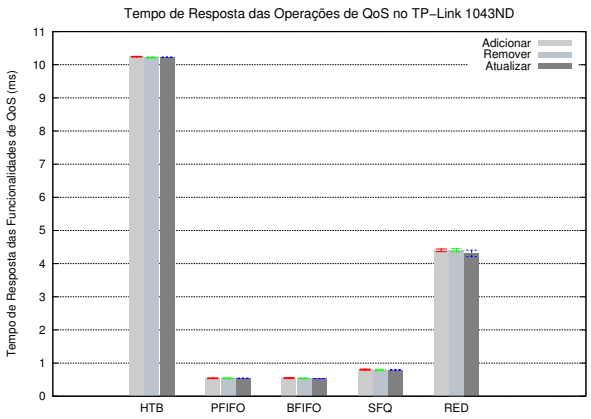
O gráfico apresentado na Figura 11(b) apresenta um nível de saturação do processador. O testes são realizados variando o número de filas nas portas de saída dos dispositivos e variando o limite da largura de banda para essas filas.

Nove fuxos são inseridos individualmente, isto é, no príncipio é instanciado uma única fila que possui o limite de 5 *Mbps*, onde um fluxo TCP é gerado de H1 para H2 durante 120 segundos. Na próxima rodada, são instanciados duas filas que possuem limite de 5 *Mbps* cada uma e dois fluxos são inseridos, uma para cada fila. E, assim por diante até a oitava rodada. Na nona rodada, uma única fila com capacidade de 40 *Mbps* é instanciado. Dessa forma, em todas as execuções, o consumo de processamento e memória RAM também são observados.

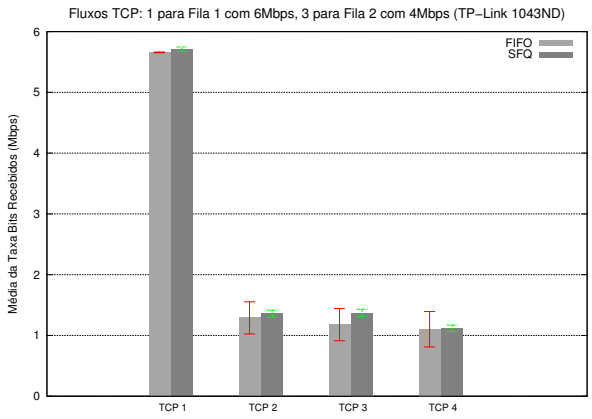
O nível de saturação da memória permenace constante em 7 % durante todas as etapas dos testes, como está apresentado na Figura 11(b). No entanto, a sobrecarga da unidade de processamento aumenta, conforme a utilização do limite de banda estabelecido para as filas. Em outras palavras, a saturação do processador aumenta em função do somatório dos recursos utilizados pelos fluxos. Isto pode ser concluido a partir da observação da oitava rodada e da nona rodada, sendo que o teste com oito filas de de 5 *Mbps*, apresenta a mesma sobrecarga que o teste com uma única fila de 40 *Mbps*. Portanto, o que ocasiona a sobrecarga no processamento, durante a comutação, não é o número de filas existentes nas portas de saída do *switch*.

5.1.6 Avaliação dos Escalonadores de Pacotes

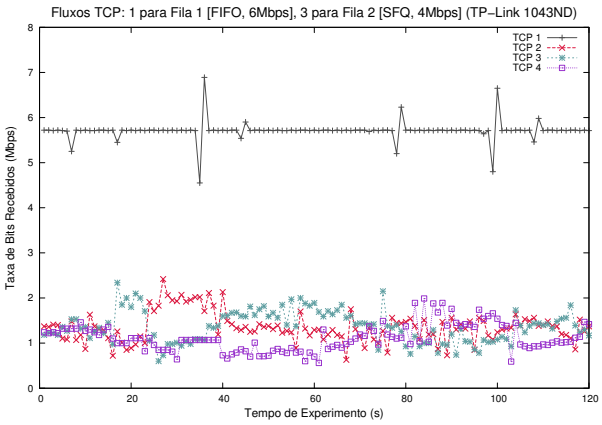
A Figura 12(a) apresenta um gráfico que ilustra a média do tempo de resposta das funções que executam operações sobre os escalonadores de pacotes. Tal tempo de reposta envolve somente o atraso das operações de QoS no nível do dispositivo. Logo, outros tipos de atrasos, como de comunicação do controlador com o *switch*, não estão incluídos. Para essa avaliação 40 mensagens de QoS são transmitidas, do controlador para o dispositivo.



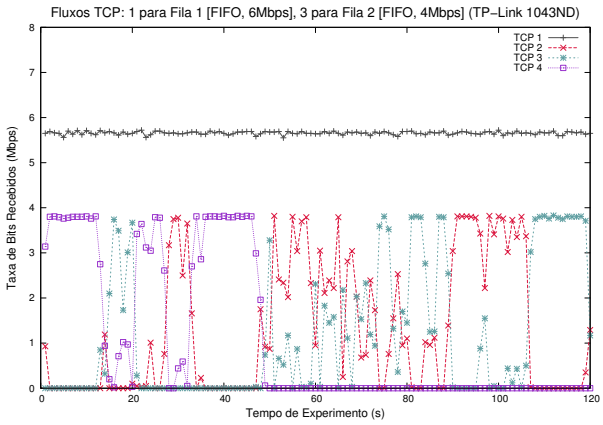
(a) Tempo de Resposta das Operações de QoS



(b) Desempenho do SFQ



(c) Maior estabilidade dos fluxos com a justiça do SFQ



(d) Menor estabilidade presenciados pelos fluxos

Figura 12: Desempenho dos escalonadores de pacotes

A Figura 12(a) apresenta que os tempos são bem reduzidos, sendo a média do maior atraso está em torno de 10 ms para operações sobre o HTB. Esse escalonador, é utilizado, na proposta, para associar limites de largura de banda para as filas. Sendo assim, os valores de média desse escalonador representa o tempo para adição, remoção ou atualização de parâmetros do HTB, no *kernel*. Tais tempos, são referentes ao dispositivo TP-Link que apresenta uma capacidade de processamento relativamente baixa, de 400 Mhz. Logo se estima que esse tempo, e dos demais escalonadores de pacotes, será menores sobre nos equipamentos que tenham uma capacidade de processamento maior.

Dentre todos os tempos de respostas, o HTB apresenta o maior tempo. Seguindo do RED, SFQ e FIFO (BFIFO/PFIFO). A justificativa para essa dispariadade do HTB em relação aos demais, ocorre devido ao maior número de parâmetros que são passados do espaço do usuário para o espaço do *kernel* do sistema operacional e por ele ser uma disciplina de fila com classes. Logo, mais complexos que os demais escalonadores, o que demanda maior tempo de processamento. A idéia é análoga para o RED. O RED apesar de ser um escalonador de pacote sem classe, sua estrutura de dados apresenta maiores parâmetros que precisam ser processados no *kernel* do sistema operacional.

Para verificar a eficiência da utilização dos escalonadores de pacotes no QoSFlow, um experimento foi realizado com o SFQ. Os resultados estão representados nos gráfico das Figuras 12(b), 12(c) e 12(d). A Figura 12(b) representa a média da taxa de *bits* recebidos utilizando FIFO e SFQ. Enquanto que os gráficos da Figura 12(c) e 12(d), representam os valores brutos coletados no intervalo de 120 segundos.

O gráfico da Figura 12(b) apresenta a justiça do SFQ com os três fluxos TCP (TCP 2, TCP 3 e TCP 4) que são direcionados para a fila 2 que possui um limite de 4 $Mbps$. Esses fluxos atingem 1.36 Mb/s, 1.37 Mb/s e 1.11 Mb/s, respectivamente. Enquanto que com FIFO, esses fluxos atingem 1.28 Mb/s, 1.17 Mb/s e 1.10 Mb/s, respectivamente. Entretanto, o SFQ apresenta uma desempenho mais próximo do FIFO quando existe um único fluxo. Isto é observado pelo desempenho do fluxo TCP 1, onde este é direcionado para a fila 1 que possui um limite de 6 $Mbps$. Sendo assim, o fluxo atinge um valor de 5.7 Mb/s e 5.6 Mb/s, respectivamente, com SFQ e FIFO.

Analizando os gráficos das Figuras 12(c) e 12(d), o fluxo TCP 1 viola o limite de banda estabelecido em alguns instantes como 36 s, 80 s e 100 s com SFQ, o que não ocorre com FIFO. Estes picos ocorrem com SFQ, devido a atualização de índices *hash* para evitar colisões de índices. Além disso, as amostras dos TCP 2, TCP 3 e TCP 4 apresentam uma variabilidade menor com SFQ, ou seja, esse escalonador oferece maior estabilidade aos fluxos.

Sendo assim, o SFQ lida melhor com múltiplos fluxos que estão na mesma fila. Na 12(d) se observa que a utilização de banda ociosa é desproporcional entre os fluxos TCP 2, TCP 3 e TCP 4 que são enfileirados na fila 2 com FIFO. Em outras palavras, existem momentos em que um fluxo domina os demais utilizando todo o recurso que foi alocado. Ao contrário do que ocorre com a justiça do SFQ, conforme é observado pela Figura 12(c), todos os fluxos tentam utilizar uma quantidade proporcional do recurso

alocado. Inclusive, nesse gráfico se nota que nenhum fluxo fica em "silêncio", isto é, a todo instante os *bits* estão sendo recebidos pelo destino, visto que nenhum dos fluxos atingem o eixo x do gráfico, quando outro fluxo utiliza o recurso compartilhado da fila 2. Isto é importante para algumas aplicações que necessitam de transmissão contínua, por exemplo, aplicações de tempo real.

5.2 Cenário com Mininet

5.2.1 Configuração de Teste

Mininet é um ambiente para prototipação de redes OpenFlow que utiliza virtualização para prototipar rapidamente uma rede com centenas de nós. As principais características positivas é a sua flexibilidade, escalabilidade e o realismo. Ambientes de simuladores por ser simplificado, é pobre em realismo, pois o código desenvolvido não é o mesmo que é implantado em um cenário real. Enquanto que, os ambientes de virtualização com máquinas virtuais possuem pouca escabilidade [38].

O ambiente de prototipação é instalável em máquinas *desktop*, sobre servidores como Xen ou sobre máquinas virtuais como VirtualBox e VMware. A comunidadde dos desenvolvedores do Mininet de Stanford disponibilizam uma máquina virtual com todas as dependências previamente instaladas em sua página Web [6]. Dessa forma, os pesquisadores de rede podem baixar, configurar, prototipar rapidamente sua topologia de rede e executar experimentos. Essa máquina virtual pode ser facilmente copiado e distribuído para outros pesquisadores. Essa máquina virtual possui aproximadamente um tamanho de 800 MB.

Nos experimentos com Mininet, a máquina virtual disponibilizada para *download* pela comunidade de Stanford é utilizada. Entretanto, para que Mininet possa oferecer suporte a proposta QoSFlow, o *switch* QoSFlow teve de ser integrado ao Mininet. Basicamente, esse processo consistiu em instalar o *switch* da proposta na máquina virtual e depois criar um *script* Python para poder instanciar *switches* QoSFlow na topologia dos experimentos.

A máquina virtual do Mininet foi utilizado sobre o VirtualBox no *Notebook* HP DM4-1055br atuando como máquina hospedeira. Essa máquina possui processador Core i5 2.0 Ghz e memória RAM de 3 GB com sistema operacional Ubuntu 11.10 i686 (versão do *kernel* 3.0.0-12). O controlador ficou situado sobre essa mesma máquina hospedeira, mas executando fora da máquina virtual.

O cenário da Figura 13 é considerado nos testes. A rede consiste em 2 nós hospedeiros e 1 *switch* QoSFlow virtualizados. O objetivo dos testes verficar o uso da proposta em um ambiente virtual com Mininet cujo é um ambiente com recursos de processamento e memória compartilhados entre os nós, ou *switches*, virtuais. Em outras palavras, verficar se o controle de tráfego ocorre sobre nós virtualizados no Mininet,

[6] http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet

sabendo-se que, os testes realizados no cenário físico com *switches* TP-Link, alcançaram êxitos satisfatórios, como são observados na Seção 5.1.

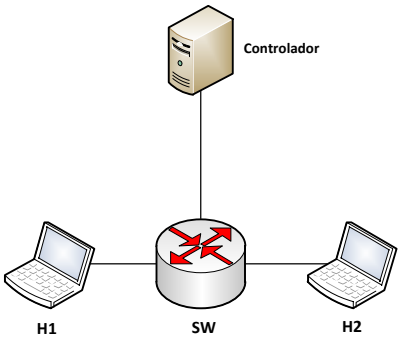


Figura 13: Cenário da rede utilizada no Mininet

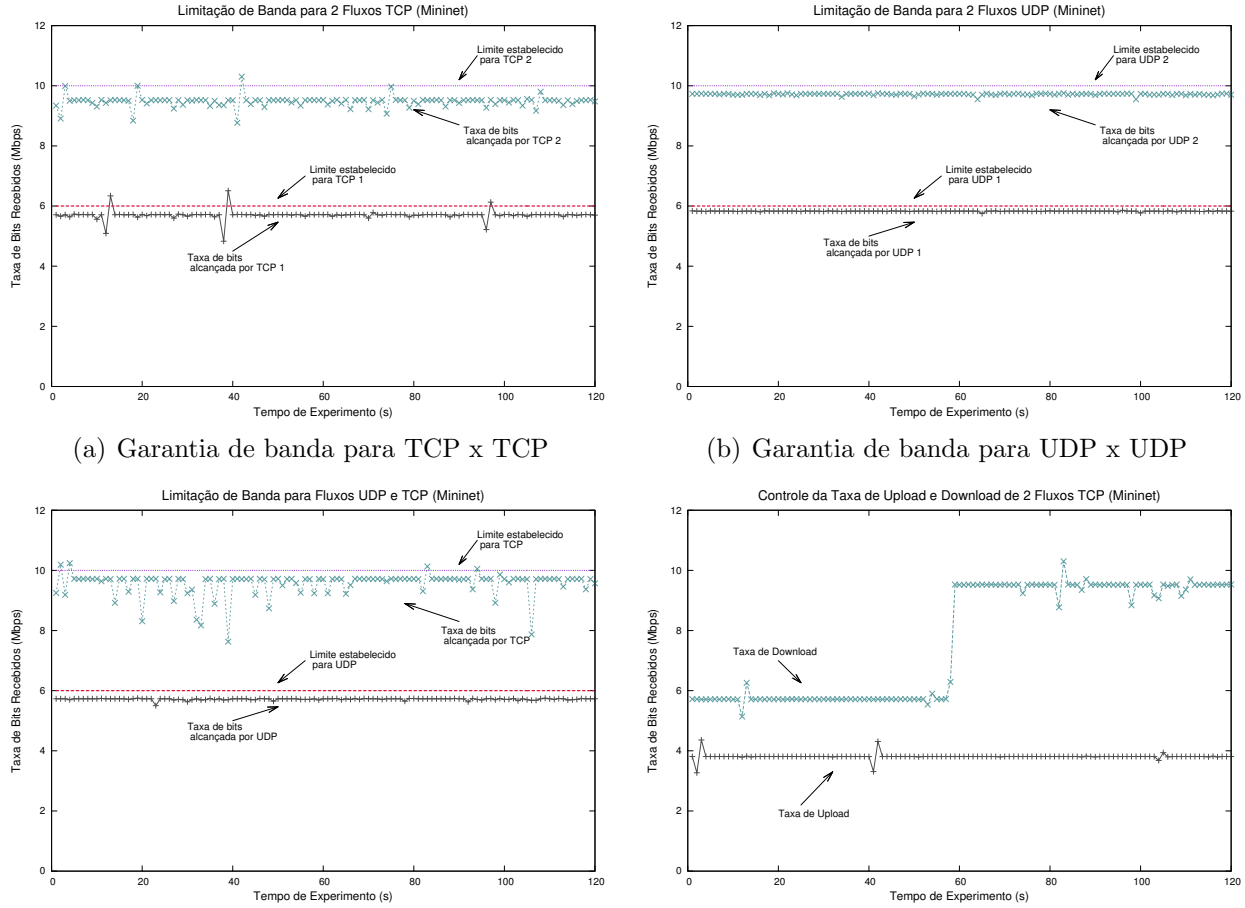
5.2.2 Procedimento

Os testes consistem em inserir tráfegos TCP e UDP na rede através da ferramenta `iperf`. De forma análoga ao procedimento apresentado na Seção 5.1, fluxos são gerados a partir de H1 para H2 durante o intervalo de experimentação de 120 s, onde em cada segundo é extraído uma amostra da largura de banda dos fluxos. Os resultados são obtidos e avaliados graficamente na Figura 14.

Testes semelhantes, ao cenário com TP-Link, são realizados para validar o desempenho da proposta QoSFlow no cenário da rede virtualizada com Mininet. As várias etapas dos testes consistem dois fluxos em cada etapa: ocorrem para fluxos (i) TCP x TCP, (ii) UDP x UDP e (iii) TCP x UDP para verificar o isolamento de recursos e (iv) um fluxo TCP de H1 para H2 e ao mesmo tempo, mais um fluxo TCP de H2 para H1 para avaliar o controle dinâmico da taxa de *uplink* e *downlink*.

5.2.3 Resultados

Conforme os resultados gráficos apresentados nas Figuras 14(a), 14(b), 14(c), os limites são estabelecidos para os valores de 6 *Mbps* e 10 *Mbps* durante 120 segundos de experimentação. Apesar da existência de algumas amostras fora do padrão de tráfego (*outliers*), o *switch* QoSFlow garante os limites de banda que foram configurados para os fluxos. O desempenho dos fluxos UDP x TCP apresentaram-se piores que os fluxos TCP x TCP e UDP x UDP. Entretanto, o *switch* QoSFlow, no Mininet, conseguiu manter os limites estabelecidos. Isto é, não permite que os fluxos extrapolem os limites que foram estabelecidos.



(a) Garantia de banda para TCP x TCP

(b) Garantia de banda para UDP x UDP

(c) Garantia de banda para UDP x TCP

(d) Controle da taxa de download e upload

Figura 14: Controle da largura de banda no ambiente Mininet com QoSFlow

O gráfico da Figura 14(d) ilustra o controle dinâmico da largura de banda sobre dois fluxos TCP que foram inseridos simultaneamente em sentidos opostos, ou seja, fluxos foram gerados de H1 para H2 e de H2 para H1. Nesse teste, as amostras de *uplink* foram coletados em H1 e de *downlink*, em H2 durante um intervalo de 120 segundos, um ajuste na taxa de *download* foi realizado no instante 60 segundos. O ajuste no limite foi feita de 6 *Mbps* para 10 *Mbps*. Nesse sentido, notou-se que ocorreram mudanças na taxa de *bits* recebidos do fluxo TCP de *download* que estava sendo transmitido.

As disparidades de *outliers* nos resultados observados, se deve, provavelmente ao compartilhamento de recurso (processamento e memória) do Mininet não ser perfeita, pois pelas amostras obtidas no cenário físico com os equipamentos TP-Link 1043ND, o QoSFlow conseguiu, além de garantir a largura de banda, também conseguiu fazer com que os tráfegos TCP e UDP funcionassem adequadamente, tanto é que não foram observados situações atípicas para esses fluxos, conforme apresenta os gráficos da Figura 14.

5.3 Estudo de Caso com QoE

5.3.1 Configuração de Teste

Esta seção visa apresentar resultados e análises de um estudo de caso realizado com Qualidade de Experiência (QoE) [39]. O QoE são medições utilizadas para determinar a satisfação do usuário com relação a rede de comunicações pela qual os fluxos multimídias estão sendo transmitidos. Essa avaliação, considera conexões fim-a-fim e aplicações que executam e transmitem vídeos sobre uma rede. Diferentemente das métricas e avaliações de QoS que ocorrem o núcleo da rede. As métricas de QoE podem ser vistas como uma consequência de QoS da rede, ou seja, se os equipamentos e os algoritmos de encaminhamento são eficientes, como consequência, as métricas de QoE são melhores.

Evalvid[7] é um arcabouço de ferramentas para avaliar a qualidade de vídeos transmitidos por uma rede de computadores, seja sobre uma topologia simulada ou sobre uma topologia física. O arcabouço possui ferramentas para codificar, decodificar, transmitir e receber vídeos para analizar parâmetros de QoE como o PSNR (*Peak Signal-to-Noise Ratio*). Porém, além da capacidade de medição de métricas de QoE, o Evalvid possibilita a medição de parâmetros de QoS como taxa de perdas de pacotes, atraso e *jitter*.

A topologia da rede utilizada foi em linha com três *switches* físicos com QoSFlow (instalado sobre o *hardware* TP-Link 1043ND) e mais duas estações H1 e H2, como ilustra a Figura 15.

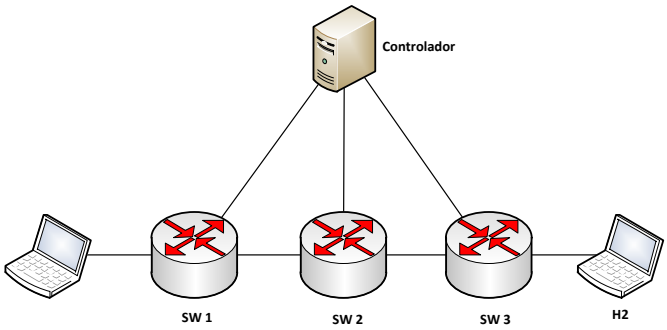


Figura 15: Topologia da rede utilizada nos experimentos com transmissões de vídeos. Os *swicthes* são equipamentos TP-Link 1043ND habilitados com QoSFlow.

Os hospedeiros H1 e H2, consistem em um *notebook* Acer (processador Intel Core i5 2.4Ghz, memória RAM de 6 GB, 1 porta Gigabit Ethernet e 1 porta sem fio IEEE 802.11b/g/n) com sistema operacional Fedora 17 i686 (versão do *kernel* 3.6.8-2) e um *netbook* Asus (processador AMD Dual Core 800 Mhz, memória RAM de 2 GB, 1 porta Gigabit Ethernet, 1 porta sem fio IEEE 802.11b/g) com sistema operacional Debian 6.0 x86_64 (versão do *kernel* 3.2.0-3). Esses hospedeiros são utilizados para atuar como gerador e receptor de fluxos de vídeos.

[7]http://www2.tkn.tu-berlin.de/research/evalvid/EvalVid/docevalvid.html

O controlador da rede, ficou situado remotamente em um *Notebook* HP DM4-1055br com processador Core i5 2.0 *Ghz* e memória RAM de 3 *GB* com sistema operacional Ubuntu 11.10 i686 (versão do *kernel* 3.0.0-12).

5.3.2 Procedimento

Nos experimentos são dois fluxos de vídeos idênticos são transmitidos de H1 para H2 sobre uma fila com capacidade para 10 *Mbps*. Dentre os vários vídeos existentes no site de tutorial do Evalvid[8] é selecionado um vídeo com movimento e duração relativamente longa, chamado Highway. Esse vídeo possui 2000 quadros, tem duração de aproximadamente 1 minuto e 20 segundos, o qual possui uma resolução de 352 x 288 e esse vídeo, apresenta uma visão de uma estrada a partir de um veículo em movimento sobre essa estrada. Os quadros de vídeo são codificados em MPEG-4 (*Moving Picture Experts Group - 4*) e transmitidas a 26.4 $kbyte/s$.

Na avaliação de QoE a métrica PSNR é considerada. O PSNR é uma métrica objetiva que é definido como, a razão entre máximo de sinal recebido pelo erro quadrático médio e é comumente expresso em decibel (dB). Sendo assim, quato maior for valor de PSNR, melhor será a qualidade do vídeo comprimido que é trasmitido pela rede e reconstruído no receptor. Em tais avaliações, um nível de confiaça de 95 % é considerado no cálculo do intervalo de confiança para média do PSNR, onde o impacto do uso dos escalonadores de pacotes FIFO e SFQ são avaliados. Além disso, avaliações visuais do vídeo também são considerados.

5.3.3 Resultados

Pela observação do gráfico da Figura 16, o SFQ apresentou desempenho muito superiores ao FIFO. Isto justifica-se pela justiça do SFQ com os fluxos. Porém, o SFQ apresentou um desempenho muito melhor que o esperado. O desempenho do SFQ foi superior em torno de 48.57 % e 68.57 % com relação ao FIFO com video 1 e video 2, respectivamente. Apesar de o gráfico da Figura 16 não ilustrar PSNR de referência, o SFQ atigiu exatamente o mesmo valor de PSNR, ou seja, de 35.92 dB.

A Figura 17 apresenta uma captura de tela, no mesmo instante, do fluxo de vídeo transmitido para ilustrar a qualidade do vídeo observada visualmente pelo usuário. Essas figuras são referentes ao fluxo de vídeo 2, ou seja, que de acordo com o gráfico da Figura 16, é onde o FIFO apresentou o melhor caso no valor de PSNR.

Um baixo valor de PSNR sinaliza alto valor de perdas de pacotes. Os vídeos são tolerantes a falhas em geral, no entanto, uma taxa de perda elevada pode decrementar consideravelmente a qualidade do vídeo. Pelo gráfico da Figura 16, pode-se inferir que houve altas taxa de perdas de pacotes com o escalonador de pacotes FIFO.

[8] www2.tkn.tuberlin.de/research/evalvid/cif.html

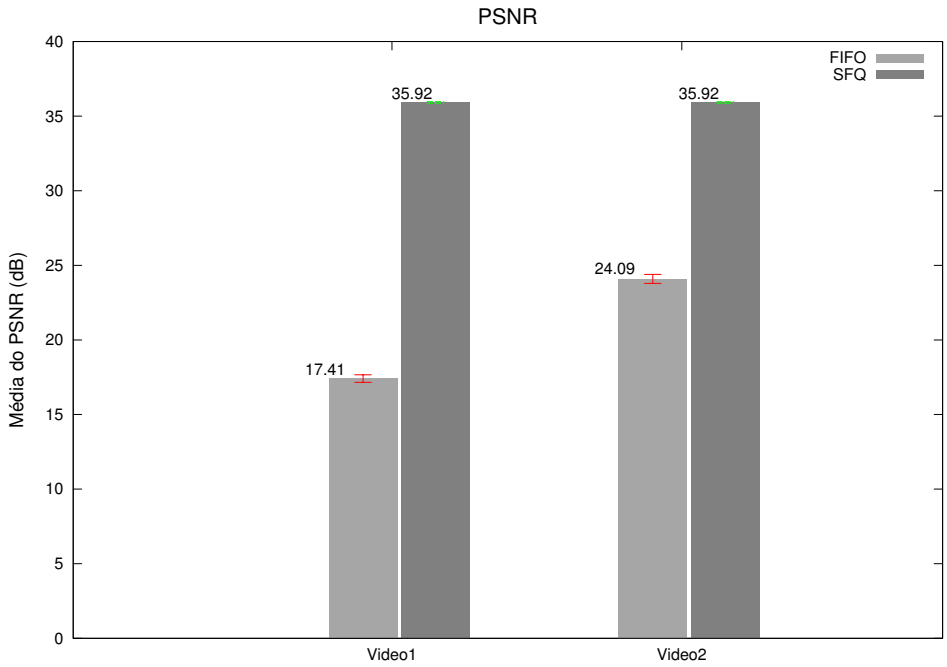


Figura 16: Métrica PSNR

(a) Vídeo original

(b) Desempenho do FIFO

(c) Desempenho do SFQ

Figura 17: Qualidade do vídeo observada pelo usuário

Pelas Figuras 17(b) e 17(c) é observado que o SFQ apresenta desempenho muito satisfatórios com relação ao tradicional escalonador de pacotes FIFO (principal e única forma de escalonamento de pacotes da atual especificação do protocolo OpenFlow). Comparando a Figura 17(b) com a Figura 17(a), a qualidade do vídeo fica muito "danificado" com política de escalonamento FIFO, enquanto que o SFQ, não apresenta diferenças. Em ambos os casos, recursos são ofertados, porém o escalonador FIFO não se apresenta eficiente para situações com múltiplos fluxos em um fila. Portanto, a utilização de escalonadores de pacotes além do FIFO, se torna necessário e importante no provisionamento de QoS em redes OpenFlow.

5.4 Conclusões do Capítulo

Este capítulo apresentou a avaliação dos resultados dos experimentos realizados sobre ambiente físico e virtual. Para os ambientes físicos, os dispositivos TP-Link 1043ND foram habilitados com QoSFlow e para experimentos no ambiente virtual, utilizou-se o Mininet.

Nos experimentos, fluxos TCP e UDP foram transmitidos diversas vezes para avaliar a característica de limitação de banda da proposta. Além disso, avaliou-se QoE sobre o escalonador de pacotes SFQ. Os resultados apresentaram-se satisfatórios em todos os cenários.

CAPÍTULO 6

Conclusões

A separação do plano de controle para fora da "caixa preta" dos dispositivos e por permitir a programação dinâmica de regras de encaminhamento nos *switches* habilitados com OpenFlow, foi um dos fatores contribuintes para o sucesso da arquitetura SDN/OpenFlow.

Entretanto, devido as deficiências de QoS presentes atualmente nos domínios OpenFlow, muitas das vezes, os fluxos proseguem com o processo de melhor esforço. Quando se é exigido um níveis de qualidade de serviço, os mesmos são configurados manualmente através de ferramentas Linux ou *scripts* preparados pelo administrador. Por conta da falta de um controle que automatize tais tarefas, as configurações estão sempre sujeitas a possíveis ocorrência de maiores erros de configuração. Mesmo que os *scripts* possam vir a oferecer um certo nível de automatização, a manutenção dos mesmos estariam sujeitos a ocorrência de erros por parte do operador e por consequência, sujeitos a erros de configuração de QoS.

O trabalho apresentou o QoSFlow, uma proposta de controle automatizado de QoS em rede OpenFlow, onde foram agregados novas funcionalidades para manipular QoS através das aplicações executadas no controlador. A proposta do trabalho consistiu em oferecer suporte ao controle de escalonadores de pacotes FIFO, SFQ, RED e HTB do *kernel* do Linux e dessa maneira, oferecer para o domínio OpenFlow os diversos tipos de escalonadores de pacotes para transmissão.

Conjunto de API de QoS foram aderidos no controlador para o desenvolvimento das aplicações de QoS da proposta. Além dessas interfaces de programação, foi criado uma versão do Mininet para dar suporte ao QoSFlow. Nesse sentido, os pesquisadores poderão realizar experimentos com QoS para validar suas propostas de trabalho.

Inúmeros experimentos foram executados para validar a proposta do trabalho. O QoSFlow apresentou-se como um candidato em potencial para lidar com os problemas de QoS encontrados atualmente no OpenFlow, visto que as configurações de largura de banda e escalonadores de pacotes funcionaram de forma eficiente.

Os resultados dos experimentos com vídeo, inclusive, apresentaram que a tradicional forma de enfileirar (FIFO) pode não ser o suficiente para manter uma boa qualidade de serviço da rede e a qualidade de experiência percebida pelo usuário final. Portanto, é essencial que o OpenFlow esteja preparado para melhor absorver as necessidades do usuário. Nesse quesito, a implementação do *datapath* QoSFlow foi desenvolvido de forma modular e de modo a facilitar a adição de uma nova disciplina de enfileiramento para ser suportado pelo OpenFlow.

Como trabalhos futuros, pretende-se realizar testes com escalonador de pacotes RED e aprimorar mais a proposta QoSFlow. Atualmente, somente as políticas de encaminhamento e alguns comandos de QoS são possíveis de serem executados na arquitetura. Portanto, como um dos vários trabalhos futuros, pretende-se realizar um estudo dirigido sobre as formas de políticas de QoS que existem nas arquiteturas tradicionais e como essas políticas poderão vir a ser proveitosa na arquitetura do QoSFlow. Assim como, um repositório de políticas de QoS deverá ser acrescentado na arquitetura.

Outro ponto a ser trabalhado, é no porte das mensagens de QoS para a especificação padrão 1.3 do OpenFlow. Além disso, uma plataforma Web de administração da rede será desenvolvida para a arquitetura do QoSFlow.

Referências

[1] "Openflow specification 1.0," 2012, "[Acessado Janeiro-2012]". [Online]. Available: http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf

[2] O. N. Foundation, "Openflow specification," 2012. [Online]. Available: https://www.opennetworking.org/

[3] W. Kuokkwee, M. Othman, S. Shamala, and A. Ariffin, "Enhanced dynamic bandwidth allocation proportional to queue length with threshold value for vbr traffic." *Int. Arab J. Inf. Technol.*, vol. 4, no. 2, pp. 117–124, 2007.

[4] K. Ji and W. Kim, "Optimal bandwidth allocation and qos-adaptive control co-design for networked control systems," *International Journal of Control, Automation, and Systems*, vol. 6, no. 4, pp. 596–606, 2008.

[5] G. Feng, H. Wang, B. Li, and Z. He, "Dynamic self-configuration of user qos for next generation network," in *Network and Parallel Computing, 2009. NPC'09. Sixth IFIP International Conference on.* IEEE, 2009, pp. 80–85.

[6] G. Gardasevic and Z. Bojkovic, "Architectural framework for application level qos adaptation in next generation networks," in *Networking and Services, 2009. ICNS'09. Fifth International Conference on.* IEEE, 2009, pp. 409–414.

[7] D. Clark, R. Braden, and S. Shenker, "Integrated services in the internet architecture: an overview," 1994.

[8] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," 1998.

[9] S. Civanlar, M. Parlakisik, A. Tekalp, B. Gorkemli, B. Kaytaz, and E. Onem, "A qos-enabled openflow environment for scalable video streaming," in *GLOBECOM Workshops (GC Wkshps), 2010 IEEE.* IEEE, 2010, pp. 351–356.

[10] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S. Lee, and P. Yalagandula, "Automated and scalable qos control for network convergence," *Proc. INM/WREN*, 2010.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[12] S. Paul, J. Pan, and R. Jain, "Architectures for the future networks and the next generation internet: A survey," *Computer Communications*, vol. 34, no. 1, pp. 2–42, 2011.

[13] "Exploring networks of the future," http://www.geni.net, 2012, "[Online; accessado Janeiro-2012]". [Online]. Available: http://www.geni.net

[14] "Protogeni," http://www.protogeni.net/, 2012, "[Online; accessado Janeiro-2012]". [Online]. Available: http://www.protogeni.net/

[15] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying nox to the datacenter," *Proc. HotNets (October 2009)*, 2009.

[16] "Beacon home," https://openflow.stanford.edu/display/Beacon/Home, 2012, [Online; accessado Janeiro-2012]. [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home

[17] "Floodlight controller," http://floodlight.openflowhub.org/, 2012, "[Online; accessado Janeiro-2012]". [Online]. Available: http://floodlight.openflowhub.org/

[18] "Pox controller," http://www.noxrepo.org/, 2012, "[Online; accessado Janeiro-2012]". [Online]. Available: http://www.noxrepo.org/

[19] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 19–24.

[20] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks," *OSDI, Oct*, 2010.

[21] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. USENIX Association, 2010, pp. 3–3.

[22] J. Kempf, S. Whyte, J. Ellithorpe, P. Kazemian, M. Haitjema, N. Beheshti, S. Stuart, and H. Green, "Openflow mpls and the open source label switched router," in *Proceedings of the 23rd International Teletraffic Congress*. ITCP, 2011, pp. 8–14.

[23] D. Eastlake 3rd., "Iana considerations and ietf protocol usage for ieee 802 parameters," RFC 5342 (Best Current Practice), Internet Engineering Task Force, September 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5342.txt

[24] *RFC 791 Internet Protocol - DARPA Inernet Programm, Protocol Specification*, Internet Engineering Task Force, September 1981. [Online]. Available: http://tools.ietf.org/html/rfc791

[25] J. Postel, "RFC 790: Assigned numbers," Sep. 1981, obsoleted by RFC0820 [**?**]. Obsoletes RFC0776 [**?**]. Status: HISTORIC. [Online]. Available: ftp://ftp.internic.net/rfc/rfc776.txt, ftp://ftp.internic.net/rfc/rfc790.txt, ftp://ftp.internic.net/rfc/rfc820.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc776.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc790.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc820.txt

[26] "Openflow specification 1.3," 2012, "[Acessado Dezembro-2012]". [Online]. Available: https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf

[27] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific.* IEEE, 2012, pp. 1–8.

[28] D. M. F. Mattos, O. C. M. B. Duarte, and R. de Janeiro-RJ-Brasil, "Qflow: Um sistema com garantia de isolamento e oferta de qualidade de serviço para redes virtualizadas," 2012.

[29] D. M. F. Mattos, N. C. Fernandes, O. C. M. B. Duarte, and R. de Janeiro-RJ-Brasil, "Xenflow: Um sistema de processamento de fluxos robusto e eficiente para migraç ao em redes virtuais," in *Simpósio Brasileiro de Redes de Computadores (SBRC)*, 2011.

[30] B. Sonkoly, A. Gulyas, F. Nemeth, J. Czentye, K. Kurucz, B. Novak, and G. Vaszkun, "On qos support to ofelia and openflow," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, oct. 2012, pp. 109 –113.

[31] M. Shin, K. Nam, and H. Kim, "Software-defined networking (sdn): A reference architecture and open apis," in *ICT Convergence (ICTC), 2012 International Conference on.* IEEE, 2012, pp. 360–361.

[32] B. Hubert, T. Graf, G. Maxwell, R. Van Mook, M. Van Oosterhout, P. B. Schroeder, J. Spaans, and P. Larroy, *Linux Advanced Routing & Traffic Control HOWTO*, Online publication, Linux Advanced Routing & Traffic Control, http://lartc.org/, Apr. 2004. [Online]. Available: http://lartc.org/lartc.pdf

[33] P. McKenney, "Stochastic fairness queueing," in *INFOCOM '90, Ninth Annual Joint Conference of the IEEE Computer and Communication Societies. The Multiple Facets of Integration. Proceedings, IEEE*, jun 1990, pp. 733 –740 vol.2.

[34] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *Networking, IEEE/ACM Transactions on*, vol. 1, no. 4, pp. 397 –413, aug 1993.

[35] V. Floyd, S. Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Trans. Netw.*, vol. 3, no. 4, pp. 365–386, 1995.

[36] M. Devera and D. Cohen, "Htb linux queuing discipline manual," *Also available from: http://luxik. cdi. cz/˜ devik/qos/htb/manual/userg. html*, 2002.

[37] Openwrt. [Online]. Available: http://www.openwrt.org

[38] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466

[39] H. J. Kim and S. G. Choi, "A study on a qos/qoe correlation model for qoe evaluation on iptv service," in *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*, vol. 2, feb. 2010, pp. 1377 –1382.

APÊNDICE A – Esquema em JSON

A.1 Disciplina de Fila ou Escalonador de Pacotes

Esquema em JSON definido pelo administrador e utilizado para adicionar, remover ou trocar um escalonador de pacotes. Essa mensagem é recebida e interpretada pelo componente `QoSFlow Manager`.

```
{
   "command" : <string>
   "dp" : <integer>
   "qdisc" : <string>
   "out_port" : <string>
   "cid" : <integer>
   key : <object>
}
```

A linha `key : <object>` deve ter um dos seguintes parâmetros: `"pfifo"`, `"bfifo"`, `"sfq"` ou `"red"`. Logo abaixo, seguem a estrutura dos parâmetros mencionados.

```
"pfifo" : {
    "limit" : <integer>
}

"bfifo" : {
    "limit" : <integer>
}

"sfq" : {
    "perturb" : <integer>,
    "quantum" : <integer>
}

"red" : {
    "treshold" : <integer>,
    "limit" : <integer>
}
```

A.2 Classe ou Fila

Esquema em JSON definido pelo administrador e utilizado para adicionar, remover or trocar um valor de largura de banda de uma fila. Essa mensagem é recebida e interpretada pelo componente `QoSFlow Manager`.

```
{
   "command" : <string>
   "dp" : <integer>
   "class" : <string>
   "out_port" : <string>
   "cid" : <integer>
   "htb" : {
       "rate" : <integer>,
       "ceil" : <integer>,
       "prio" : <integer>
   }
}
```

A.3 Política de Encaminhamento

Esquema em JSON definido pelo administrador e utilizado para definir regras de encaminhamento para os fluxos. Essa mensagem é recebida e interpretada pelo componente `QoSFlow Manager`.

```
{
   "command" : <string>
   "dp" : <integer>
   "controller" : <string>
   "idle" : <integer>
   "hard" : <integer>
   "table_entry" : <string>
   "action" : {"cid" : <integer>, "out_port" : <string>}
   "condition" : <object>
}
```

A.4 Período de Monitoramento de QoS

Esquema em JSON definido pelo administrador e utilizado para comandar o componente `QoSFlow Monitor` a consultar estatísticas de QoS em intervalos de tempo específico.

```
{
   "command" : <string>
   "dp" : <integer>,
   "interval" : <integer>
}
```

A.5 Consulta de QoS

Esquema em JSON definido pelo administrador e utilizado para coletar informações de QoS das portas e filas de comutador. Essa mensagem é recebida e interpretada pelo componente `QoSFlow Monitor`.

```
{
   "command" : <string>
   "dp" : <integer>
   "out_port" : <string>
   "cid" : <integer>
}
```

O parâmetro `"cid"` deve ser omitido caso a estatística de fila não seja de interesse, ou seja, somente informações de QoS relativos às portas.

A.6 Estatística de Porta

Esquema em JSON definido pelo componente `QoSFlow Monitor` para enviar as estatísticas das portas para a interface do administrador da rede.

```
[
   {
      "dp" : <integer>
      "devices" :
      [
          {
             "port_name" : <string>
             "rx_packets" : <integer>
             "tx_packets" : <integer>
             "rx_bytes" : <integer>
             "tx_bytes" : <integer>
             "rx_dropped" : <integer>
             "tx_dropped" : <integer>
          },
```

```
          ....(se existirem mais portas)
      ]
   },
   ....(se existirem mais datapaths)
]
```

A.7 Estatística de Fila

Esquema em JSON definido pelo componente `QoSFlow Monitor` para enviar as estatísticas das filas para a interface do administrador da rede.

```
[
   {
       "dp" : <integer>
       "devices" :
       [
          {
             "port_name" : <string>
             "queue_id" : <integer>
             "tx_packets" : <integer>
             "tx_bytes" : <integer>
             "tx_errors" : <integer>
          },
          ....(se existirem mais portas)
       ]
   },
   ....(se existirem mais datapaths)
]
```

A.8 Notificação de Erro

Essa mensagem é definida pelo componente `Gerente` ou `Monitor QoSFlow` em caso de ocorrência de erros ocorridos detectados pelo plano de controle.

```
{
"error" : <string>
}
```

APÊNDICE B – Classes do QoSFlow no Plano de Controle

B.1 Classe QoSFlow

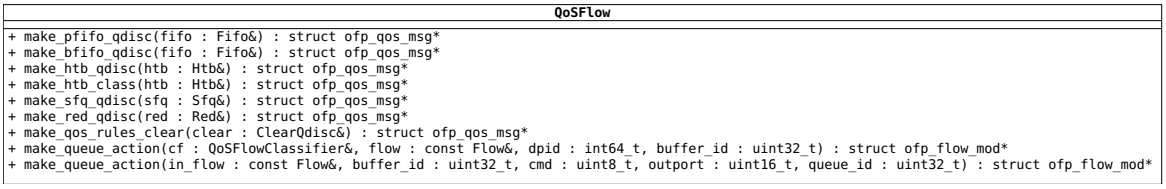


Figura 18: Classe utilizada para contruir mensagens de QoS dos escalonadores de pacotes e políticas de encaminhamento.

B.2 Classe das APIs de QoS

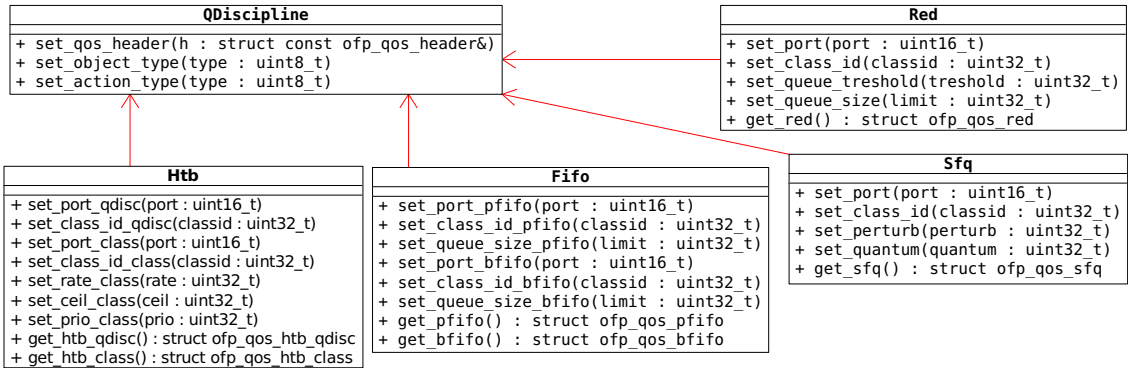


Figura 19: Digrama de classes das APIs de QoS para instanciar os escalonadores de pacotes HTB, PFIFO, BFIFO, SFQ e RED através de objetos.

B.3 Classe Gerente QoSFlow

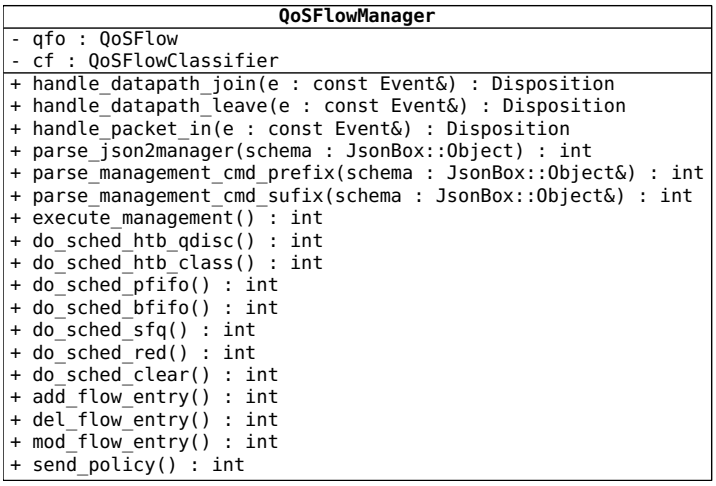


Figura 20: Classe do componente Gerente QoSFlow do controlador.

B.4 Classe Monitor QoSFlow

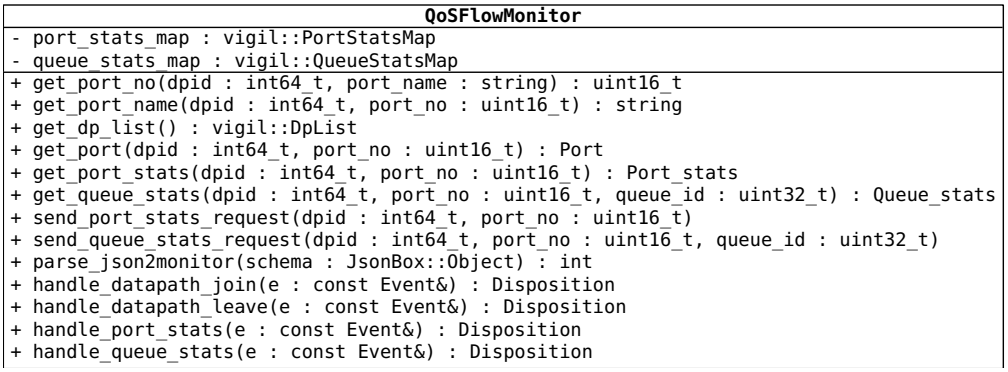


Figura 21: Classe do componente Monitor QoSFlow do controlador.

APÊNDICE C – Mensagens de QoS

Estruturas de dados relacionados a QoS. Essas estruturas foram extendidas ao protocolo OpenFlow e são utilizados para manipular QoS na rede.

```
/**
 * The * MAIN * data structure to "agregate" queuing discipline
 * into the 'msg' field. The sender must fill this message
 * with one queueing discipline type before to send.
 */
struct ofp_qos_msg {
        struct ofp_header hdr;    /* OpenFlow protocol message header */
        uint8_t sched_type;       /* If is HTB, SFQ, PFIFO, BFIFO, RED */
        uint8_t reserved[3];
        uint8_t body[0];          /* The 'body' can be one of the schedulers */
};

/* Header to encapsulate all qos messages (controller --> datapath) */
struct ofp_qos_header {
        uint8_t object_type;
        uint8_t action_type;
};
OFP_ASSERT(sizeof(struct ofp_qos_header) == 2);


/* Classless queueing discipline */

/**
 * Simplest usable qdisc, pure First In, First Out behaviour.
 */
struct ofp_qos_pfifo {
        struct ofp_qos_header qos_hdr;
        uint16_t port_no;
        uint32_t class_id;

        uint32_t limit;
};
OFP_ASSERT(sizeof(struct ofp_qos_pfifo) == 12);
```

```
/**
 * Simplest usable qdisc, pure First In, First Out behaviour.
 */
struct ofp_qos_bfifo {
        struct ofp_qos_header qos_hdr;
        uint16_t port_no;
        uint32_t class_id;

        uint32_t limit;
};
OFP_ASSERT(sizeof(struct ofp_qos_bfifo) == 12);

/**
 * Stochastic Fairness Queueing reorders queued traffic.
 * Queues are dequeued in turn (round-robin).
 */
struct ofp_qos_sfq {
        struct ofp_qos_header qos_hdr;
        uint16_t port_no;
        uint32_t class_id;

        uint32_t perturb;
        uint32_t quantum;
};
OFP_ASSERT(sizeof(struct ofp_qos_sfq) == 16);

/**
 * Random Early Detection simulates physical congestion by randomly
 * dropping  packets when nearing configured bandwidth allocation.
 */
struct ofp_qos_red {
        struct ofp_qos_header qos_hdr;
        uint16_t port_no;
        uint32_t class_id;

        uint32_t treshold;
        uint32_t limit;
};
OFP_ASSERT(sizeof(struct ofp_qos_red) == 16);
```

```
/**
 * The Hierarchy Token Bucket implements a rich linksharing hierarchy
 * of classes with an emphasis on conforming to existing practices.
 * HTB facilitates guaranteeing bandwidth to classes, while also
 * allowing specification of upper limits to inter-class sharing.
 * It contains shaping elements, based on TBF and can prioritize classes.
 *
 * To see how HTB works, see: http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm
 */
struct ofp_qos_htb_qdisc {
        struct ofp_qos_header qos_hdr;
        uint16_t port_no;
        uint32_t class_id;
};
OFP_ASSERT(sizeof(struct ofp_qos_htb_qdisc) == 8);

struct ofp_qos_htb_class {
        struct ofp_qos_header qos_hdr;
        uint16_t port_no;
        uint32_t class_id;

        uint32_t rate;
        uint32_t ceil;
        uint32_t prio;
};
OFP_ASSERT(sizeof(struct ofp_qos_htb_class) == 20);
```