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

VERÔNICA SOUZA LEAL SALIBA GOMES

ISOLAMENTO DE RECURSOS EM REDES DEFINIDAS POR *SOFTWARE* VIRTUALIZADAS BASEADAS EM *OPENFLOW*

BELÉM-PA

Abril / 2013

VERÔNICA SOUZA LEAL SALIBA GOMES

ISOLAMENTO DE RECURSOS EM REDES DEFINIDAS POR *SOFTWARE* VIRTUALIZADAS BASEADAS EM *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: Dr. Antônio Jorge Gomes Abelém

BELÉM-PA

Abril / 2013

VERÔNICA SOUZA LEAL SALIBA GOMES

ISOLAMENTO DE RECURSOS EM REDES DEFINIDAS POR *SOFTWARE* VIRTUALIZADAS BASEADAS EM *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. Raimundo Viégas Junior
Universidade Federal do Pará

Prof. Dr. Mauro Margalho Coutinho
Universidade da Amazônia

Dedico o sucesso deste trabalho aos meus familiares, amigos e às pessoas que proporcionaram apoio incondicional e incansável no decorrer desta etapa de minha vida.

Agradecimentos

Agradeço primeiramente à Deus pela oportunidade e presença constante em minha vida.

Dedico meus sinceros agradecimentos aos meus pais, esposo e demais familiares, pela dedicação e incentivo recebido durante estes anos.

Ao professor Doutor Antônio Jorge Gomes Abelém, pela orientação, compreensão e contribuição na realização deste trabalho.

À equipe do laboratório GERCOM, em especial aos colegas Fernando Farias, Airton Ishimori e Hugo Toda pela ajuda em diversos momentos.

A todos que auxiliaram direta e indiretamente na concretização desta etapa.

Resumo

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

Isolamento de recursos em Redes Definidas por *Software* virtualizadas baseadas em *Openflow*

Orientador: Dr. Antônio Jorge Gomes Abelém
Palavras-chave: Redes Definidas por *Software*; *OpenFlow*; *FlowVisor*; virtualização; QoS

As redes definidas por software vêm ganhando considerável projeção no âmbito das comunidades de pesquisa para evolução da Internet, caracterizando-se pela definição de uma interface que permite a programação centralizada dos elementos da rede. No contexto de aplicação deste paradigma, o *OpenFlow* tem se destacado como uma importante solução, incluindo um padrão aberto que possibilita a utilização de equipamentos em redes comerciais e de pesquisa para experimentação de novas propostas, podendo aplicar uma camada de virtualização para execução de múltiplos experimentos em paralelo sobre uma única infraestrutura através da ferramenta *FlowVisor*. A solução habilita o compartilhamento de recursos de rede através da definição de diferentes fatias de redes virtuais. Apesar de seu potencial, tal iniciativa ainda enfrenta alguns desafios e limitações em sua arquitetura, dentre as quais a mais importante é a necessidade de novos métodos para garantir o isolamento real de recursos entre as diferentes fatias. Este trabalho apresenta uma proposta para o aperfeiçoamento das funções de isolamento do *FlowVisor*. Novos módulos foram desenvolvidos, permitindo a aplicação de mecanismos de controle de tráfego e gerenciamento de recursos entre as redes virtuais. Dentre as funcionalidades aplicadas inclui-se uma interface para programação de aspectos de QoS nas portas de dispositivos de rede, permitindo o controle da alocação de recursos diretamente pelo *FlowVisor*. Os resultados de experimentos realizados para validação da proposta demonstram que a solução é capaz de isolar as fatias, mitigando as interferências entre elas.

Abstract

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

Title

Advisor: Dr. Antônio Jorge Gomes Abelém
Key words: *Software Defined Networkin; OpenFlow; FlowVisor; Virtualization; QoS.*

Software-Defined Networking have been gaining considerable projection on the research communities in the Internet's evolution, characterized by defining a interface for centralized programming the network elements. In the context of this paradigm application, the OpenFlow has emerged as an important solution, including an open standard that allows the use of equipment in commercial and research networks for testing new proposals. A virtualization layer can be applied to perform multiple experiments in parallel on a single infrastructure through the tool called FlowVisor. The solution enables sharing network resources by defining different slices of virtual networks. Despite its potential, this initiative still faces some challenges and limitations in its architecture, among them the most important is the need for new methods to ensure real resources isolation between different slices. This work presents a proposal to improve FlowVisor isolation functions. New modules were developed, allowing the application of mechanisms for traffic control and resources management between virtual networks. Among the features implemented, an interface for programming QoS aspects on network devices ports were included, allowing resource allocation control directly by FlowVisor. The experiments performed to validate the proposal demonstrate that the solution is capable of isolating the slices, mitigating interference between them.

Sumário

Lista de Abreviaturas

ARPANet	Advanced Research Projects Agency Network
IF	Internet do Futuro
GENI	Global Environment for Network Innovations
FIRE	Future Internet Research and Experimentation
FIBRE	Future Internet Experimentation Between Brazil and Europe
SDN	Software Defined Networking
API	Application Programming Interface
QoS	Quality of Service
MAC	Media Access Control
IP	Internet Protocol
TCP	Transmission Control Protocol
VLAN	Virtual Local Area Network
SSL	Secure Socket Layer
NOX	Network Operating System for OpenFlow
PCP	Priority Code Point
VNode	Virtualization Node
VNP	Virtualization Node Project
FIFO	First In, First Out
XML	Extensible Markup Language
RPC	Remote Procedure Call
HTB	Hierarchical Token Buckets
PFIFO	Packet First In, First Out
BFIFO	Byte First In, First Out
SFQ	Stochastic Fairness Queueing
RED	Random Early Detection
UDP	User Datagram Protocol

DDoS	Distributed Denial-of-Service
Mbps	Megabit por segundo

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

Este capítulo introdutório inclui uma visão geral sobre os objetivos do trabalho, apresentando uma contextualização ao tema abordado e as principais motivações e desafios considerados para o desenvolvimento da proposta.

1.1 Visão geral

Ao longo dos anos a Internet tem evoluído de forma considerável. Ao final de 2011, mais de um terço da população mundial estava on-line, ou seja, 2,3 bilhões de pessoas, podendo a Internet alcançar até 60% de usuários em todo o mundo em 2015 [ITU-D 2012]. Apesar do amplo sucesso, a arquitetura da Internet vem apresentando algumas limitações para atender às novas necessidades exigidas pelas aplicações, como desempenho, priorização de serviços, suporte à mobilidade, confiabilidade e segurança.

A Internet evoluiu a partir da ARPANet (*Advanced Research Projects Agency Network*), uma rede de computadores à base de comutação de pacotes concebida, originalmente, para atender a demanda do Departamento de Defesa dos Estados Unidos da América em criar uma rede de controle e comando tolerante a falhas. A ARPANet inicial era uma rede isolada e fechada, mas que logo foi interligada a outras redes, fundamentalmente de pesquisas [Kurose and Ross 2010]. Uma breve análise do surgimento da Internet permite compreender que a rede foi inicialmente idealizada para atender e interligar um número limitado de nós confiáveis, cujas demandas incluíam aplicações simples, como troca de mensagens e transferência de arquivos. Este cenário culminou no desenvolvimento de uma arquitetura essencialmente simples, porém disponível para novas aplicações, o que, por um lado, permitiu a rápida evolução e extensão da rede.

Por outro lado, considera-se a simplicidade do modelo como característica prepon-

derante para ocorrência do problema designado, por alguns autores, como engessamento ou "ossificação"da Internet (tradução do inglês *ossified*) [Paul et al. 2011], [Correia et al. 2011], [Moreira et al. 2009]. O incremento sucessivo de soluções e protocolos na arquitetura original da rede, com o objetivo de atender às novas exigências dos usuários e aplicações, ocorreu sem grandes mudanças em sua estrutura básica. A aplicação de contínuas adaptações, cada vez mais substanciais, acabou determinando a ocorrência de novos problemas. Com isto, é possível verificar as limitações do atual projeto em suportar as crescentes demandas.

Neste cenário, pesquisas são realizadas com o intuito de desenvolver arquiteturas alternativas para a chamada Internet do Futuro (IF) [Correia et al. 2011], [Pan et al. 2011], buscando a formulação de soluções que proporcionem um crescimento ordenado e eficiente da rede. Conforme ressalta [Paul et al. 2011] o redesenho deve ocorrer visando atender às necessidades atuais, enquanto, ao mesmo tempo, assegura a flexibilidade suficiente para incorporar de forma adequada as demandas futuras.

Entretanto, considerando os possíveis impactos de novas propostas no ambiente atual, sua aplicação efetiva está sujeita a realização de experimentos para comprovação e validação de suas funcionalidades, demandando, para isto, a utilização de infraestruturas de teste o mais próximas da real. Este fato é um dos principais desafios enfrentados pelos pesquisadores para incorporação de suas novas ideias. Os atuais produtos e equipamentos de rede possuem sistemas fechados com soluções proprietárias e, ainda, existe a necessidade dos dados de experimentação serem mantidos isolados do tráfego de produção.

Admitindo-se tais limitações, portanto, iniciativas para a definição de infraestruturas e métodos de experimentação tornam-se cada vez mais evidentes. Dentre os projetos desenvolvidos em diversos países destacam-se: o GENI (*Global Environment for Network Innovations*) [Geni 2013], nos Estados Unidos; o projeto AKARI [Akari 2013], no Japão; o programa FIRE (*Future Internet Research and Experimentation*) [Fire 2013], na Europa; e o projeto FIBRE (*Future Internet Experimentation Between Brazil and Europe*) [Fibre 2013], incluindo apoio para experimentação conjunta entre pesquisadores na Europa e no Brasil. Impulsionados pela capacidade de abstração de elementos da rede e pela garantia de um controle remoto sobre o fluxo de dados, as redes programáveis (SDN (*Software Defined Networking*) [Greene 2009]) e virtualizadas vêm sendo adotadas por esses projetos [Farias et al. 2012].

A abordagem aplicada pelas redes programáveis ou definidas por *software* está pautada na dissociação entre as funções de encaminhamento de pacotes, desempenhadas pelo plano de dados, que continuam a residir nos dispositivos de rede, e as lógicas de decisão aplicadas pelo plano de controle (ações que são empregadas sobre os pacotes que transcendem o plano de dados), que são movidas para um elemento externo. A possibilidade de desenvolver novas aplicações para atuarem no plano de controle garante aos pesquisadores a programabilidade e domínio sobre o comportamento da rede.

Desta forma, considerando os aspectos supracitados, torna-se essencial a aplicação de métodos para garantir a comunicação entre o plano de controle e o plano de da-

dos. Uma solução que vem ganhando destaque, neste contexto, é o *framework OpenFlow* [McKeown et al. 2008]. Fruto do programa de pesquisa em Internet do Futuro de Stanford, o *OpenFlow* é uma abordagem que surgiu com o propósito de oferecer aos pesquisadores a possibilidade de testar seus protocolos experimentais diretamente sobre as redes de produção (como redes de um campus, redes metropolitanas ou em um *backbone* de rede de ensino e pesquisa), sem, no entanto interferir no tráfego original da rede. O *OpenFlow* é considerado, por alguns autores, como a primeira interface padrão projetada especificamente para SDN, estabelecendo-se como impulsor do paradigma [Greene 2009], [ONF 2012].

Dentre os principais elementos apresentados pelo *OpenFlow* inclui-se: a definição de um protocolo aberto de controle, para manipulação da tabela de encaminhamento dos dispositivos; e o fornecimento de uma API (Interface de Programação de Aplicações), simples e extensível, implementada em uma entidade remota, denominada controlador *OpenFlow*, que possibilita a programação do comportamento dos fluxos de pacotes.

Desde sua definição, diversas pesquisas, nas mais variadas áreas como virtualização, protocolos ou arquiteturas, são realizadas com o objetivo de contribuir para a evolução e criação de novos elementos que possibilitem a extensão das funcionalidades das redes *OpenFlow*. Neste sentido, o *FlowVisor* [Sherwood et al. 2009] tem se destacado como um destes elementos, podendo ser integrado à rede com o intuito de criar uma camada de virtualização para ambientes *OpenFlow* e permitir o compartilhamento de recursos sobre um mesmo substrato físico.

O *FlowVisor* é apresentado como uma abordagem para permitir que o mesmo *hardware* do plano de dados possa ser compartilhado entre múltiplas redes virtuais, cada uma definindo lógicas de encaminhamento distintas. Sendo assim, diferentes redes podem utilizar os recursos físicos de forma independente e aplicando seu próprio controlador.

A ferramenta atua como um *proxy* transparente entre os dispositivos de rede e múltiplos controladores *OpenFlow*, criando uma camada de virtualização sobre a camada de abstração de hardware e garantindo o compartilhamento de recursos disponíveis através da criação de várias fatias (do inglês *slices*), cada uma delas definindo uma rede virtual distinta. Assim, além da separação entre tráfego de pesquisa e produção, é possível a execução simultânea de diversos experimentos, sem que ocorra interferência entre eles.

Embora o *FlowVisor* seja capaz de lidar com múltiplos controladores, atribuindo uma fatia para cada um deles, atualmente a definição de uma solução que forneça um controle eficiente para alocação de recursos nas redes virtuais, como provisionamento de largura de banda [Takahashi 2012], ainda é um problema pouco explorado. Além da necessidade de mecanismos para suportar o compartilhamento, as configurações que podem ser realizadas nos dispositivos são dependentes de ferramentas externas. Em ambientes virtualizados, considerando a utilização mútua de uma quantidade limitada de recursos, a ausência de mecanismos que garantam o isolamento pode resultar em interferências entre os diferentes ambientes virtuais, de forma que uma única fatia pode causar perturbação em toda infraestrutura.

Considerando tais limitações, este trabalho propõe uma solução para aperfeiçoamento das funções de isolamento do *FlowVisor*. Novos módulos foram desenvolvidos, permitindo a aplicação de mecanismos específicos de controle de tráfego e gerenciamento de recursos entre as redes virtuais. Dentre as funcionalidades aplicadas inclui-se uma interface para programação de aspectos de QoS nas portas dos dispositivos de rede, permitindo o controle da alocação de recursos diretamente pelo *FlowVisor*.

A abordagem de desenvolvimento aplicada inclui a integração do *FlowVisor* com subcomponentes do plano de dados do arcabouço de *software* denominado QoSFlow [Ishimori et al. 2012b]. A solução QoSFlow apresenta funcionalidades para administração dinâmica de recursos de QoS (*Quality of Service*), como largura de banda, tamanho da fila, ou atraso, diretamente nas portas de dispositivos habilitados com *OpenFlow*. Desta forma, o desenvolvimento de um módulo de gerenciamento e controle de tráfego na ferramenta *FlowVisor* e a inclusão de funcionalidades para classificação de fluxos, através da atribuição de uma fila a cada *slice*, garante a separação dos recursos compartilhados entre as redes virtuais e a atenuação de possíveis interferências entre os ambientes.

Para validação da solução, foram aplicados cenários de testes distintos. Foram definidos ambientes onde fluxos em diferentes fatias competiam por largura de banda em um link compartilhado. Desta forma, foi possível a comparação do comportamento da rede com e sem a aplicação dos mecanismos definidos para o isolamento de recursos. Demonstrou-se que, com a implementação de recursos para limitação do uso da largura de banda disponível, a solução é capaz de isolar as redes virtuais, resultando na mitigação de interferências entre os fluxos concorrentes.

1.2 Motivação e desafios

A escala de esforços em pesquisas voltadas para o desenvolvimento de uma próxima geração da Internet, ao mesmo tempo em que comprova sua importância, evidencia a necessidade de seu aperfeiçoamento para sustentar as exigências futuras. Apesar do crescente número de pesquisas voltadas para a definição de novos protocolos e arquiteturas para Internet do Futuro, muitas ideias são refutadas devido à incapacidade de validação de seu comportamento e desempenho em condições reais. Conforme destaca [Moreira et al. 2009] a realização de experimentos em ambientes reais em larga escala é uma tarefa complexa. Além da ausência de flexibilidade para o controle do funcionamento interno dos equipamentos de rede, existe uma incerteza quanto aos possíveis impactos da execução de experimentos em ambientes de produção.

Desta forma, soluções para a definição e aperfeiçoamento de alternativas para pesquisa experimental, tornam-se cada vez mais relevantes. Dentre elas, as Redes Definidas por *Software* destacam-se como abordagem para habilitar o controle e programabilidade da rede, possibilitando a execução de experimentos em ambientes de produção, sem, no entanto prejudicar a infraestrutura atual.

Neste contexto, o *OpenFlow* destaca-se como iniciativa que vem se projetando de

maneira significativa nas comunidades de pesquisa em Internet do Futuro. A ideia básica de operação da solução consiste na separação entre o plano de dados e o plano de controle dos dispositivos de rede. Conforme definido por [McKeown et al. 2008], uma interface padrão ou protocolo aberto é capaz de realizar a comunicação entre os dispositivos e um processo de controle externo, responsável por controlar as decisões de encaminhamento das tabelas de fluxos dos equipamentos, determinando remotamente as ações de processamento associadas a cada fluxo. Sendo assim, ao habilitar o *OpenFlow* em seus produtos, os fabricantes fornecem aos pesquisadores condições para execução de experimentos em ambientes reais, sem a necessidade de expor o funcionamento interno de seus dispositivos.

No entanto, apenas a aplicação dos componentes básicos definidos originalmente para as redes *OpenFlow* não permite que múltiplos experimentos sejam conduzidos em um único ambiente. Para isso, foi proposta uma camada de virtualização para o *OpenFlow*, habilitada pela ferramenta denominada *FlowVisor*. Com o *FlowVisor*, múltiplas redes lógicas, identificadas como fatias, podem compartilhar a mesma topologia de rede e a mesma infraestrutura física. O *FlowVisor* utiliza o *OpenFlow* como uma camada de abstração de hardware situando-se logicamente entre o plano de controle e o de encaminhamento [Sherwood et al. 2009].

O cenário exposto, portanto, vem sendo considerado como um importante instrumento para as atividades de validação de novas propostas. Porém, ainda estão em aberto algumas questões para definição de uma solução completa de virtualização. O *FlowVisor* ainda apresenta algumas limitações em sua arquitetura, dentre as quais a mais importante é a necessidade de novos métodos para garantir o isolamento real de recursos entre as diferentes fatias de rede. Além da necessidade de mecanismos para suportar o compartilhamento, as configurações que podem ser realizadas nos dispositivos são dependentes de ferramentas externas, como as realizadas estaticamente por linha de comando ou por outro protocolo de configuração, dedicado para este fim.

Trabalhos relacionados à virtualização, como [Kanada et al. 2012], [Sherwood et al. 2009] ou [Sherwood et al. 2010a], demonstram que a correta separação ou isolamento de recursos entre as redes virtuais definidas, incluindo: topologia, largura de banda, capacidade de processamento ou tabelas de encaminhamento, é um dos requisitos essenciais para alcançar uma completa virtualização. Especificamente em relação à largura de banda, por exemplo, destaca-se que cada fatia deve possuir sua própria fração, sendo que, a ausência de mecanismos que garantam o isolamento pode resultar em interferências entre os diferentes ambientes, de forma que uma única rede virtual pode comprometer o rendimento de toda a infraestrutura.

Desta forma, este trabalho apresenta uma solução para o aperfeiçoamento do *FlowVisor*, buscando a separação efetiva de recursos compartilhados entre as fatias de rede. Novos módulos foram desenvolvidos, agregando funções específicas de controle de tráfego à ferramenta. As funções desenvolvidas permitem a integração com dispositivos de rede *OpenFlow* com QosFlow habilitado. Assim sendo, torna-se possível a programação dinâmica de aspectos de QoS nas portas dos dispositivos, proporcionando a definição de classes e disciplinas de fila e, consequentemente, o controle de tráfego diretamente a partir

do *FlowVisor*.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é definir uma proposta para aperfeiçoamento da ferramenta *FlowVisor*, de modo a agregar funções para garantir o isolamento de recursos em ambientes virtuais de Redes Definidas por Software baseadas em *Openflow*.

1.3.2 Objetivos Específicos

Como desdobramento do objetivo geral, os seguintes objetivos específicos foram definidos:

- **Avaliação do estado da arte:** verificação do estado da arte, viabilizando o mapeamento das lacunas e dos problemas relevantes a serem explorados, além da identificação de pesquisas anteriores relacionadas à área de pesquisa abordada;
- **Avaliação do tratamento de mensagens *OpenFlow* pelo *FlowVisor*:** avaliar o escopo das mensagens *OpenFlow* v1.0 implementadas pelo *FlowVisor*, definindo as necessidades de alterações e adaptações;
- **Avaliação da solução QosFlow:** analisar a arquitetura e as implementações realizadas pelo arcabouço de software QosFlow [Ishimori et al. 2012b], auxiliando no desenvolvimento dos módulos necessários para comunicação com os dispositivos habilitados com QosFlow;
- **Definição da arquitetura:** modelagem da arquitetura de desenvolvimento proposta para solução;
- **Alteração do pacote OpenFlowJ [OpenFlowJ 2012]:** atualização do pacote que implementa as mensagens do protocolo *OpenFlow*, utilizado pelo *FlowVisor*, para que assim, interprete e envie mensagens QoSFlow;
- **API de QoS:** desenvolvimento de uma API de gerenciamento para interagir com dispositivos QoSFlow, bem como possibilitar a atribuição ou modificação de filas às fatias de rede;
- **Conversor de mensagens:** implementação de um conversor para substituir ações nas mensagens *Openflow* recebidas a partir dos controladores, tornando transparente às aplicações a utilização de filas pelo *FlowVisor*;

- **Análise dos resultados:** realização de experimentos e avaliação dos resultados encontrados e, quando possível, compará-los com estudos anteriores para validação e comprovação da viabilidade da proposta.

1.4 Contribuições

Dentre os avanços alcançados com o aperfeiçoamento do *FlowVisor*, através da incorporação de funções específicas de controle de tráfego, incluem-se a atenuação de interferências entre as redes virtuais definidas, possibilitando a programação específica de aspectos de QoS nos dispositivos de rede *OpenFlow* com QosFlow habilidado, diretamente pela ferramenta, sem a necessidade de atuação manual ou a utilização mecanismos externos.

1.5 Organização do texto

O restante do documento está dividido seguindo o ordenamento descrito abaixo:

- Capítulo 2: São apresentados os principais conceitos relacionados ao tema, incluindo uma descrição do funcionamento do *framework Openflow* e da ferramenta *FlowVisor*.
- Capítulo 3: Apresenta os principais trabalhos relacionados aos temas abordados neste trabalho.
- Capítulo 4: Neste capítulo é apresentada a proposta desenvolvida, incluindo os procedimentos utilizados para implementação e a arquitetura da solução.
- Capítulo 5: Inclui os cenários aplicados para validação da proposta e a análise dos resultados encontrados.
- Capítulo 6: Apresenta as considerações finais sobre a pesquisa e as perspectivas de trabalhos futuros.

CAPÍTULO 2

Referencial Teórico

Este capítulo apresenta os principais conceitos relacionados ao tema abordado neste trabalho, descrevendo as motivações para o desenvolvimento da solução e servindo de base para compreensão da proposta apresentada e avaliação dos resultados obtidos.

A Seção 2.1 inclui uma contextualização sobre as redes definidas por *software* e sua aplicabilidade no âmbito das pesquisas voltadas à Internet do Futuro. A Seção 2.2 apresenta o OpenFlow e os detalhes de sua arquitetura. Na Seção 2.3 são evidenciados os detalhes de funcionamento da ferramenta *FlowVisor*. A Seção 2.4 conclui este capítulo.

2.1 Internet do Futuro e Redes Definidas por *Software*

A expansão da Internet, aliada ao crescimento na quantidade de usuários e às novas necessidades apresentadas pelas aplicações, vem ocorrendo através do incremento sucessivo de soluções e protocolos na arquitetura original da rede, porém, sem grandes mudanças em sua estrutura básica. A aplicação de contínuas adaptações, cada vez mais substanciais, acabou determinando a ocorrência de novos problemas. Com isto, é possível verificar a incompatibilidade das novas e crescentes demandas com o atual projeto da Internet. Deste modo, pesquisas são realizadas com o intuito de desenvolver arquiteturas alternativas para a chamada Internet do Futuro, criando-se soluções que proporcionem um crescimento ordenado e eficiente da rede [Farias et al. 2011].

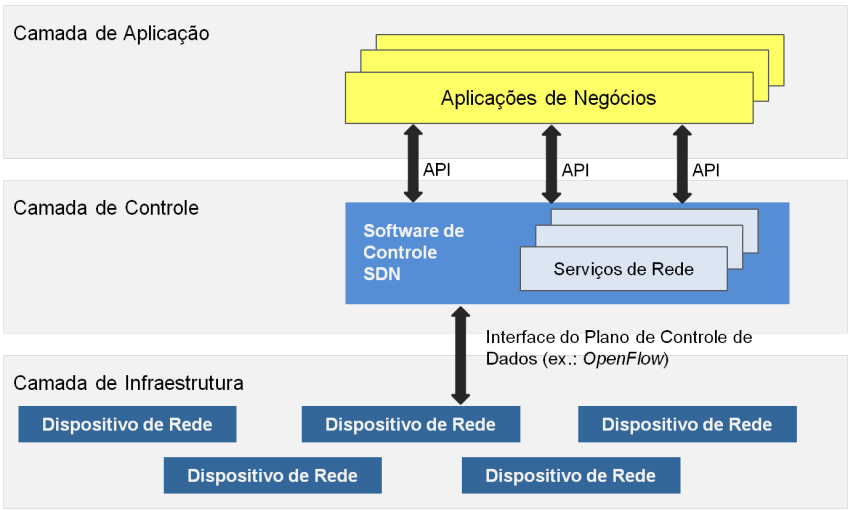
Para [Pan et al. 2011], a arquitetura da Internet do Futuro não inclui uma simples melhoria em um tópico ou tema específico. Deve haver um redesenho de toda a arquitetura, levando-se em consideração, concomitantemente, diversas questões como: segurança, mobilidade, confiabilidade e desempenho. Além disso, as soluções devem ser

suficientemente flexíveis para incorporar de forma adequada mudanças futuras.

Considerando estes aspectos, um importante desafio enfrentado pelos pesquisadores está na realização de experimentos para comprovação da funcionalidade e viabilidade de suas novas propostas. Existe a necessidade de que experimentos sejam realizados em larga escala, considerando condições o mais próximas do ambiente real. Porém, conforme destaca [Moreira et al. 2009] esta é uma tarefa complexa. Além dos projetos de equipamentos e produtos comerciais não serem abertos, existe uma forte relutância dos administradores de rede em permitir que os experimentos utilizem o tráfego de produção.

Neste contexto, as redes programáveis, ou SDN (*Software Defined Networking*) têm emergido consideravelmente, ganhando ampla projeção e espaço nas discussões das comunidades de pesquisa e indústrias da área. Conforme define [Silva et al. 2012], o paradigma SDN provê algumas abstrações, separando o *software* que controla os elementos de rede do *hardware*, fornecendo uma interface aberta para controle e modificação de seu comportamento. A abordagem aplicada está pautada, portanto, na dissociação entre as funções de encaminhamento desempenhadas pelo plano de dados, que continuam a residir nos dispositivos de rede, e as lógicas de decisão aplicadas pelo plano de controle, que são movidas para um elemento externo.

Em [ONF 2012] é apresentada uma visão lógica da arquitetura SDN, conforme adaptada na Figura 1. A inteligência da rede está centralizada em controladores, que mantêm, em uma camada intermediária, uma visão global da rede física. Conforme definido por [Guedes 2012], o controlador age como um sistema operacional para a rede provendo o controle direto dos dispositivos e oferecendo uma interface mais eficiente para os desenvolvedores, isolando os detalhes de implementação de cada componente presente na camada de infraestrutura. Em uma camada superior encontram-se as aplicações, desenvolvidas com o intuito de implementar novas soluções e funcionalidades sobre os dispositivos.



Adaptada de [ONF 2012]

Figura 1: Arquitetura das Redes Definidas por *Software*

Com isto, a rede é apresentada, logicamente, às aplicações como um dispositivo único. A aplicação do paradigma simplifica o controle e operação da rede, uma vez que garante o gerenciamento de toda a infraestrutura a partir de um único ponto lógico, independentemente do fornecedor da plataforma dos dispositivos. Além disso, a possibilidade de desenvolver novas aplicações para atuarem no plano de controle garante aos pesquisadores a programabilidade e domínio sobre o comportamento da rede.

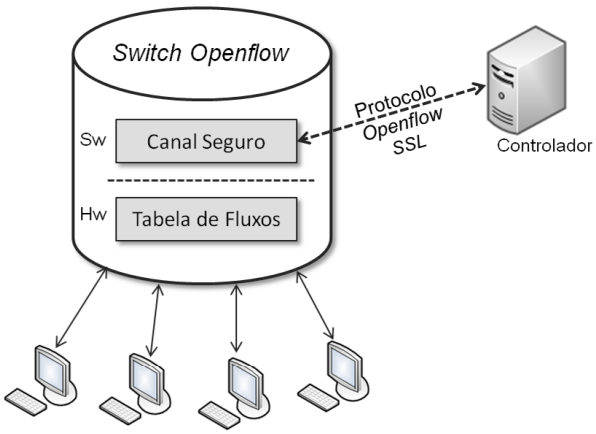
2.2 *OpenFlow*

O *framework OpenFlow* [McKeown et al. 2008] tem se destacado no âmbito das redes definidas por *software*, fornecendo meios factíveis para validação de soluções alternativas para a Internet do Futuro. O *OpenFlow* é considerado o primeiro padrão de interface de comunicação entre as camadas de controle e encaminhamento da arquitetura SDN [ONF 2012]. Fruto de programas de pesquisa realizados na Universidade de Stanford, a solução surgiu como uma abordagem para oferecer aos pesquisadores a possibilidade de testar seus protocolos e arquiteturas experimentais diretamente sobre ambientes de produção, sem, no entanto, comprometer o tráfego original da rede. Um mecanismo que é executado nos comutadores possibilita a manipulação de suas tabelas de encaminhamento, no plano de dados, por um elemento remoto denominado controlador.

Entender o funcionamento e os avanços trazidos por esta solução exige antes uma melhor compreensão de como os equipamentos de rede operam atualmente. Os dispositivos, como switchs e roteadores, são verdadeiras "caixas-pretas", sendo que, apenas os próprios fabricantes têm condições de modificar sua inteligência. Conforme destaca [McKeown et al. 2008], a prática comercial no campo de redes é restringir a padronização de interfaces externas apenas ao plano de encaminhamento de pacotes, estando oculta toda a flexibilidade interna do dispositivo. Os elementos internos ainda diferem de fornecedor para fornecedor, não havendo uma plataforma padrão para que os pesquisadores tenham condições de experimentar suas novas ideias.

O ponto principal do *OpenFlow* é buscar reverter estas limitações, permitindo que o *software* que define as ações na rede seja retirado dos dispositivos e alocado em um servidor externo de gerenciamento. Em um roteador ou *switch* clássico, o encaminhamento de pacotes (*datapath* ou plano de dados) e as decisões de roteamento de alto nível (*controlpath* ou plano de controle) estão agregadas no mesmo dispositivo. Conforme explica [Campista et al. 2010] o *OpenFlow* separa estas duas funções. A função de encaminhamento de pacotes continua a residir nos comutadores/roteadores. Porém, as decisões de controle, como as definições de roteamento, por exemplo, são movidas para um novo elemento de rede, denominado Controlador, normalmente localizado em um servidor. Os comutadores e controladores se comunicam através do protocolo *OpenFlow*. Portanto, seguindo o paradigma das SDNs, a adoção de controladores desagregados da camada física garante que a programabilidade dos comutadores seja realizada externamente, suprimindo as dependências de fabricante.

A Figura 2 inclui uma representação dos elementos que compõem a arquitetura de um comutador *OpenFlow*. Conforme destaca [McKeown et al. 2008], um *switch OpenFlow* é composto de pelo menos três partes: (1) uma Tabela de Fluxos, com uma ação associada a cada entrada, determinando como o fluxo deve ser processado, (2) um canal seguro, que conecta o *switch* ao controlador remoto, permitindo que pacotes sejam enviados entre eles utilizando (3) o protocolo *OpenFlow*, que fornece um padrão aberto de comunicação. As funções e características dos principais componentes estão melhor detalhadas nos parágrafos seguintes.



Adaptada de [McKeown et al. 2008]

Figura 2: Especificação do comutador *Openflow*

A Tabela de Fluxos define entradas, que consistem em campos do cabeçalho dos pacotes, associados a ações, que estabelecem como os pacotes devem ser processados e para onde devem ser encaminhados. Além disso, possui contadores, utilizados para acompanhamento de estatísticas ou remoção de fluxos inativos. Cada fluxo é, portanto, identificado na tabela a partir de informações contidas em seu cabeçalho, como endereços MAC (*Media Access Control*), endereços IP (*Internet Protocol*), portas TCP (*Transmission Control Protocol*) etc [Campista et al. 2010]. Os campos de cabeçalho disponíveis para comparação com fluxos de entrada variam conforme a versão do protocolo *OpenFlow*. A Tabela 1 inclui uma representação dos campos definidos na versão 1.0.0 do protocolo [Openflow 2009].

Porta de Entrada	ID VLAN	VLAN PCP	Ethernet			IP				TCP/UDP	
			MAC Orig	MAC Dest	Tipo	End Orig	End Dest	Proto	ToS	Porta de Orig	Porta de Dest

Tabela 1: Campos do pacote utilizados para combinação com entradas de fluxos

Portanto, ao identificar um fluxo, o *OpenFlow* executa a ação relacionada, conforme descriminado pela tabela. Podem se tratar de ações como: enviar o pacote para uma porta de saída específica (`OFPAT_OUTPUT`); encaminhar o pacote através de uma fila ligada a uma porta (`OFPAT_ENQUEUE`); ou até alterar determinados campos do pacote (*Modify-Field*), como alterações em bits associados a VLAN (*Virtual Local Area Network*). Os

pacotes recebidos que não possuem entradas correspondentes são enviados ao controlador, utilizando-se mensagens `OFPT_PACKET_IN`, através do canal seguro de comunicação, que estabelece a ação a ser aplicada ao pacote, podendo configurar novas entradas na tabela de fluxos.

Um Canal Seguro é aplicado para evitar que a rede sofra ataques de elementos mal intencionados. O canal assegura, portanto, confiabilidade na troca de informações entre o *switch* e o controlador. A interface utilizada para acesso ao tráfego é o protocolo SSL (*Secure Socket Layer*) [Farias et al. 2012]. Através desta interface, o controlador configura e gerencia o *switch*, recebe eventos e envia pacotes através das portas do comutador, utilizando como padrão de comunicação do protocolo *OpenFlow*.

O protocolo *OpenFlow* disponibiliza, portanto, um padrão aberto para estabelecimento de comunicação entre o *switch* e o controlador. Conforme documento de especificação do *switch Openflow* [Openflow 2009], o protocolo suporta três tipos de mensagem: controlador-switch, assíncrona e simétrica, cada uma delas incluindo múltiplos subtipos. O primeiro tipo inclui as mensagens iniciadas a partir do controlador e aplicadas para gerenciar diretamente ou inspecionar o estado do dispositivo, por exemplo, as mensagens de consulta ou definição de parâmetros do *switch*, modificação de tabelas de fluxos, ou de solicitação de estatísticas. As mensagens assíncronas, em contrapartida, são iniciadas pelo dispositivo de rede, sem solicitação prévia do controlador, e utilizadas para informar sobre eventos da rede ou alterações de estado do *switch*, como a chegada de um novo pacote que não possua entrada de fluxo correspondente, ou sobre a ocorrência de um erro. Por fim, o terceiro tipo inclui as mensagens que podem ser enviadas tanto a partir do controlador quanto de um *switch* e sem solicitação, é o caso das mensagens trocadas durante o estabelecimento de conexão ("`HELLO`") ou as de "`Echo Request/Reply`".

Outro importante elemento nas redes *OpenFlow* é o controlador. Este é o componente onde são incluídas as lógicas de decisão, que originam as instruções a serem encaminhadas. As aplicações implementadas no controlador fornecem uma interface para execução de funções como: atualização das tabelas de fluxo, configuração do *switch* ou troca de mensagens sobre capacidades do dispositivo. Os controladores, portanto, são definidos como sistemas operacionais de rede, fornecendo uma interface genérica de alto nível para programação de aplicações que realizam o gerenciamento e controle da rede [Gude et al. 2008].

Diferentes tipos de controladores de rede já foram desenvolvidos dentro do contexto das redes *OpenFlow*. Dentre eles, incluem-se, por exemplo: o NOX [Gude et al. 2008], cuja infraestrutura básica e principais funções foram implementadas em C++; o *Floodlight* [Floodlight 2013], incluindo o núcleo e módulos principais baseado na linguagem Java e distribuído segundo a licença Apache; ou o Maestro [Cai et al. 2013], uma plataforma escalável de controladores para *switches OpenFlow* em Java.

Portanto, por se tratar de um padrão aberto capaz de reduzir a complexidade de gerenciamento da rede, garantindo o domínio sobre seu comportamento, a plataforma *OpenFlow* tem oferecido relevante suporte à inovação, oportunizando o desenvolvimento

de novos mecanismos e sua validação em ambientes reais. Aos poucos a solução vem ganhando maior apoio em todo o setor de redes, sendo que vários tipos de implementação vêm sendo desenvolvidas para aperfeiçoamento e adequação dos ambientes baseados em *OpenFlow*.

2.3 *FlowVisor:* Virtualização sobre *OpenFlow*

Ao criar condições favoráveis para que o comportamento da rede seja controlado, o *OpenFlow* abriu espaço para utilização da rede como ambiente de experimentação. Porém, o emprego da arquitetura básica definida para o *OpenFlow*, limitada a utilização de um único controlador para manipulação dos dispositivos da rede, acaba restringindo a capacidade de instalação de novas aplicações. Ainda que o acesso ao controlador seja compartilhado, existe a limitação de uma única interface disponível e a necessidade de inclusão de mecanismos para evitar interferências entre os diferentes experimentos.

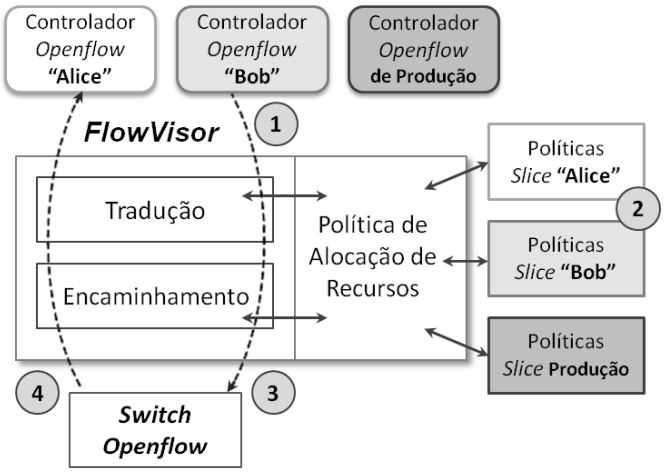
Neste sentido, com o objetivo de superar as dificuldades supracitadas, a utilização de técnicas de virtualização dos recursos de rede em conjunto com a solução *OpenFlow* vem se projetando como uma excelente alternativa [Farias et al. 2011]. Com a aplicação deste princípio, os recursos da rede física podem ser apresentados separadamente a cada desenvolvedor, habilitando, efetivamente, a execução de múltiplos experimentos diretamente sobre os ambientes de produção.

Neste contexto, através da definição de uma camada de virtualização para o *OpenFlow*, o *FlowVisor* [Sherwood et al. 2009], [Sherwood et al. 2010a] é apresentado como uma abordagem para permitir que o mesmo hardware do plano de dados possa ser compartilhado entre múltiplas fatias de redes virtuais (do inglês *slices*), cada uma com lógicas de encaminhamento distintas. Cada *slice* está vinculado a um controlador. Sendo assim, diferentes redes podem utilizar os recursos de forma independente e aplicando seu próprio controlador. De acordo com [Sherwood et al. 2010a] uma camada de abstração de *hardware* é provida pelo *OpenFlow* e como camada de virtualização se tem *FlowVisor*.

O *FlowVisor* é um controlador específico que opera de forma transparente entre os *datapaths OpenFlow* e múltiplos controladores. As mensagens *OpenFlow* enviadas de e para os controladores são interceptadas pelo *FlowVisor*, podendo ser reescritas, e encaminhadas de acordo com as redes virtuais estabelecidas. As fatias de rede são definidas através da união, interseção e/ou diferença de valores dos diversos campos de cabeçalho disponíveis para identificação de fluxos *OpenFlow* [Guedes et al. 2012]. Alguns dos campos, conforme representados na Tabela 1 da Seção 2.2 são implementados pelo *FlowVisor*. Este conjunto de fluxos a ser controlado por cada *slice*, é denominado de "espaço de fluxo"(ou *flowspace*).

Desse modo, os controladores não necessitam sofrer alterações e, devido à interceptação transparente do *FlowVisor*, os mesmos acreditam estar se comunicando diretamente com os dispositivos da rede que constituem o plano de dados [Sherwood et al. 2010b].

A Figura 3 ilustra, de maneira geral, os procedimentos implementados pelo *FlowVisor* para gerenciamento das fatias de rede. Além do controlador de produção, outros dois controladores são representados, identificados por "Alice"e "Bob", o que determina, consequentemente, a presença de três *slices*. A dinâmica de processamento das mensagens interceptadas pelo *FlowVisor* está representada pelos círculos numerados. Inicialmente, mensagens *OpenFlow* originadas de um controlador são interceptadas (1). Aplicando a política do espaço de fluxos definida para a fatia (2), as mensagens são reescritas de forma transparente, a fim de garantir o controle de determinada fatia da rede que compõe o plano de dados (3). Da mesma forma, as mensagens provenientes dos comutadores para o plano de controle (4) são novamente interceptadas e encaminhadas ao controlador correspondente, de acordo com a política de definição da rede virtual.



Adaptada de [Sherwood et al. 2009]

Figura 3: Visão geral de funcionamento do *FlowVisor*

Portanto, a aplicação de uma camada de virtualização nas redes *OpenFlow* destina-se a proporcionar melhorias na alocação de recursos e um melhor aproveitamento da infraestrutura, permitindo que múltiplas redes sejam executadas, simultaneamente, sobre um mesmo substrato físico.

Ressalta-se, no entanto, que no contexto das redes virtuais, a correta separação de alguns recursos é considerada indispensável para garantir uma completa virtualização. Dentre eles [Sherwood et al. 2010b] destaca: topologia, largura de banda, capacidade de processamento e tabelas de encaminhamento. Especificamente em relação à largura de banda, os autores descrevem que cada fatia deve possuir sua própria fração, de tal forma que, falhas no isolamento deste recurso acarretariam em interferências entre os *slices*, podendo comprometer o rendimento de cada um ou de toda a infraestrutura.

Neste mesmo sentido, [Kanada et al. 2012] evidenciam como um requisito chave para alcançar a virtualização de rede o isolamento de recursos entre fatias, ou seja, permitir que uma rede virtual utilize os recursos necessários para fornecer o desempenho esperado, mesmo quando um congestionamento ocorre na rede física ou em outras fatias. De acordo com os mesmos autores, duas funções de controle de tráfego, geralmente utilizadas para

garantia de QoS, isto é, modelagem e policiamento de tráfego, podem ser aplicadas para o isolamento de recursos.

Embora o *FlowVisor* seja capaz de lidar com múltiplos controladores, atribuindo uma fatia para cada um deles, atualmente a definição de uma solução que forneça um controle eficiente para alocação de recursos nas redes virtuais, como provisionamento de largura de banda [Takahashi 2012], ainda é um problema pouco explorado. Apesar das funcionalidades apresentadas em [Sherwood et al. 2009] e [Sherwood et al. 2010b] incluírem mecanismos para isolamento de recursos como largura de banda, topologia ou processamento, algumas soluções apresentam limitações, demandando a definição de controles mais efetivos.

Além disso, as configurações que devem ser realizadas nos dispositivos para suportar o compartilhamento e alocação de recursos são dependentes de ferramentas externas, como as realizadas estaticamente pelo comando `dpctl` do *framework OpenFlow* ou por um outro protocolo de configuração, dedicado para este fim.

Consequentemente, esta é uma das principais questões ponderadas na avaliação do *FlowVsior*, quanto a sua estabilidade para aplicação de redes virtuais e adoção efetiva em ambientes de produção. Segundo [Sherwood et al. 2010b] a exposição de um controle mais refinado dos elementos de encaminhamento permitirá a resolução de problemas de isolamento ainda existentes. Os autores apontam os ganhos obtidos com o aperfeiçoamento das funções de isolamento, que incluem: possibilitar aos pesquisadores a validação de suas ideias em larga escala e com maior realismo; a indústria pode garantir com maior segurança a qualidade de seus novos produtos; e permitir aos operadores executar várias versões da rede em paralelo, e, quando necessário, reverter para estados reconhecidamente estáveis.

2.4 Conclusões do capítulo

As redes de computadores, de uma maneira geral, têm se expandido de forma considerável nos últimos tempos. Novas aplicações, cada vez mais robustas, estão sendo demandadas pelos usuários. As características dos serviços se tornam cada vez mais desafiadoras, determinando, portanto, a necessidade de atendimento a requisitos não mais suportados pela estrutura atual da rede. Neste contexto, pesquisas são realizadas com o intuito de definir uma nova arquitetura.

Este cenário conduz à necessidade de ambientes apropriados para validação efetiva das novas propostas e, com esta finalidade, as redes definidas por software do protocolo *Openflow* tem se mostrado como uma tecnologia promissora para definição de redes programáveis e virtualizadas.

Além das vantagens introduzidas pelas redes programáveis na experimentação de novos protocolos e arquiteturas, [ONF 2012] apontam outros benefícios da adoção de uma rede definida por software baseada em *Openflow*, dentre eles destacam-se: controle cen-

tralizado dos dispositivos, independente de suas plataformas; oportunidade de automação de tarefas manuais, reduzindo a complexidade de operação da rede; criação de condições favoráveis para inovações; e o aumento da confiabilidade e segurança da rede, eliminando a necessidade de configuração individual e direta dos dispositivos.

Além disso, a integração do *Openflow* com a ferramenta *FlowVisor* capacita a virtualização de elementos da rede, possibilitando que uma mesma infraestrutra física seja compartilhada entre várias topologias lógicas. Desta forma, estabelecem-se condições para que, além do tráfego de produção, mais de um experimento seja executado simultaneamente sobre a rede.

Dentre as funcionalidades apresentadas pelo *FlowVisor* destacam-se vários mecanismos de isolamento, que buscam evitar que um *slice* interfira no funcionamento de outro. No entanto, a aplicação destas funções e gerenciamento de ambientes baseados em *OpenFlow/FlowVisor* é um assunto que ainda não foi totalmente explorado. Dessa forma, faz-se necessário a investigação e aplicação de soluções para o gerenciamento pleno desse tipo de ambiente, proporcionando uma melhor operação das redes constituídas nesse modelo.

Este trabalho explora uma das principais limitações da atual arquitetura do *FlowVisor*, relacionada a carência de métodos para o isolamento de recursos entre as redes virtuais definidas, buscando a manutenção do desempenho de cada fatia, independente de eventos que possam ocorrer nos demais ambientes virtuais.

CAPÍTULO 3

Trabalhos Relacionados

O objetivo deste capítulo é apresentar uma avaliação comparativa dos principais trabalhos e pesquisas relacionados ao tema desta dissertação. Diversas outras iniciativas, não incluídas nesta análise, foram investigadas durante a etapa de levantamento bibliográfico. Desta forma, são apresentados apenas os conceitos que serviram de embasamento para as tomadas de decisões e definição do escopo da proposta.

Alguns trabalhos apresentam os mecanismos adotados por diferentes versões do *FlowVisor* na tentativa de promover controle de recursos entre os *slices*. Em [Sherwood et al. 2009] é apresentado como solução para alocação de banda a marcação do campo VLAN PCP (*Priority Code Point*) nos pacotes. Conforme destacam os autores, a utilização de bits de VLAN PCP é apenas uma solução de curto prazo, demandando em novas versões um controle mais direto e preciso de QoS.

Versões superiores do *FlowVisor*, por sua vez, empregam o protocolo OpenFlow v1.0. Apesar de encontrar-se definida por esta versão do protocolo melhorias que permitem encaminhar pacotes através filas configuradas em portas dos dispositivos, o *FlowVisor* não prevê mecanismos suficientes para implementar o controle de recursos.

A análise adicional de trabalhos que implementaram propostas para isolamentos de recursos em outros ambientes virtuais foram indispensáveis para auxiliar na determinação da abordagem empregada neste trabalho. Em [Kanada et al. 2012], por exemplo, é sugerida e validada a aplicação de funções de controle de tráfego para garantir o isolamento.

Em outro trabalho, [Ishimori et al. 2012b], com o objetivo de automatizar e suprir as deficiências de gerenciamento de QoS (*Quality of Service*) apresentadas pelo protocolo *OpenFlow*, propõe o QosSFlow. Trata-se de um arcabouço de software que inclui funcionalidades para administração dinâmica de recursos de QoS, como largura de

banda, tamanho da fila, ou atraso. A arquitetura QosFlow apresentada não contempla sua aplicação em cenários virtuais utilizando *OpenFlow*, devido ao não suporte destes às novas mensagens QoSFlow.

3.1 Abordagens sobre o *FlowVisor*

Um dos primeiros trabalhos a apresentar o *FlowVisor* pode ser encontrado em [Sherwood et al. 2009]. Neste, são incluídos os mecanismos aplicados pelo *FlowVisor* para o isolamento de recursos entre os *slices*, dentre eles, os utilizados para isolamento de largura de banda. Esta versão da ferramenta foi implementada em C e, conforme destaca o autor, apesar da versão do protocolo *Openflow* utilizada no desenvolvimento não implementar mecanismos de gerenciamento de QoS, o *FlowVisor*, ainda assim, conseguia alavancar características de isolamento de largura de banda nos *switchs*, utilizando para tanto a atribuição do campo VLAN PCP nos pacotes. Conforme destacam os próprios autores, a utilização de bits de VLAN PCP é apenas uma solução de curto prazo, demandando em novas versões um controle mais direto e preciso de QoS. Além disso, a definição de cada classe de tráfego deveria ser configurada diretamente em cada *datapath* por linha de comando.

Outros trabalhos consideram a mesma abordagem. Em [Min et al. 2012] são apresentados esquemas para o controle de admissão e garantia de largura de banda mínima. A alocação da banda requerida por um fluxo específico ainda é realizada através da marcação dos bits de prioridade de VLAN nos pacotes, porém de forma dinâmica, incluindo novos módulos e interfaces de configuração.

Versões superiores do *FlowVisor*, por sua vez, empregam o protocolo OpenFlow v1.0. Apesar do protocolo definir a ação `OFPAT_ENQUEUE`, utilizada para encaminhar pacotes através de uma fila configurada em uma porta do *datapath*, o *FlowVisor* não previa mecanismos para implementar este controle. Apenas a ação de `OFPAT_OUTPUT`, que encaminha os pacotes diretamente por uma porta, encontrava-se definida. Portanto, mesmo que os controladores fornecessem suporte a QoS, ao receber mensagens do tipo `OFPAT_ENQUEUE` estas eram descartadas pelo *FlowVisor*.

A versão 0.10 do FlowVisor foi liberada ao final de 2012 [Al-Shabibi 2012], incluindo algumas melhorias em relação ao tratamento das mensagens `OFPAT_ENQUEUE`, além da possibilidade de atribuição de filas aos *flowspaces*, através da definição de novos parâmetros para as entradas de fluxos. Ações de `OFPAT_OUTPUT` também podem ser reescritas para ações de `OFPAT_ENQUEUE`, conforme as regras definidas. A principal limitação da solução ainda reside no fato de serem necessárias que as definições de fila no *datapath* sejam realizadas por ferramentas externas e fortemente dependente de atuação manual, o que restringe o gerenciamento completo das classes de serviço pelo *FlowVisor*.

3.2 Isolamento de recursos em ambientes virtuais

Compreender as técnicas aplicadas para isolamento de recursos em diferentes ambientes virtuais e os resultados encontrados em alguns trabalhos, foi essencial para definição da abordagem implementada na solução proposta.

Em [Kanada et al. 2012] são propostos métodos para evitar interferências entre redes virtuais. De acordo com os autores, duas funções de controle de tráfego, geralmente utilizadas para garantia de QoS, ou seja, modelagem e policiamento de tráfego, podem ser utilizadas para o isolamento de recursos de rede. Com a modelagem de tráfego é possível a alocação de fila aos *slices*, evitando interferências entre eles. O policiamento, por sua vez, é aplicado por *link*, evitando interferências entre fluxos dentro de um mesmo *slice*.

No trabalho analisado, os dois métodos foram implementados, porém, os autores destacam que a modelagem de tráfego é suficiente para garantir o isolamento de recursos entre as fatias de rede, evitando interferências tanto de largura de banda quanto de atraso. A utilização do método em conjunto com a alocação por fluxo deve ser aplicado se o objetivo é a garantia completa de QoS. As implementações e validações foram realizadas em um protótipo VNode (*Virtualization Node*) baseado na arquitetura do projeto VNP (*Virtualization Node Project*) [Nakao 2010].

Em [Skoldstrom and Yedavalli 2012] os autores investigam, além de modelos de virtualização para redes *OpenFlow*, mecanismos para impor a alocação de recursos entre as diferentes fatias da rede. Especificamente em relação à alocação de recursos, afirmam que a aplicação de ferramentas clássicas de QoS como classificação, medição, policiamento, modelagem e escalonamento nas portas de entrada e saída, podem limitar os congestionamentos.

No contexto da virtualização as técnicas de QoS citadas podem ser aplicadas para garantir a utilização justa dos recursos e o isolamento entre diferentes redes virtuais. Os autores destacam, ainda, que a especificação do protocolo *OpenFlow* v1.1 não proporciona muito suporte para ferramentas de QoS. Desta forma, são sugeridas algumas extensões ao protocolo. Considerando os modelos analisados e as sugestões apresentadas, os autores definem um sistema de virtualização flexível, ainda não implementado, que permite alocação justa do *datapath* e o controle de recursos nas redes virtuais.

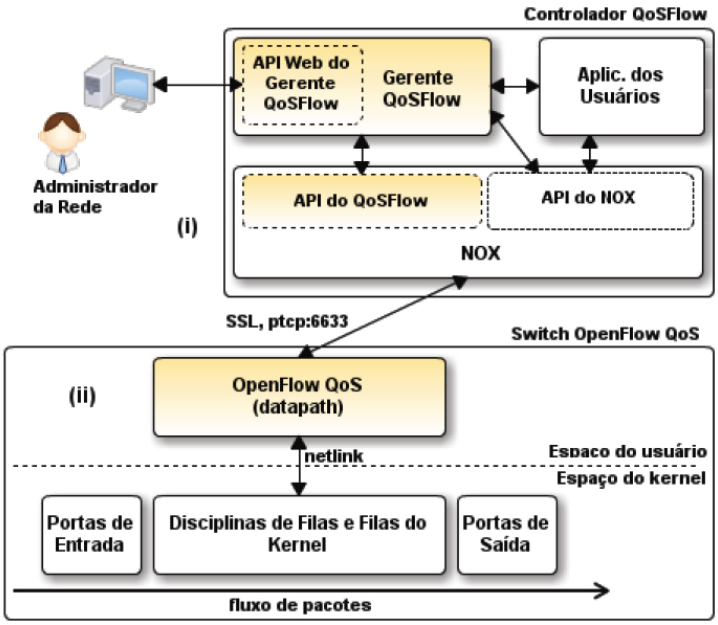
Apresentando um modelo para isolamento de recursos em redes *OpenFlow* virtualizadas, [El-azzab et al. 2011] propõem a inclusão de diferentes níveis de isolamento entre os *slices* de um *switch* virtualizado, cada um combinando um ou mais recursos disponíveis (interface, memória e processamento). Uma modelagem matemática é definida para possibilitar a correta identificação e validação dos fluxos atribuídos a cada *slice*. Os autores, porém, não apresentam detalhes das possíveis implementações realizadas para aplicação do modelo sobre o *FlowVisor* ou dos mecanismos para permitir a alocação dos recursos.

3.3 QosFlow

O arcabouço de software QosFlow [Ishimori et al. 2012b] foi proposto com o objetivo de suprir as deficiências de gerenciamento de QoS apresentadas pelas redes *OpenFlow*. Conforme destacam os autores, nestas redes, as configurações que podem ser realizadas nos *datapaths*, com o objetivo de garantir qualidade de serviço, são dependentes de ferramentas externas.

O QoSFlow inclui funcionalidades para administração dinâmica de recursos de QoS, como largura de banda, tamanho da fila, ou atraso. Através do controlador é possível a programação de aspectos de QoS diretamente nas portas de dispositivos habilitados com *OpenFlow*.

A arquitetura modular da solução, representada na Figura 4, é composta por dois elementos principais: (i) o controlador e (ii) o *datapath* QosFlow. O módulo controlador foi desenvolvido como uma API (*Application Programming Interface*) sobre o NOX, onde novas mensagens de controle foram implementadas para permitir o gerenciamento dos recursos de QoS no plano de dados. As alterações no *datapath OpenFlow* original incluem subcomponentes para o recebimento e processamento das mensagens de QoS, adicionadas ao protocolo. As ações de QoS são aplicadas automaticamente com base nas informações contidas no cabeçalho do pacote.



Fonte [Ishimori et al. 2012b]

Figura 4: Arquitetura do QosFlow

As alterações realizadas no protocolo incluem a definição de um novo tipo de mensagem, identificada no cabeçalho *OpenFlow* como `OFPT_QOS_QUEUEING_DISCIPLINE`. As mensagens são interpretadas pelo *datapath* permitindo a configuração das primitivas de QoS na porta dos dispositivos. Como exemplo de ações disponíveis [Ishimori et al. 2012b]

cita a configuração de disciplinas de fila FIFO (*First In, First Out*) e HTB (*Hierarchical Token Buckets*) e o gerenciamento de largura de banda na rede.

Em [Ishimori et al. 2012a] é apresentada uma lista não exaustiva incluindo os tipos de problemas que podem ser resolvidos adotando-se o QosFlow para maximizar a utilização dos recursos de rede. São eles: limitar a largura de banda total para uma taxa conhecida; limitar a largura de banda para um determinado usuário, serviço, ou cliente; reservar largura de banda para uma determinada aplicação ou usuário; gerenciar a banda excedente; permitir a distribuição equitativa da largura de banda não reservada.

Apesar das vantagens ofertadas pela solução, incluindo, ainda, a eliminação da tarefa de configuração manual em cada *switch* da rede, ou a independência do manuseio de ferramentas externas, a arquitetura QosFlow apresentada não contempla sua aplicação em cenários virtuais utilizando *OpenFlow*, devido ao não suporte destes às novas mensagens QoSFlow. Além disso, a concentração das tarefas de configuração dos parâmetros de QoS em controladores NOX restringe o modelo de controlador a ser aplicado sobre o *FlowVisor*, bem como, descentraliza-se funções de gerenciamento que devem ser executadas na camada de virtualização, uma vez que esta mantém uma visão geral e a administração sobre as redes virtuais definidas.

Desta forma, a aplicação de funções de gerenciamento QoSFlow no *FlowVisor* permite sua adoção no contexto de redes *OpenFlow* virtualizadas, concentrando diretamente no *FlowVisor* os mecanismos para configuração de classes e disciplinas de filas aos nós *OpenFlow*.

3.4 Conclusões do capítulo

Através da análise de trabalhos de pesquisa relacionados ao tema abordado nesta dissertação foi possível a identificação das principais deficiências que ainda precisam ser exploradas e superadas.

O primeiro foco da pesquisa incluiu a avaliação das principais funcionalidades adotadas sobre diferentes versões do *FlowVisor* para o isolamento de recursos. A Tabela 2 apresenta uma visão geral das abordagens, incluindo um comparativo entre as características analisadas em cada uma.

Destaca-se que o *FlowVisor* foi totalmente reescrito em Java na versão 0.6. A versão 0.8.X foi incluída na análise por se tratar da versão adotada durante o desenvolvimento deste trabalho. A proposta implementada está identificada na tabela como *FlowVisorQoS*. Percebe-se, portanto, que o *FlowVisor* ainda é fortemente dependente de atuação manual, mantendo em ferramentas externas a responsabilidade pela configuração dos parâmetros necessários no plano de dados. Além disso, não estão disponíveis mecanismos que garantam um controle mais preciso de QoS, como a possibilidade de configuração de escalonadores de fila, por exemplo.

O estudo de outros trabalhos relacionados à alocação e isolamento de recursos

Tabela 2: Análise comparativa de versões do *FlowVisor*

Características	FlowVisor até 0.6	FlowVisor0.8.X	FlowVisor0.10	*FlowVisorQoS*
VLAN PCP	Sim	Não	Não	Não
Tratamento de mensagens ENQUEUE	Não	Não	Sim	Sim
Atribuição de filas aos *slices*	Não	Não	Sim	Sim
Configuração direta no *datapath*	Não	Não	Não	Sim
Disponibilidade de modelagem de tráfego	Não	Não	Não	Sim

em redes virtuais, permitiram identificar os pontos chaves a serem abordados durante a definição da proposta. Dentre eles, destaca-se:

- Utilização de técnicas de QoS, como modelagem de tráfego, para garantir o isolamento entre as fatias de rede;
- Aplicar mecanismos para permitir a atribuição de classes de serviços aos *slices*;
- Tornar transparente a camadas superiores a utilização de controle de trafego pela camada de virtualização;
- Proporcionar flexibilidade a solução, a fim de evitar interações externas e reduzir os custos operacionais;
- Superar as limitações do *OpenFlow* quanto ao gerenciamento de QoS, o que é possível através da adoção da solução QosFlow;

CAPÍTULO 4

Proposta

Este capítulo abrange os procedimentos selecionados para o desenvolvimento da solução proposta e detalhes de sua arquitetura, com ênfase nas funcionalidades que foram acrescentadas ao *FlowVisor*. O principal objetivo da proposta é agregar novas funções à ferramenta que garantam o isolamento de recursos entre os ambientes virtuais. Dentre as funcionalidades aplicadas inclui-se uma interface para programação de aspectos de QoS nas portas de dispositivos de rede, permitindo o controle da alocação de recursos diretamente pelo *FlowVisor*.

Nas Seções seguintes encontram-se os detalhes do desenvolvimento e implementação, que estão estruturadas da seguinte forma: A Seção 4.1 apresenta uma visão geral do escopo e objetivos da proposta. A Seção 4.2 abrange os procedimentos de desenvolvimento aplicados e as particularidades da solução, incluindo nas Subseções 4.2.1 e 4.2.2 os detalhes de funcionamento da arquitetura. Por fim, a Seção 4.3 conclui o capítulo.

4.1 Visão geral e escopo da proposta

A solução proposta inclui o desenvolvimento de novos módulos na ferramenta *FlowVisor*, permitindo o aperfeiçoamento das funções de isolamento de recursos, que são essenciais para evitar interferências entre diferentes redes virtuais. Sua estrutura está, portanto, fundamentada na arquitetura SDN/*OpenFlow* vitualizada.

Dentre as funcionalidades implementadas destaca-se a definição de um módulo de controle capaz de interagir com *datapaths OpenFlow* com QosFlow habilitado. Conforme melhor detalhado no Capítulo 3, o QosFlow capacita o gerenciamento de qualidade de serviço em domínios *OpenFlow*. Assim sendo, torna-se possível a programação dinâmica de aspectos de QoS nas portas dos dispositivos de rede, proporcionando a definição de

classes e disciplinas de fila e, consequentemente, o controle de tráfego nos *datapaths*, diretamente a partir do *FlowVisor*.

Adicionalmente, a interface desenvolvida permite a atribuição de filas aos *slices*. Desta forma, o conjunto de fluxos conferido à determinada rede virtual será direcionado para sua respectiva fila, conforme especificações definidas. É incluído, ainda, um módulo para coleta de estatísticas de QoS a partir de *datapaths* QosFlow.

O *FlowVisor* está baseado em Java e utiliza um pacote com uma implementação, também em Java, do protocolo *OpenFlow* v1.0, denominado OpenFlowJ [OpenFlowJ 2012]. Durante o desenvolvimento deste trabalho, a versão 0.8.5 era a ultima versão estável do *FlowVisor* e foi utilizada para elaboração da proposta. Porém, após a realização das implementações e execução dos primeiros experimentos, outras versões foram lançadas. Evoluções da solução, em trabalhos futuros, irão considerar uma adaptação à versão mais recente da ferramenta.

4.2 Solução de Integração de QoS sobre virtualização *OpenFlow*

A necessidade de melhorias nas funções desempenhadas pelo *FlowVisor* no controle de recursos entre *slices*, levou a definição de um documento de projeto apresentado em [Takahashi 2012]. O documento descreve alguns passos essenciais para resolução do problema e serviu de embasamento para definição dos procedimentos adotados no desenvolvimento desta proposta. As seguintes alterações são apontadas no documento:

- (a) Atualização do pacote OpenFlowJ para suportar mensagens de obtenção das configurações de filas a partir do comutador *OpenFlow*;
- (b) Implementação de API para atribuição de fila aos slices; e
- (c) Substituição de ações nos pacotes recebidos.

Na proposta, porém, não são considerados os mecanismos para configuração de filas nos *datapaths*, assumindo o autor que as filas já foram apropriadamente configuradas por ferramentas externas.

Desta forma, considerando o projeto descrito anteriormente e suas limitações quanto à configuração apropriada dos *datapaths*, os procedimentos adotados durante o desenvolvimento da solução proposta neste trabalho consistiram em três etapas principais:

- (a) Atualização do pacote OpenFlowJ, para que o *FlowVisor* interprete e envie mensagens QoSFlow;
- (b) Desenvolvimento de uma API de gerenciamento para interagir com o *datapath* QoSFlow, transmitindo mensagens de configuração de parâmetros de QoS, bem como possibilitar a atribuição ou modificação de filas aos *slices*; e

- (c) Implementação de um conversor para substituir ações de `OFPAT_OUTPUT` por `OFPAT_ENQUEUE` nas mensagens Openflow recebidas a partir dos controladores.

A Figura 5 representa a arquitetura da solução. Seus componentes são descritos nas Subseções 4.2.1 e 4.2.2.

4.2.1 Apresentação da Arquitetura

Conforme descrito anteriormente, a arquitetura definida para a solução está fundamentada na arquitetura das redes definidas por software virtualizadas baseadas em *OpenFlow*. Novos módulos foram incluídos no *FlowVisor* e outros, já existentes, foram alterados para suportar as funcionalidades adicionais. Na Figura 5 estão em destaque estas inclusões e alterações.

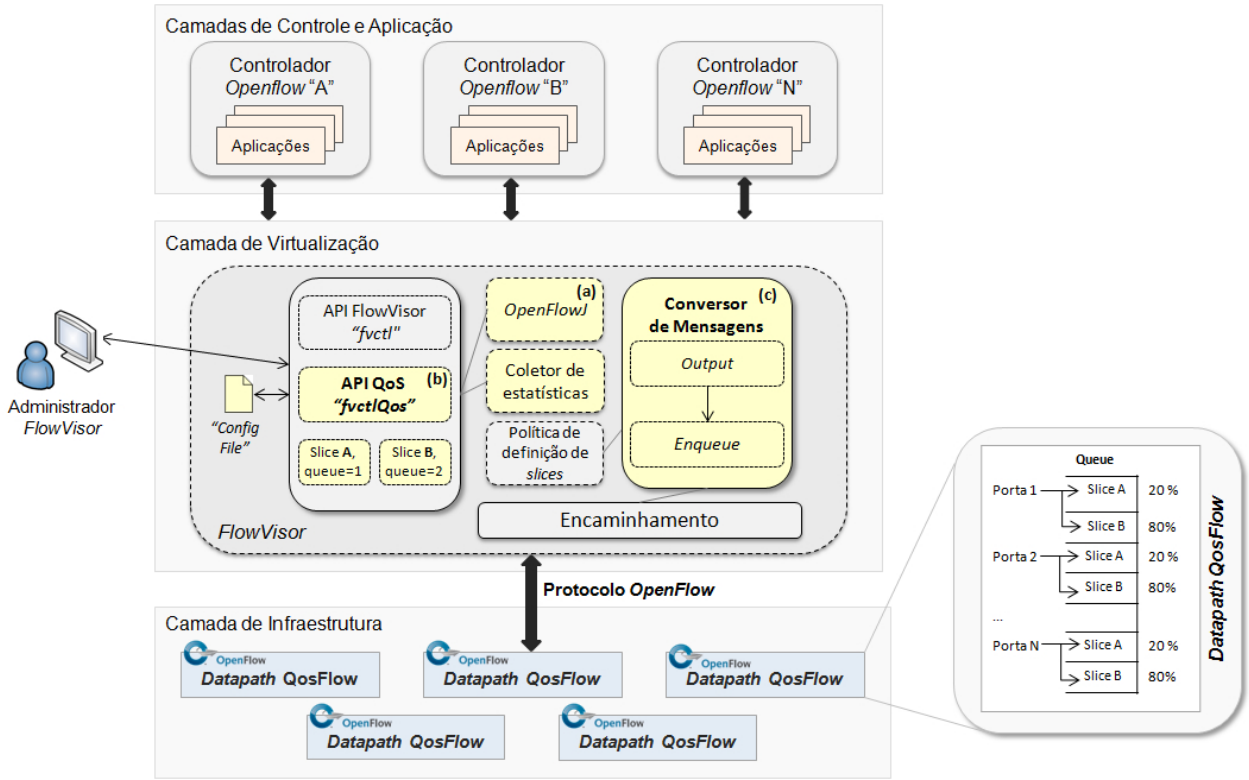


Figura 5: Arquitetura da solução proposta

As alterações no pacote OpenFlowJ incluem a definição de mensagens *OpenFlow* do tipo `OFPT_QOS_QUEUEING_DISCIPLINE`, que são interpretadas pelo *datapath* QosFlow, permitindo a configuração de classes, disciplinas de fila e seus parâmetros. Um maior detalhamento a cerca das implementações realizadas está incluído na subseção 4.2.2 e diagramas de classes representando as principais alterações foram incluídos no Apêndice B deste trabalho.

Na versão adotada do *FlowVisor* as configurações, monitoramento e administração da ferramenta são realizadas utilizando uma ferramenta de configuração em linha

de comando identificada como "*fvctl*", que envia os comandos a uma chamada de procedimento remoto em XML-RPC (*Extensible Markup Language-Remote Procedure Call*). Optou-se pela inclusão de uma nova interface de configuração, identificada como "*fvctlQoS*", destinada apenas ao tratamento de aspectos de QoS e atribuição de filas aos *slices*, mantendo-se inalterada a API original do *FlowVisor "fvctl"*. As informações das classes configuradas e a identificação das filas atribuídas a cada *slice* são armazenadas no arquivo de configuração da ferramenta.

O papel do conversor de mensagens é tornar transparente aos controladores a utilização de filas. Desta forma, o controlador envia mensagens com ações de `OFPAT_OUTPUT` e o *FlowVisor*, a fim de garantir o controle na utilização dos recursos altera as ações para `OFPAT_ENQUEUE`, conforme as configurações de fila estabelecidas para cada *slice*.

4.2.2 Detalhamento dos Componentes

Os elementos que compõem a arquitetura, conforme retratados na Figura 5, estão melhor detalhados a seguir:

- **Alterações no pacote OpenFlowJ:** permite a interação do *FlowVisor* com *datapaths* QosFlow. As extensões aplicadas pelo QosFlow ao software *switch* do *OpenFlow* incluem a definição de um novo tipo de mensagem *OpenFlow*, identificada como `OFPT_QOS_QUEUEING_DISCIPLINE`. As mensagens são interpretadas pelo *datapath*, permitindo a configuração de escalonadores de pacotes e de seus parâmetros. As alterações no pacote abrangem, portanto, a definição de novas mensagens *OpenFlow*, que disponibilizam as estruturas necessárias para configuração das seguintes disciplinas de fila: HTB (*Hierarchical Token Buckets*); PFIFO e BFIFO (*Packet* e *Byte First In, First Out*); SFQ (*Stochastic Fairness Queueing*) e RED (*Random Early Detection*).

- **Módulo API "fvctlQoS":** nova interface em linha de comando, implementada para permitir a interação com o *FlowVisor* para transmissão de mensagens de QoS aos *datapaths*, possibilitando a definição dos aspectos de QoS nos dispositivos de rede com QosFlow habilitado. Além disso, a API inclui comandos que permitem configurar ou modificar a atribuição de fila aos *slices*.

- **Conversor de Mensagens:** O objetivo do conversor de mensagens é tornar transparente aos controladores a utilização de filas. Desta forma, não são necessárias alterações em suas implementações para suportar o isolamento de recursos. Quando mensagens `OFPT_FLOW_MOD` ou `OFPT_PACKET_OUT`, com ações de `OFPAT_OUTPUT`, são recebidas pelo *FlowVisor*, a partir de um controlador, caso o slice de destino esteja atribuído à uma fila (*class_id* $\neq 0$), o *FlowVisor* substitui a ação `OFPAT_OUTPUT` por uma `OFPAT_ENQUEUE`. O campo "*queue_id*" da ação `OFPAT_ENQUEUE` é preenchido com base na definição realizada para o *slice* e armazenada no arquivo de configuração do *FlowVisor*.

- **Política de definição dos slices:** Não foram implementadas alterações nas definições de *flowspace*. Porém, como consequência da atribuição de filas aos *slices*, os conjuntos de fluxos de domínio daquele *slice* são mapeados para suas respectivas filas de acordo com as configurações realizadas.

- ***Datapath* QosFlow:** O *datapath OpenFlow* habilitado com QosFlow interpreta as mensagens de configuração recebidas a partir do *FlowVisor*, aplicando automaticamente as definições de QoS estabelecidas.

- **Módulo Coletor de estatísticas:** Possibilita a obtenção, a partir de um *datapath*, de estatísticas de QoS, como o número total de pacotes e *bytes* transmitidos por uma fila associada a uma porta de saída. As solicitações estão baseadas na mensagem de requisição *OpenFlow* do tipo `OFPT_STATS_REQUEST` e subtipo `OFPST_QUEUE`.

- **Arquivo de Configuração:** Algumas alterações foram implementadas no arquivo de configuração utilizado pelo *FlowVisor*, incluindo novos campos para armazenamento das informações de fila atribuída aos *slices*, bem como, as classes configuradas e seus respectivos atributos.

4.2.3 Sequência de Atividades

Merecem destaque a alteração de duas sequências de atividades que são desempenhadas pelo *FlowVisor* com a aplicação da proposta. A primeira delas diz respeito aos procedimentos realizados na ferramenta durante a configuração de classes e atribuição aos *slices*. A Figura 6 inclui um fluxograma contemplando as atividades executadas, onde foram destacados os passos introduzidos pela solução desenvolvida. Algumas etapas implementadas foram suprimidas, por exemplo, as mensagens de erro de sintaxe para os comandos ou as confirmações de existência de *slices* e *datapaths* antes de realizar as configurações, fornecendo assim uma visão geral e simplificada do diagrama.

Ao receber uma requisição para configuração de escalonadores no *datapath*, através de interface "fvctlQos", o *FlowVisor* executa as ações para criação de mensagens *OpenFlow* do tipo `OFPT_QOS_QUEUEING_DISCIPLINE` e as envia ao comutador a fim de serem configuradas as filas em suas portas. As novas mensagens enviadas incluem a configuração e definição de parâmetros específicos para diferentes disciplinas de fila disponíveis nos *datapaths* QosFlow, como HTB, PFIFO/BFIFO, SFQ ou RED. Ao confirmar a configuração, as definições das classes são armazenadas no arquivo de configuração. Desta forma, torna-se possível a atribuição das classes definidas aos *slices*.

A segunda sequência, por sua vez, está relacionada às atividades executadas a partir do momento em que o *datapath* envia uma mensagem do tipo `OFPT_PACKET_IN` ao controlador, considerando que não foi encontrada em sua tabela uma entrada de fluxo correspondente para o pacote.

Quando um *datapath* QoSFlow recebe um novo fluxo, que não possui uma ação correspondente em sua tabela de fluxos, então uma mensagem `OFPT_PACKET_IN` é envi-

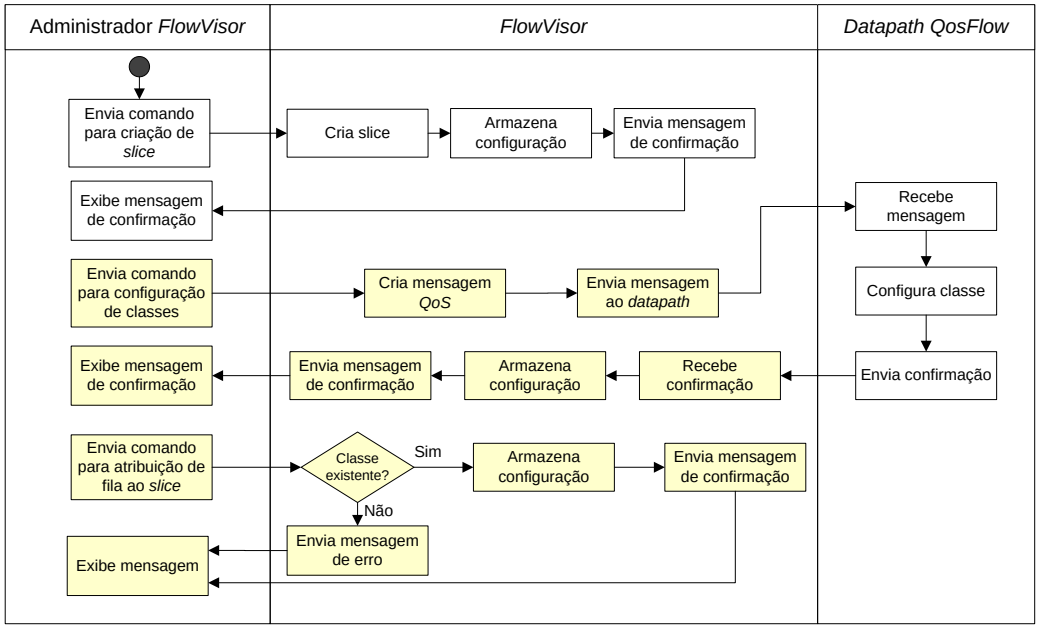


Figura 6: Fluxograma de definição de classes e atribuição aos *slices*

ada ao controlador, sendo interceptada pelo *FlowVisor*. O *FlowVisor* envia a mensagem ao controlador responsável pelo gerenciamento do fluxo correspondente, de acordo com as regas de *flowSpace* estabelecidas. O controlador então pode enviar mensagens `OFPT_FLOW_MOD` e/ou `OFPT_PACKET_OUT` ao *datapath*. O *FlowVisor* intercepta estas mensagens, verifica a classe de fluxo adequada ao slice, e, caso esta esteja configurada, realiza a substituição das ações `OFPAT_OUTPUT` das mensagens por ações `OFPAT_ENQUEUE`, antes de enviá-las ao *datapath* de destino.

Um fluxograma para permitir a melhor visualização desta sequência foi incluído no Apêndice A deste trabalho. Trata-se de uma adaptação dos diagramas encontrados na documentação da arquitetura do *FlowVisor*, incluindo-se as etapas para conversão das mensagens.

4.3 Conclusões do capítulo

Este capítulo apresentou detalhes do desenvolvimento da solução proposta neste trabalho. Através da implementação de módulos específicos o *FlowVisor* é habilitado para o gerenciamento e alocação de recursos de QoS diretamente em *datapaths* com QosFlow habilitado.

Uma interface foi desenvolvida para recebimento de comandos de configuração de escalonadores de fila. As requisições são interpretadas pelo *FlowVisor*, que cria mensagens *OpenFlow* e as envia diretamente ao *datapath* para configuração das filas e seus parâmetros nas portas do dispositivo.

Além disso, a interface disponibiliza comandos para atribuição das filas configuradas aos *slices*, garantindo que cada um utilizará apenas a fatia de recursos a ele conferida.

A aplicação de um conversor para substituição de ações nas mensagens recebidas a partir dos controladores torna transparente a utilização do controle de tráfego.

As implementações realizadas permitiram a execução de experimentos para validação da proposta, constatando-se sua capacidade para isolamento dos recursos de rede. Foram realizados testes em ambientes onde fluxos em fatias distintas competiam por largura de banda em um link compartilhado. Maiores detalhes dos experimentos e os resultados encontrados estão reproduzidos no Capítulo 5.

CAPÍTULO 5

Avaliação de Resultados

Este capítulo apresenta os testes executados para validação da proposta, avaliação de seu desempenho e comprovação de sua funcionalidade. A metodologia aplicada abrange o estabelecimento de ambientes onde fluxos em fatias distintas competiam por largura de banda em um link compartilhado, propiciando a comparação do comportamento da rede, com e sem a aplicação dos mecanismos definidos para o isolamento de recursos.

5.1 Metodologia dos Experimentos

Os testes foram realizados com o intuito de confirmar a capacidade da solução em limitar a largura de banda disponibilizada a diferentes *slices* e, consequentemente, atestar a diminuição de interferências entre os fluxos concorrentes.

Os experimentos foram conduzidos com a aplicação de topologias *OpenFlow* básicas. Nos comutadores utilizados em todos os experimentos foi empregado o software *datapath* QosFlow. Para os Cenários A e B, o modelo dos dispositivos utilizados consistiam em um TP-Link TL-WR1043ND, com sistema operacional OpenWRT versão 10.3.1 (*backfire*). Apesar de se tratar de um comutador sem fio, foram utilizados apenas as interfaces ethernet cabeadas. No Cenário C, por sua vez, o aplicativo Mininet [Lantz et al. 2010], capaz de simular um ambiente *OpenFlow*, foi utilizado para instanciar clientes, servidores e *datapaths QosFlow*.

Os controladores NOX *OpenFlow* aplicados, o *FlowVisor* e o Mininet foram instalados em máquinas virtuais com sistema operacional Ubuntu 11.04, sobre uma máquina física com processador Intel Core 2 Duo e utilizando, cada uma, respectivamente: 512MB, 1GB e 1GB de memória. Nos Cenários A e B as instâncias cliente e servidor foram aplicadas sobre máquinas físicas, com as seguintes configurações:

- Servidor: Sistema operacional Fedora release 17; Kernel Linux 3.6.8-2.fc17i686.PAE; Processador AMD C-50 x2 (800Mhz); Memória 1.6 GiB; e placa de rede 10/100Mbps.
- Clientes: Sistema operacional Ubuntu 11.04; Kernel 2.6.38-15-generic; Processador Intel Celeron M353 900 MHz; Memória 1GB DDR2; e placa de rede 10/100Mbps.

Três cenários foram aplicados durante a realização dos testes. Conforme representado na Figura 7, o primeiro consiste em um cenário semelhante ao apresentado em [Sherwood et al. 2009] e [Sherwood et al. 2010b] durante testes de validação de isolamento de largura de banda, compreendendo uma topologia *OpenFlow* básica.

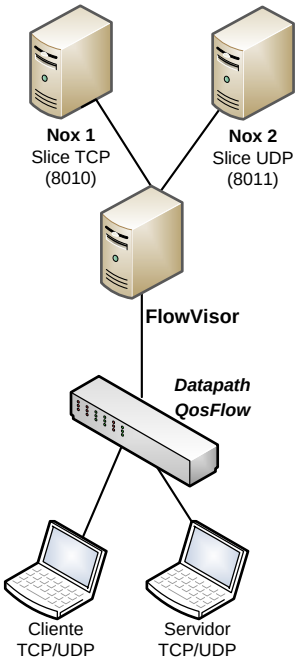


Figura 7: Cenário de Teste A

No segundo, por sua vez, foram incluídos novos dispositivos, considerando-se, ainda, a presença de três controladores, conforme Figura 8.

Por fim, no terceiro cenário, representado na Figura 9, utilizou-se o Mininet para definição de uma topologia mais complexa, permitindo atestar o comportamento da solução com um número ainda maior de *slices*. Os cenários e resultados encontrados estão mais bem detalhados nos subitens seguintes.

5.2 Detalhamento dos cenários

5.2.1 Cenário A

Para os experimentos executados no primeiro cenário, uma topologia OpenFlow básica foi utilizada. Conforme representado na Figura 7, 02 (duas) máquinas foram

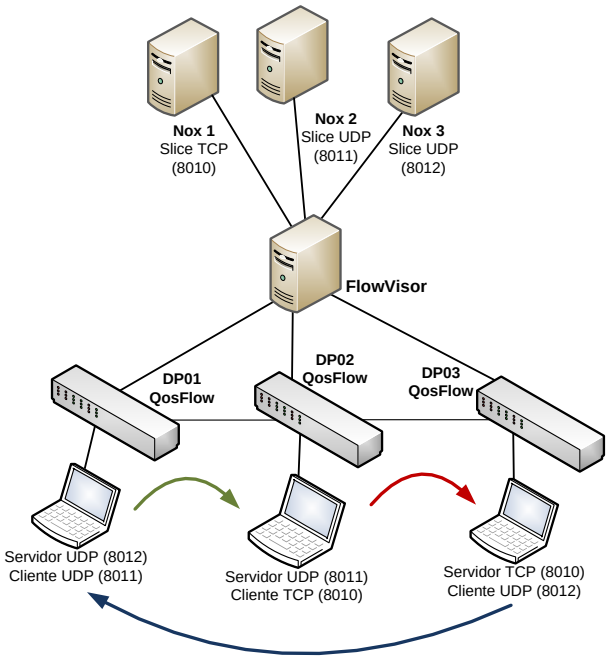


Figura 8: Cenário de Teste B

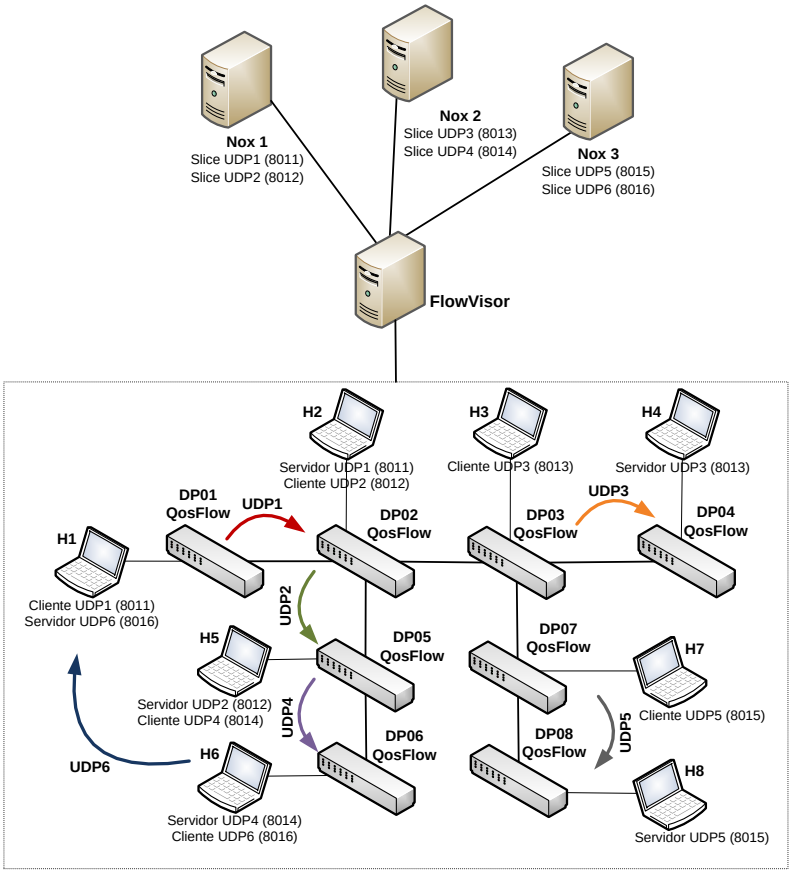


Figura 9: Cenário de Teste C

utilizadas: uma como origem dos fluxos, rodando duas instancias cliente, e outra como um destino comum executando duas instancias servidor, conectadas através de um *datapath*

Openflow com QoSFlow habilitado.

Dois controladores NOX foram aplicados para avaliação e encaminhamento de fluxos em *slices* distintos. O experimento executado consistiu na aplicação de fluxos TCP (*Transmission Control Protocol*) e UDP (*User Datagram Protocol*), competindo por largura de banda em um *link* compartilhado. Conforme destaca [Sherwood et al. 2009] estes padrões de tráfego são representativos de um cenário onde um *slice* de produção (TCP) compartilha um link com, por exemplo, um slice (UDP) executando um experimento DDoS (*Distributed Denial-of-Service*).

Para geração dos tráfegos foi adotada a ferramenta iperf [Iperf 2012], configurada no modo TCP (porta 8010) para produção do fluxo a ser transmitido pelo primeiro *slice* e no modo UDP (porta 8011) para o fluxo do segundo. As configurações de *flowSpace* foram realizadas para tratamento do tráfego iperf com base nas portas de origem e destino, conforme estabelecidas para cada protocolo.

5.2.2 Cenário B

Neste cenário novos nós foram aplicados à topologia, incluindo três *datapaths* com QosFlow habilitado e três controladores NOX conectados ao *FlowVisor*. Para cada um dos três nós, conectados a *datapaths* distintos, foram iniciadas uma instância cliente e uma instância servidor.

Três *slices* foram configurados: o primeiro para lidar com tráfego TCP na porta 8010 (representado em vermelho na Figura 8), o segundo com fluxo UDP na porta 8011 (em verde), e o terceiro também com fluxo do tipo UDP, porém na porta 8012 (em azul). As setas representam o direcionamento estabelecido para cada fluxo entre as instâncias cliente/servidor. A ferramenta iperf também foi utilizada neste caso para geração dos tráfegos em cada *slice*.

Em cada um dos dois cenários anteriormente definidos, os experimentos executados consistiram em duas etapas. Na primeira os fluxos foram gerados em cada *slice* sem os mecanismos de isolamento habilitados. Na segunda etapa os recursos para isolamento de largura de banda foram aplicados, através da configuração de disciplinas de filas e classes no *datapath*, além da atribuição de cada slice a uma destas classes. Os resultados encontrados estão detalhados na Seção 5.3.

5.2.3 Cenário C

O objetivo do terceiro cenário aplicado foi a verificação do comportamento da solução com a presença de um número ainda maior de *slices*. Desta forma, seis *slices* foram utilizados, cada um deles configurado para tratamento de fluxos UDP em portas distintas (respectivamente: 8011, 8012, 8013, 8014, 8015 e 8016). Os fluxos aplicados pelos *slices* estão representados em cores diferenciadas na Figura 9. Três controladores NOX foram utilizados, cada um com duas aplicações distintas sendo instanciadas. A

ferramenta Mininet foi utilizada para definição dos *datapaths QosFlow* e das instâncias cliente e servidor, conforme representados na Figura 9.

Neste caso, são apresentados apenas os resultados gráficos para a etapa onde os mecanismos propostos para alocação de banda foram aplicados, considerando que nos cenários anteriores já foi possível atestar a incapacidade de isolamento pelo *FlowVisor* sem a adoção da solução proposta.

5.3 Analise dos resultados

5.3.1 Resultados para os experimentos no Cenário A

Conforme destacado anteriormente, na primeira fase do experimento os fluxos foram gerados em cada *slice* sem os recursos de isolamento habilitados. Através do gráfico da Figura 10 é possível observar que o compartilhamento de banda entre *slices* TCP e UDP faz com que a presença de um afete o desempenho do outro. No tempo t=30seg o tráfego UDP, configurado para utilizar aproximadamente a taxa de recebimento de bits completa alcançada na comunicação cliente/servidor (50Mbps), é iniciado. Percebe-se a utilização de considerável fatia da banda disponível, causando queda na taxa de transmissão do fluxo TCP. A duração definida para o fluxo foi de 60 segundos e quando finalizado percebe-se o retorno do TCP ao seu estado inicial.

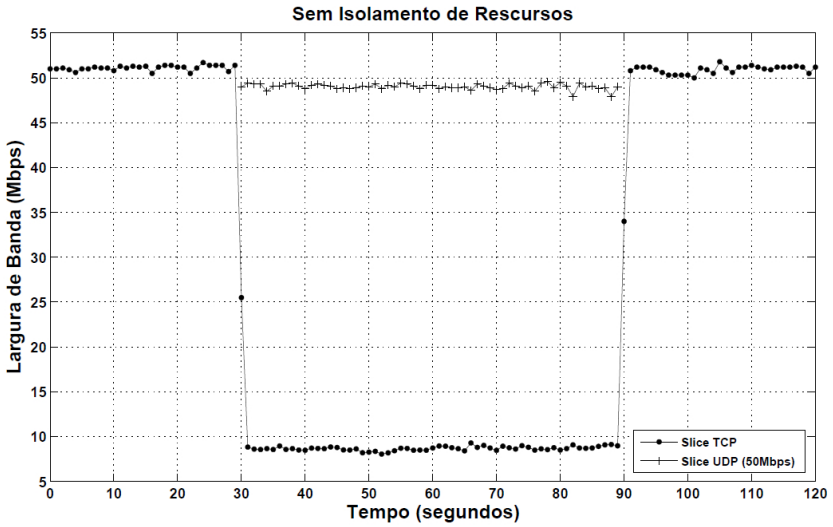


Figura 10: Resultado para o Cenário A sem isolamento de recursos

As características inerentes a cada um dos protocolos utilizados contribuiu para a ocorrência das interferências entre os fluxos no cenário aplicado, permitindo a avaliação do comportamento da rede, com e sem a aplicação de mecanismos para isolamento de recursos. Quando fluxos TCP e UDP tentam competir por largura de banda, esta não é equitativamente partilhada.

O comportamento padrão do protocolo UDP é considerado mais simples, não

provendo mecanismos de controle de fluxo ou retransmissões, buscando utilizar a largura de banda tanto quanto disponível.

Em contrapartida, as características básicas do protocolo TCP incluem a utilização de janelas deslizantes para controle de fluxo, confirmações e retransmissões. Congestionamentos na rede são detectados pelo protocolo através da perda de pacotes. Quando isso ocorre, o pacote é retransmitido, e ao mesmo tempo, o TCP reduz o tamanho da sua janela de congestionamento, efetivamente reduzindo sua taxa de saída para evitar congestionamentos adicionais. Na ausência de congestionamento, o TCP aumenta o tamanho de sua janela de congestionamento e a taxa de saída.

Na segunda etapa os recursos para isolamento de largura de banda foram aplicados, através da configuração de disciplinas de filas e classes no *datapath*, além da atribuição de cada *slice* a uma destas classes, conforme descrito a seguir:

- Slice TCP (8010): atribuído à "Classe 1", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 15Mbps;
- Slice UDP (8011): atribuído à "Classe 2", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 6Mbps;

Da mesma forma que na etapa anterior, iniciou-se pela aplicação do fluxo TCP, seguido pela transmissão do tráfego UDP. Conforme representado da Figura 11, mesmo com o início do fluxo UDP no tempo t=20s, o fluxo TCP consegue manter a taxa de transmissão configurada para sua classe. Da mesma forma, a taxa verificada para o UDP não ultrapassou o limite de 6Mbps estabelecido para a "Classe 2". Destaca-se que, neste caso, a utilização de largura de banda configurada para o UDP foi de 30Mbps, considerando a queda na taxa de transmissão constatada quando os mecanismos de QoS do QoSFlow são habilitados no *datapath*.

5.3.2 Resultados para os experimentos no Cenário B

Este cenário foi proposto para verificar o comportamento da rede com um maior número de *datapaths* e para certificar que as interferências verificadas no cenário anterior não foram totalmente provenientes da concorrência nas placas de rede da máquina cliente e/ou servidor.

Como no cenário anterior, duas etapas foram executadas. Na primeira, fluxos foram gerados em cada *slice*, em tempos distintos, sem a configuração das propriedades de controle de tráfego. O gráfico da Figura 12 apresenta os resultados encontrados. O inicio de fluxos UDP nos tempos t=20seg e t=40seg, cada um com duração de 40 segundos e configurados para transmissão a uma largura de banda de 25Mbps, afetam a taxa de

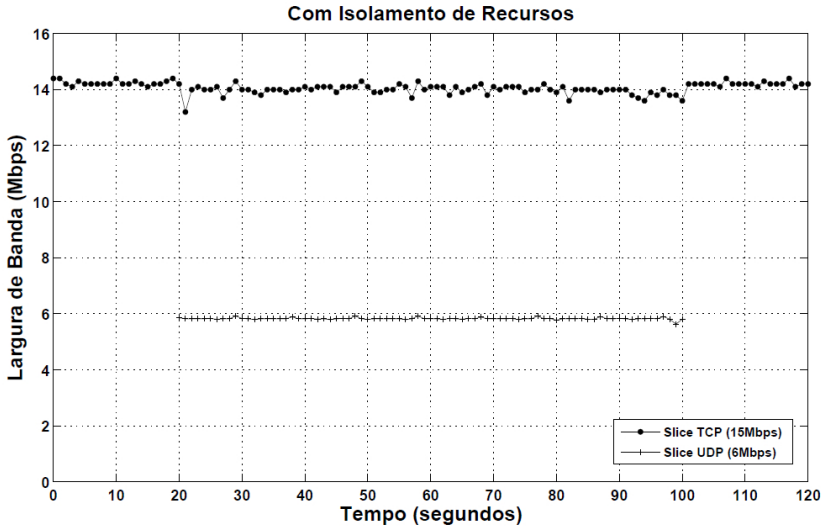


Figura 11: Resultados para o Cenário A com isolamento de recursos habilitado

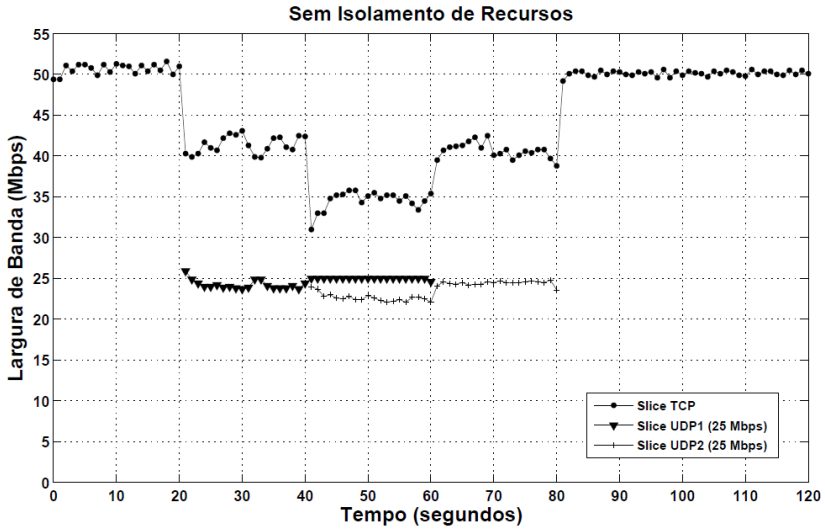


Figura 12: Resultados para o Cenário B sem isolamento de recursos

transmissão no slice TCP. É perceptível, ainda, uma pequena interferência entre os fluxos UDP, no intervalo de tempo em que são transmitidos em conjunto.

Como descrito anteriormente, na segunda etapa os mecanismos para isolamento de largura de banda foram habilitados, através da configuração de disciplinas de filas e classes nos *datapaths*, além da atribuição de cada slice a uma destas classes, conforme descrito a seguir:

- Slice TCP (8010): atribuído à "Classe 1", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 10Mbps;
- Slice UDP (8011): atribuído à "Classe 2", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 5Mbps;

- Slice UDP (8012): atribuído à "Classe 3", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 3Mbps;

Da mesma forma que na etapa anterior, iniciou-se pela aplicação do fluxo TCP, seguido pela transmissão de cada um dos tráfegos UDP, nos tempos t=20seg e t=40seg, cada um com duração de 40 segundos. O gráfico da Figura 13 apresenta os resultados encontrados.

Mesmo com o aumento no número de nós e o estabelecimento de mais uma classe, foi possível constatar a capacidade de isolamento, onde para cada *slice* verificou-se a manutenção da taxa média de banda garantida, conforme configurado em cada classe.

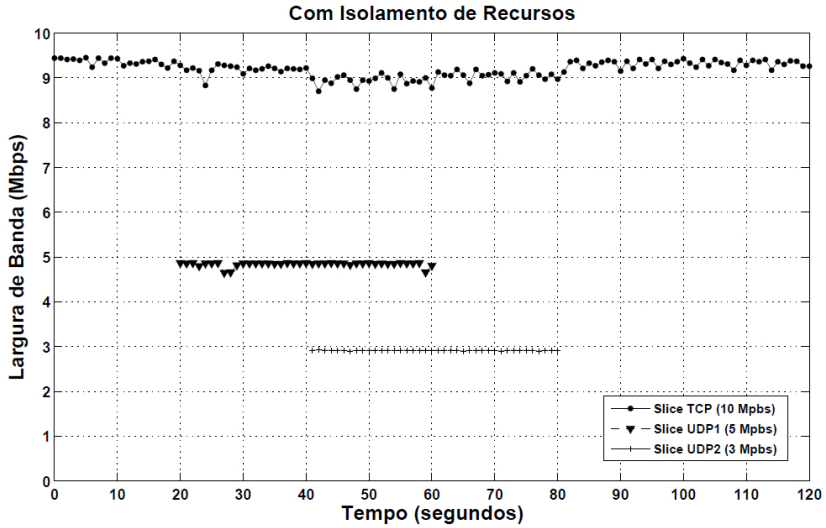


Figura 13: Resultados para o cenário B com isolamento de recursos habilitado

5.3.3 Resultados para os experimentos no Cenário C

Conforme apresentando anteriormente, o Cenário C foi proposto para verificação do comportamento da rede com a presença de um número ainda maior de *slices*. Neste caso, considerando as maiores interferências causadas por fluxos UDP, conforme verificado nos cenários anteriores, em todos os *slices* este tipo de tráfego foi adotado. Portanto, as disciplinas de filas e classes foram configuradas nos *datapaths* e atribuídas a cada um dos seis *slices* conforme descrito a seguir:

- Slice UDP (8011): atribuído à "Classe 1", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 1Mbps;
- Slice UDP (8012): atribuído à "Classe 2", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 1Mbps;

- Slice UDP (8013): atribuído à "Classe 3", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 2Mbps;
- Slice UDP (8014): atribuído à "Classe 4", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 2Mbps;
- Slice UDP (8015): atribuído à "Classe 5", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 3Mbps;
- Slice UDP (8016): atribuído à "Classe 6", configurada para utilizar o algoritmo de controle de tráfego HTB (*Hierachical Token Bucket*), com taxa de transmissão garantida de 3Mbps;

É importante destacar que, apesar das vantagens proporcionadas pelo Mininet, permitindo a execução de diversos elementos em um único sistema, algumas restrições são encontradas, dentre elas, destaca-se a limitação de recursos, uma vez que as capacidades de processamento e de rede são compartilhadas entre os *hosts* e *switchs* virtuais. Sendo assim, maiores taxas de largura de banda ocasionaram maior instabilidade ao experimento. Os resultados encontrados estão representados no gráfico da Figura 14. Apesar de pequenas interferências verificadas nos fluxos configurados com taxas superiores a 1mbps, o que pode ser ocasionado pelas limitações do Mininet ou da máquina virtual onde este foi instalado, ainda assim nenhum dos *slices* ultrapassou o limite de banda alocado para seus fluxos.

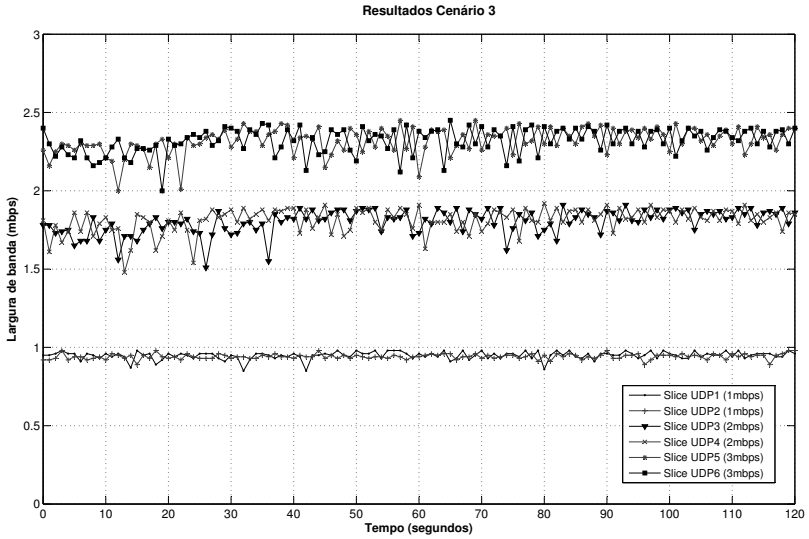


Figura 14: Resultados para o cenário C com isolamento de recursos habilitado

5.4 Conclusões do Capítulo

Os experimentos realizados para validação da solução permitiram a comparação do comportamento da rede, com e sem a aplicação dos mecanismos definidos para o isolamento de recursos. Com a definição de classes nos dispositivos QosFlow e sua atribuição aos *slices*, foi possível a limitação da largura de banda disponível a cada rede virtual.

Em um ambiente compartilhado, onde fluxos de *slices* distintos competem pelo uso da banda, o isolamento foi capaz de evitar que o desempenho de uma fatia fosse afetado por eventos nas demais redes virtuais.

A aplicação destes mecanismos sobre ambientes *OpenFlow* virtualizados ainda é um problema pouco explorado, sendo que não foram encontradas propostas semelhantes na literatura, o que impossibilita a definição de outros parâmetros de comparação quanto ao desempenho da solução.

CAPÍTULO 6

Conclusões

6.1 Conclusões

A Internet desempenha hoje um papel de extrema importância no contexto social, interconectando bilhões de pessoas e sustentando serviços cada vez mais heterogêneos. Porém, a rápida expansão da rede aliada às novas exigências apresentadas pelas aplicações, não previstas em seu projeto inicial, trouxe consigo a necessidade de contínuas adaptações, cada vez mais substanciais. O incremento sucessivo de soluções e protocolos na arquitetura original da rede acabou determinando a ocorrência de novos problemas e o engessamento do projeto atual, demonstrando sua incompatibilidade com as recentes e progressivas demandas.

Desta forma, pesquisas são realizadas com o intuito de desenvolver soluções e arquiteturas alternativas para a Internet, o que tem demandado da comunidade de pesquisa o investimento em iniciativas para implantação de métodos que garantam a experimentação de suas novas propostas.

Neste contexto, as Redes Definidas por *Software* baseadas em *OpenFlow* evidenciam-se como uma alternativa para habilitar o controle e programabilidade da rede, possibilitando a execução de experimentos diretamente sobre ambientes de produção, sem, no entanto interferir no tráfego original da rede.

A incorporação de virtualização nestes ambientes, através do *FlowVisor*, agrega funcionalidades para possibilitar a utilização simultânea da rede, que passa a compartilhar os recursos físicos entre diferentes redes virtuais. Esta solução, porém, ainda apresenta restrições para sua plena utilização, como a necessidade de novos métodos para mitigar as interferências entre fluxos de diferentes fatias de rede.

Portanto, este trabalho procurou dirimir uma das principais limitações do *Flow-*

Visor, relacionada a necessidade de mecanismos para o isolamento de recursos entre as diferentes redes virtuais definidas. Novos módulos foram desenvolvidos e implementados na ferramenta, habilitando o gerenciamento e alocação de recursos de QoS diretamente nos comutadores de rede. Para isto, realizou-se a integração com uma variante dos *datapaths OpenFlow* disponibilizada pelo arcabouço de *software* QosFlow.

Os experimentos de validação realizados possibilitaram a comprovação da funcionalidade da proposta quanto a mitigação de interferências entre fluxos de *slices* distintos, garantindo a limitação do uso de largura de banda em cada fatia.

6.2 Trabalhos Futuros

Outros trabalhos futuros envolverão a evolução da solução, incluindo: aplicação de mecanismos de controle de admissão; implementação de novos métodos para permitir a alocação de recursos entre os fluxos dentro de um mesmo *slice*; e possibilitar a sua aplicação em ambientes que implementem hierarquia de *FlowVisors*, através da troca de informações de configuração de filas entre estes controladores específicos e a definição de políticas para redistribuição dos recursos disponíveis à fluxos dentro de um mesmo *slice*. Sendo assim, objetiva-se a inclusão e aperfeiçoamento de diferentes aspectos de QoS no *FlowVisor*, garantindo-se um controle completo de qualidade de serviço diretamente pela ferramenta.

Referências

[Akari 2013] Akari (2013). Akari architecture design project for new generation network. Internet site: http://akari-project.nict.go.jp/eng/index2.htm. [Online; accessado Janeiro-2013].

[Al-Shabibi 2012] Al-Shabibi, A. (2012). Flowvisor 0.10.0 released. Internet site: https://mailman.stanford.edu/ pipermail/openflow-discuss/2012-December/003900.html. [Online; accessado Dezembro-2012].

[Cai et al. 2013] Cai, Z., Cox, A. L., and Ng, E. T. S. (2013). Maestro: A system for scalable openflow control. tech. rep. tr10-11. Technical report, Department of Computer Science: Rice University.

[Campista et al. 2010] Campista, M. E. M., Ferraz, L. H. G., Moraes, I. M., Lanza, M. L. D., Costa, L. H. M. K., and Duarte, O. C. M. B. (2010). Interconexão de redes na internet do futuro: Desafios e soluções. *Minicursos do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC*, pages 47–101.

[Correia et al. 2011] Correia, L. M., Abramowicz, H., Johnsson, M., and Wünstel, K. (2011). *Architecture and Design for the Future Internet*. Springer.

[El-azzab et al. 2011] El-azzab, M., Bedhiaf, I. L., Lemieux, Y., and Cherkaoui, O. (2011). Slices isolator for a virtualized openflow node. In *Proceedings of the 2011 First International Symposium on Network Cloud Computing and Applications*, NCCA '11, pages 121–126, Washington, DC, USA. IEEE Computer Society.

[Farias et al. 2011] Farias, F., Júnior, J., Salvatti, J., Silva, S., Abelém, A., Salvador, M., and Stanton, M. (2011). Pesquisa experimental para a internet do futuro: Uma proposta utilizando virtualização eo framework openflow. *Minicurso,Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC*, pages 1–61.

[Farias et al. 2012] Farias, F., Salvatti, J., and Abelém, A. (2012). Experimentação no futuro da internet: pesquisa utilizando virtualização e framework openflow. *V Congresso Internacional Software Livre e Governo Eletrônico - CONSEGI*.

[Fibre 2013] Fibre (2013). Future internet experimentation between brazil and europe. Internet site: http://www.fibre-ict.eu/. [Online; accessado Janeiro-2013].

[Fire 2013] Fire (2013). Future internet research and experimentation. Internet site: https://www.opennetworking.org/. [Online; accessado Janeiro-2013].

[Floodlight 2013] Floodlight (2013). Floodlight is an open sdn controller. Internet site: http://floodlight.openflowhub.org/. [Online; accessado Janeiro-2013].

[Geni 2013] Geni (2013). Exploring networks of the future. Internet site: http://www.geni.net. [Online; accessado Janeiro-2013].

[Greene 2009] Greene, K. (2009). Tr10: Software-defined networking. MIT Technology Review. Internet site: http://www.technologyreview.com/web/22120/. [Online; accessado Dezembro-2012].

[Gude et al. 2008] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110.

[Guedes et al. 2012] Guedes, D., Vieira, L., Vieira, M., Rodrigues, H., and Nunes, R. (2012). Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. *Minicurso,Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC*, pages 160–210.

[Iperf 2012] Iperf (2012). Internet site: http://iperf.sourceforge.net. [Online; accessado Dezembro-2012].

[Ishimori et al. 2012a] Ishimori, A., Farias, F., Carvalho, I., and Cerqueira, E.and Abelém, A. (2012a). Automatic qos management on openflow software-defined networks. *7th API Think-Tank - Software Defined Networking*.

[Ishimori et al. 2012b] Ishimori, A., Salvatti, J., farias, F. F., L., G., Granville, L., E., C., and G., A. A. J. (2012b). Qosflow: Gerenciamento automático da qualidade de serviço em infraestruturas de experimentação baseadas em framework openflow. *III Workshop de Pesquisa Experimental da Internet do Futuro/Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC/WPEIF*.

[ITU-D 2012] ITU-D (2012). International telecommunication union. measuring the information society. executive summary 2012. Internet site: http://www.itu.int. [Online; accessado Janeiro-2013].

[Kanada et al. 2012] Kanada, Y., Shiraishi, K., and Nakao, A. (2012). Network-resource isolation for virtualization nodes. In *Fourth International Conference on Communication Systems and Networks - COMSNETS*, pages 1–2.

[Kurose and Ross 2010] Kurose, J. F. and Ross, K. W. (2010). *Redes de Computadores e a Internet: uma abordagem top-down*. Addison Wesley, 5 edition.

[Lantz et al. 2010] Lantz, B., Heller, B., and McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. In *HotNets'10*, pages 19–19.

[McKeown et al. 2008] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, 38(2):69–74.

[Min et al. 2012] Min, S., Kim, S., Lee, J., Kim, B., Hong, W., and Kong, J. (2012). Implementation of an openflow network virtualization for multi-controller environment. In *14th International Conference on Advanced Communication Technology (ICACT), 2012*, pages 589–592.

[Moreira et al. 2009] Moreira, M., Fernandes, N., Costa, L., and Duarte, O. (2009). Internet do futuro: Um novo horizonte. *Minicurso, Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC*, pages 1–59.

[Nakao 2010] Nakao, A. (2010). Virtual node project: Virtualization technology for building new-generation networks. *NICT News No. 393*, pages 1–6.

[ONF 2012] ONF (2012). Software-defined networking:the new norm for networks. Technical report, Open Networking Foundation. [Online; accessado Janeiro-2013].

[Openflow 2009] Openflow (2009). Openflow switch specification, version 1.0.0 (wire protocol 0x01). Internet site: http://www.openflowswitch.org/documents/openflow-spec-v1.0.0.pdf. [Online; accessado Outubro-2012].

[OpenFlowJ 2012] OpenFlowJ (2012). Openflow java. Internet site: https://openflow.stanford.edu/fisheye/browse/OpenFlowJ. [Online; accessado Novembro-2012].

[Pan et al. 2011] Pan, J., Paul, S., and Jain, R. (2011). A survey of the research on future internet architectures. *Communications Magazine, IEEE*, 49(7):26–36.

[Paul et al. 2011] Paul, S., Pan, J., and Jain, R. (2011). Architectures for the future networks and the next generation internet: A survey. *Computer Communication*, 34(1):2–42.

[Sherwood et al. 2010a] Sherwood, R., Chan, M., Covington, A., Gibb, G., Flajslik, M., Handigol, N., Huang, T.-Y., Kazemian, P., Kobayashi, M., Naous, J., Seetharaman, S., Underhill, D., Yabe, T., Yap, K.-K., Yiakoumis, Y., Zeng, H., Appenzeller, G., Johari, R., McKeown, N., and Parulkar, G. (2010a). Carving research slices out of your production networks with openflow. *ACM SIGCOMM Computer Communication Review*, 40(1):129–130.

[Sherwood et al. 2009] Sherwood, R., Gibb, G., Yap, K.-k., Appenzeller, G., Casado, M., Mckeown, N., and Parulkar, G. (2009). Flowvisor: A network virtualization layer flowvisor. *OpenFlow Switch*, page 15.

[Sherwood et al. 2010b] Sherwood, R., Gibb, G., Yap, K.-K., Appenzeller, G., Casado, M., McKeown, N., and Parulkar, G. (2010b). Can the production network be the testbed? In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA. USENIX Association.

[Silva et al. 2012] Silva, F., Pereira, J., Rosa, P., and Kofuji, S. (2012). Enabling future internet architecture research and experimentation by using software defined networking. In *European Workshop on Software Defined Networking (EWSDN), 2012*, pages 73–78.

[Skoldstrom and Yedavalli 2012] Skoldstrom, P. and Yedavalli, K. (2012). Network virtualization and resource allocation in openflow-based wide area networks. In *Communications (ICC), 2012 IEEE International Conference on*, pages 6622–6626.

[Takahashi 2012] Takahashi, M. (2012). Design documents: Enqueue action support. Internet site: https://openflow.stanford.edu/display/DOCS/ENQUEUE+action+support. [Online; accessado Setembro-2012].

APÊNDICE A – Fluxograma

A Figura 15 apresenta uma visão geral do fluxo de atividades executadas pelo *FlowVisor*, quando do recebimento de mensagens *PacketIn* a partir dos *datapaths*, incluindo o repasse ao controlador e a conversão das ações nas mensagens antes de retornarem ao comutador. Algumas etapas foram incluídas com base nos diagramas encontrados na documentação da atual arquitetura do *FlowVisor*, sendo que, algumas etapas foram suprimidas, fornecendo assim uma visão geral e simplificada do diagrama.

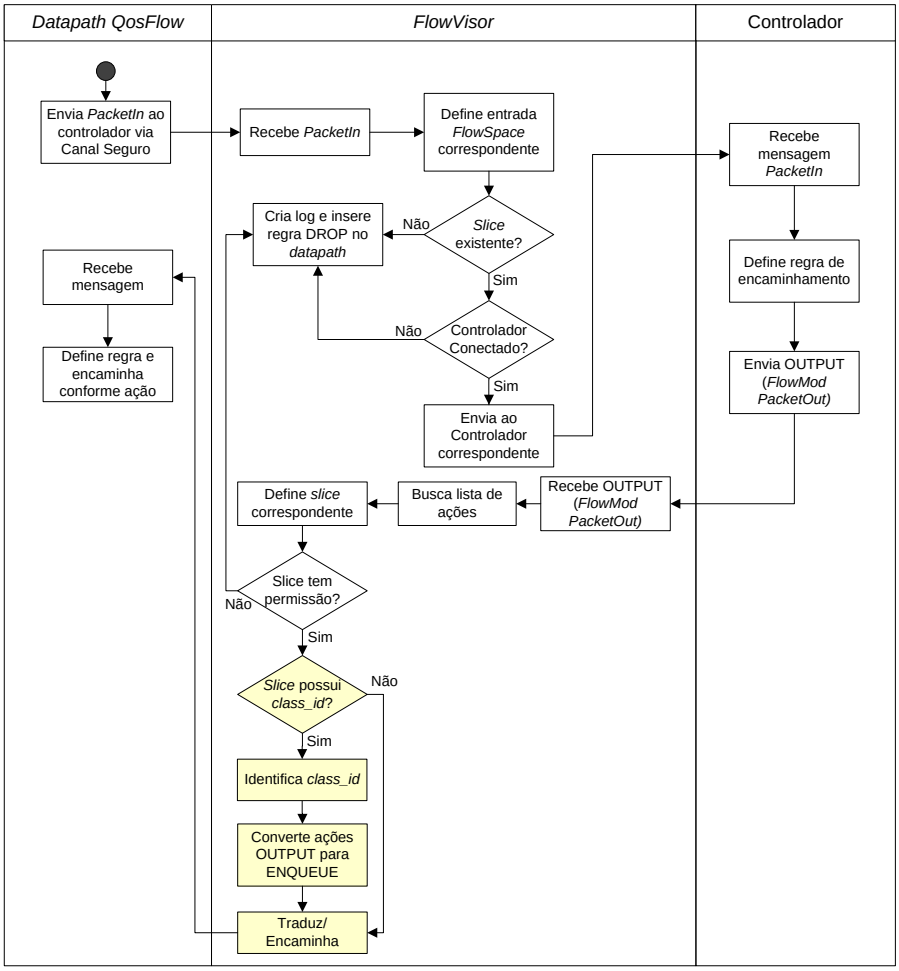


Figura 15: Fluxograma de atividades incluindo conversão de mensagens $OUTPUT$

APÊNDICE B – Diagramas de Classe

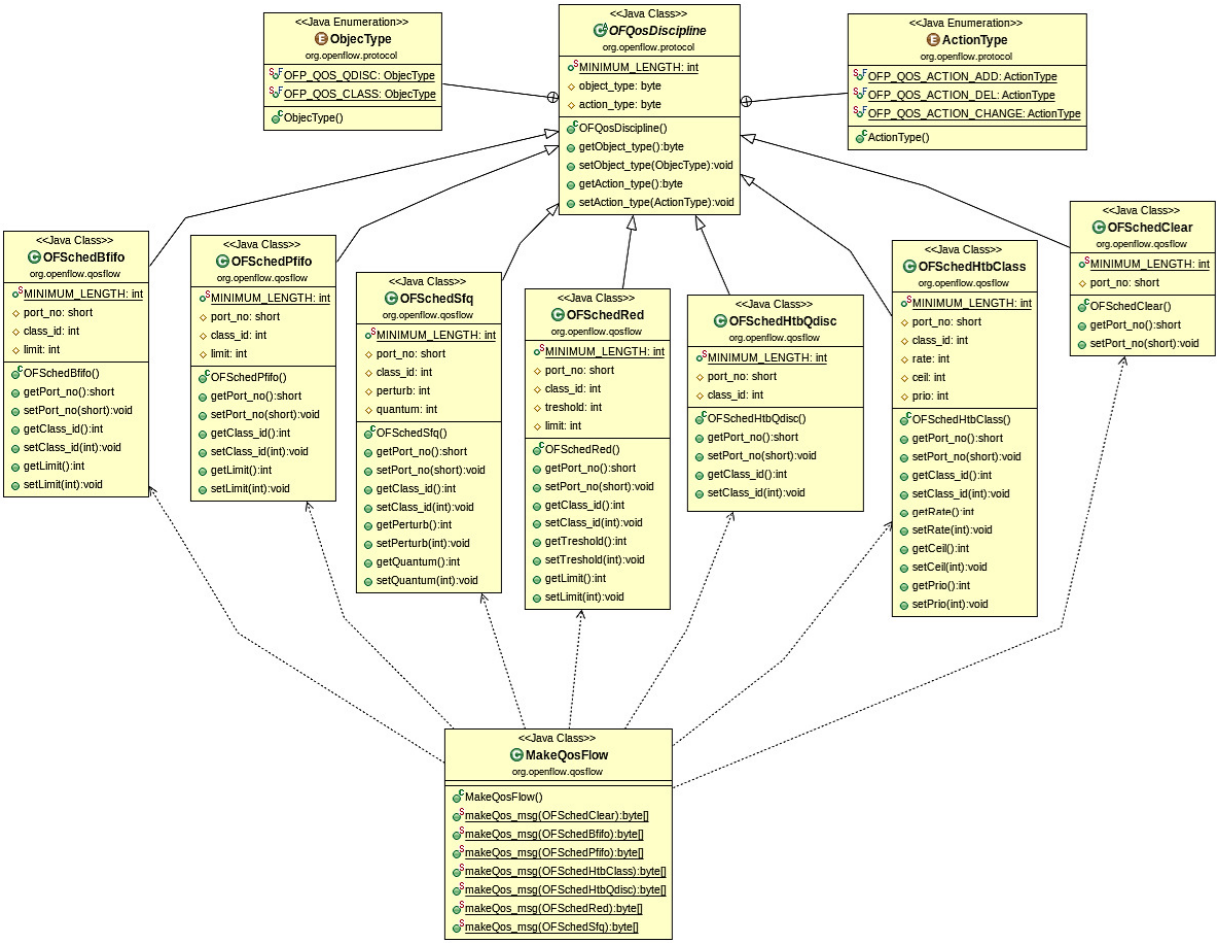


Figura 16: Diagrama de Classes com alterações implementadas no pacote OpenFlowj - Parte 1

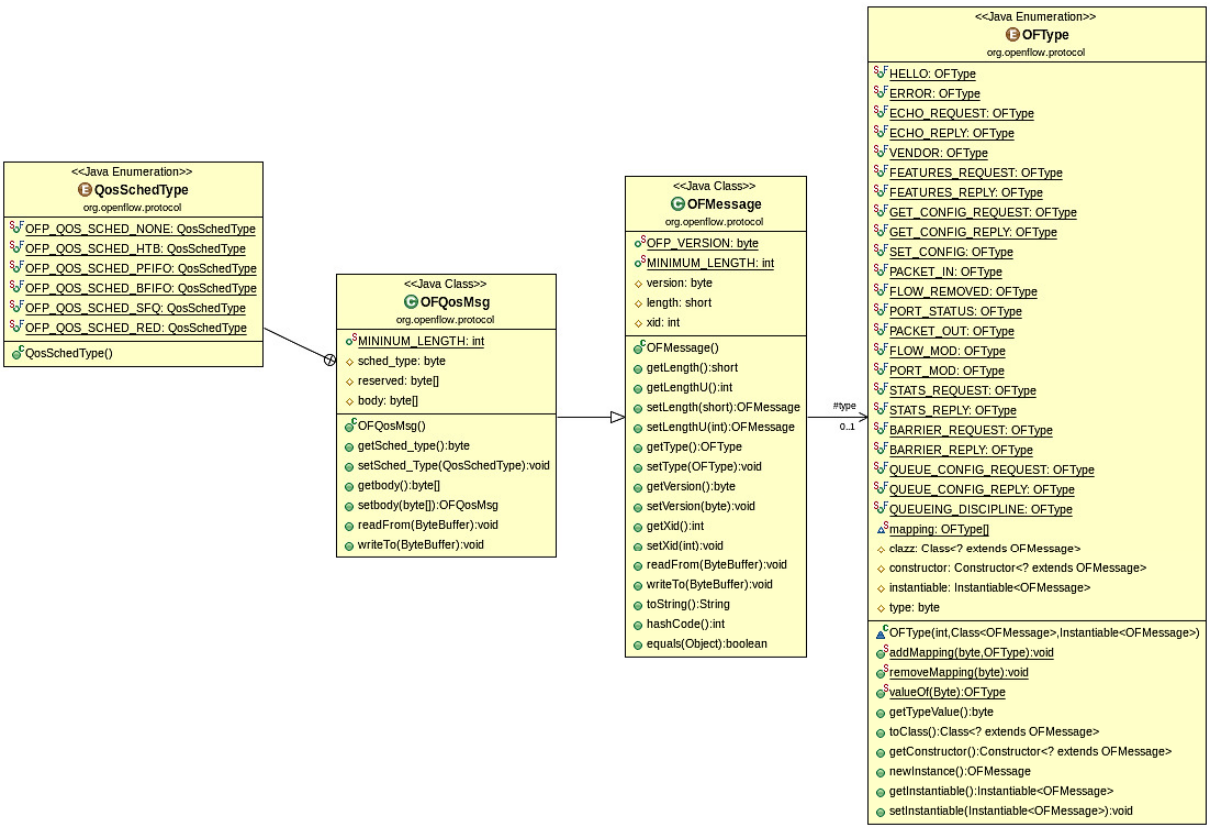


Figura 17: Diagrama de Classes com alterações implementadas no pacote OpenFlowj - Parte 2