UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UM ESTUDO SOBRE A APLICAÇÃO DE UMA REDE DE CIRCUITOS DINÂMICOS EM DOMÍNIOS OPENFLOW

Daniel de Arêa Leão Marques

Orientador:
Sidney Cunha de Lucena

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2012

UM ESTUDO SOBRE A APLICAÇÃO DE UMA REDE DE CIRCUITOS DINÂMICOS EM DOMÍNIOS OPENFLOW

Daniel de Arêa Leão Marques

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

Sidney Cunha de Lucena, D.Sc. – UNIRIO

Carlos Alberto Vieira Campos, D.Sc. – UNIRIO

Marcel William Rocha da Silva, D.Sc. – UFRJ

RIO DE JANEIRO, RJ – BRASIL

SETEMBRO DE 2012

Marques, Daniel de Arêa Leão.

M357 Um estudo sobre a aplicação de uma rede de circuitos dinâmicos em domínios openflow / Daniel de Arêa Leão Marquês , 2012.

99f. ; 30 cm

Orientador: Sidney Cunha de Lucena.

Dissertação (Mestrado em Informática) – Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2012.

À minha mãe Nadia e o meu pai Reinaldo,
por seu apoio incondicional.

Agradecimentos

Dedico este trabalho primeiramente à Deus, por me conceder determinação e força inabaláveis.

Aos meus pais, Reinaldo e Nadia, e meus irmãos, Tatiana e Luiz Felipe, pelo apoio e dedicação incondicional durante todos estes anos.

Ao Sidney, primeiro professor da primeira aula da graduação, orientador do projeto final de graduação e desta trabalho de dissertação , acima de tudo, meu amigo. Sidney, sem o seu apoio e o seu braço amigo, este trabalho jamais teria sido concluído.

Ao Professor Carlos Alberto, amigo que me apoiou e se preocupou em todos os momentos. Foi uma enorme honra ter sido seu aluno.

Aos professores do DIA/PPGI, com quem convivi durante seis anos e tive o prazer de participar de suas aulas.

À equipe da secretaria, que sempre me ajudou e teve muita paciência com os meus pedidos.

Aos meus colegas de mestrado, Carlos, Pablo, Anna e Thiago pelo apoio tanto técnico, nas horas de aperto, como por sua fiel amizade nos momentos mais críticos.

Aos meus colegas de vida, Thiago, Paulo, Henrique, Jonas e Larissa. Obrigado pelo apoio e por sua amizade durante todos esses anos.

"Do not pray for an easy life, pray for the strength to endure a difficult one."
Bruce Lee

Marques, Daniel de Arêa Leão. **Um Estudo sobre a Aplicação de uma Rede de Circuitos Dinâmicos em Domínios OpenFlow.** UNIRIO, 2012. 99 páginas. Dissertação de Mestrado Departamento de Informática Aplicada. UNIRIO.

Resumo

O OpenFlow é uma tecnologia que abstrai o plano de controle do plano de dados e permite que este plano se torne uma interface programática, assim ditando o comportamento da rede através de aplicações. Uma rede de circuitos dinâmicos é uma arquitetura de redes que permite agendar circuitos virtuais com banda garantida. Estas duas tecnologias foram recentemente integradas em caráter experimental, já que o OpenFlow está se tornando um novo paradigma na área de redes e a rede de circuitos dinâmicos necessita englobar novas tecnologias. No entanto, até a completude desta dissertação não houve um estudo extensivo avaliando os resultados desta integração, assim como não há estudos sobre as melhores opções de ferramentas e arquiteturas para esta integração, provocando uma carência de respostas a questões de desempenho e flexibilidade. Logo, como objetivo para este trabalho, é proposto um estudo aprofundado fazendo um levantamento dos possíveis mecanismos a serem utilizados para provisionar circuitos sobre uma rede OpenFlow que, de fato, garantam isolamento, banda reservada e policiamento da banda negociada. Os resultados deste estudo apontaram para o uso do *framework* QoSFlow como forma de implementar as soluções de QoS num domínio OpenFlow. Além disso, foi possível verificar limitações e validar funcionalidades de um conjunto de ferramentas utilizadas para a emulação de ambientes virtuais de teste e *softwares* que implementam *switches* OpenFlow. Por fim, é detalhado como deve ser implementada a integração da ferramenta de DCN OSCARS 0.6 e o *framework* QoSFlow.

Palavras-chave: Internet do Futuro, Redes Híbridas, Redes Definidas por Software, Qualidade de Serviço, Redes de Circuitos Dinâmicos

Abstract

OpenFlow is a technology which separates the data plane from the control plane and allows the latter one to become a programmatic interface, in this way dictacting the behaviour through the use of applications. Dynamic Circuit Network is an architecture that permits to schedule virtual circuits with guaranteed bandwidth. Both technologies were recently integrated in a experimental way, since OpenFlow is becoming a new network paradigm and the Dynamic Circuit Network needs to integrate new technologies. However, up to the completion of this dissertation there was no extensive research which evaluated the results of this integration, so there are no studies about which are the best options of tools and architectures for this integration, causing a need of answers about questions of performance and flexibility. Therefore, the objective of this work is to propose an extensive study in which is made a survey about the possible tools which can be used to provision circuits over an OpenFlow network, that in fact guarantee isolation, reserved bandwidth and policy of the negotiated bandwidth. Results of this study pointed out for the use of QoSFlow framework as a way to implement the QoS solutions in OpenFlow domain. Besides that, it was possible to verify the limitations and validate the functionalities of a set of tool used for the emulation of virtual testbeds and for implement OpenFlow software switches. Finally, it is detailed how should be implemented an integration of the DCN's tool OSCAR version 0.6 with the QoSFlow framework.

Keywords: Future Internet, Hybrid Network, Software Defined Network, Quality of Service, Dynamic Circuit Network

Lista de Figuras

1 Rede de Circuitos global. 5

2 Plano de Controle e Plano de Dados segregados. 6

3 Arquitetura DCN segundo (VOLLBRECHT; CASHMAN; LAKE, 2008). 7

4 Algoritmo do Token Bucket (H3C, 2012). 12

5 Exemplificação da estrutura hierárquica do HTB, extraída de (MOTA, 2012). . . 13

6 Algoritmo do WFQ, extraída de (H3C, 2012). 14

7 Visão da interação dos componentes do OSCARS (ESNET, 2012c). 16

8 *Workflow* entre os componentes do OSCARS (OSCARS, 2012) 17

9 Diagrama de estados de criação de agendamento (ESNET, 2012b). 18

10 Diagrama de estados de reserva (DICE, 2010). 19

11 Arquitetura completa do DRAGON (LEHMAN; SOBIESKI; JABBARI, 2006). . . . 20

12 Arquitetura de Redes Definidas por Software (FOUNDATION, 2012b). 21

13 Arquitetura do switch OpenFlow baseada em (CONSORTIUM, 2009b). 22

14 Arquitetura do QoSFlow (ISHIMORI et al., 2012). 24

15 Arquitetura OSCARS com OpenFlow, baseada no documento (INTERNET2, 2011). 29

16 Arquitetura OSCARS com QoSFlow. 30

17 Arquitetura utilizada no experimento de funcionalidades de QoS. 46

18 Resultado referente ao policiamento de banda - Vazão de banda do *software switch* referência com banda exigida 5 Mbps usando UDP. 47

19 Resultado referente ao policiamento de banda - Vazão de banda do OVS exigida 5 Mbps usando UDP. 48

20 Resultado referente ao policiamento de banda - Vazão de banda do *software switch* referência exigida 5 Mbps usando TCP/UDP. 48

21 Resultado referente ao policiamento de banda - Vazão de banda do OVS exigida 5 Mbps usando TCP/UDP. 49

22 Resultado referente ao policiamento de banda - Vazão de banda do *software switch* referência exigida 3 Mbps usando UDP. 49

23 Resultado referente ao policiamento de banda - Vazão de banda do OVS exigida 3 Mbps usando UDP. 50

24 Resultado referente ao policiamento de banda - Vazão de banda do *software switch* referência exigida 3 Mbps usando TCP/UDP. 50

25 Resultado referente ao policiamento de banda - Vazão de banda do OVS exigida 3 Mbps usando TCP/UDP. 51

26 Resultado referente à banda mínima do *switch* referência. 52

27 Resultado referente à banda mínima do OVS. 53

28 Arquitetura utilizada no experimento de performance. 55

29 Arquitetura referente a segunda etapa do experimento de performance. 56

30 Arquitetura utilizada no experimento 3. 58

31 Teste de vazão realizado no experimento 3 com bandas requeridas de 3 Mbps e 1 Mbps. 60

32 Teste de vazão realizado no experimento 3 com bandas requeridas de 3 Mbps e 1 Mbps. 60

33 Integração entre as soluções OSCARS e QoSFlow. 64

Lista de Tabelas

1 Campos de um fluxo (CONSORTIUM, 2009b). 23

2 Campos dos pacotes utilizados para correspôndencia nos fluxos (CONSORTIUM, 2009b). 23

3 Componentes de uma entrada na *Meter Table*. 32

4 Comparação qualitativa entre as três arquiteturas propostas. 34

5 Média da vazão máxima da primeira avaliação do experimento. 57

6 Média da vazão máxima (em Mbps) da segunda avaliação do experimento 2. . . 57

7 CPU e Memória da segunda avaliação do Experimento 2. 58

Lista de Siglas

API	Application Programming Interface
CERN	The European Organization for Nuclear Research
CPqD	Centro de Pesquisa e Desenvolvimento em Telecomunicações
DANTE	Delivery of Advanced Network Technology to Europe
DC	Domain Controller
DCN	Dynamic Circuit Network
DRAGON	Dynamic Resource Allocation via GMPLS Optical Networks
ESNet	Energy Sciences Network
GMPLS	Generalized Multiprotocol Label Switching
HTB	Hierarchical Token Bucket
IDCP	Inter-domain Controller Protocol
IP	Internet Protocol
IDC	Inter-domain Controller
LDP	Label Distribution Protocol
LSA	Link-state Advertisement
LSP	Label Switched Path
LSR	Label Switching Router

LXC	Linux Containers
MAC	Media Access Control
MPLS	Multiprotocol label switching
NORDUnet	Nordic Infrastructure for Research & Education
OSPF	Open Shortest Path First
ONF	Open Network Foundation
OVS	Open vSwitch
PQ	Priority Queuing
QoS	Quality of Service
RAM	Random Access Memory
RNP	Rede Nacional de Ensino e Pesquisa
RSVP	Resource reservation Protocol
RFC	Request for Comments
RSVP	Resource Reservation Protocol
SDN	Software Defined Networking
SOR	Sistema Operacional de Redes
TB	Token Bucket
TE	Traffic Engineering
VLAN	Virtual Local Area Network
WFQ	Weighted fair queuing

Sumário

1 Introdução **1**

1.1 Considerações iniciais 1

1.2 Motivação, Objetivos e Contribuições 3

1.3 Estrutura do texto da dissertação 4

2 Revisão bibliográfica **5**

2.1 Redes de Circuitos Dinâmicos 5

2.1.1 Protocolos para Uso em DCN 8

2.1.1.0.1 MPLS - Multiprotocol label switching 8

2.1.1.0.2 RSVP - Resource Reservation Protocol 9

2.1.1.0.3 GMPLS - Generalized Multiprotocol Label Switching 9

2.1.1.0.4 OSPF-TE 10

2.1.2 Qualidade de Serviço 10

2.1.2.1 Escalonadores 11

2.1.2.1.1 Token Bucket 12

2.1.2.1.2 Hierarchical Token Bucket 12

2.1.2.1.3 Priority Queuing 13

2.1.2.1.4 Weighted Fair Queuing 13

2.1.2.2 Políticas de Tráfego 14
2.1.2.2.1 Best Effort 14
2.1.2.2.2 Expedicted Forward 14
2.1.2.2.3 Assured-Forward 15
2.1.2.2.4 Scavenger 15
2.1.3 Soluções de software para DCNs 15
2.1.3.1 OSCARS - On-demand Secure Circuits and Advance Reservation System 15
2.1.3.2 DRAGON: Dynamic Resource Allocation in GMPLS Optical Networks 19
2.2 Redes Definidas por Software 20
2.2.1 OpenFlow 21
2.2.2 QoSFlow 24
2.2.3 Controladores 25
2.2.4 Switches OpenFlow 26

3 Uso de DCN sobre Redes SDN **28**

3.1 Arquiteturas 28
3.1.1 Arquitetura básica 29
3.1.2 Arquitetura com QoSFlow 30
3.1.3 Arquitetura com QoSFlow utilizando OpenFlow 1.3 31
3.1.4 Análise qualitativa das arquiteturas 33
3.2 Ferramentas OpenFlow para Implementação das Arquiteturas DCN sobre SDN 34

3.2.1 Switch Referência OpenFlow 34
3.2.2 Switch QoSFlow 35
3.2.3 Open vSwitch 36
3.2.4 NOX-Classic 37
3.2.5 NOX QoSFlow 37
3.2.6 Mininet 37
3.2.7 Virtualizador LXC 38
3.2.8 Virtualização de enlaces utilizando veth 38
3.2.9 Switch OpenFlow em Hardware 39
3.3 Comparativo entre as Opções de Ferramentas 40
3.3.1 Switches OpenFlow 40
3.3.2 Ambientes 41

4 Análise Experimental de DCN sobre um domínio OpenFlow 42

4.1 Abordagens utilizadas 42
4.1.1 Metodologia experimental 43
4.2 Experimentos relativos ao suporte de QoS 43
4.2.1 Experimento - Funcionalidades de QoS 45
4.2.1.1 Objetivo 45
4.2.1.2 Topologia e Configuração 46
4.2.1.3 Policiamento de banda 47
4.2.1.4 Garantia de banda mínima 52
4.2.1.5 Resultados 54

4.2.2 Experimento - Performance 55
4.2.2.1 Objetivo 55
4.2.2.2 Topologia e Configuração 55
4.2.2.3 Resultados 57
4.2.3 Experimento - Funcionalidades do QoSFlow 58
4.2.3.1 Objetivo 58
4.2.3.2 Topologia e Configuração 58
4.2.3.3 Resultados 59
4.2.4 Limitações dos dispositivos e cenários de teste 61
4.3 Integração entre as soluções OSCARS e QoSFlow 62

5 Conclusão **65**

5.1 Contribuições 66
5.2 Limitações 67
5.3 Trabalhos futuros 67

Anexo A – Códigos da aplicação para o NOX **68**

Anexo B – Código JSON referente às mensagens do OSCARS **76**

Referências **79**

1 Introdução

1.1 Considerações iniciais

A comunidade científica hoje em dia está cada vez mais realizando projetos de grande escala colaborativos, conforme notado em (GUOK et al., 2008) e (LEHMAN; SOBIESKI; JABBARI, 2006). Estas cooperações são extremamente distribuídas, ou seja, estão espalhadas com os seus recursos computacionais ao redor do planeta, e dependem de redes de alta performance para que os seus resultados se integrem, e consequentemente alcancem sucesso. Destes projetos colaborativos, o mais célebre de todos é o LHC (*Large Hadron Collider*) (COLLIDER, 2012), que se encontra no CERN (CERN, 2012). No entanto, segundo (LEHMAN; SOBIESKI; JABBARI, 2006) e (GUOK et al., 2008), o requerimento de serviços de rede de tais pesquisas está além do que a tradicional infraestrutura de *best effort*[1] é capaz de oferecer. Mesmo com a sua evolução ao longo dos anos, a infraestrutura tradicional ainda não oferece previsibilidade, flexibilidade e garantias.

Para solucionar o problema de falta de garantia previamente apresentado, foi desenvolvida uma nova solução de arquitetura de redes, denominada de Redes de Circuitos Dinâmicos (termo também conhecido como *Dynamic Circuit Network* ou DCN) (INTERNET2, 2012a). Esta arquitetura foi proposta por um conjunto de organizações, envolvendo a ESnet (ESNET, 2012a), a Internet2 (INTERNET2, 2012a) e a Dante (DANTE, 2012). A DCN da ESnet, por exemplo, é composta por uma infraestrutura de rede chamada *Science Data Network* (GUOK et al., 2008), que é uma rede separada do núcleo da rede IP de produção da ESnet e possui um conjunto de

[1] O *best effort* é um serviço prestado pelo protocolo IP, que busca encaminhar o datagrama o mais rápido possivel, no entanto este não oferce garantias de entrega, atraso ou variação de atraso.

políticas de qualidade de serviço específico, e um conjunto de sistemas que torna esta rede um recurso gerenciável para estas exigentes aplicações científicas.

Através dos sistemas da DCN é possível agendar o provisionamento automático de circuitos que oferecem banda garantida. A operação de provisionamento ocorre no plano de controle, que realizará o cálculo de rotas para que a banda exigida pelo usuário seja assegurada. Após esta etapa as configurações obtidas serão encaminhadas para os equipamentos, que se encontram no plano de dados.

Outro conceito que segrega o plano de dados do plano de controle, são as Redes Definidas por *Software* (FOUNDATION, 2012b) (ou *Software Defined Network*, SDN). Neste conceito o plano de controle se torna uma interface programática, e por ele é possível ditar como será o comportamento do plano de dados, desta forma simplificando e otimizando o processo de configuração de uma rede.

O maior expoente do conceito SDN, é o OpenFlow (MCKEOWN et al., 2008). Esta tecnologia surgiu da inviabilidade dos pesquisadores realizarem os seus experimentos de rede utilizando ambientes reais, já que a infraestrutura de redes tem um papel crítico no dia-a-dia de empresas e universidades, assim impedindo que novos protocolos e idéias sejam testados. No entanto, através do OpenFlow é possível que estes cientistas controlem a tabela de fluxo dos equipamentos da rede, assim podendo segregar os tráfegos de produção dos tráfegos relacionados a pesquisa.

O OpenFlow administra o plano de controle da rede através de um sistema operacional de redes (SOR). Neste SOR é possível criar aplicações que controlem o plano de dados de um domínio OpenFlow, já que através deste protocolo é possível configurar a tabela de fluxo do insumos que utilizam o OpenFlow.

A DCN e o OpenFlow, são alguns dos atuais expoentes do estado da arte das tecnologias para provimento de conectividade: o primeiro tem uma grande adesão em redes acadêmicas e organizações, como ESNet (ESNET, 2012a), NorduNet (NORDUNET, 2012), GÉANT (GEANT, 2012) e RNP (RNP, 2012), e o segundo apresenta cada vez mais trabalhos sedimentados em sua arquitetura, que dos quais destacam-se o FlowVisor (SHERWOOD et al., 2009) e o RouteFlow

(NASCIMENTO et al., 2011a).

1.2 Motivação, Objetivos e Contribuições

Recentemente foi apresentada em (BOOTE et al., 2011) a integração destas duas tecnologias (DCN e SDN) e, apesar da união ter sido concretizada, a construção de circuitos dinâmicos de forma eficiente e satisfatória utilizando a arquitetura DCN sobre uma rede OpenFlow não foi devidamente estudada, provocando uma carência de respostas às perguntas relacionadas a: desempenho (Qual a vazão máxima que os equipamentos OpenFlow conseguem alcançar? Qual seu consumo de memória e processamento?) e flexibilidade (Quais funcionalidade e tecnologias estes insumos suportam?). E como a questão não foi completamente abordada, logo não há um consenso de como estabelecer uma rede de circuitos dinâmicos (tecnologia usada nas grandes rede academicas) sobre um domínio OpenFlow (que está se tornando um novo paradigma de redes).

Os métodos atualmente propostos para se provisionar circuitos em redes OpenFlow não consideram todos os aspectos de QoS necessários para que isto seja realizado de forma satisfatória. Por outro lado também, os mecanismos existentes para dotar uma rede OpenFlow de funcionalidades como suporte a QoS não estão completamente avaliados e validados.

Como objetivo para este trabalho, é proposto um estudo aprofundado fazendo um levantamento dos possíveis mecanismos a serem utilizados para provisionar circuitos sobre uma rede OpenFlow que de fato garantam isolamento, banda reservada e policiamento da banda negociada (Qualidade de Serviço). Além das contribuições do estudo realizado, foi proposta uma nova arquitetura integrando o DCN com SDN, porém com ênfase no gerenciamento de QoS, onde foi realizada uma avaliação das funcionalidades e levantados os detalhes desta integração, e por fim a criação de uma ferramenta que auxilia na criação de circuitos dinâmicos baseados em VLAN.

1.3 Estrutura do texto da dissertação

Esta dissertação é dividida em cinco capítulos. No Capítulo 2 é feita uma revisão bibliográfica, onde são apresentados os conceitos de DCN e SDN, com as suas respectivas ferramentas e protocolos utilizados pelos paradigmas.

O Capítulo 3 são apresentadas novas formas de integração entre o OpenFlow e DCN, e as ferramentas disponíveis para os experimentos são descritas.

Capítulo 4 são realizados os experimentos de funcionalidade de QoS, desempenho em ambientes virtualizados, uma avaliação experimental do *switch* QoSFlow e o detalhamento de como deve ser implementada a solução proposta.

No Capítulo 5 é feita uma conclusão para este trabalho, onde são apresentadas as contribuições, limitações e trabalhos futuros.

2 *Revisão bibliográfica*

Neste capítulo serão abordados os principais conceitos utilizados neste trabalho e também serão apresentadas as ferramentas mais relevantes de cada área envolvida para a construção desta dissertação.

Sobre a organização deste capítulo: a primeira seção abordará os conceitos, protocolos e principais ferramentas que estão relacionadas com Redes de Circuitos Dinâmicos; A segunda seção abordará Redes Definidas por Software, onde serão apresentados seus conceitos, o protocolo OpenFlow e a sua arquitetura, e ferramentas.

2.1 Redes de Circuitos Dinâmicos

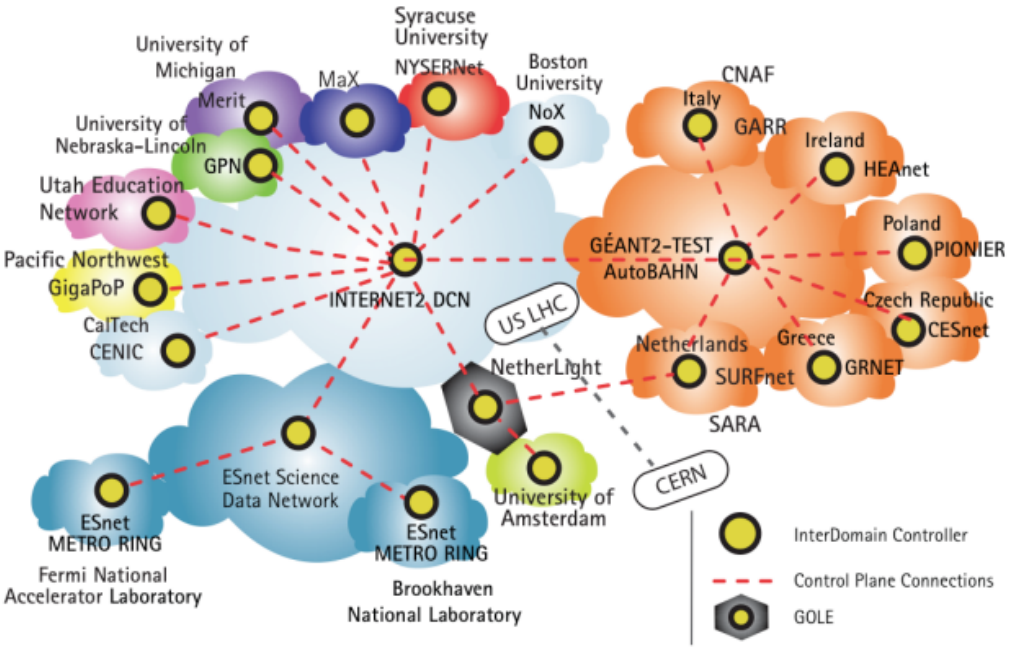

Figura 1: Rede de Circuitos global.

Uma Rede de Circuitos Dinâmicos[1], é uma rede óptica que fornece banda dedicada para as aplicações exigentes (INTERNET2, 2012a), como videoconferências em alta definição e apoio no processamento de grandes quantidades de dados. Nesta rede existe um sistema que cria circuitos *peer-to-peer* onde o usuário final ou aplicação enviam requisições por conexões. A Figura 1 apresenta uma visão de como era a rede de circuitos global no ano de 2008, demonstrando a rede da Internet2 (INTERNET2 DCN) conectando as redes acadêmicas de algumas universidades e organizações americanas, com redes de outros países como Itália e Polônia via GÉANT2-TEST, e outras instituições como *Fermi National Accelerator Laboratory* através da *Science Data Network* da ESnet.

Segundo (LEHMAN; SOBIESKI; JABBARI, 2006), a motivação para o uso desta tecnologia vem da necessidade das pesquisas em grande escala, que atualmente são colaborações multidisciplinares distribuídas e que dependem de redes de alta performance, pois o modelo de *best-effort*, atualmente vigente nas redes do dia-a-dia, apresenta falta de previsibilidade e flexibilidade, e não oferece quaisquer garantias.

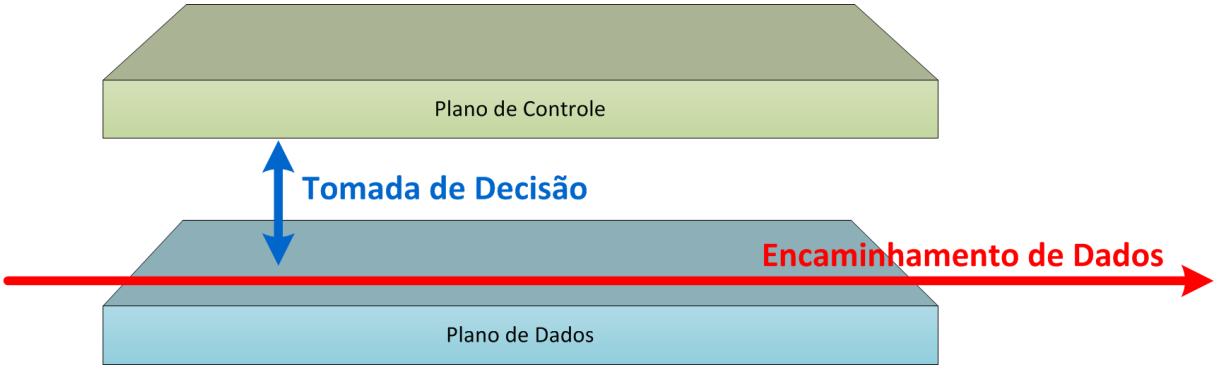

Figura 2: Plano de Controle e Plano de Dados segregados.

Na arquitetura de uma rede circuitos dinâmicos, o plano de controle é segregado do plano de dados. No plano de controle, são realizados os cálculos de rotas para que os circuitos criados forneçam a banda exigida pelos usuários e aplicações, e o plano de dados comuta os dados de acordo com as decisões de roteamento do primeiro plano. A figura 2 faz uma representação do conceito descrito demonstrando os planos de dados e de controle separados, onde o primeiro apenas encaminha os dados de acordo com as decisões do plano de controle.

[1]Redes de Circuitos Dinâmicos pode ser também conhecida como Dynamic Circuit Network (DCN).

Para que os circuitos sejam criados em uma rede DCN, o usuário agenda uma requisição com o Plano de Controle da rede, que, conforme a disponibilidade dos recursos permitirá o agendamento do pedido ou irá indeferi-lo.

O Plano de Controle do DCN se subdivide em duas partes lógicas: DC (*Domain Controller*) e IDC (*Inter-domain Controller*). O primeiro tem atuação restrita a um dado domínio e é responsável por gerenciar os recursos locais e por provisionar e cancelar as conexões que passam pelo respectivo domínio. Já o segundo aceita as requisições de criação de circuitos, calcula o caminho e coordena estas requisições com outros domínios (VOLLBRECHT; CASHMAN; LAKE, 2008), conforme o caminho.

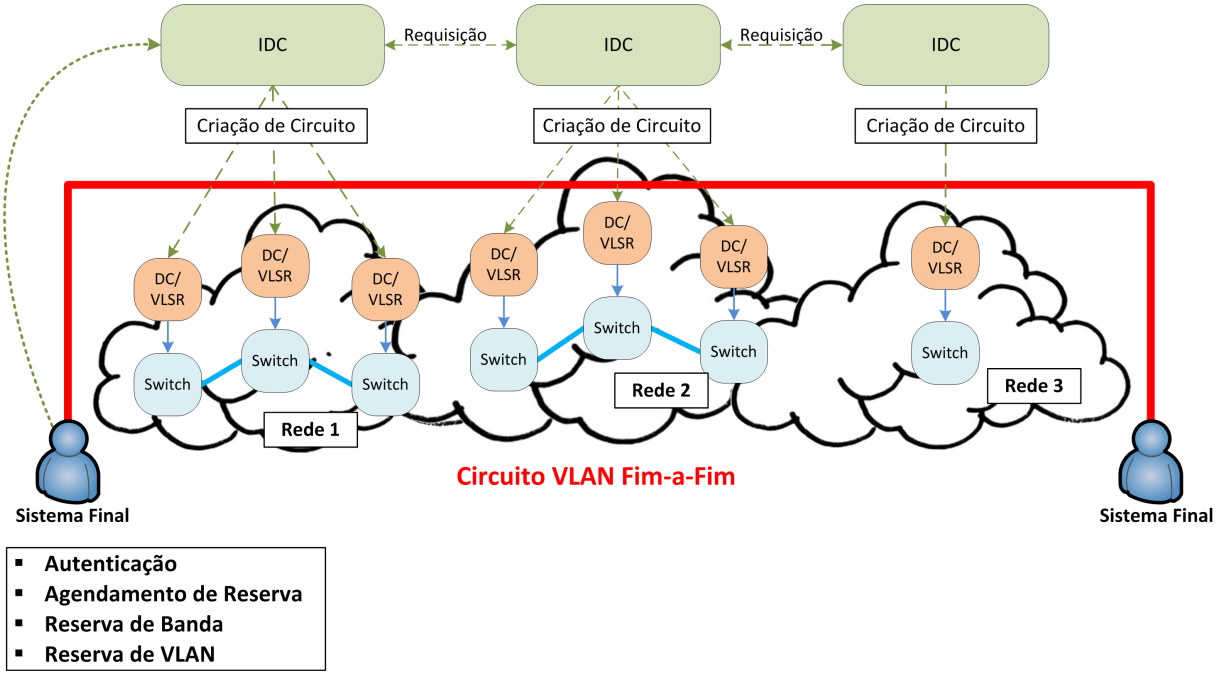

Figura 3: Arquitetura DCN segundo (VOLLBRECHT; CASHMAN; LAKE, 2008).

Utilizando o exemplo da Figura 3, um usuário de uma determinada rede que deseja criar um circuito em um dado período irá realizar uma requisição para o seu IDC, que avaliará as condições da rede (por exemplo: agendamento de outros circuitos e banda exigida) no período desejado para determinar se a reserva será aceita ou se será indeferida. Caso seja aceito o pedido, o IDC irá notificar os outros *Interdomain Controllers* da rede sobre o provisionamento do circuito, para que, assim, no momento de criação possam solicitar aos seus DCs a efetivação da conexão requisitada, criando então um circuito ponto-a-ponto entre os elementos finais que tro-

carão dados. Ao final do período reservado para o circuito, os IDCs solicitarão aos respectivos DCs a remoção do circuito.

2.1.1 Ferramentas da DCN

Numa rede de circuitos dinâmicos, a única tarefa que um usuário deve fazer é a reserva de banda. Desta forma, o cliente não precisa configurar uma rota, ou realizar a marcação dos pacotes. Todos os mecanismos necessários para prover a garantia de banda e o estabelecimento/remoção dos circuitos são coordenados pelo sistema gerenciador da DCN (GUOK, 2006).

Para que o sistema gerenciador da DCN possa realizar as suas funções é necessário um conjunto de protocolos de encaminhamento e roteamento, para garantia de banda e criação dos circuitos, uma rede que ofereça a qualidade de serviço exigida com os devidos mecanismos de enfileiramento, e um conjunto de *softwares* que atuem no inter-domínio e no intra-domínio.

Os protocolos de roteamento e encaminhamento têm por objetivo, numa DCN, criar os circuitos dos usuários de maneira que a sua banda mínima exigida seja garantida. Os protocolos mais relevantes são o GMPLS, MPLS, RSVP e OSPF-TE.

2.1.1.0.1 MPLS - Multiprotocol label switching

Segundo Farrel (2004), o MPLS (*Multiprotocol label switching*) (ROSEN; VISWANATHAN; CALLON, 2001) é um protocolo que comuta o seus pacotes através da verificação de etiquetas. O encaminhamento ocorre da seguinte forma: os pacotes que participam de uma rede MPLS possuem etiquetas atreladas (*label*), e ao passarem por um nó, este, irá ler o rótulo do pacote e consultará uma tabela para determinar qual será o próximo salto para o qual o pacote deverá ser enviado, ao ter realizado esta etapa, o nó anexa um novo rótulo no pacote.

Em uma rede MPLS o caminho por onde um pacote é comutado é denominado de LSP (*Label Switched Path*), este caminho é composto por nós que no protocolo recebem a alcunha de LSR (*Label Switching Router*). Conforme já supracitado, quando um pacote passa por um LSR, este recebe um cabeçalho de "calço" (*Shim Header*), que conterá uma etiqueta fornecida

de acordo com as informações da tabela *Label Forwarding Information Base* (LFIB) do LSR, que é uma base que indica como encaminhar os pacotes.

Para que a LFIB seja preenchida é necessário que os protocolos de distribuição de rótulos as configurem. Estes protocolos são protocolos de controle baseados em IP, que possuem a finalidade de distribuir as etiquetas em um ambiente MPLS. O LDP (*Label Distribuition Protocol*) é um dos mais fundamentais deste tipo e funciona da seguinte forma: este busca LSRs com capacidade LDP ajacente e então é iniciada uma sessão onde os parceiros LDP irão anunciar os seus rótulos uns aos outros.

2.1.1.0.2 RSVP - Resource Reservation Protocol

Farrel (2004) descreve o RSVP como um protocolo de sinalização que tem o fim de alocar recursos ao longo do caminho seguido pelos dados dentro da rede. O seu funcionamento consiste no envio de uma mensagem que coletará informações sobre a disponibilidade de recursos na rede até chegar ao nó de destino. Ao chegar no nó de egresso, este calcula quais recursos precisarão ser reservados na rede e envia uma mensagem que fará um caminho de volta para o nó de origem, reservando os recursos para cada nó que passa. Quando chega na origem a alocação é completada.

O RSVP tradicional não possui a capacidade de distribuir rótulos, no entanto, a sua extensão, o RSVP-TE, é capaz de realizar esta tarefa, assim podendo ser integrado ao conjunto de protocolos de distribuição de rótulos do MPLS.

2.1.1.0.3 GMPLS - Generalized Multiprotocol Label Switching

Segundo Farrel (2004), o GMPLS é um conjunto de extensões aos protocolos de sinalização de engenharia de tráfego e tem por objetivo oferecer um plano de controle comum que permita a interação com diferentes tecnologias como o MPLS, Ethernet e ATM. Também tem como característica possuir o seu plano de controle segregado do plano de dados, conceito que é utilizado na implementação de DCNs, dentre outros motivos para suportar equipamentos que não possuem o protocolo MPLS embarcado.

O uso do protocolo GMPLS está frequentemente relacionado ao provisionamento de circuitos conforme denotado em e-VLBI (DRAGON, 2012), um projeto de rádio astronomia em que uma ferramente que gerencia do plano de controle GMPLS, faz provisões de recursos dinamicamente, conectando os colaboradores através de sua rede.

2.1.1.0.4 OSPF-TE

O OSPF-TE (KATZ; KOMPELLA; YEUNG, 2003) (KOMPELLA; REKHTER, 2005) é uma extensão do protocolo de roteamento OSPF que permite engenharia de tráfego e o uso em redes que não sejam IP. Nesta versão com engenharia de tráfego, são adicionadas novas informações na mensagem LSA[2], que permitirá que os equipamentos levem em consideração a banda disponível nos enlaces para o cálculo de rota.

2.1.2 Qualidade de Serviço

QoS (Qualidade de Serviço ou *Quality of Service*) (TANENBAUM, 2003) é um termo utilizado para indicar um conjunto de requisitos que devem ser cumpridos para que uma determinada aplicação obtenha da rede um nível de serviço satisfatório (qualidade de serviço). A qualidade de serviço obtida por uma dada aplicação ou por um conjunto de usuários de uma rede tradicionalmente está relacionada com três métricas de desempenho: perda de pacotes, atraso fim-a-fim unidirecional e *jitter* (variação do atraso). De acordo com o tipo de aplicação, estas medidas não podem ultrapassar determinados valores máximos. Este é o caso de aplicações multimídia, ou seja, que façam uso de vídeo e voz. Maior ainda será a sensibilidade as estas medidas se a aplicação multimídia for interativa (por exemplo: videoconferência). No caso de aplicações elásticas, ou seja, aquelas que fazem uso de TCP para a transmissão de dados com recuperação de erro, valores altos para as métricas de QoS não impedem destas aplicações funcionarem. Entretanto, valores altos para estas métricas podem fazer com o que a transferência de dados leve muito tempo para terminar. Dependendo do contexto, esta demora excessiva pode inviabilizar determinadas aplicações de rede que possuem um limite de tempo para serem

[2] O LSA (*Link-state Advertisement*) é uma mensagem que informa a topologia local de um equipamento de rede para os seus vizinhos, mediante que estes estejam na mesma área.

executadas, ou ainda, representar um aumento no custo da rede para um determinado usuário. Este é o caso, por exemplo de aplicações científicas que envolvem a transferência de grandes volumes de dados em grandes distâncias, como os dados provenientes do LHC situado no CERN (CERN, 2012).

De maneira que uma rede possa fornecer garantias de QoS, é necessária a adoção de arquiteturas e mecanismos capazes de fornecer um tratamento prioritário para determinadas classes de trafego em situações de congestionamento. São duas as arquiteturas utilizadas para este fim, a arquitetura de serviços diferenciados (*DiffServ*) e a arquitetura de serviços integrados (*IntServ*). A primeira implica uma abordagem baseada em classes de tráfego, onde os pacotes são marcados de maneira a serem classificados em diferentes classes. De acordo com a classe o pacote passará por um dado mecanismo de escalonamento nas filas das interfaces de saída dos roteadores da rede. A segunda arquitetura utiliza o conceito de circuito, ou seja, para uma dada aplicação requisitando garantia de QoS é estabelecido um circuito onde os recursos necessários são previamente reservados através do RSVP. Caso o caminho não possua os recursos necessários, o circuito não é estabelecido.

Em ambas as arquiteturas, faz se necessário o uso de mecanismos de escalonamento de fila, para privilegiar determinadas classes, conforme o perfil de cada uma, e de policiamento de tráfego, para evitar que uma dada aplicação, ou que o tráfego agregado de uma determinada classe, ultrapasse os limites "negociados". Dentre os diversos mecanismos para escalonamento de fila e para policiamento de tráfego citamos aqui alguns que serão úteis no contexto deste trabalho:

2.1.2.1 Escalonadores

Através dos mecanismos de escalonamento, é definido o modo como os pacotes são enfileirados, selecionados e encaminhados pelo enlace. Dentre os diversos mecanismos para escalonamento de fila, os que podem ser utilizados no contexto de DCN são:

2.1.2.1.1 Token Bucket

Segundo (TANENBAUM, 2003), o *Token Bucket* é um algoritmo de controle de transmissão de pacotes que impõe um controle de saída. Este algoritmo consiste em um balde (*bucket*) que contém símbolos (*token*) e estes são repostos a uma determinada taxa. Para que um pacote seja transmitido, este deve capturar uma determinada quantidade de símbolos e eliminá-los para poder trafegar. Caso a taxa de pacotes seja superior à taxa de reposição de símbolos, os pacotes terão que esperar até novos símbolos surgirem. A figura 4 demonstra como é o algoritmo.

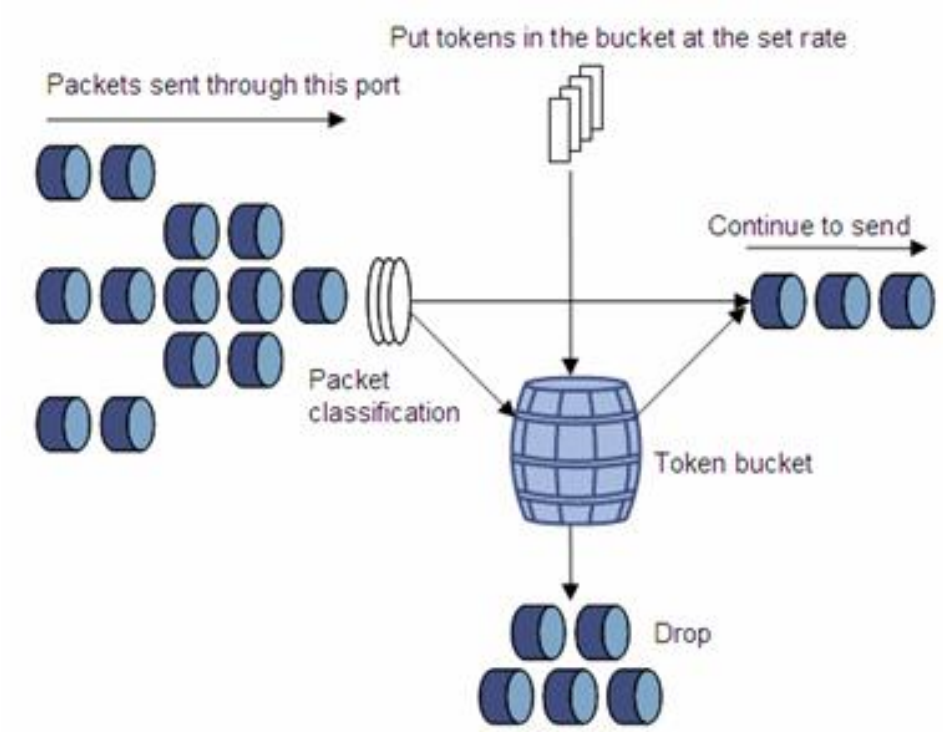

Figura 4: Algoritmo do Token Bucket (H3C, 2012).

2.1.2.1.2 Hierarchical Token Bucket

O HTB é um escalonador de pacotes que utiliza uma estrutura hierárquica, onde através da instanciação de classes é possível dividir o tráfego usando diferentes prioridades. No HTB existem três tipos de classe: *root*, *inner* e *leaf*. Pelas classes *root* passa todo o tráfego da interface, *inner* podem ter classes pais e filhas e *leaf* são classes terminais que só possuem pai.

Através da hierarquia de compartilhamento, as classes filhas (*leaf* e *inner*) pegam emprestados *tokens* das classes-pai no momento que ultrapassam a taxa mínima de envio. Estas classes

continuarão pegando os *tokens* emprestados até que estes se esgotem, indicando que a taxa máxima foi atingida. Neste momento começará a haver enfileiramento de pacotes até que mais *tokens* estejam disponíveis. É importante notar que em uma classe são definidos limites de banda máxima e mínima e prioridades conforme indicado pela Figura 5.

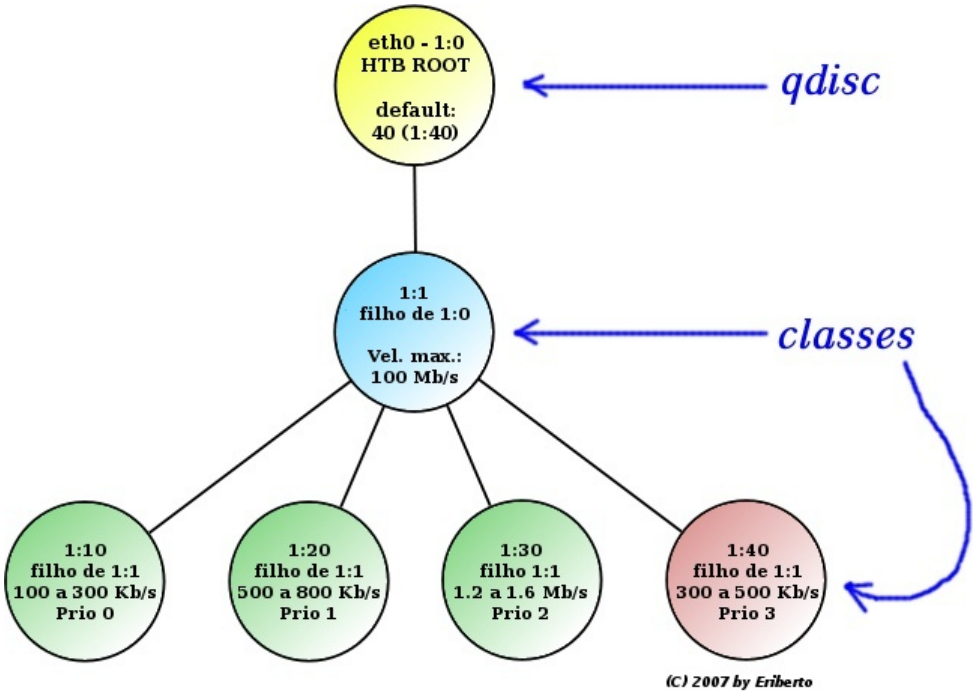

Figura 5: Exemplificação da estrutura hierárquica do HTB, extraída de (MOTA, 2012).

2.1.2.1.3 Priority Queuing

O *Priority Queuing* (PQ) faz uso de um conjunto de n-filas cada qual com o seu nível de prioridade. Uma fila de prioridade `i` só poderá ter seu pacotes servidos depois que uma fila de prioridade `i + 1` tiver os seus pacotes processados.Ou seja, a fila de prioridade mais baixa só terá os seus pacotes servidos quando todas as demais filas estiverem vazias. Da mesma maneira, enquanto existir pacotes na fila de alta prioridade, nenhum pacote das outras filas poderá ser processado.

2.1.2.1.4 Weighted Fair Queuing

WFQ (*Weighted Fair Queuing*) é um escalonador de pacotes cíclico ponderado que gerencia múltiplas filas. As filas que possuem maior prioridade receberão um peso maior, isto é, serão

atendidas por um periodo mais longo. Ao término do processamento da fila de maior prioridade as filas com peso menor serão atendidas de forma ponderada, conforme o peso de cada uma. A Figura 6 apresenta como é o algoritmo.

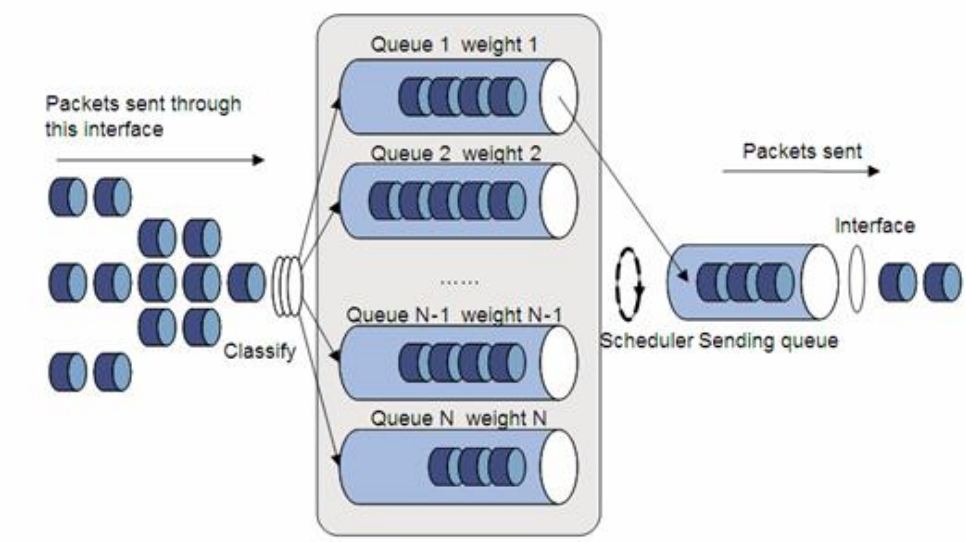

Figura 6: Algoritmo do WFQ, extraída de (H3C, 2012).

2.1.2.2 Políticas de Tráfego

As Políticas de Tráfego definem regras de QoS, definindo como os pacotes devem ser encaminhados e qual a sua prioridade. Dependendo da política escolhida um ou mais mecanismos de escalonamento podem ser usados.

2.1.2.2.1 Best Effort

É referente a uma abordagem em que a rede não faz nenhuma diferenciação entre os pacotes que transitam por ela. Este serviço apenas busca enviar os pacotes o quanto antes, sem que haja garantias de entrega ou de QoS para os seus usuários.

2.1.2.2.2 Expedicted Forward

O *Expedicted Forward* (DAVIE et al., 2002) é uma política de serviço em que o tráfego associado a esta classe deve ser enviado com banda assegurada até o limite máximo, baixo *jitter*, baixa latência e baixa perda. Esta política é implementada com o uso da *Priority Queueing* e de um

mecanismo de *Token Bucket* para limitar o volume de tráfego nesta classe.

2.1.2.2.3 Assured-Forward

O AF (*Assured-Forward*) (HEINANEN et al., 1999) é uma política que oferece diferentes níveis de encaminhamento assegurado. Neste serviço as classes são definidas, e nelas são alocados determinadas quantidades de recursos (*buffer* e banda). Os pacotes de um determinado fluxo são atribuídos a uma classe, e nesta, estes recebem podem um nível de precedência de descarte, entre n-níveis possíveis, sendo que quanto menor o nível de precedência, menor a possibilidade de descarte. Nesta política o envio é garantido mediante que a taxa alocada na classe não seja ultrapassada.

2.1.2.2.4 Scavenger

Scavenger ou *Lower Effort* (BLESS; NICHOLS; WEHRLE, 2003) é uma política voltada para tráfegos de baixa prioridade, que no qual todos os outros tráfegos recebem precedência sobre o tráfego do *Scavenger* no consumo da banda.

Este serviço é usado em redes que possuem tráfego de encaminhamento opcional, ou seja, é um tráfego que somente consumirá a banda da rede caso haja disponibilidade na mesma.

2.1.3 Soluções de software para DCNs

Segundo um estudo de popularidade realizado por (SALMITO, 2012) as soluções de inter-domínio e intra-domínio mais adotadas pelas organizações, inclusive pela RNP (RNP, 2012), são o OSCARS e o DRAGON, respectivamente.

2.1.3.1 OSCARS - On-demand Secure Circuits and Advance Reservation System

OSCARS é uma ferramenta que faz o provisionamento de circuitos virtuais com garantia de banda ponta a ponta e em múltiplos domínios. Este software utiliza um modelo de *Web Services* para implementar a comunicação com o sistema e entre os domínios e sua arquitetura é composta dos seguintes elementos, que são representados pela Figura 7:

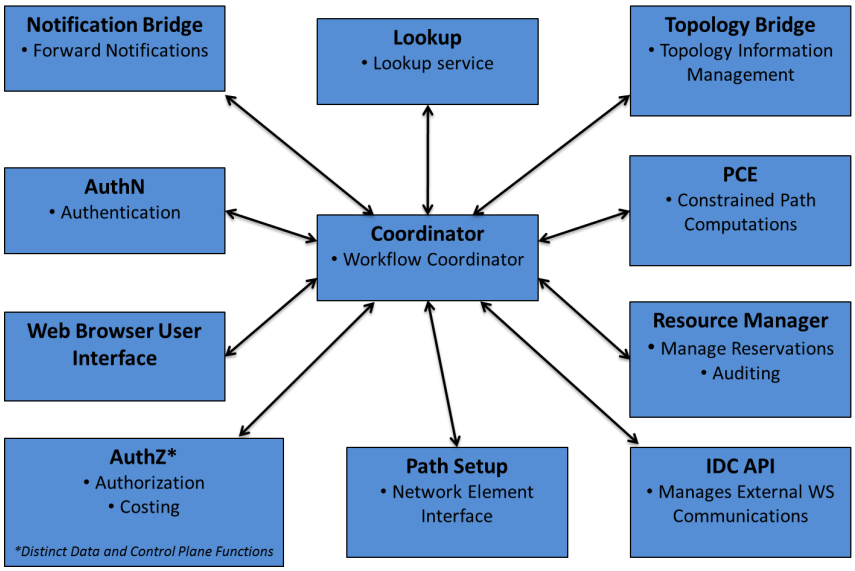

Figura 7: Visão da interação dos componentes do OSCARS (ESNET, 2012c).

- ***Coordinator*** - Responsável por criar/remover/modificar reservas de cliente e inter-domínio.
- ***Notification Bridge*** - Encaminha as notificações do componente Coordinator.
- ***AuthN*** - Faz a autenticação de usuários, serviços e domínios.
- ***AuthZ*** - Utiliza uma lista de atributos do usuário para permitir ou indeferir as ações do mesmo.
- ***Web Browser User Interface*** - Interface Web do Usuário.
- ***Lookup*** - Localiza serviços que controlam um domínio.
- ***Path Setup*** - Se comunica com os equipamentos da rede, configurando-os para a criação de circuitos. Também é conhecido pelo acrônimo PSS.
- ***Topology Bridge*** - Gerência as topologias, armazenando internamente ou usando de um serviço externo como o PerfSONAR.
- ***Path Computation Engine*** - Também conhecido como PCE, é responsável pelo cálculo de rota no intra-domínio.
- ***Resource Manager*** - Auditoria e gerência de reservas de circuitos.

- ***IDC API*** - API externa para programação de aplicativos.

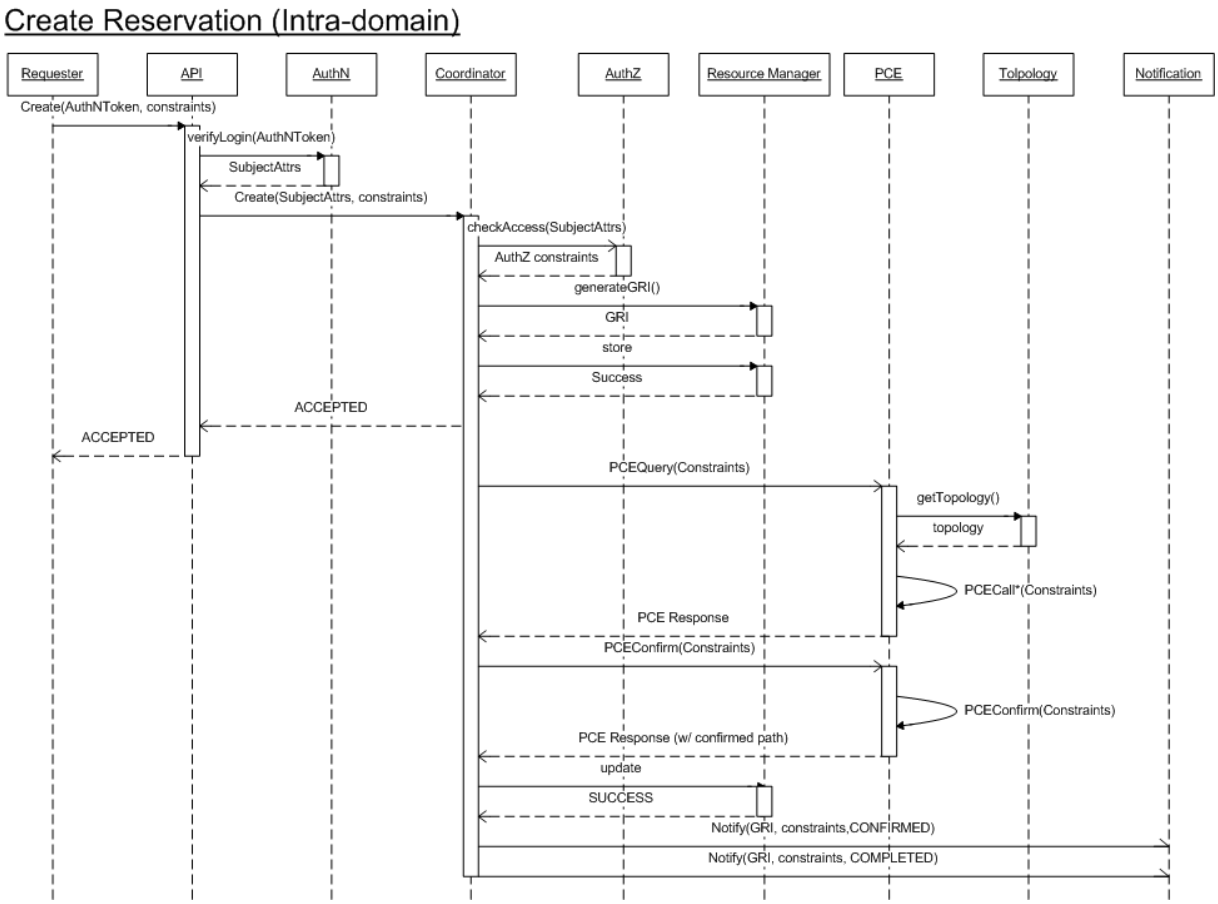

Figura 8: *Workflow* entre os componentes do OSCARS (OSCARS, 2012)

A Figura 8 apresenta como os componentes do OSCARS se comunicam no momento de uma reserva. Para realizar o provisionamento de um circuito, o usuário deverá acessar a `API` do OSCARS e passará por uma etapa de autenticação (`AuthN`), onde é verificado as permissões do usuário. Caso o usuário possua as permissões necessárias, este poderá realizar o agendamento de um circuito (`Resource Manager`), onde será determinado o período de vigência do mesmo, a origem e destino do circuito, a etiqueta do circuito e a banda desejada. Se os recursos exigidos pelo usuário estiverem disponíveis o `PCE`, calculará o caminho entre origem e destino, baseado na topologia (`Topology`). Após o cálculo o componente `Notification` irá encaminhar a mensagem de criação de circuito para outros OSCARS e para o `PSS`, elemento responsável por configurar os equipamentos.

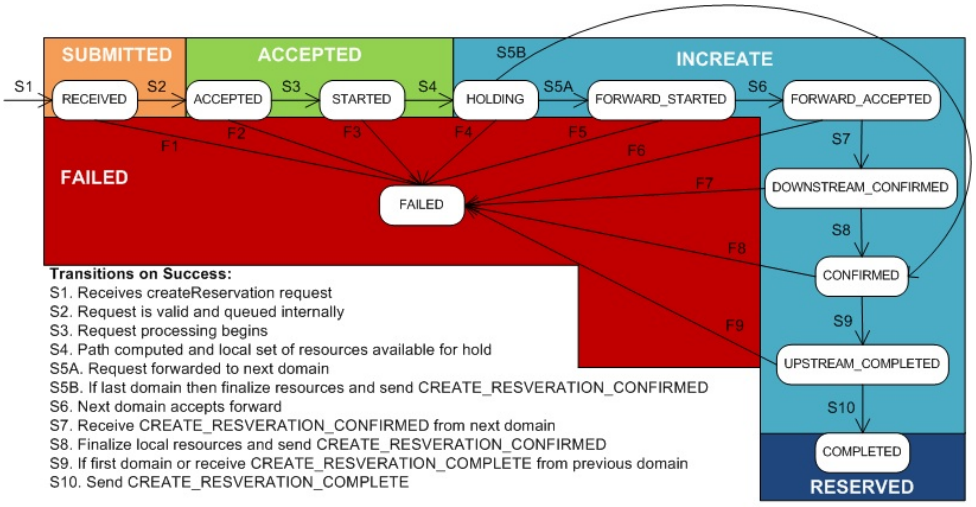

Figura 9: Diagrama de estados de criação de agendamento (ESNET, 2012b).

A Figura 9 é representado graficamente o processo de notificação citado anteriormente. É demonstrado que quando o usuário faz o pedido de reserva de um circuito, o estado deste é representado como `RECEIVED` pelo OSCARS. Caso os requisitos exigidos pelo usuário estejam disponíveis, a reserva passará a ser aceita, caso não esteja disponível, a reserva falhará (`FAILED`). Após o circuito ser aceito, este passará por um processo de cálculo de rota, representado pelos fluxos `S3` e `S4`, e então este provisionamento será encaminhado para outros domínios em um processo macro denominado de `INCREATE` (representado pelos fuxos `S5A`, `S6` e `S7`). Se os domínios por onde passa o circuito tiverem disponibilidade de recursos, o pedido de reserva será confirmado (`CONFIRMED`). No término deste processo, uma mensagem será encaminhada para o domínio de origem da reserva confirmando que o pedido foi reservado.

O OSCARS utiliza o IDCP (*Inter-domain Controller Protocol*), protocolo focado nas soluções de problemas de redes de circuitos dinâmicos (DICE, 2010). A Figura 10, extraída do IDCP, representa graficamente os estados de reserva de um circuito. Depois do aprovisionamento ter sido confirmado, é necessário que as configurações sejam repassadas para os equipamentos de rede. Isto é representado pelo estado `INSETUP` e, se for uma reserva já existente sendo modificada, o estado será `INMODIFY`. Quando completadas as modificações, os circuitos passarão para o estado `ACTIVE` e, no momento que o seu período de reserva acabar, o provisionamento será derrubado, conforme indicado pelo estado `INTEARDOWN`. Quando o processo de término

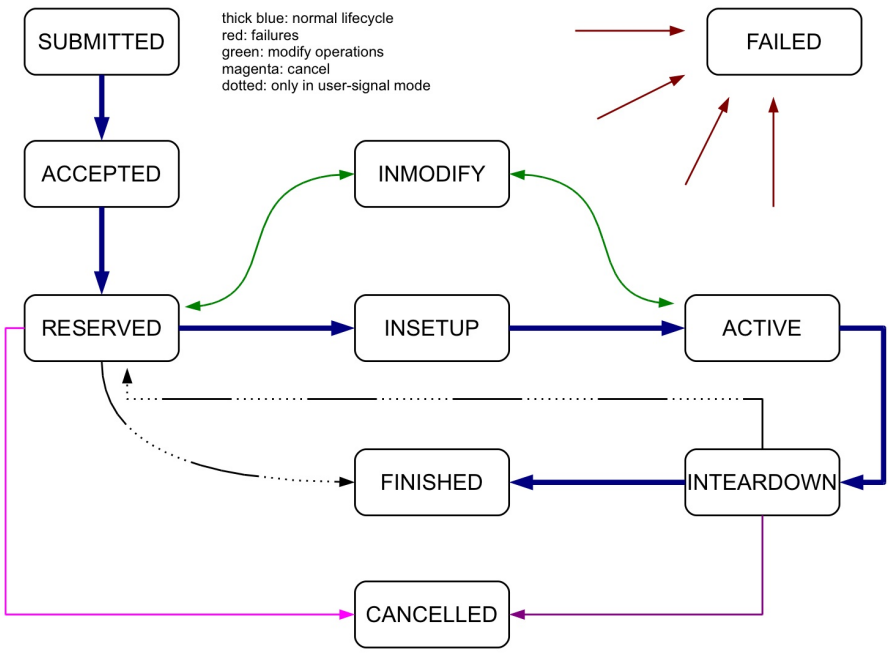

Figura 10: Diagrama de estados de reserva (DICE, 2010).

de agendamento chega ao seu fim, com sucesso, o circuito passará para o estado `FINISHED`. Circuitos podem ser cancelados (`CANCELLED`), no entanto para chegar esta etapa, também é necessário passar pelo processo de `INTEARDOWN` caso já estejam ativos. No momento que os circuitos terminam ou são cancelados, os recursos da rede ficam novamente disponíveis para o agendamento de novas requisições.

2.1.3.2 DRAGON: Dynamic Resource Allocation in GMPLS Optical Networks

Aplicativo que atua como um controlador de domínio, auxiliando no provisionamento dinâmico de circuitos em redes heterogêneas no intradomínio (LEHMAN; SOBIESKI; JABBARI, 2006).

A arquitetura do DRAGON consiste nos seguintes componentes, e está representada na figura 11:

- **NARB** (*Network-Aware Resource Broker*) - É uma entidade que representa um domínio e faz o roteamento interno baseado nos recursos disponíveis na rede. O NARB também pode ser responsável para o roteamento interdomínio.

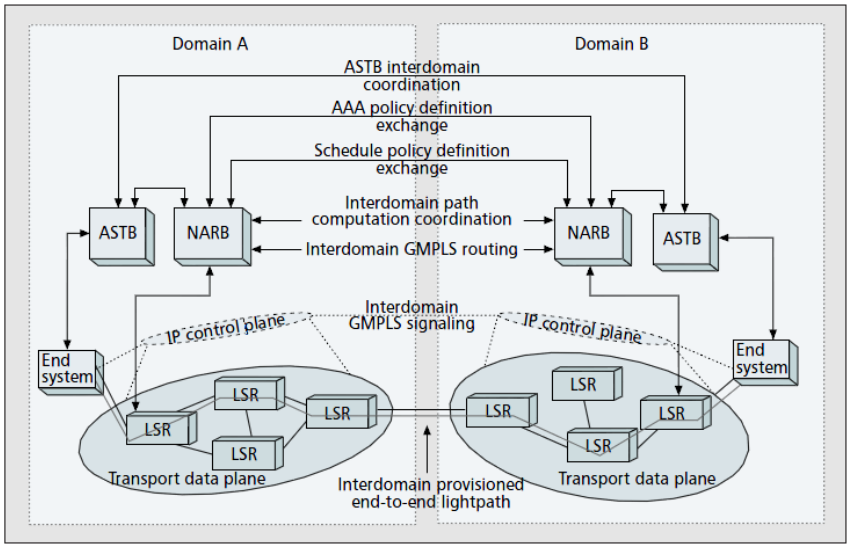

Figura 11: Arquitetura completa do DRAGON (LEHMAN; SOBIESKI; JABBARI, 2006).

- **ASTB** (*Application-Specific Topology Builder*) - Aceita os pedidos dos usuários ou sistemas finais para múltiplas conexões de rede, e utiliza o NARB para determinar se o caminho pedido está disponível com as restrições de segurança e agendamento exigidas.

- **VLSR** (*Virtual Label Switch Router*) - Permite que componentes que não compreendem GMPLS possam ser integrados a rede.

2.2 Redes Definidas por Software

Segundo a ONF [3] (FOUNDATION, 2012b), **Redes Definidas por Software** (também conhecido como SDN, de Software Defined Network) é uma arquitetura de redes emergente em que o plano de controle da rede é dissociado do plano de encaminhamento e é possível programá-lo diretamente.

Conforme apresentado na Figura 12, a inteligência da rede é centralizada na aplicação de controle, que possui uma visão global da infraestrutura. Isto torna os equipamentos da rede em uma única abstração para as aplicações e mecanismos de política. No momento em que toda a rede é considerada uma única entidade, configurar ou alterar o seu comportamento se torna uma atividade mais simples, pois para os operadores de rede e administradores bastaria programar

[3] Acrônimo de Open Network Foundation.

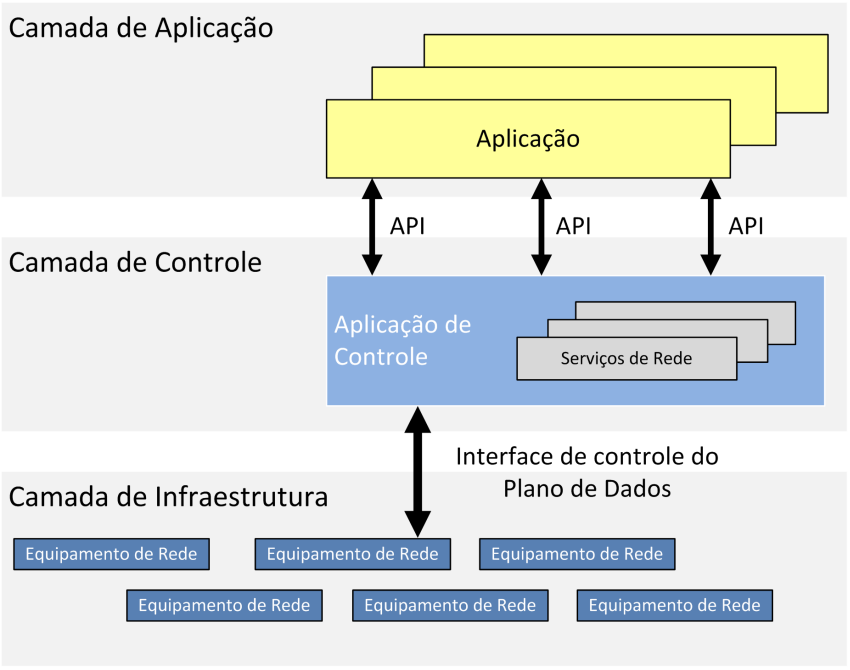

Figura 12: Arquitetura de Redes Definidas por Software (FOUNDATION, 2012b).

uma aplicação, em vez de configurar cada entidade manualmente.

Além da vantagem supracitada, o SDN permite que os hardwares sejam mais simples e baratos, uma vez que é possível controlar todos os componentes através de um protocolo comum, cenário que não ocorre em ambientes de produção visto que cada fabricante tem a sua própria interface.

2.2.1 OpenFlow

O protocolo OpenFlow é uma das vertentes do conceito de SDN, apresentado em (MCKEOWN et al., 2008) e tem origem nos trabalhos SANE (CASADO et al., 2006) e Ethane (CASADO et al., 2007). O primeiro buscava melhorar a segurança da rede através da centralização das decisões de encaminhamento e acesso por um servidor central, já o segundo foi uma das primeiras arquiteturas de SDN e que por ventura se tornaria este protocolo.

Segundo notado em (MCKEOWN et al., 2008), o OpenFlow surgiu da necessidade de se experimentar novos protocolos em ambientes reais (como campus de universidades), já que é impossível testá-los na infraestrutura de produção, visto que esta tem um papel essencial no nosso dia-a-dia. Isto faz com que a maioria das novas ideias da comunidade científica não

sejam averiguadas em cenários reais.

O OpenFlow oferece um protocolo aberto para programar a tabela de fluxos de diferentes *switches*, assim permitindo que pesquisadores controlem os seus próprios fluxos, por exemplo, escolhendo por quais rotas os seus pacotes passarão e que processamento receberão. Com isto, administradores de redes poderiam segregar fluxos de produção e de pesquisa, e os pesquisadores teriam a liberdade necessária para que experimentem novos protocolos em cenários reais.

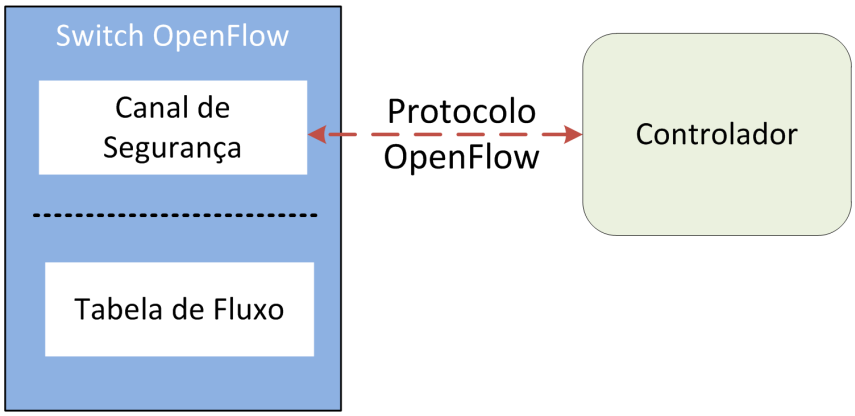

Figura 13: Arquitetura do switch OpenFlow baseada em (CONSORTIUM, 2009b).

A Figura 13 apresenta a arquitetura de um *switch* OpenFlow, este consiste em quatro partes: uma **tabela de fluxos** (*flow table*), que realiza a correspondência dos pacotes com os fluxos presentes nesta tabela; um **canal de segurança** (*secure channel*) que conecta o *switch* com um controlador externo, assim permitindo que pacotes e comandos sejam enviados entre o controlador e o *switch*; o **protocolo OpenFlow**, que provê uma forma padronizada e aberta do controlador se comunicar com o *switch*; e o **controlador** é o elemento responsável por determinar como manusear estes pacotes sem uma entrada de fluxo válida, e por gerenciar a tabela de fluxo do *switches* através da adição ou da remoção de fluxos.

A Tabela de fluxos contém um conjunto de entradas de fluxos, contadores de atividades, e um conjunto de atividades para serem aplicadas nos pacotes que correspondam com os fluxos.

Todos os pacotes que são processados pelo *switch* são comparados com a tabela de fluxo. Se uma correspondência é encontrada, então a ação daquela entrada de fluxo é realizada no pacote (por exemplo: adicionar um *tag* de VLAN ao pacote ou encaminhar o pacote para uma

determinada porta). Se nenhuma correspondência é encontrada, o pacote é encaminhado para o controlador através do canal de segurança, para este determinar como irá processá-lo.

Campos do cabeçalho	Contadores	Ações

Tabela 1: Campos de um fluxo (CONSORTIUM, 2009b).

Um fluxo é formado por três campos, conforme apresentado na tabela 1, e que consistem em:

- **Campos do cabeçalho** - são os campos com os quais os pacotes devem ser correspondidos.
- **Contadores** - estatísticas que mantém o controle do número de pacotes e bytes de cada fluxo, e o tempo transcorrido desde que o último pacote correspondeu ao fluxo.
- **Ações** - define como os pacotes devem ser processados pelo *switch*.

Ingress port	Ether src	Ether dst	Ether type	VLAN ID	VLAN prio	IP src	IP dst	IP proto	IP TOS	TCP/UDP src port	TCP/UDP dst port

Tabela 2: Campos dos pacotes utilizados para correspôndencia nos fluxos (CONSORTIUM, 2009b).

A Tabela 2 lista os campos de cabeçalhos com os quais um pacote que está entrando no *switch* deve ser comparado. Cada cabeçalho possui um valor específico. A tabela apresentada neste texto é referente a versão do OpenFlow 1.0.0.

As ações ditam como o *switch* deve controlar os pacotes correspondentes com um fluxo, sendo que cada fluxo pode ser associado com nenhuma ação ou com mais de uma ação. Se nenhuma ação de encaminhamento estiver presente no fluxo, o pacote será descartado.

As possíveis ações que um *switch* OpenFlow pode realizar estão enumeradas abaixo:

- Encaminhamento (*Forward*) - O *switch* OpenFlow deve suportar o encaminhamento de pacotes para as portas físicas e virtuais.
- Enfileiramento (*Enqueue*) - A ação de enfileiramento encaminha um pacote para uma fila anexada a uma porta. Esta fila ditará como sera o padrão de encaminhamento dos pacotes.

- Descarte (*Drop*) - Um fluxo sem nenhuma ação especificada, indica que todos os pacotes que corresponderem com este serão descartados.

- Modificação de Campo (*Modify-Field*) - Consiste na modificação dos cabeçalhos dos pacotes. Um exemplo desta ação é a modificar o MAC de origem.

2.2.2 QoSFlow

O **QoSFlow** de (ISHIMORI et al., 2012), surgiu de uma questão pouco explorada, o gerenciamento de QoS (*Quality of Service*). No OpenFlow, para se configurar o QoS, é necessário o uso de ferramentas externas ao protocolo, como o `dpctl`, capaz de configurar filas nas portas dos *switches* de forma estática.

O QoSFlow é um*framework* que possibilita o gerenciamento de QoS em domínios OpenFlow, através da introdução de primitivas de QoS. As primitivas são um agregado de funções capazes de gerenciar recursos de QoS, que quando invocadas por um controlador OpenFlow, programam dinamicamente os aspectos de QoS nas portas dos *switches* OpenFlow.

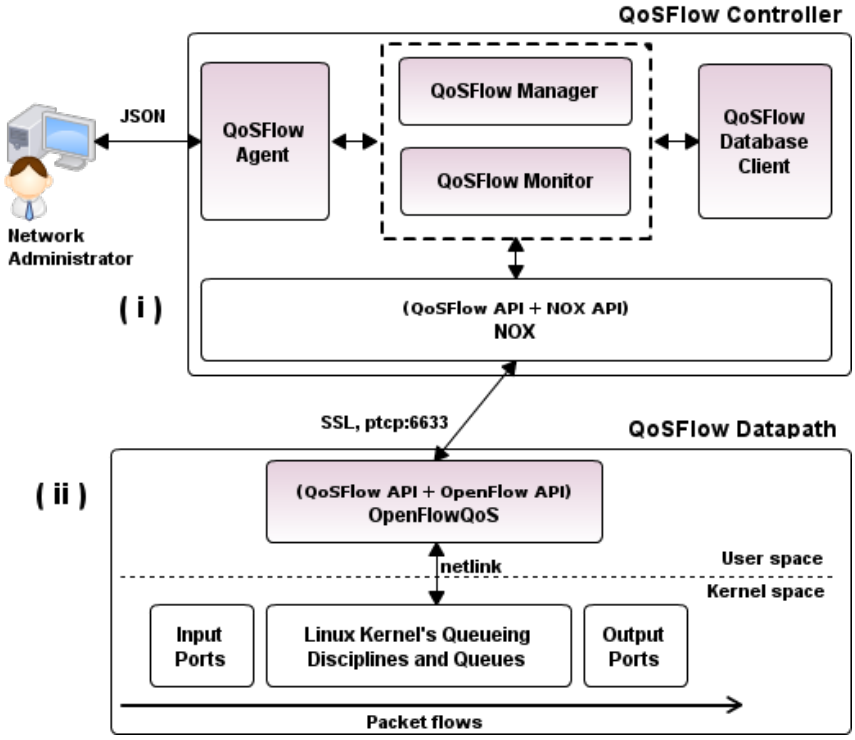

Figura 14: Arquitetura do QoSFlow (ISHIMORI et al., 2012).

A Figura 14, apresenta a arquitetura atualmente utilizada no QoSFlow. Nesta, os seguintes elementos se destacam:

- ***Network Administrator*** - Componente onde o usuário define as políticas de QoS de um domínio QoSFlow.
- ***QoSFlow Agent*** - É uma interface que recebe as polítcas de QoS, via mensagens JSON, e aplica as suas diretivas nos componentes *QoSFlow Manager* e no *QoSFlow Monitor*.
- ***QosFlow Manager*** - Componente que configura os escalonadores de pacotes em uma porta do *switch*, limita a largura de banda e cria regras.
- ***QoSFlow Monitor*** - Usado para coletar estátiscas do QoS e informações adicionais como a identificação do *switch* e o nome de portas.
- ***QoSFlow Database Client*** - É uma base de dados que armazena os recursos de QoS disponíveis e consumidos pelas aplicações do usuário.

É importante notar que, para as funcionalidades do QoSFlow serem usufruídas, é necessário que o *switch* e o controlador possuam a API do QoSFlow, assim sendo necessário o uso de um *firmware* específico para estes equipamentos. Caso não haja esta modificação ou outros componentes da arquitetura do OpenFlow não possuam suporte ao QoSFlow, as capacidades do último não poderão ser utilizadas.

2.2.3 Controladores

Os controladores, conforme apresentado na seção 2.2.1, são os elementos responsáveis pelo gerenciamento da tabela de fluxo dos *switches* e também são os responsáveis por manusear os pacotes que não possuem uma entrada de fluxo válida. Nesta seção estão enumerados os controladores de maior relevância que utilizam o protocolo OpenFlow 1.0:

- **NOX-Classic** - É um controlador de *switches* OpenFlow que possui API's em C++ e Python. Foi descontinuado em Março de 2012 (NOX-CLASSIC, 2012).

- **POX e NOX** - Sucessores do NOX-Classic. O primeiro tem como base a linguagem Python e é focado para o público com interesse em pesquisa, o segundo tem como base a linguagem C++ e é voltado para o público que necessita de performance (MCCAULEY, 2012).

- **NOX QoSFlow** - É o NOX-Classic com a API extendida para o uso do QoSFlow.

- **Trema** - É um *framework* para o desenvolvimento de controladores OpenFlow. As linguagens que este controlador suporta são C e Ruby (TREMA, 2012).

- **Beacon** - É um controlador multi-plataforma baseado na linguagem Java, que suporta operações baseadas em eventos e em *threads* (BEACON, 2012).

- **Floodlight** - Ramificação do controlador Beacon também é um controlador baseado na linguagem Java (FLOODLIGHT, 2012).

- **Maestro** - Escrito em Java, é uma plataforma escalável de controle da rede (CAI; NG., 2010).

- **OESS** - Conjunto de softwares que são utilizados para configurar e controlar dinamicamente circuitos virtuais baseados em VLAN (NDDI, 2012).

2.2.4 Switches OpenFlow

Conforme já apresentado na seção 2.2.1, os *switches* OpenFlow são equipamentos que encaminham os pacotes de acordo com as decisões tomadas pelo controlador. Alguns fabricantes de *switches* e roteadores (HB, Brockade, Cisco, Juniper e etc) provem, em alguns de seus modelos, a implementação de APIs OpenFlow. Entretanto, existem soluções de *switching* baseadas em *software* que podem ser usadas tanto em um ambiente virtualizado quanto embarcadas em *hardwares*, específico (NETFpga) ou não (arquiteturas x68), rodando o sistema operacional Linux. Dentre os softwares *switches* mais utilizados têm se os seguintes:

- **Switch Referência OpenFlow** - É uma implementação mínima da pilha OpenFlow (CONSORTIUM, 2009a).

- **Open vSwitch** - É um *switch* virtual multicamadas e flexível. Tem compatibilidade com o OpenFlow.

- **Switch QosFlow** - É um *switch* OpenFlow 1.0 com a API do QoSFlow.

- **Pantou** - Transforma pontos de acesso em *switches* OpenFlow 1.0 (CONSORTIUM, 2012b).

3 *Uso de DCN sobre Redes SDN*

O foco deste capítulo é apresentar formas de como integrar o uso de DCN sobre redes SDN. A primeira etapa realiza um levantamento de possíveis arquiteturas que integrem estes dois ambientes e ao final desta parte, é feita uma análise qualitativa entre as arquiteturas apresentadas. Na segunda etapa são elencadas as ferramentas que podem viabilizar a concretização desta integração.

3.1 Arquiteturas

Para a integração das tecnologias de SDN e DCN, são apresentadas três arquiteturas:

- **Arquitetura básica** - Arquitetura desenvolvida pela Internet2, faz a integração dos dois conceitos através das ferramentas OSCARS e NOX. Esta arquitetura foi apresentada no evento SuperComputer de 2011 (SC11, 2012).
- **Arquitetura com QoSFlow** - Variação da arquitetura básica, utiliza o conceito do QoSFlow. Nesta arquitetura é possível gerenciar o QoS em domínios OpenFlow, diferente do que ocorre na arquitetura previamente apresentada.
- **Arquitetura com QoSFlow utilizando OpenFlow 1.3** - Proposta de arquitetura utilizando o QoSFlow, porém aplicado com a especificação do OpenFlow 1.3 (FOUNDATION, 2012a).

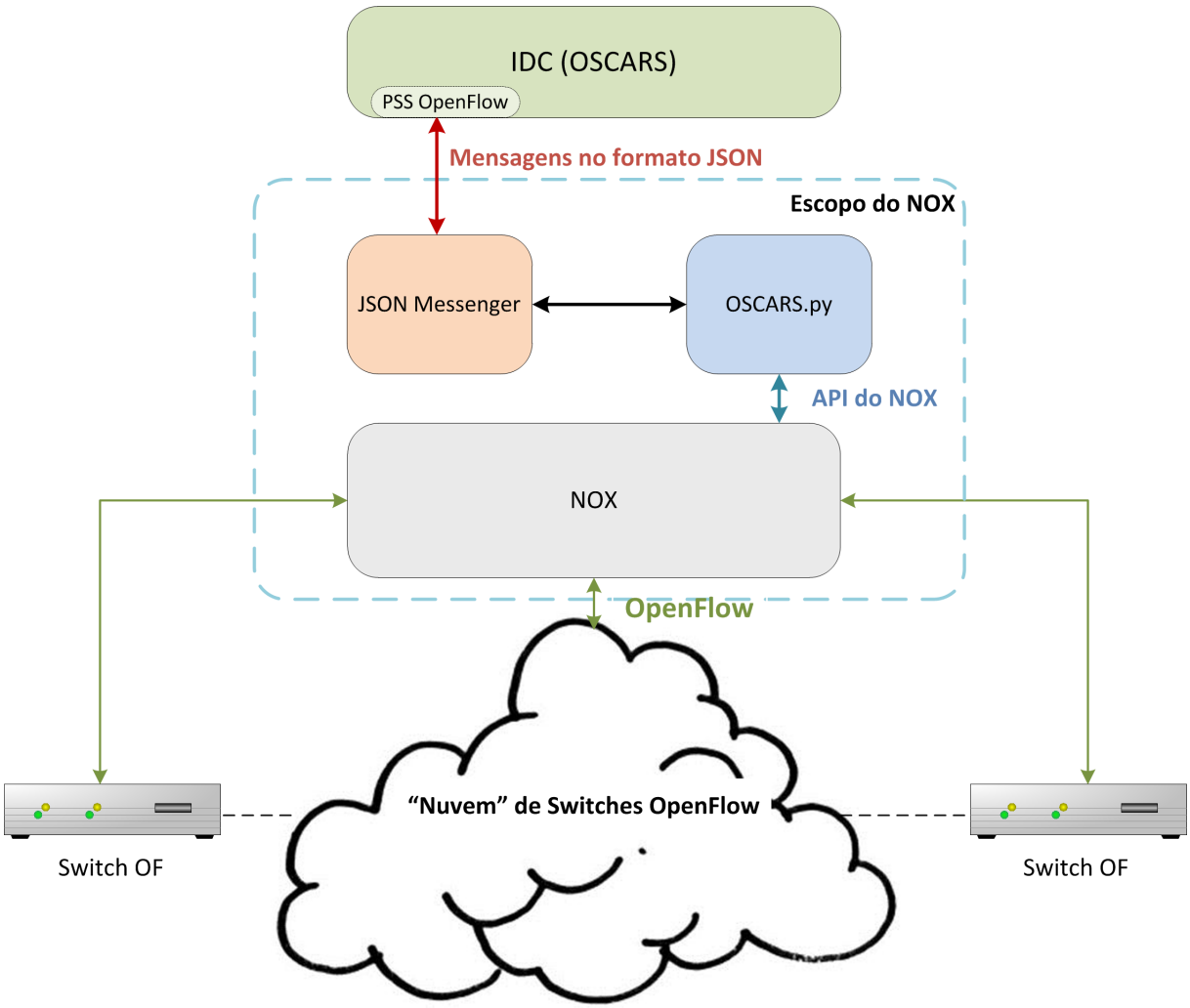

Figura 15: Arquitetura OSCARS com OpenFlow, baseada no documento (INTERNET2, 2011).

3.1.1 Arquitetura básica

Arquitetura apresentada na Figura 15 é o modelo atualmente vigente para integração do DCN sobre o SDN, consistindo no uso da ferramenta OSCARS em conjunto com o NOX e *switches* do protocolo OpenFlow 1.0.

O seu funcionamento nesta arquitetura começa com o OSCARS, que calcula o caminho de um circuito em uma determinada topologia, levando em consideração outros circuitos já existentes e agendados. Após esta etapa, o PSS OpenFlow[1] se comunica com a aplicação JSON Messenger[2] (via mensagens JSON) que repassará as mensagens para o OSCARS.py. No OS-

[1] É o mecanismo que o OSCARS utiliza para interagir com componentes controladores da arquitetura SDN.
[2] É uma aplicação padrão do NOX, que permite a interação com o controlador a partir de mensagens no formato JSON.

CARS.py, estas mensagens serão analisadas e convertidas em comandos de adição ou remoção de fluxos, que deverão ser encaminhados para os *switches* de um domínio OpenFlow.

Apesar desta arquitetura integrar o DCN sobre o SDN, possui como limitação a incapacidade de instanciar automaticamente filas nas portas dos *switches* para que a funcionalidade de QoS seja habilitada. Isto acontece porque o protocolo OpenFlow não configura filas, conforme apontado na seção 5.3.4 da especificação do OpenFlow 1.0 (CONSORTIUM, 2009b), assim, sendo possível apenas configurar o QoS estaticamente.

3.1.2 Arquitetura com QoSFlow

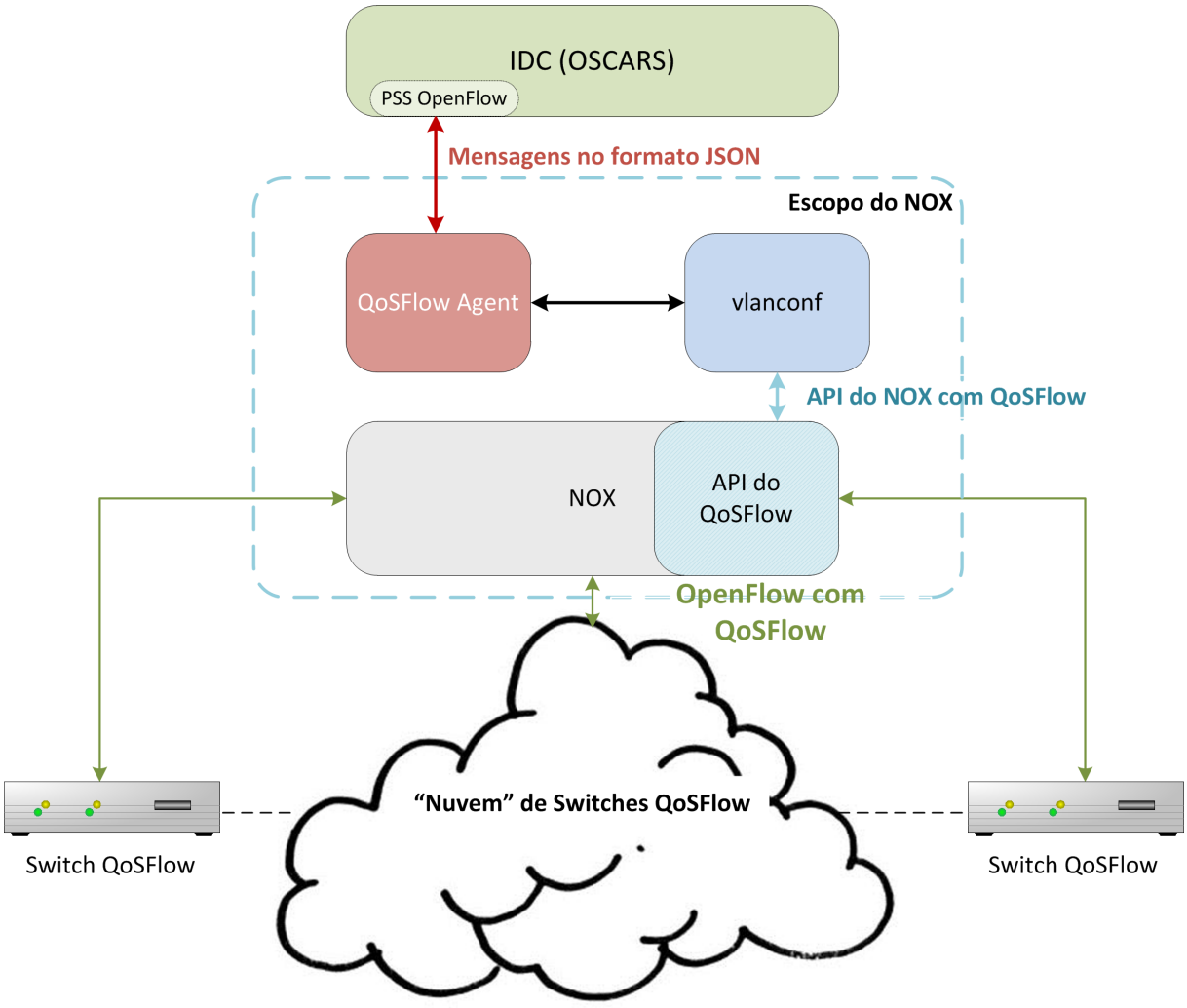

Figura 16: Arquitetura OSCARS com QoSFlow.

A segunda arquitetura, apresentada na figura 16 possui como diferencial o uso do *framework* do QoSFlow, que possibilita instanciar automaticamente filas nas portas dos *switches*,

funcionalidade que o protocolo OpenFlow não apresenta.

Funcionalmente este cenário é similar ao primeiro, o OSCARS calculará o caminho e notificará o controlador OpenFlow através de mensagens JSON. No entanto, nesta proposta será necessário modificar o componente QoSFlow Agent para que este possa compreender as mensagens do IDC. Este componente encaminhará as requisições para criação/remoção de circuitos para um segundo aplicativo sendo executado no NOX, o `vlanconf`[3], que será o responsável por configurar circuitos VLAN bidirecionais (ou seja com caminhos de ida e volta), utilizando o protocolo OpenFlow, e instanciar as diretivas de QoS, através da disciplina HTB, nas portas dos *switches* de ingresso através do uso da API do QoSFlow.

O componente `vlanconf`, criado para este trabalho, possui a funcionalidade de apoiar a criação de circuitos, que utilizam VLAN, e configurar o QoS de acordo com as necessidades do usuário.

Apesar desta arquitetura ser capaz de configurar os circuitos VLAN com QoS automaticamente, esta exige que o controlador OpenFlow, e os *switches* estejam aptos a usarem o QoSFlow, caso não possuam esta tecnologia não será possível configurar o QoS através do controlador.

3.1.3 Arquitetura com QoSFlow utilizando OpenFlow 1.3

Esta terceira proposição utilizará o mesmo funcionamento apresentado na seção 3.1.2, porém será contextualizado na versão 1.3 do protocolo OpenFlow (FOUNDATION, 2012a).

Nesta nova versão do protocolo, é integrada a ***Meter Table***, que consiste numa tabela de medidores por fluxo. Através desta tabela é possível implementar diversas ações de QoS, como por exemplo limitação de banda, podendo ainda estas serem combinadas com as filas configuradas pra uma dada porta.

Os medidores (*meters*) aferem a taxa de pacotes associados a um dado fluxo e permitem o controle desta taxa. Os medidores são vinculados diretamente a um fluxo, diferente das filas que são vinculadas as portas de um *datapath*. Qualquer fluxo pode especificar um medidor no seu

[3] O nome Vlanconf é devido a função que este exerce de configurar caminhos VLAN.

Identificador do Medidor	Bandas de Medidor	Contadores

Tabela 3: Componentes de uma entrada na *Meter Table*.

conjunto de ações. O medidor irá mensurar e controlar a taxa de pacotes dos fluxos associados a ela. Uma entrada nesta tabela é composta por três campos, conforme apresentado na tabela 3:

- **Identificador do Medidor** (*Meter Identifier*) - É o identificador único de um dado medidor. Trata-se de um número inteiro não negativo de 4 bytes.
- **Bandas de Medidor** (*Meter Bands*) - É uma lista desordenada de valores de banda, onde cada entrada indica também a forma como o pacote deve ser processado
- **Contadores** - São atualizados quando um pacote é processado por um medidor.

Um medidor pode ter mais de uma banda associada. Conforme a taxa de pacotes medida para um dado fluxo, o respectivo medidor aplicará a banda cuja taxa possui o maior valor abaixo da taxa medida. No caso do valor medido ser inferior a qualquer das taxas da *meter bands*, nenhuma banda será aplicada.

Além da taxa, cada banda de medidor possui os seguintes campos:

- **Tipo de banda** - Define como os pacotes são processados pela banda.
- **Contadores** - Contabilizam os pacotes processados por uma banda.
- **Argumentos específicos** - Tratam-se de argumentos opcionais conforme o tipo de banda especificado.

As formas de processamento dos pacotes conforme o tipo de banda são:

- **Descarte** (*Drop*)
- **Remarcação DSCP** (*Dscp remark*)

Tendo em vista a possibilidade de remarcação automática do campo DSCP de acordo com a taxa de um determinado fluxo. Com esta ação seria possível realizar políticas de QoS mais sofisticadas. Por exemplo, este seria o caso do serviço Scavenger, em que caso um fluxo ultrapasse a taxa delimitada por uma banda, este teria os pacotes marcados com um novo valor DSCP e seria encaminhado para uma fila de baixa prioridade.

3.1.4 Análise qualitativa das arquiteturas

Nesta seção será feita uma análise qualitativa entre as arquiteturas propostas. Esta avaliação começará pela primeira arquitetura, onde apenas as funcionalidades padrão do protocolo OpenFlow são utilizadas. Estas consistem na adição e na remoção de *tags* VLAN e é importante ressaltar que o protocolo OpenFlow não configura QoS nas portas dos *switches*, porém é possível realizar esta tarefa através de ferramentas externas. Entretanto, apesar da configuração da QoS ser possível, esta não é feita de forma automática, o que poderia prejudicar o funcionamento da DCN, já que o operador teria que configurar os equipamentos manualmente.

A segunda arquitetura utiliza o *framework* do QoSFlow. Neste cenário além de presentes as funcionalidades do OpenFlow já supracitadas, é possível configurar automaticamente o QoS nas portas dos *switches*, ou seja, sem a necessidade de utilizar ferramentas externas para esta atividade. Apesar deste arcabouço apresentar mais funcionalidades, este ainda não atenderia por completo todas as exigências do DCN, já que para o uso de políticas de QoS mais sofisticadas como o *Scavenger*, seria necessário alterar a marcação de prioridade dos pacotes que extrapolarem um determinado limite de banda, e para isto seria necessário o uso de uma ferramenta externa como o `IPTABLES`.

A terceira arquitetura é baseada na segunda arquitetura e faz uso do protocolo OpenFlow 1.3, desta forma esta também é capaz de configurar o QoS sem o uso de ferramentas externas. Também ao utilizar a versão 1.3 do OpenFlow é possível remarcar os pacotes que ultrapassem uma determinada banda sem a necessidade de uma ferramenta externa, conforme já denotado na seção 3.1.3. Com isto, esta arquitetura se apresenta como a solução mais completa, viabilizando a implementação de mais políticas de QoS, que antes não seriam possíveis de serem utilizadas

de maneira automática e nativa.

	Adição de tag VLAN	Remoção de tag VLAN	Configuração de QoS	Remarcação do campo DSCP
Arquitetura Básica	X	X		
QoSFlow	X	X	X	
QoSFlow com OF 1.3	X	X	X	X

Tabela 4: Comparação qualitativa entre as três arquiteturas propostas.

Tendo em vista esta análise, foi considerado que a arquitetura que melhor atende as necessidades de uma rede DCN é a opção do QoSFlow utilizando o OpenFlow 1.3. Atualmente o CPQD (CPQD..., 2012) disponibilizou uma versão do *software switch* em (CPQD, 2012). A Tabela 4 apresenta uma visão resumida desta avaliação qualitativa.

3.2 Ferramentas OpenFlow para Implementação das Arquiteturas DCN sobre SDN

Nesta seção serão apresentadas as ferramentas do OpenFlow selecionadas para este trabalho e ao final desta, será realizada uma avaliação qualitativa entre as mesmas.

3.2.1 Switch Referência OpenFlow

O *switch* referência, conforme já denotado na seção 2.2.4, é uma implementação mínima da pilha do OpenFlow 1.0, que adiciona funcionalidades de um *switch* OpenFlow à um computador com distribuição Linux e diversas placas de rede. Neste arcabouço de *software* estão disponibilizados os seguintes componentes:

- **dpctl** - É um componente que monitora e administra *datapaths* OpenFlow. É capaz de mostrar o estado atual do *switch*, incluindo suas funcionalidades, configuração e entradas nas tabelas de fluxo.
- **ofprotocol** - É um programa que instancia o canal de segurança entre o *switch* OpenFlow e um controlador, onde são enviadas mensagens OpenFlow de uma ponta a outra.

- **ofdatapath** - É a implementação de um *datapath* OpenFlow, na forma de um processo. Monitora uma ou mais interfaces de redes, encaminhando pacotes entre elas de acordo com as entradas na tabela de fluxo do *datapath*. Quando utilizado com o `ofprotocol`, conecta o *datapath* a um controlador OpenFlow.

- **controller** - É um exemplo de controlador OpenFlow que pode funcionar como um *hub* ou um *switch*.

O *software* deste *switch* atualmente só executa no espaço de usuário (*userspace*), como um processo. No entanto, em versões anteriores, este tinha suporte para atuar no espaço do kernel, onde poderia ter melhor desempenho.

O motivo para escolha desta ferramenta para uso nos experimentos é que trata-se da implementação padrão de um *datapath* OpenFlow.

3.2.2 Switch QoSFlow

O switch QoSFlow é uma derivação do switch referência, porém possui a API do QoSFlow, o que possibilita a implementação de QoS nas portas dos *datapaths* sem o uso de ferramentas externas. Este switch, até o momento, só pode ser utilizado na plataforma OpenWRT (OPENWRT, 2012).

Por ser uma derivação do primeiro switch apresentado, possui os mesmos *softwares*. No entanto, para fazer o uso de sua inovadora API, utiliza mais dois componentes:

- **qf-manager** - Configura escalonadores de pacotes e/ou disciplinas de enfileiramento em um porta do switch.

- **qf-monitor** - Coleta as estatísticas e outras informações do *datapath* como identificação e nome das portas.

- **qf-conn** - Permite o `qf-monitor` e o `qf-manager` conectarem-se com um controlador de rede.

3.2.3 Open vSwitch

Segundo Pettit et al. (2010), o Open vSwitch (ou OVS) é um *switch* virtual multicamadas, projetado para ser flexível e portátil. Suporta as funcionalidades de um switch de borda avançado: visibilidade do fluxo de tráfego através de NetFlow, sFlow, e *mirroring* (SPAN e RSPAN); políticas de ACL (Access Controls Lists) e QoS; e suporte para controle centralizado. Também oferece *port bonding*[4], tunelamentos GRE e IPsec, e policiamento de tráfego por máquina virtual. Além destas funcionalidades, o Open vSwitch também oferece suporte ao protocolo OpenFlow[5].

Open vSwitch apresenta duas formas de funcionamento: os pacotes podem ser tratados no *user space*, ou tratados diretamente no kernel (*kernel space*). O modo de funcionamento preferencial é o último, que consome menos memória e permite uma maior escalabilidade. Também é notado que OVS já foi implementado em hardware real, assim não se limitando apenas a implementações a nível de sofware.

Os componentes envolvidos para instanciar e controlar um *datapath* OpenFlow são:

- **ovs-vswitchd** - É um *daemon* que controla e gerencia os switches OVS em uma máquina local.
- **ovs-vsctl** - É um programa que configura `ovs-vswitchd` através de uma interface de alto nível. Através deste é possível instanciar novos *softwares switches*, adicionar portas aos mesmos e configurar o QoS nas portas dos *datapaths*.
- **ovs-ofctl** - É um componente para monitorar e administrar os *switches* OpenFlow. Também pode mostrar o estado dos *datapaths* OpenFlow, incluindo funcionalidades, configuração e entradas na tabela de fluxo.

[4] É um termo designado para descrever o método de agregação de *link*.

[5] A meta de desenvolvimento do Open vSwitch pretende que a aplicação forneça suporte para todas as versões do protocolo OpenFlow.

3.2.4 NOX-Classic

O NOX-Classic é apresentado em Gude et al. (2008) como um sistema operacional de redes que permite que programas sejam escritos sobre a sua plataforma centralizada. Estes aplicativos enxergam a rede como apenas um único nó computacional e controlam a rede de acordo com o comportamento programado neles.

A API do NOX-Classic permite o desenvolvimento de aplicações tanto em C++ como em Python, sendo que a primeira oferece melhor performance que a segunda, conforme notado em McCauley (2012).

McCauley (2012) noticiou que, em Março de 2012, o NOX-Classic foi descontinuado, e o projeto se subdiviu em dois: NOX, que usa somente a API em C++, e o POX, que usa a API em Python.

Apesar do fim do projeto original, esta ferramenta foi elencada pois continua sendo um dos controladores mais utilizados no meio do OpenFlow.

3.2.5 NOX QoSFlow

É uma derivação do controlador NOX-Classic (versão Zaku) com a API do QoSFlow. Através deste componente é possível configurar o gerenciamento de QoS nos *datapaths* de um domínio OpenFlow.

3.2.6 Mininet

Segundo (LANTZ; HELLER; MCKEOWN, 2010), o Mininet é um ambiente para prototipação de aplicativos para SDN, onde o usuário pode instanciar a sua própria rede virtual para testar a novas funcionalidades desenvolvidas por ele.

A ferramenta para instanciar o seu ambiente virtual faz uso dos seguintes componentes:

- **Nós** - Utiliza a virtualização por *containers*, onde somente a pilha de redes é isolada

- **Enlaces** - Os enlances são feitos através de pares Ethernet virtuais (Veth), que atuam conectando dois elementos da rede.

- ***Switches*** - São utilizados os *software switches* do OVS e a versão de referência do OpenFlow.

A plataforma Mininet é ideal para os estudos sobre SDN, devido ao seu grande potencial de escalabilidade, conforme notado Lantz, Heller e McKeown (2010), e a facilidade que o usuário tem em criar as topologias neste sistema, visto que esta ferramenta fornece uma API que facilita a criação da rede, sem que haja a necessidade de cada ferramenta ser configurada manualmente. Além disto, o Mininet foi escolhido pelo OpenFlow Consortium, como a forma indicada de se começar a pesquisar o conceito de SDN (CONSORTIUM, 2012a).

3.2.7 Virtualizador LXC

O LXC (LXC, 2012) é um virtualizador de *containers*, que constrói as suas máquinas virtuais a partir do mecanismo de `chroot`, criando sistemas completos e adicionando gestão de recursos e mecanismo de isolamento à infraestrutura existente.

Dentre os diversos tipos de virtualização existentes, como a para-virtualização (Xen e KVM) e virtualização completa (VMWare e VirtualBox), a virtualização baseada em *containers* se mostra como a que menos onera o sistema hospedeiro em termos de consumo de CPU e memória (CORREA et al., 2011).

Há duas opções de virtualizadores baseados em containers que são mais referenciadas na literatura: o OpenVZ (BHANAGE et al., 2011) e o LXC (CORREA et al., 2011). Destas, o LXC se mostra como melhor opção pelo fato de suportar o OpenVswitch, o que o OpenVZ não suporta.

3.2.8 Virtualização de enlaces utilizando veth

Virtual Ethernet device (veth) é um software que cria um par de interfaces virtuais de rede em uma ligação ponto a ponto, onde é instanciada uma nova pilha de rede em sistemas

operacionais UNIX (KELLY; GASPARAKIS, 2010). Cada interface de rede deste enlance ponto a ponto é vinculada a um *software switch* ou a um *container*.

3.2.9 Switch OpenFlow em Hardware

Existem variadas soluções para se trabalhar com *datapaths* baseados em hardware. Dentre as mais usadas, uma delas faz uso de interfaces NetFPGA (NAOUS et al., 2008) combinadas a uma plataforma com arquitetura x86. Com a correta seleção de uma distribuição Linux e as devidas modificações de kernel e adição de drivers, as interfaces NetFPGAs se comportarão como switches OpenFlow. Esta solução é bastante versátil e o programador tem total controle sobre o funcionamento do datapath.

Existem também diversos fabricantes consagrados no mercado de redes que produzem *switches* dos mais variados portes capazes de funcionar como um *datapath* OpenFlow. São exemplos de fabricantes, a HP, Cisco, Juniper, Brocade etc. Neste tipo de solução, tem-se uma dependência das alterações realizadas pelos fabricantes nos firmwares de seus equipamentos para suportar o protocolo OpenFlow 1.0 (até o momento não há registro de fabricantes implementando versões mais recentes do OpenFlow).

Existe um terceiro tipo de solução de menor custo que, de uma certa maneira, combina vantagens e desvantagens das soluções anteriores. Esta faz uso da implementação do Linux OpenWRT, que é uma distribuição voltada para sistemas embarcados, tradicionalmente usada em roteadores sem fio (OPENWRT, 2012). Uma vez instalado o OpenWRT, este pode ser modificado de maneira a tornar o roteador sem fio num *switch* OpenFlow (CONSORTIUM, 2012b) de baixo custo.

O hardware escolhido foi o TP-LINK WR1043ND, que é um roteador sem fio homologado pela OpenWRT.org assim como pelo (CONSORTIUM, 2012b). Este equipamento, além de poder funcionar como *switch* OpenFlow, também pode funcionar como *switch* QoSFlow, através de procedimento similar. Em (ISHIMORI, 2012) é mostrado que o equipamento foi validado com sucesso para uso com QoSFlow. Abaixo estão as especificações do equipamento:

- **CPU** - Atheros AR9132@400MHz
- **Memória RAM** - 32 MB
- **Interfaces** - 4 interfaces LAN e 1 WAN, além da interface *wireless*.

3.3 Comparativo entre as Opções de Ferramentas

Nesta seção será realizada uma análise qualitativa das opções relacionadas a *switches* OpenFlow e ambiente de virtualização. Esta análise objetiva traçar um comparativo qualitativo entre as possíveis implementações das arquiteturas apresentadas na seção 3.2.

3.3.1 Switches OpenFlow

Dentre as opções de switches OpenFlow, o *switch* referência apresenta somente as funcionalidades padrão do protocolo OpenFlow que são:l adicionar, remover fluxos e configurar filas nas portas através do comando `dpctl`. Entretanto, não é possível configurar o QoS através do controlador, e a única configuração possível de qualidade de serviço é a garantia de banda mínima através do comando `dpctl`, que configura as filas através do `tc`. Outra limitação é que este *switch* só pode ser usado no modo *user space*, o que reduz seu desempenho frente a outras soluções.

O segundo *switch* que será avaliado será o *switch* QoSFlow, neste também é apresentada as funcionalidades padrão do OpenFlow, no entanto devido a sua API extendida, é possível configurar o QoS utilizando diferentes mecanismo de escalonamento, como o HTB e FIFO. Neste switch, o QoS pode ser configurado via um controlador QoSFlow, sem a intervenção de uma ferramenta externa.

O terceiro *switch* é o OVS. Nesta ferramenta, apesar de oferecer inúmeras funcionalidades, seu comportamento com relação a configuração de QoS é similar ao *switch* referência, pois para configurar as filas nas portas é necessário o uso de ferramentas externas, no caso do componente `ovs-vsctl`. No entanto, este *switch* também é capaz de configurar mais opções de QoS, sendo

possível ter garantias de banda mínima e limite de banda. O OVS apresenta uma vantagem com relação aos outros componentes, nele é possível ser utilizado no modo *kernel space*.

Tendo em vista as qualidades levantadas, o *switch* que melhor atende aos requisitos do DCN, é o QoSFlow, uma vez que este é capaz de configurar o QoS automaticamente, sem a intervenção de uma ferramenta externa. O OVS é a melhor opção como *software switch* pois além de oferecer mais opções de QoS que o *switch* referência, este possui melhor escalabilidade devido ao modo *kernel space*.

3.3.2 Ambientes

Ambientes, neste trabalho, é o termo designado para as ferramentas de virtualização que ajudam a compor os nós clientes de uma rede. Para esta pesquisa foram escolhidos dois ambientes: LXC e o Mininet, ambos baseados na na virtualização por *containers* para instanciar os nós da rede.

Apesar dos virtualizadores serem semelhantes, o Mininet apenas isola a pilha de rede, e o restante dos componentes do sistema são compartilhados com o hospedeiro, por exemplo o sistemas de arquivos. Já o LXC faz o isolamento completo, ou seja cada nó possui o seu próprio sistema de arquivos e sua própria pilha de rede.

O impacto destas opções na virtualização é que o Mininet tem perda de funcionalidades, no entanto a sua máquina virtual é extremamente leve, ou seja, consome menos memória e processamento do hospedeiro as suas máquinas virtuais. Já o LXC, por isolar mais elementos, possui mais funcionalidades , entretanto, ao escolher esta opção, as suas máquinas virtuais são mais pesadas (consomem mais memória e processamento), o que pode afetar no desempenho dos *software switches* que utilizam o *user namespace*.

4 *Análise Experimental de DCN sobre um domínio OpenFlow*

Neste capítulo serão mostrados os experimentos que buscam verificar e validar as funcionalidades que podem ser utilizadas para provisionar circuitos em um domínio OpenFlow. A primeira etapa deste capítulo consiste numa avaliação dos *software switches*, buscando verificar como podem ser utilizados os mecanismos de QoS em diferentes ambientes. Serão mostrados três experimentos com objetivos distintos, envolvendo a verificação das funcionalidades de QoS para um ambiente DCN com *software switches*, a perda de desempenho em ambientes totalmente virtualizados e a instalação de mecanismos de QoS num *hardware switch* QoSFlow, através de ações do próprio controlador. Previamente aos experimentos, serão mostradas as abordagens utilizadas e algumas limitações dos dispositivos e cenários disponíveis para os testes, tendo como base um cenário real de uso de DCN.

Por fim, será detalhada a integração entre o OSCARS 0.6 e a solução QoSFlow para possibilitar DCNs sobre domínios OpenFlow.

4.1 Abordagens utilizadas

Nesta seção serão apresentadas e discutidas as metodologias utilizadas, as medidas extraídas para a realização dos experimentos e as limitações dos dispositivos e cenários disponíveis para os testes.

4.1.1 Metodologia experimental

Em uma rede DCN, os circuitos provisionados pelos usuários devem ter sua banda garantida, ou seja, a taxa de envio nestes circuitos pode excursionar até a banda máxima reservada pelo usuário, sem que haja perda de pacotes ao longo do caminho do circuito ou grande variação do atraso (*jitter*). Da mesma maneira a taxa de envio não pode ultrapassar a banda reservada sob pena de perda de pacotes (policiamento de tráfego). Logo, a partir desta situação, os *switches* OpenFlow devem garantir a banda reservada e, juntamente, realizar o policiamento da taxa de envio no circuito, sendo que esta limitação corresponde a banda reservada. Caso estas exigências não sejam atendidas, o equipamento será impróprio para uso numa arquitetura DCN.

A duração dos experimentos (30 segundos para cada rodada) foi baseada na RFC 2889 (Mandeville e Perser (2000)), que fornece metodologias para aferir as capacidades de encaminhamento, controle de congestionamento, latência e outras características em *switches* legados. As medidas extraídas envolvem vazão máxima de cada circuito configurado e o consumo de CPU e de memória dos *software switches*. Para extração destas medidas, as ferramentas `iperf` e `pidstat` foram usadas como apoio. O `iperf` é utilizado para gerar tráfego num circuito e verificar a vazão máxima alcançada por este tráfego. O `pidstat` é utilizado para extrair dados sobre utilização de CPU e de memória do processo que executa os *software switches*.

4.2 Experimentos relativos ao suporte de QoS

Estes experimentos visam avaliar os *datapaths* OpenFlow relacionados no Capítulo 3. Os aspectos à serem avaliados envolvem as funcionalidades necessárias para atender às exigências de uma DCN. Também será verificado a redução de desempenho que ocorre na implementação de uma DCN em ambiente totalmente virtualizado. Tal avaliação, permite verificar ate que ponto é possivel emular um arquitetura DCN utilizando virtualização de sistema no caso relativo aos recursos computacionais utilizados nos experimentos.

Na avaliação dos possíveis *datapaths* OpenFlow, é dada uma atenção especial aos *software switches* e suas funcionalidades. Isto porque é possível embarcar tais *softwares* em um *hard-*

ware dedicado a comutação ou mesmo em *hardwares* comerciais tipicamente utilizados como *desktops* e servidores, apropriadamente adicionados de interfaces de rede.

Os experimentos realizados para avaliação da funcionalidade dos *software switches*, que são aplicações que atuam como *switches* virtuais, no entanto podendo controlar tanto as portas físicas de um computador como portas virtuais.

Os seguintes equipamentos foram utilizados na realização dos experimentos:

- Desktop:
  - CPU: Intel Core i7-2600
  - Memória RAM: 8.00 GB
  - SO Base: Ubuntu 11.04
  - 2 interfaces de rede.
    - Intel 82579V Gigabit Network Connection - 1.0Gbps
    - D-Link DFE-520TX 10/100Mbps - 100Mbps
  - Ambientes virtuais:
    - VirtualBox executando o Controlador (NOX); SO CentOS 6.03.
    - VirtualBox executando o virtualizador LXC, *switch* referência e OVS 1.7.1; SO Ubuntu 12.04.

O experimento que foi realizado para avaliar o *switch* QoSFlow, utilizou os seguintes recursos:

- Desktop:
  - CPU: Intel Core i7-2600
  - Memória RAM: 8.00 GB
  - SO Base: Ubuntu 11.04

  - 2 interfaces de rede.
    - Intel 82579V - 1 Gbps
    - D-Link DFE-520TX - 100Mbps
  - Ambientes virtuais:
    - VirtualBox executando o Controlador (NOX QoSFlow); SO CentOS 6.03.

- Hardware Switch:
  - TP-LINK WR1043ND com OpenWRT.

- Notebooks:
  - Macbook Pro
    - SO Base: Mac OS
    - 1 interface de rede.
      - Broadcom Corporation NetXtreme - 1 Gbps
    - VirtualBox executando dois sistemas finais; SO Ubuntu Server 12.04.
  - Macbook
    - SO Base: Mac OS
    - 1 interfaces de rede.
      - Broadcom Corporation NetXtreme - 1 Gbps
    - VirtualBox executando dois sistemas finais; SO Ubuntu Server 12.04.

4.2.1 Experimento - Funcionalidades de QoS

4.2.1.1 Objetivo

O experimento objetiva verificar as funcionalidades de policiamento de banda e garantia de banda mínima disponíveis nos *switches* OpenFlow, Open vSwitch e a implementação referência de um *switch* OpenFlow. A Figura 17 mostra o arranjo realizado para este experimento.

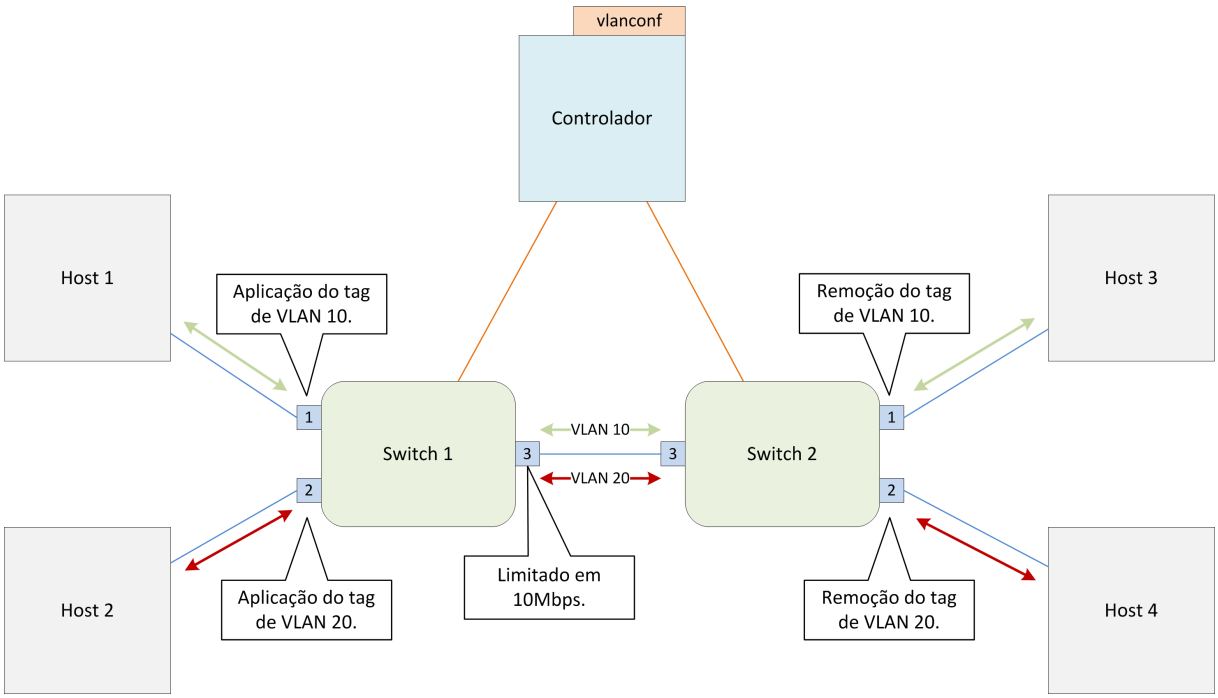

Figura 17: Arquitetura utilizada no experimento de funcionalidades de QoS.

4.2.1.2 Topologia e Configuração

A topologia envolve dois *switches* OpenFlow interconectados, cada qual dando acesso para dois *hosts* conectados com interfaces distintas. Os enlaces entre os *switches* e os *hosts* possuem capacidade de 1 Gbps, e o enlace entre os *switches* possui capacidade máxima de 10 Mbps. Estas capacidades foram configuradas utilizando o mecanismo `tc` do Linux.

Sobre a configuração desta topologia, dois circuitos serão criados, desta forma, interconectando *hosts* distintos em cada *switch*: H1 conectado com H3 e H2 com H4. Para cada circuito é associado um *tag* de VLAN específico, configurado através do componente `vlanconf` acoplado ao controlador OpenFlow NOX-Classic. Este *tag* é adicionado aos pacotes que atravessam de um *switch* para outro, e removidos quando os pacotes são entregues aos respectivos *hosts* de destino.

A configuração de QoS foi realizada estaticamente, ou seja, sem intervenção do controlador, e através de um software externo do OpenFlow, o `dpctl`. Foi utilizado um mecanismo de escalonamento HTB na interface de saída do *switch* 1 que é conectada ao *switch* 2, realizando limitação de banda, quando disponível, ou fornecendo garantia de banda mínima.

Por fim, para a geração de tráfego neste experimento foi utilizada a ferramente `iperf` nos *hosts*.

4.2.1.3 Policiamento de banda

Para os testes de policiamento de banda foram instanciados dois circuitos concomitantemente para avaliar o policiamento. Estes experimento foi dividido em duas etapas, a primeira os equipamentos devem garantir 5 Mbps para cada circuito, desta forma saturando o enlace de 10 Mbps, e a segunda etapa os equipamentos devem garantir e policiar a banda dos circuitos criados em 3 Mbps para cada, cenário em que o enlace não é saturado. Nestas etapas foram feitas duas variações: na primeira H1 e H2 enviam tráfego UDP (com o tráfego do `iperf` configurado para 10 Mbps em ambos), na segunda variação H1 faz o envio de tráfego UDP enquanto que H2 faz o envio de tráfego TCP.

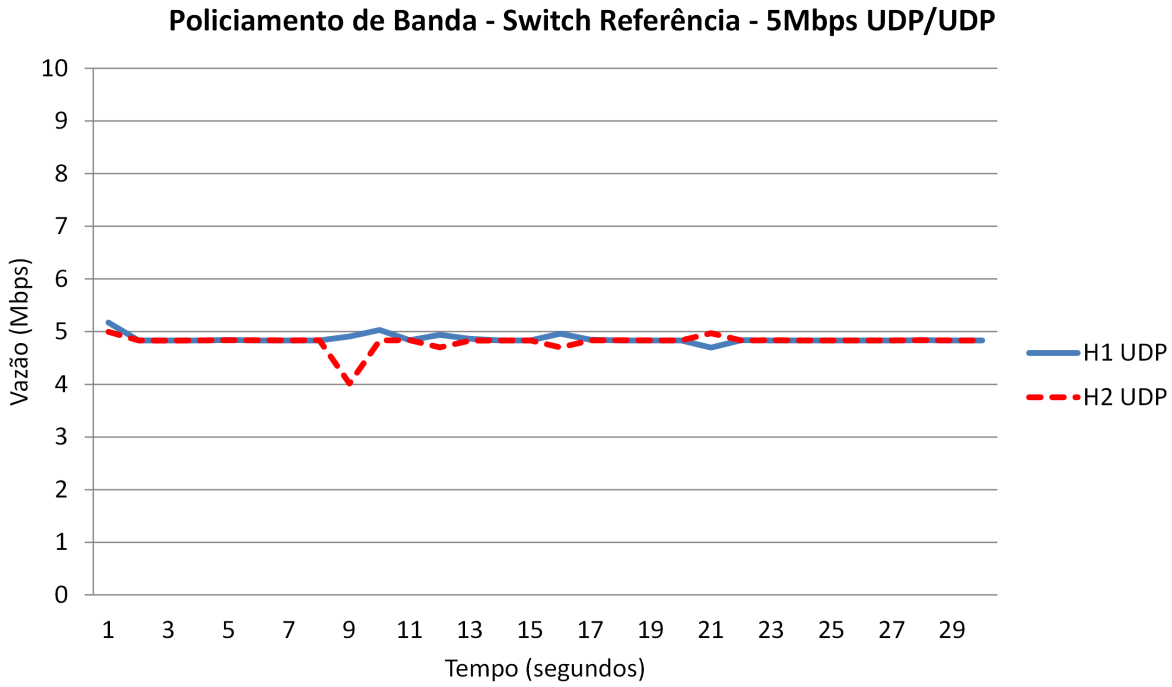

Figura 18: Resultado referente ao policiamento de banda - Vazão de banda do *software switch* referência com banda exigida 5 Mbps usando UDP.

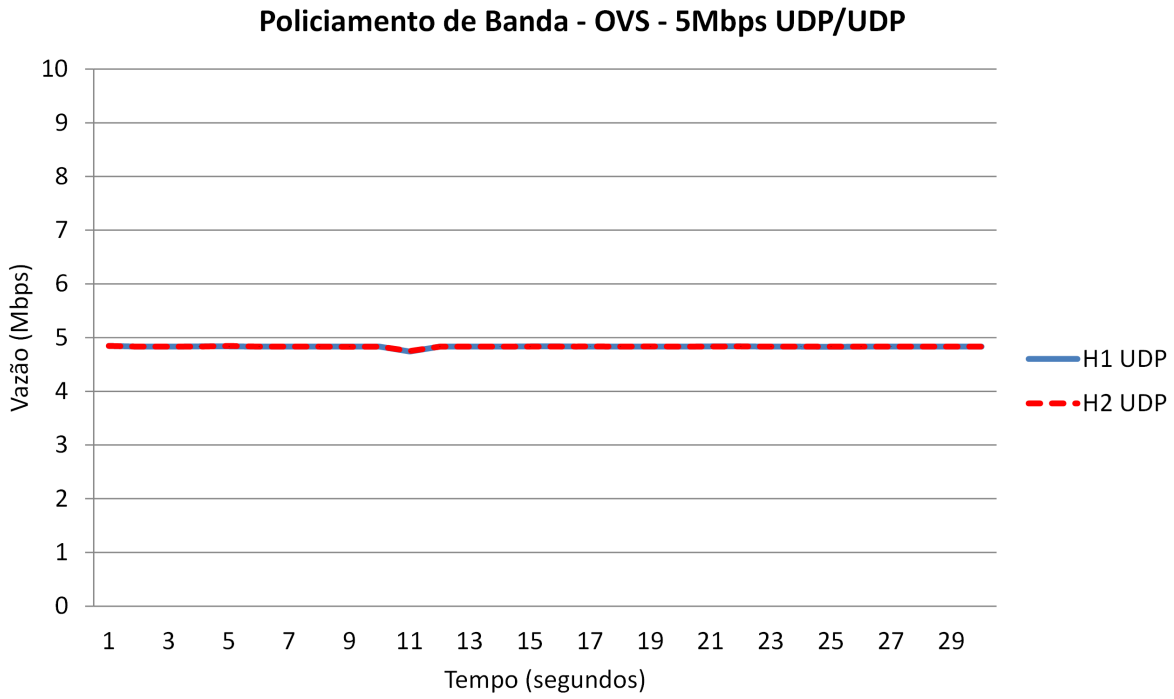

Figura 19: Resultado referente ao policiamento de banda - Vazão de banda do OVS exigida 5 Mbps usando UDP.

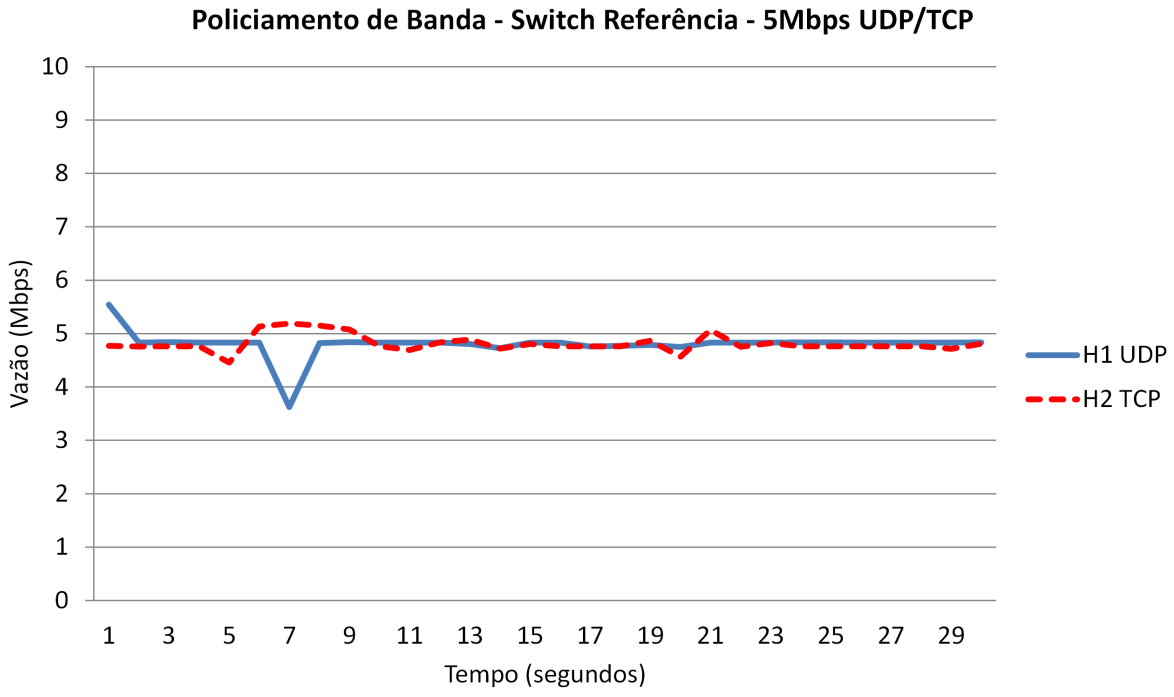

Figura 20: Resultado referente ao policiamento de banda - Vazão de banda do *software switch* referência exigida 5 Mbps usando TCP/UDP.

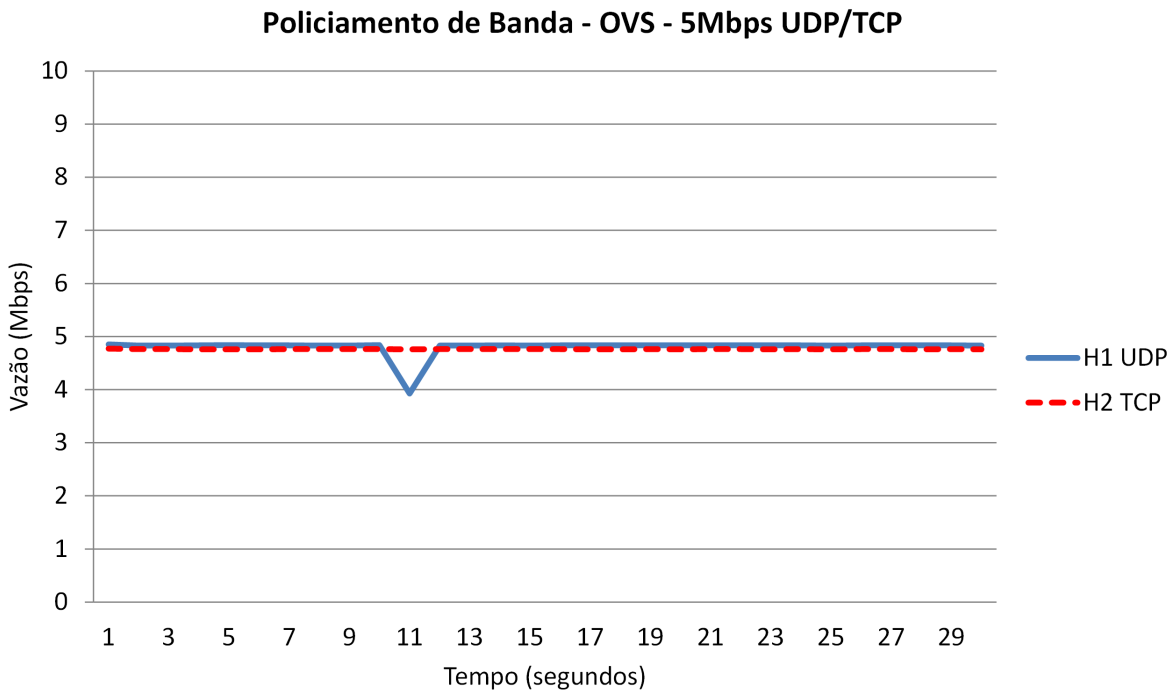

Figura 21: Resultado referente ao policiamento de banda - Vazão de banda do OVS exigida 5 Mbps usando TCP/UDP.

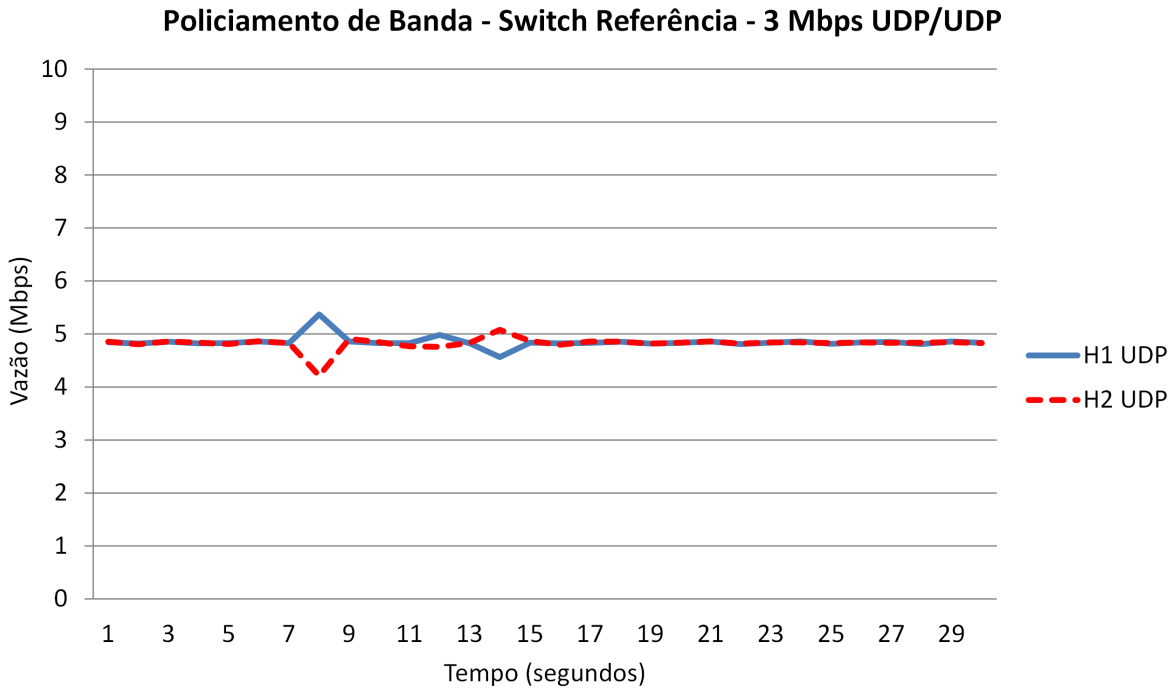

Figura 22: Resultado referente ao policiamento de banda - Vazão de banda do *software switch* referência exigida 3 Mbps usando UDP.

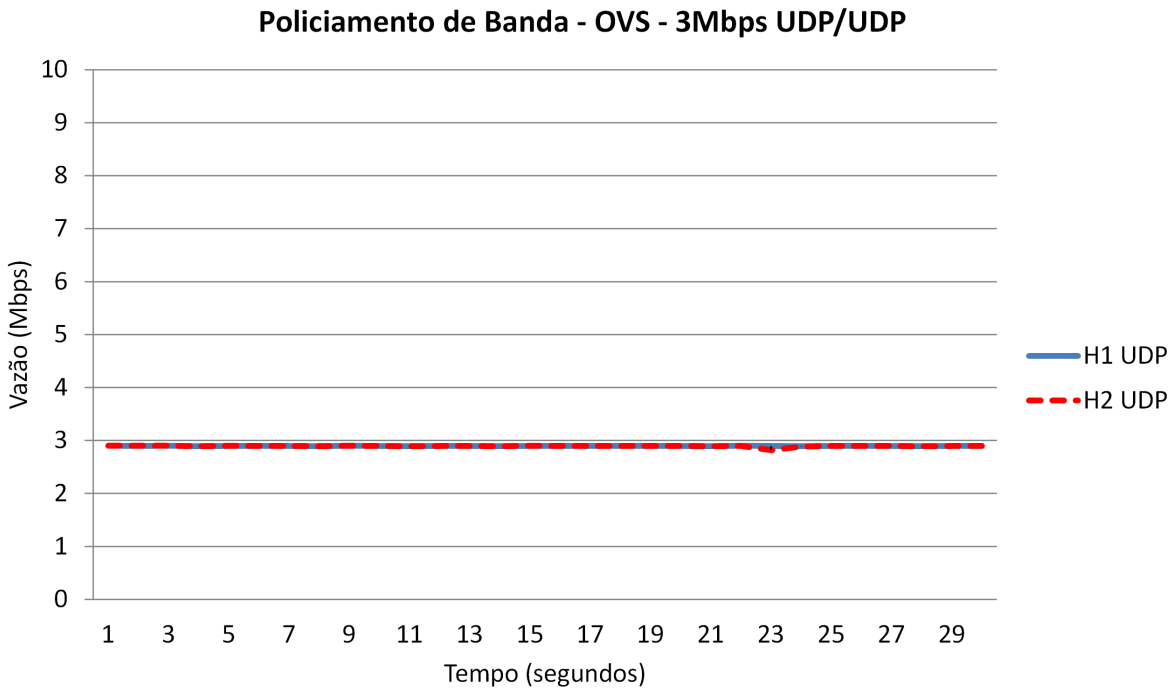

Figura 23: Resultado referente ao policiamento de banda - Vazão de banda do OVS exigida 3 Mbps usando UDP.

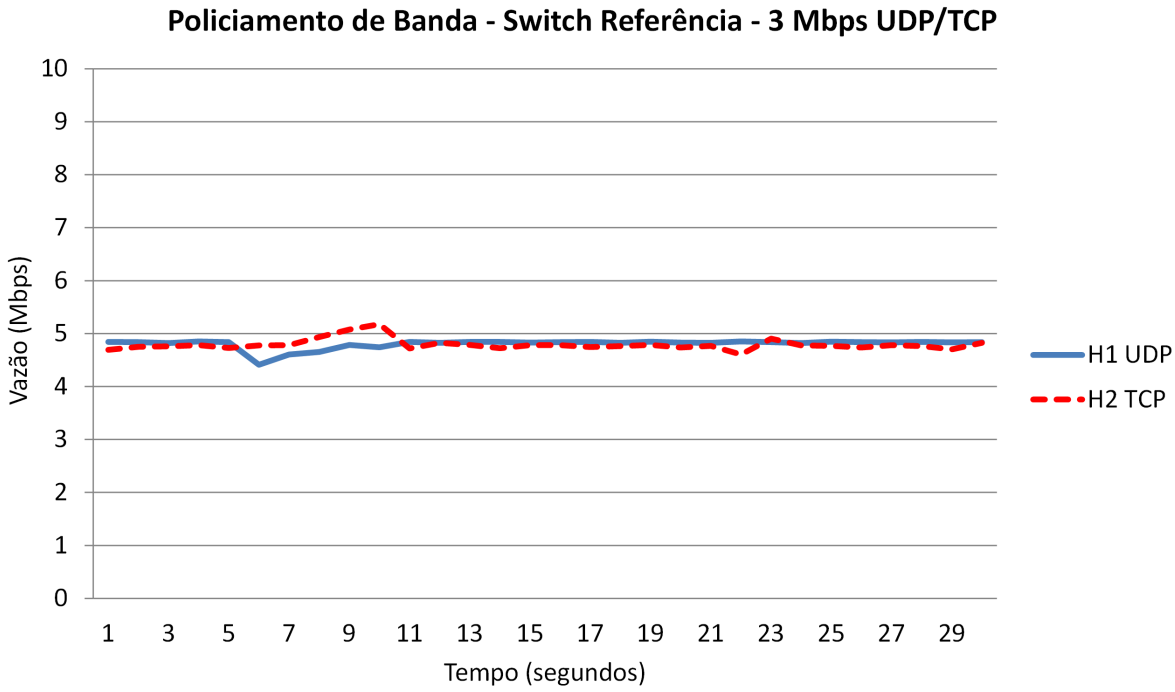

Figura 24: Resultado referente ao policiamento de banda - Vazão de banda do *software switch* referência exigida 3 Mbps usando TCP/UDP.

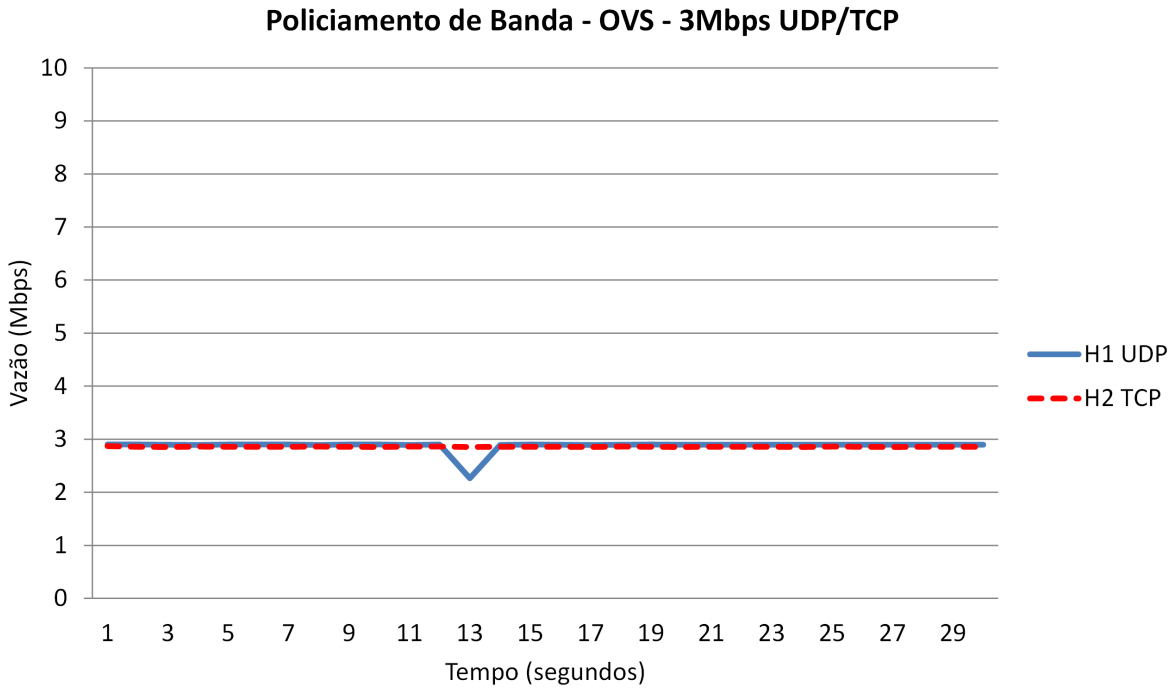

Figura 25: Resultado referente ao policiamento de banda - Vazão de banda do OVS exigida 3 Mbps usando TCP/UDP.

4.2.1.4 Garantia de banda mínima

Para a avaliação da garantia de banda mínima, é criado um circuito onde H1 faz o envio de tráfego TCP para o H3 por 30 segundos, assim criando um tráfego "de fundo". Após 10 segundos da instanciação do primeiro circuito é provisionado um segundo circuito, de tráfego UDP, do H2 para o H4, com duração de 10 segundos. Neste cenário, deve ser garantida uma banda mínima de 7 Mbps para o circuito proveniente de H1, enquanto que o circuito oriundo de H2 deve ter uma banda garantida de 3 Mbps.

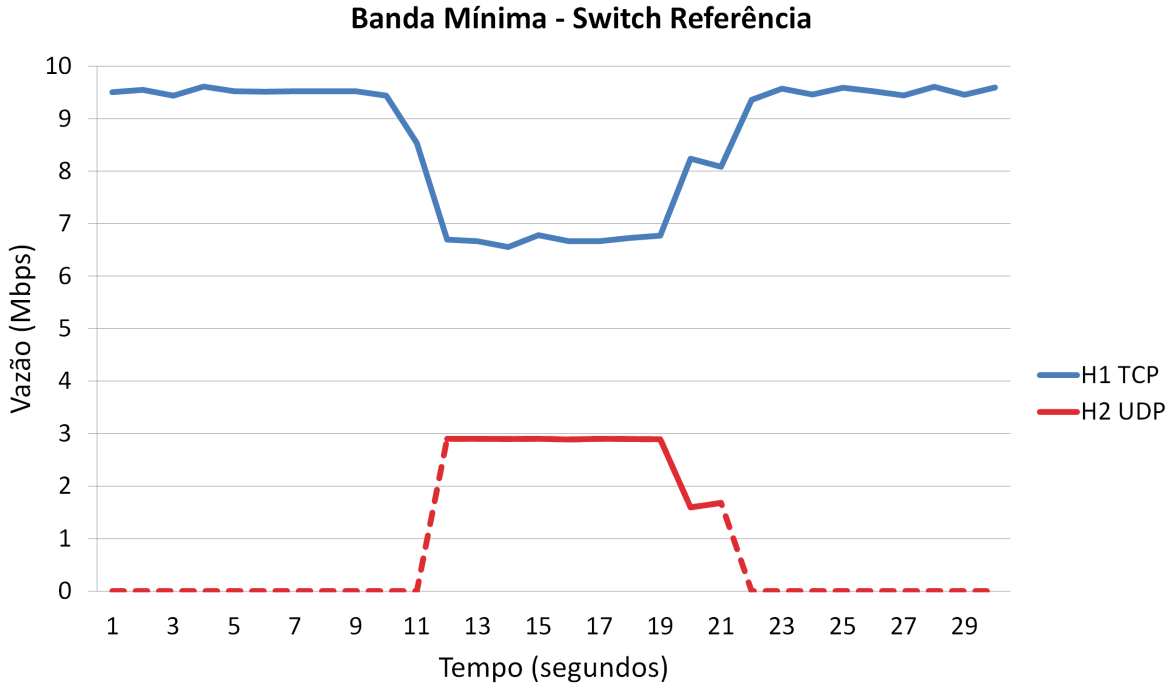

Figura 26: Resultado referente à banda mínima do *switch* referência.

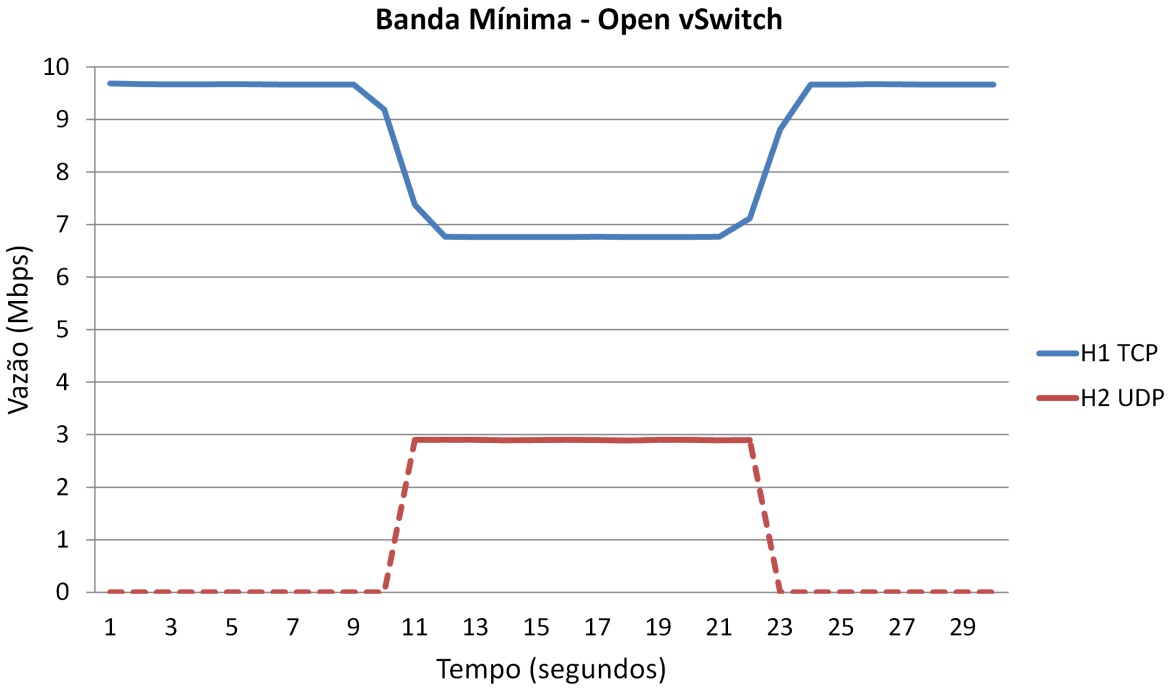

Figura 27: Resultado referente à banda mínima do OVS.

4.2.1.5 Resultados

Conforme anteriormente informado, o *software switch* referência não tem a capacidade realizar o policiamento da banda dos circuitos. Portanto, é esperado que a configuração de banda mínima garantida, unicamente, não sirva de parâmetro para se obter isolamento entre os circuitos. A incapacidade do *switch* referência de fazer o policiamento pode ser verificada nas Figuras 22 e 24, onde cada circuito extrapola o policiamento exigido (3 Mbps para cada circuito), dado que a saturação do enlace se dá em 10 Mbps. Nestes gráficos, cada circuito teve em média 5 Mbps, diferente dos 3 Mbps exigidos, assim não atendendo o policiamento exigido pelo DCN. Nas figuras 18 e 20 isto não é observado, pois o policiamento exigido é obtido através da funcionalidade de banda mínima que o equipamento oferece, uma vez que para cada circuito é garantido 5 Mbps resultando na saturação do enlace.

No OVS, o policiamento da banda foi empregado corretamente, conforme indicado nas Figuras (19, 21, 23 e 25). Isto ocorre, pois o OVS possui esta funcionalidade de policiamento da banda através do emprego do escalonador HTB, diferente do *software switch* referência, que apesar de fazer uso do HTB, apenas configura o policiamento de banda mínima.

Com relação aos resultados obtidos na avaliação de banda mínima, ambos os *softwares switches* conseguiram garantir a banda mínima exigida pelo circuito, tais resultados podem ser verificados nas Figuras 27 e 26.

Para cada rodada foi calculada a taxa média de envio para cada intervalo de 1 segundo. Cada ponto dos gráficos exibidos nas figuras representa a média das taxas médias calculadas para o respectivo intervalo.

Uma vez demonstrada a inadequação do *switch* de referência para o caso geral de uso de DCNes, os experimentos seguintes baseados em ambiente virtualizado utilizarão apenas como *switch* OpenFlow o OVS no ambiente LXC.

4.2.2 Experimento - Performance

4.2.2.1 Objetivo

Este experimento procura verificar a perda de desempenho ao se realizar uma implementação de DCN sobre domínio OpenFlow num ambiente totalmente virtualizado. Sabe-se que este tipo de cenário se presta apenas para uma emulação de cenário real e, devido a isto, é esperado que o desempenho num cenário utilizando *switches* físicos seja superior. Todavia, obter dados quantitativos sobre limites de desempenho num ambiente específico serve como referência para novos experimentos que precisem ser realizados no mesmo contexto.

4.2.2.2 Topologia e Configuração

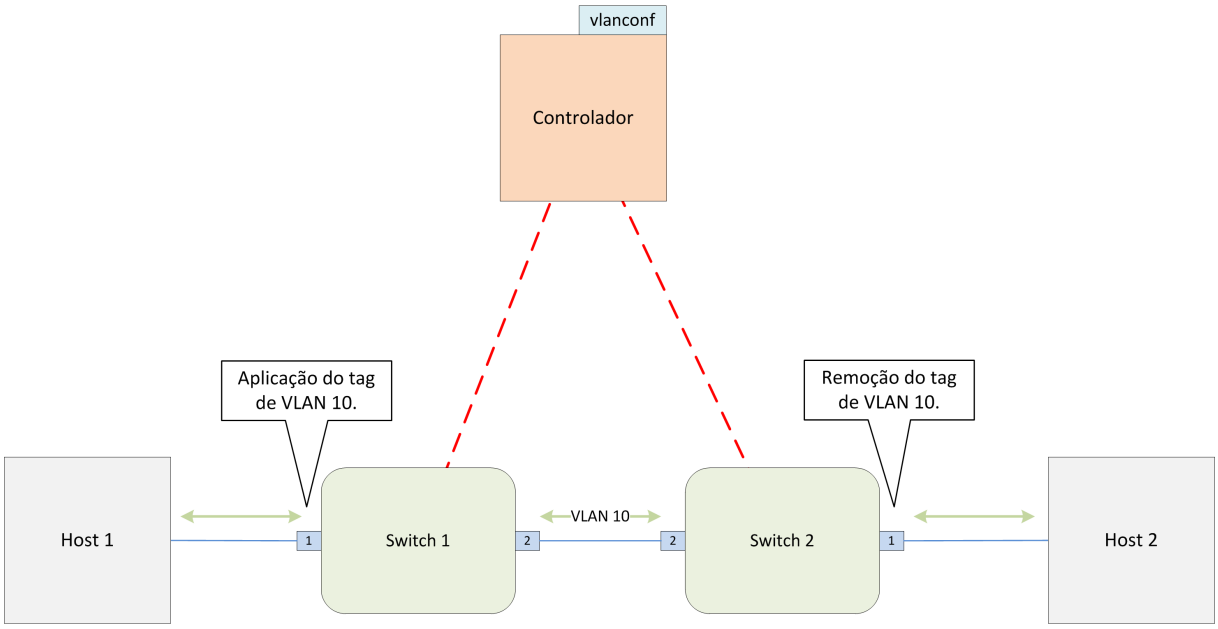

Figura 28: Arquitetura utilizada no experimento de performance.

O experimento foi dividido em duas partes. Na primeira, procurou-se verificar a vazão máxima atingida por uma aplicação entregando tráfego para outra num ambiente virtualizado composto por dois *switches* OpenFlow, implementados com OVS em ambiente LXC, conforme ilustrado na Figura 28. Neste cenário foram realizados testes com o `iperf` entregando, em rodadas distintas, tráfego UDP e TCP, ambos os casos com e sem uso de *tag* de VLAN. A segunda parte do experimento, ilustrada pela Figura 29, é similar a primeira etapa, porém sete

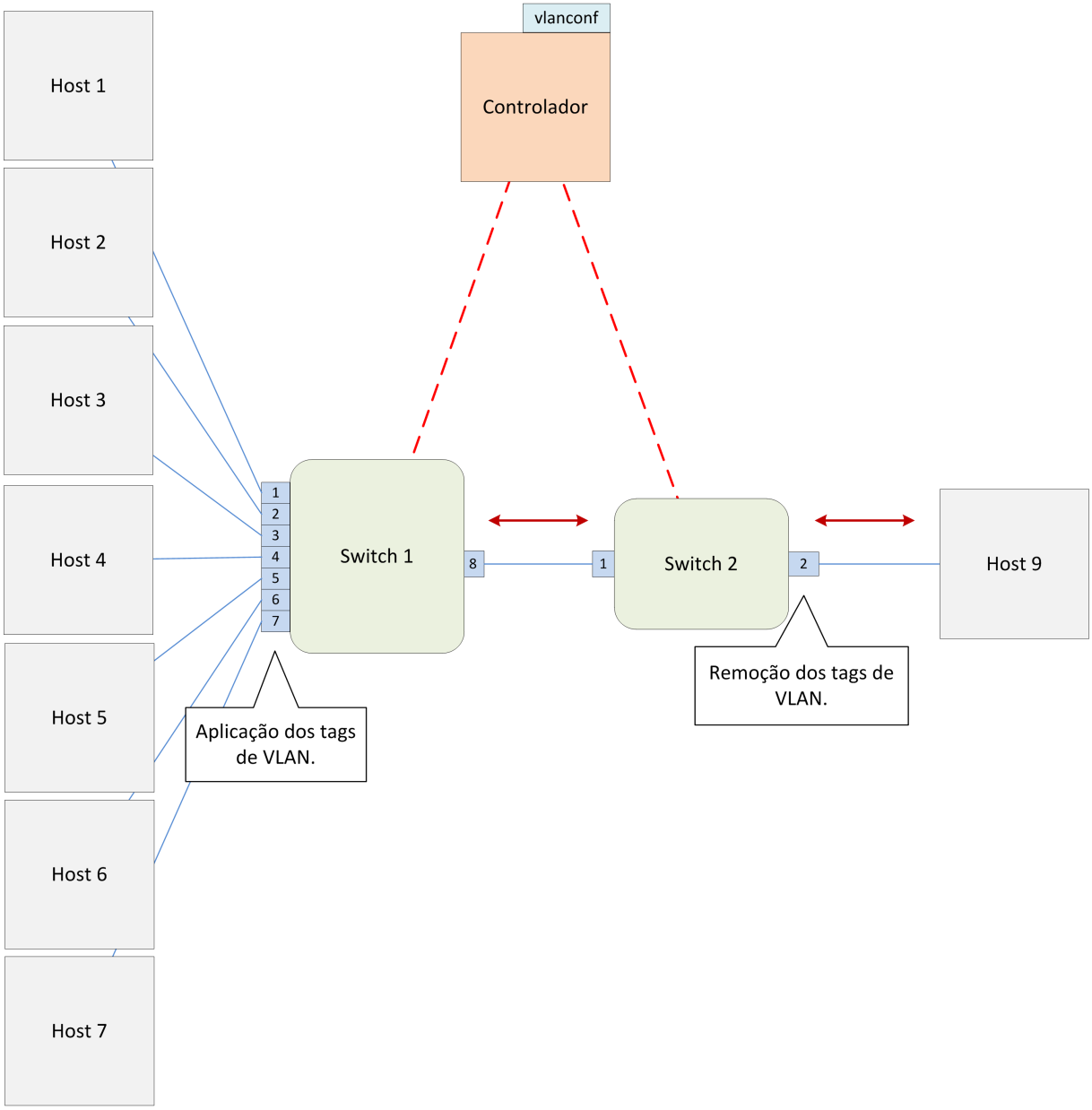

Figura 29: Arquitetura referente a segunda etapa do experimento de performance.

circuitos são instanciados, onde cada qual é associado a um *host* ligado ao *switch* 1 por uma porta específica. Em cada porta do *switch* 1, em que há um *host* conectado, foi configurada uma fila, via o escalonador HTB, onde as bandas máxima e mínima são fixadas em 100 Mbps. Por fim, em ambas as topologias os enlaces foram implementados através do *software* `veth` do Linux e possuem capacidade máxima de 1 Gbps.

4.2.2.3 Resultados

Os resultados apresentados nesta seção foram extraídos após 30 rodadas de 30 segundos, e o seu cálculo final foi através de uma média entre os valores obtidos.

A Tabela 5, referente ao primeiro experimento, apresenta os resultados de vazão máxima para envio de pacotes com *tag* e sem *tag*. No ambiente utilizado, a capacidade máxima atingida fica em torno 10% da capacidade nominal dos enlaces virtuais, ou seja, próximos a 100 Mbps, tanto nos testes com TCP e com UDP.

A Tabela 6, referente a segunda etapa do experimento, traz os resultados de vazão máxima utilizando o tráfego do tipo UDP (gerado através da aplicação `iperf`). Como se pode verificar, neste segundo teste a vazão total (somatório das médias de envio de cada *host*) obtida entre os *switches* 1 e 2 foi cerca de 30% superior aos resultados do primeiro teste (valor total de: 130,22 Mbps).

A Tabela 7, referente ao Experimento 2, apresenta o consumo médio de CPU e de memória do processo que implementa os *switches* OVS. É importante ressaltar que este processo é responsável pelo gerenciamento de todas as instâncias de *switches* OVS virtualizados no ambiente. A baixa performance na vazão deve-se ao *network namespace* (LXC, 2012) do LXC, que é um componente responsável pela virtualização de redes, não estar adequado ao kernel do sistema operacional Linux utilizado, assim isentado o OVS pela baixa performance, já que este também apresenta baixo consumo de CPU e memória.

	TCP	UDP
Com *tag*	93,42 Mbps	101,81 Mbps
Sem *tag*	94,28 Mbps	101,55 Mbps

Tabela 5: Média da vazão máxima da primeira avaliação do experimento.

Host 1	Host 2	Host 3	Host 4	Host 5	Host 6	Host 7
18,65	18,85	18,61	18,54	18,57	18,57	18,43

Tabela 6: Média da vazão máxima (em Mbps) da segunda avaliação do experimento 2.

	CPU	Memória
OVS	0,35%	0,22%

Tabela 7: CPU e Memória da segunda avaliação do Experimento 2.

4.2.3 Experimento - Funcionalidades do QoSFlow

4.2.3.1 Objetivo

O objetivo deste experimento é atestar as funcionalidades do QoSFlow que permitem a aplicação de políticas de QoS através de um controlador OpenFlow, no caso, um NOX-Classic modificado para conter uma API QoSFlow.

4.2.3.2 Topologia e Configuração

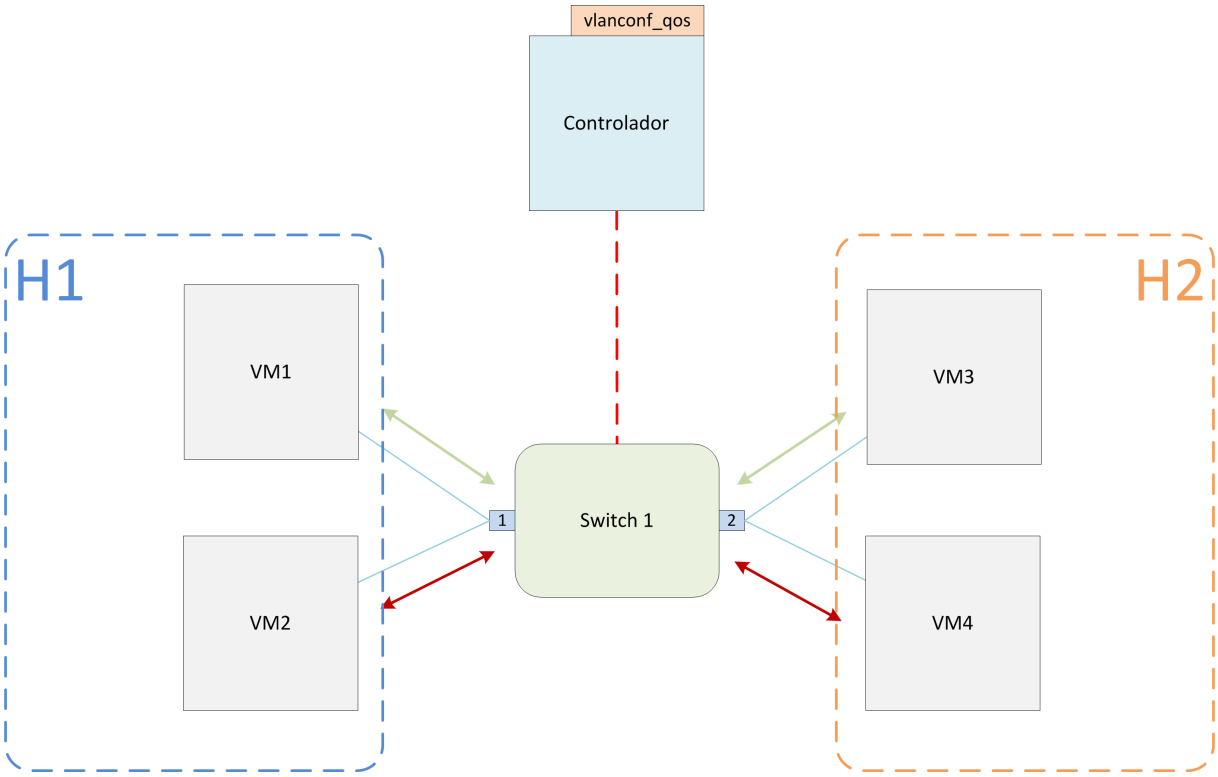

Figura 30: Arquitetura utilizada no experimento 3.

A Figura 30 traz a topologia empregada, que consiste de um arranjo simples de dois *hosts* (H1 e H2) ligados a um *switch* QoSFlow. Em cada *host*, duas máquina virtuais são criadas e é gerado via `iperf` um tráfego UDP entre pares de VMs em *hosts* distintos. O *switch* QoSFlow é controlado pelo controlador NOX QoSFlow que, através de uma aplicação que faz uso da

API do QoSFlow, utiliza o mecanismo de HTB para instanciar duas filas na porta de saída do *switch* QoSFlow para o *host* H2. Foi utilizado como *switch* QoSFlow o *hardware* TP-LINK WR1043ND com o sistema OpenWRT embarcado e alterado para se comportar como um *datapath* OpenFlow.

Para este experimento, foi utilizado o componente `vlanconf`, aplicação desenvolvida para este trabalho que faz uso da API do QoSFlow, por onde foram instanciadas filas com a disciplina HTB. Neste experimento são propostos dois testes, um de policiamento de banda e um de garantia de banda mínima.

Para o experimento de policiamento de banda, através do `vlanconf` foi configurada um policiamento de banda igual a 3 Mbps para o fluxo entre a VM1 e a VM3 e 1 Mbps para o fluxo entre a VM2 e a VM4. Todo o tráfego gerado foi limitado mediante parametrização do `iperf` em 10 Mbps. Para a diferenciação dos fluxos foi utilizado o endereço MAC de origem da VM1 e da VM2.

No experimento de banda mínima foi instanciado uma garantia de banda mínima de 7 Mbps para o fluxo entre a VM1 e a VM3, enquanto que no fluxo da VM2 para a VM4 foi instanciada uma garantia de banda de 3 Mbps. Neste cenário, primeiro é instanciado do fluxo VM1-VM3, com duração de 30 segundos, e após 10 segundos é instanciado o fluxo VM2-VM4, com duração de 10 segundos. Neste cenário o `iperf` na VM1 e VM2 gera tráfego UDP com vazão de 20 Mbps.

4.2.3.3 Resultados

A Figura 31 demonstra que o policiamento de banda foi realizado conforme o proposto. No gráfico apresentado foi realizada uma única rodada com duração de 30 segundos apresentado o policiamento de banda aplicado através do QoSFlow.

A aplicação de banda mínima é representada pela Figura 32 que apresenta uma única rodada de 30 segundos. Conforme apresentado no gráfico é possível verificar que a banda mínima exigida foi respeitada para os dois circuitos.

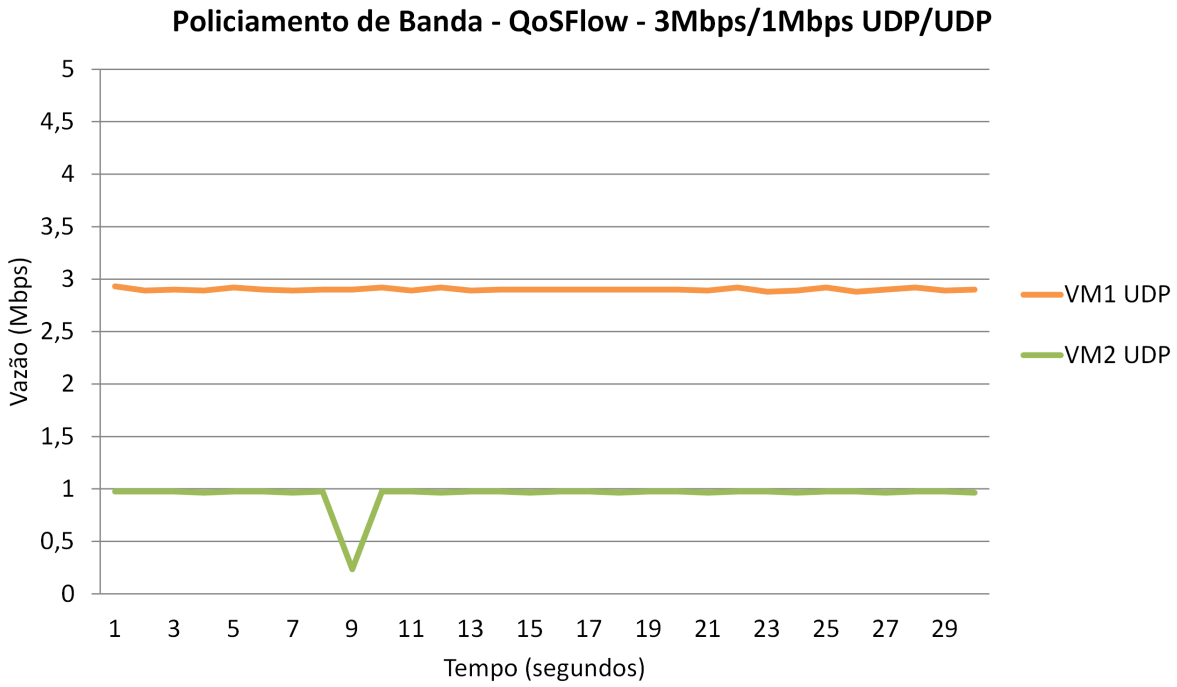

Figura 31: Teste de vazão realizado no experimento 3 com bandas requeridas de 3 Mbps e 1 Mbps.

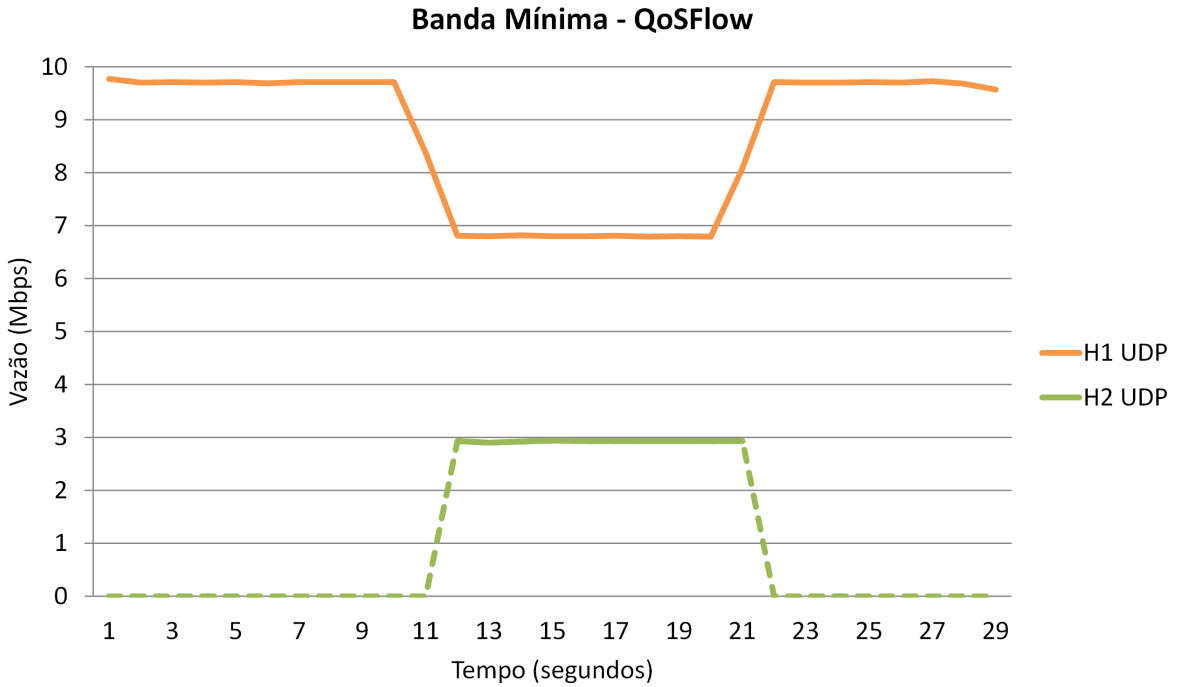

Figura 32: Teste de vazão realizado no experimento 3 com bandas requeridas de 3 Mbps e 1 Mbps.

Conforme apresentados pelas Figuras 31 e 32 pode se verificar que o propósito do arcabouço do QoSFlow conseguiu ser atingido através da ação de um aplicativo de rede ligado ao

controlador NOX QoSFlow.

4.2.4 Limitações dos dispositivos e cenários de teste

Ao longo dos experimentos, foram constatadas diversas limitações de funcionalidades e de desempenho relacionadas aos dispositivos (*switches* OpenFlow) e aos ambientes de virtualização. Segue abaixo as descrições destas limitações:

- O *software switch* de referência (OpenFlow 1.0) garante apenas a banda mínima mas não limita a banda máxima, obrigando que haja outro dispositivo diferente fazendo policiamento na entrada do domínio OpenFlow. A ausência desta funcionalidade pode trazer riscos na garantia de isolamento mútuo entre os circuitos e entre outras possíveis classes de tráfego (por exemplo, trafégo de produção não vinculado a circuitos).

- O *switch* de referência implementa oito filas por interface de saída, porém, por razões de implementação, isto não ocorre no Open vSwitch. Todavia, sabe-se que o valor de oito filas por porta é amplamente utilizado nos *switches* legados de diversos fabricantes (muitos deles têm limitação de 4 filas por porta), não impedindo o uso destes em arquiteturas de serviços diferenciados, de maneira que não deve afetar o uso em DCNs. Sabe-se também que, quanto maior o número de filas por porta, maior é o comprometimento do dispositivo em termos de desempenho.

- Durante a execução dos experimentos, foi percebido que, em distribuições Linux mais recentes (Ubuntu 11.10 e Ubuntu 12.04), o *switch* referência não consegue encaminhar os pacotes de acordo com o *tag* de VLAN. Ou seja, a partir do *tag* de VLAN, o *switch* de referência não consegue determinar para qual porta encaminhar o pacote.

- O ambiente de virtualização do Mininet, máquina virtual versão: mininet-vm-ubuntu11.10-052312 , possui uma versão de OVS muito antiga (1.2) e incapaz de realizar configurações de QoS, o que impede o uso do Mininet com OVS para a virtualização de um ambiente DCN.

- No ambiente de virtualização LXC, a implementação do *switch* de referência apresenta vazão máxima bastante inferior à implementação do OVS, conforme pode ser visto no experimento 2. Esta redução no desempenho não se verificou quando o *switch* de referência é usado no Mininet. Uma explicação para isto reside nas diferentes formas de como se dá a virtualização no LXC e no Mininet. No LXC, os *containers* virtualizam a pilha de redes e apresentam isolamento do *filesystem*, enquanto que no Mininet a virtualização por *containers* replica apenas a pilha de redes, consequentemente apresentando uma virtualização que consome menos processamento e memória, porém com menos funcionalidades.

- O *hardware switch* usado para o experimento com QoSFlow, um roteador wireless TP-LINK WR1043ND rodando o sistema OpenWRT alterado conforme (ISHIMORI, 2012), apresentou problemas na aplicação de *tag* de VLAN. Todavia, isto não o inviabiliza para uso em DCN. No caso, o circuito deverá ser individualmente reconhecido por outros parâmetros diferentes do *tag* de VLAN. Vale notar que este é um problema referente à implementação do *switch* QoSFlow no equipamento indicado, o que pode não ocorrer em outras implementações que venham a surgir.

4.3 Integração entre as soluções OSCARS e QoSFlow

Conforme descrito na 3.1.1, o OSCARS utiliza mensagens JSON para instruir o controlador NOX como realizar uma configuração de circuito num domínio OpenFlow. A Figura 33 faz uma representação gráfica desta integração.

Nestas mensagens JSON são enviadas as seguintes informações (Anexo B):

- Tipo de ação:
  - ***Setup*** - Instancia um circuito pela rede.
  - ***Teardown*** - Finaliza um determinado circuito pela rede.
  - ***Modify*** - Altera um circuito.
  - ***Verify*** - Verifica se o circuito foi provisionado.

- O identificador de domínio, chamado GRI.
- A largura de banda a ser reservada.
- IPs dos switches ao longo do caminho do intradomínio.
- Portas dos swichtes ao longo do caminho intradomínio.
- Identificador de VLAN (*tag*).

O controlador NOX, no caso da arquitetura básica, não utiliza as informações de reserva de banda para estabelecimento de QoS, o que é realizado de forma estática e externa ao controlador através do `dpctl`. Mensagens de resposta do NOX para o OSCARS também são enviadas no formato JSON contendo status de sucesso ou falha na execução das ações.

Em se tratando da arquitetura envolvendo OSCARS e QoSFlow, as mensagens JSON enviadas pelo OSCARS devem ser tratadas de maneira a se extrair os parâmetros necessários para serem passados para a API do QoSFlow, e também para serem usados pelo componente `vlanconf`, conforme mostrado na Figura 16. Dentre as possíveis formas de se implementar este tratamento, a que se mostrou mais vantajosa é a de criar um novo componente para o QoSFlow. Este novo componente, o QoS-OSCARS, seria gerado a partir de uma modificação do componente `QoSFlow Agent`, que é preparado para receber mensagens JSON contendo as políticas de QoS a serem aplicadas num domínio OpenFlow. O componente `QoS-OSCARS`, que já estaria integrado com a API do QoSFlow, deve então ser capaz de interpretar as mensagens JSON enviadas pelo OSCARS e dela extrair os dados necessários para serem usados pelo `vlanconf`, assim como solicitar ao controlador NOX QoSFlow a aplicação da reserva de banda solicitada.

Vale notar que o componente `vlanconf` atuará de forma análoga ao componente OSCARS.py usado na arquitetura básica. Para tanto, ele deve ainda ser modificado para comportar as ações de remoção de circuito, modificação de circuito e verificação de status, assim como para enviar mensagens JSON de resposta ao OSCARS.

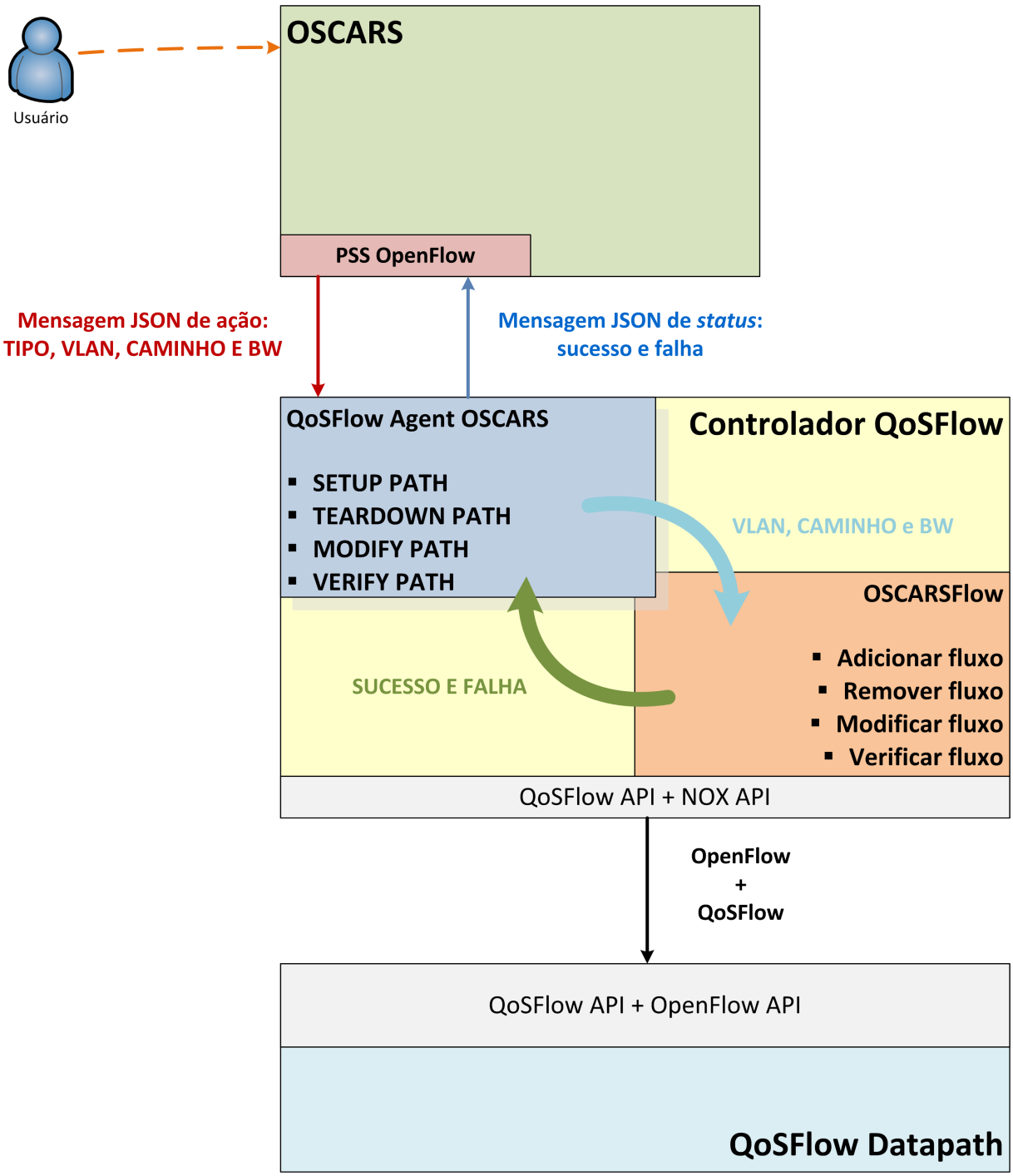

Figura 33: Integração entre as soluções OSCARS e QoSFlow.

5 *Conclusão*

Este capítulo apresenta as conclusões desta dissertação, onde serão discorridas as suas contribuições, limitações e trabalhos futuros.

Neste trabalho foram observadas duas tecnologias: o OpenFlow, que abstrai o plano de controle do plano de dados e permite que este plano se torne uma interface programática, assim ditando o comportamento da rede através de aplicações; e o DCN, que é uma arquitetura de redes que permite agendar circuitos virtuais com banda garantida. Apesar destas duas tecnologias já terem sido integradas, a construção de circuitos dinâmicos de forma eficiente e satisfatória utilizando a arquitetura DCN sobre uma rede OpenFlow não foi devidamente estudada, provocando uma carência de respostas a perguntas relacionadas sobre desempenho (Qual a vazão máxima que os equipamentos e *softwares switches* OpenFlow conseguem alcançar? Qual seu consumo de memória e processamento?) e flexibilidade (Quais funcionalidade e tecnologias estes insumos suportam?). Assim, não há um consenso de como estabelecer uma rede de circuitos dinâmicos sobre um domínio OpenFlow. Logo, como objetivo para este trabalho, é proposto um estudo aprofundado fazendo um levantamento dos possíveis mecanismos a serem utilizados para provisionar circuitos sobre uma rede OpenFlow que de fato garantam isolamento, banda reservada e policiamento da banda negociada (Qualidade de Serviço).

Para realizar este estudo foram levantadas algumas possibilidades de como se integrar DCN sobre SDNs, que das quais se destacaram: a arquitetura básica utilizada pela ESnet para integrar os dois conceitos; uma arquitetura com o *framework* QoSFlow, que permite a configuração automática de QoS; e, por fim, uma arquitetura com o QoSFlow, porém aplicada no âmbito do protocolo OpenFlow 1.3. Além destes possíveis sistemas, foram elencadas ferramentas que

poderiam ser aplicadas a uma rede OpenFlow integrada com DCN.

Durante o decorrer deste trabalho, através de avaliações qualitativas, foi possível observar que a arquitetura mais promissora é aquela na qual o DCN é integrado com o QoSFlow, no cenário de OpenFlow 1.3, pois a nova versão do protocolo OpenFlow amplifica as capacidades de QoS possíveis. Entretanto, como esta nova versão ainda não tem implementação disponível no QoSFlow, atualmente a melhor opção usar DCN integrado ao QoSFlow baseado na versão OpenFlow 1.0. Além disto, foram feitas avaliações tanto qualitativas como quantitativas das ferramentas elencadas, nas quais foram observadas as limitações de cada ferramenta e validadas as suas funcionalidades, de maneira a se obter insumos para uma adequada seleção com vistas a arquitetura integrando OSCARS e QoSFlow. No caso dos *software switches*, verificou-se que a melhor opção é uso do Open vSwitch.

5.1 Contribuições

Abaixo estão listados os benefícios que esta pesquisa trouxe:

- O estudo qualitativo e quantitativo sobre as funcionalidades das ferramentas de *software switch* do OpenFlow, o qual denotou que a solução em *software* do Open vSwitch é a mais adequada, atualmente, para o uso em uma DCN. Foi verificado também que o QoSFlow é o arcabouço de SDN, dentro do ecossistema OpenFlow, mais promissor para uso com DCN, tendo em vista suas funcionalidades de configuração automática de QoS.

- A proposta de integração da arquitetura DCN com o *framework* do QoSFlow, utilizando o OSCARS 0.6. Esta arquitetura é a mais adequada para o ambiente DCN, devido as propriedades do QoSFlow. No entanto, conforme notado em 3.1.3, a arquitetura utilizando o framework QoSFlow adaptado para o protocolo OpenFlow 1.3 seria a melhor aquitetura para a implementação de uma DCN sobre um domínio OpenFlow. Esta arquitetura não foi avaliada por ainda não haver implementações do versão 1.3 do protocolo OpenFlow.

- A criação do componente `vlanconf` para a plataforma uso no controlador NOX. Esta

aplicação possui o objetivo de apoiar a criação de circuitos VLAN, e com o uso da API estendida do QoSFlow, este *software* também é capaz de configurar o QoS, em plataformas QoSFlow.

5.2 Limitações

As limitações encontradas neste trabalho estão listadas abaixo:

- O uso de soluções virtualizadas para emulação dos ambientes de teste de QoS em switches OpenFlow, o que impede um estudo de desempenho destas soluções em um cenário real de aplicação.
- Apesar da ter sido feito um estudo de viabilidade da proposta de uso do *framework* QoSFlow numa arquitetura de DCN, é necessário realizar experimentos que comprovem que esta integração funciona adequadamente.

5.3 Trabalhos futuros

Para trabalhos futuros, são propostos as seguintes ações:

- Realizar experimentos incluindo uma gama maior de ferramentas, por exemplo, o uso do Pantou em *hardware switch* e o QoSFlow em *software switch*, assim como a avaliação de controladores diferentes.
- Integrar os ambientes do OSCARS 0.6 e QoSFlow, segundo apresentado em 4.3, para realizar provas de conceito.

ANEXO A – Códigos da aplicação para o NOX

```
1
2  #include "assert.h"
3  #include "packet-in.hh"
4  #include <boost/bind.hpp>
5  #include "openflow-action.hh"
6  #include "component.hh"
7  #include "vlog.hh"
8  #include "qosflow.hh"
9  #include "qosflow-utils.hh"
10
11 namespace {
12
13 using namespace vigil;
14 using namespace vigil::container;
15
16 Vlog_module lg("vlan");
17
18 class VlanConf
19         : public Component
20 {
21 public:
22         VlanConf(const Context* c,
23                  const json_object*)
24             : Component(c) { }
25
26         void configure(const Configuration* conf);
27         void install();
```

```
28          Disposition handle(const Event&);
29
30          void addVLAN(uint64_t dp_id, int inport, int outport, uint16_t vlan
                );
31      void queueVLAN(uint64_t dp_id, int inport, int outport, uint32_t
            queueid, uint16_t vlan);
32      void stripVLAN(uint64_t dp_id, int inport, int outport, uint16_t vlan);
33          void forwardVLAN(uint64_t dp_id, int inport, int outport);
34      void createRootHTB(uint64_t dp_id, int port);
35      void createClassHTB(uint64_t dp_id, int port, int class_id, uint32_t
            rate, uint32_t ceil);
36
37 };
38
39 void
40 VlanConf::configure(const Configuration* conf) {
41 }
42
43 void
44 VlanConf::install() {
45          register_handler<Packet_in_event>(boost::bind(&VlanConf::handle,
                this, _1));
46 }
47
48 Disposition
49 VlanConf::handle(const Event& e){
50   createRootHTB(1, 2);
51   createClassHTB(1, 2, 1, 3000000, 3000000);
52   createClassHTB(1, 2, 2, 1000000, 1000000);
53   return CONTINUE;
54 }
55
56 void
57 VlanConf::createRootHTB(uint64_t dp_id, int port){
```

```
58
59          QoSFlow qfo;
60      Htb htb;
61      struct ofp_qos_msg *msg;
62
63      /* qos header */
64      htb.set_object_type(OFP_QOS_QDISC);
65      htb.set_action_type(OFP_QOS_ACTION_ADD);
66
67      /* qos body */
68      htb.set_port_qdisc(port);    // out port
69      htb.set_class_id_qdisc(0); // queue id (root = 0)
70
71      /* get message ready to send */
72      msg = qfo.make_htb_qdisc(htb);
73
74      send_openflow_command(datapathid::from_host(dp_id), &msg->hdr, false);
75
76  }
77
78  void
79  VlanConf::createClassHTB(uint64_t dp_id, int port, int class_id, uint32_t
        rate, uint32_t ceil){
80
81      QoSFlow qfo;
82      Htb htb;
83      struct ofp_qos_msg *msg;
84
85      rate = rate/8;
86      ceil = ceil/8;
87
88      /* qos header */
89      htb.set_object_type(OFP_QOS_CLASS);
90      htb.set_action_type(OFP_QOS_ACTION_ADD);
```

```
91
92    /* qos body */
93    htb.set_port_class(port);
94    htb.set_class_id_class(class_id);
95    htb.set_rate_class(rate);          // garantia minima
96    htb.set_ceil_class(ceil);          // garantia maxima
97    htb.set_prio_class(1);      // prioridade
98
99    /* get message ready to send */
100   msg = qfo.make_htb_class(htb);
101
102   send_openflow_command(datapathid::from_host(dp_id), &msg->hdr, false);
103
104 }
105
106 void
107 VlanConf::addVLAN(uint64_t dp_id, int inport, int outport, uint16_t vlan){
108
109   boost::shared_array<uint8_t> of_raw;
110   ofp_action_list oal;
111
112   ofp_action* vlan_tag =  new ofp_action();
113   vlan_tag->set_action_vlan_vid(OFPAT_SET_VLAN_VID, vlan);
114   oal.action_list.push_back(*vlan_tag);
115
116   ofp_action* ofpa = new ofp_action();
117   ofpa->set_action_output(outport, 0);
118   oal.action_list.push_back(*ofpa);
119
120   size_t size = sizeof(ofp_flow_mod)+ oal.mem_size();
121   of_raw.reset(new uint8_t[size]);
122   of_flow_mod ofm;
123   ofm.header.version = OFP_VERSION;
124   ofm.header.type = OFPT_FLOW_MOD;
```

```
125     ofm.header.length = size;
126     ofm.match.wildcards = (OFPFW_ALL & (~OFPFW_IN_PORT));
127     ofm.match.in_port = inport;
128     ofm.cookie = 0;
129     ofm.buffer_id = -1;
130     ofm.command = OFPFC_ADD;
131     ofm.idle_timeout = 5;
132
133     ofm.pack((ofp_flow_mod*) openflow_pack::get_pointer(of_raw));
134     oal.pack(openflow_pack::get_pointer(of_raw,sizeof(ofp_flow_mod)));
135
136     send_openflow_command(datapathid::from_host(dp_id), of_raw, false);
137
138 }
139
140 void
141 VlanConf::queueVLAN(uint64_t dp_id, int inport, int outport, uint32_t
        queueid, uint16_t vlan){
142
143     boost::shared_array<uint8_t> of_raw;
144     ofp_action_list oal;
145
146     ofp_action* vlan_tag =  new ofp_action();
147     vlan_tag->set_action_vlan_vid(OFPAT_SET_VLAN_VID, vlan);
148     oal.action_list.push_back(*vlan_tag);
149
150     ofp_action* q = new ofp_action();
151     q->set_action_enqueue(outport, queueid);
152     oal.action_list.push_back(*q);
153
154     size_t size = sizeof(ofp_flow_mod)+ oal.mem_size();
155     of_raw.reset(new uint8_t[size]);
156     of_flow_mod ofm;
157     ofm.header.version = OFP_VERSION;
```

```
158     ofm.header.type = OFPT_FLOW_MOD;
159     ofm.header.length = size;
160     ofm.match.wildcards = (OFPFW_ALL & (~OFPFW_IN_PORT));
161     ofm.match.in_port = inport;
162     ofm.cookie = 0;
163     ofm.buffer_id = -1;
164     ofm.command = OFPFC_ADD;
165     ofm.idle_timeout = 5;
166 
167     ofm.pack((ofp_flow_mod*) openflow_pack::get_pointer(of_raw));
168     oal.pack(openflow_pack::get_pointer(of_raw,sizeof(ofp_flow_mod)));
169 
170     send_openflow_command(datapathid::from_host(dp_id), of_raw, false);
171 
172 }
173 
174 void
175 VlanConf::forwardVLAN(uint64_t dp_id, int inport, int outport){
176 
177     boost::shared_array<uint8_t> of_raw;
178     ofp_action_list oal;
179 
180     ofp_action* ofpa = new ofp_action();
181     ofpa->set_action_output(outport, 0);
182     oal.action_list.push_back(*ofpa);
183 
184     size_t size = sizeof(ofp_flow_mod)+ oal.mem_size();
185     of_raw.reset(new uint8_t[size]);
186     of_flow_mod ofm;
187     ofm.header.version = OFP_VERSION;
188     ofm.header.type = OFPT_FLOW_MOD;
189     ofm.header.length = size;
190     ofm.match.wildcards = (OFPFW_ALL & (~OFPFW_IN_PORT));
191     ofm.match.in_port = inport;
```

```
192     ofm.cookie = 0;
193     ofm.buffer_id = -1;
194     ofm.command = OFPFC_ADD;
195     ofm.idle_timeout = 5;
196
197     ofm.pack((ofp_flow_mod*) openflow_pack::get_pointer(of_raw));
198     oal.pack(openflow_pack::get_pointer(of_raw,sizeof(ofp_flow_mod)));
199
200     send_openflow_command(datapathid::from_host(dp_id), of_raw, false);
201
202 }
203
204
205
206 void
207 VlanConf::stripVLAN(uint64_t dp_id, int inport, int outport, uint16_t vlan)
        {
208
209     boost::shared_array<uint8_t> of_raw;
210     ofp_action_list oal;
211
212     ofp_action* strip = new ofp_action();
213     strip->set_action_strip_vlan(OFPAT_STRIP_VLAN);
214     oal.action_list.push_back(*strip);
215
216     ofp_action* ofpa = new ofp_action();
217     ofpa->set_action_output(outport, 0);
218     oal.action_list.push_back(*ofpa);
219
220     size_t size = sizeof(ofp_flow_mod)+ oal.mem_size();
221     of_raw.reset(new uint8_t[size]);
222     of_flow_mod ofm;
223     ofm.header.version = OFP_VERSION;
224     ofm.header.type = OFPT_FLOW_MOD;
```

```
225     ofm.header.length = size;
226     ofm.match.wildcards = (OFPFW_ALL & (~OFPFW_IN_PORT));
227     ofm.match.in_port = inport;
228     ofm.match.dl_vlan = vlan;
229     ofm.cookie = 0;
230     ofm.buffer_id = -1;
231     ofm.command = OFPFC_ADD;
232     ofm.idle_timeout = 5;
233
234     ofm.pack((ofp_flow_mod*) openflow_pack::get_pointer(of_raw));
235     oal.pack(openflow_pack::get_pointer(of_raw,sizeof(ofp_flow_mod)));
236
237     send_openflow_command(datapathid::from_host(dp_id), of_raw, false);
238
239 }
240
241 REGISTER_COMPONENT(container::Simple_component_factory<VlanConf>, VlanConf)
        ;
242
243 } // unnamed namespace
244
245 \end{verbatim}
```

ANEXO B – Código JSON referente às mensagens do OSCARS

```
1. SETUP PATH
{ "type" : "oscars-request", "action" : "setup", "version": "1.0",
  "gri":"testdomain-1.net-123", "bandwidth":"100Mbps",
  "path": [
  { "switch": "10.10.25.14", "add-flows": [{"port":"1", "vlan_range":
      "3000"},
{"port": "50", "vlan_range": "3000"}] },
  { "switch": "10.10.29.14", "add-flows": [{"port":"51", "vlan_range"
      :"3000"},
{"port": "2", "vlan_range": "3000"}] }
  ],
}

2. TEARDOWN PATH
{ "type" : "oscars-request", "action" : "teardown", "version": "1.0",
    "gri":"testdomain-1.net-123", "bandwidth":"100Mbps",
  "path": [
  { "switch": "10.10.25.14", "del-flows": [{"port":"1", "vlan_range":
      "3000"},
{"port": "50", "vlan_range": "3000"}] },
  { "switch": "10.10.29.14", "del-flows": [{"port":"51", "vlan_range"
      :"3000"},
```

```
18 {"port": "2", "vlan_range": "3000"}] }
19   ],
20 }
21
22 3.  MODIFY PATH
23 { "type" : "oscars-request", "action" : "modify", "version": "1.0",
24   "gri":"testdomain-1.net-123", "vlan_range":"3000", "bandwidth":"100
       Mbps",
25   "path": [
26   { "switch": "10.10.25.14", "del-flows": [{"port":"1", "vlan_range":
       "3000"},
27 {"port": "50", "vlan_range": "3000"}],
28 "add-flows": [{"port":"2", "vlan_range":"3000"},
29 {"port": "50", "vlan_range": "3000"}] },
30   { "switch": "10.10.29.14", "del-flows": [{"port":"51", "vlan_range"
       :"3000"},
31 {"port": "2", "vlan_range": "3000"}] }
32 "add-flows": [{"port":"51", "vlan_range":"3000"},
33 {"port": "1", "vlan_range": "3000"}] }
34 ],
35 }
36
37 4.  VERIFY PATH
38 { "type" : "oscars-request", "action" : "verify", "version": "1.0",
39   "gri":"testdomain-1.net-123", "vlan_range":"3000", "bandwidth":"100
       Mbps",
40   "path": [
41       { "switch": "10.10.25.14", "has-flows": [{"port":"1", "
           vlan_range":"3000"},
42 {"port": "50", "vlan_range": "3000"}] },
```

```
43      { "switch": "10.10.29.14", "has-flows": [{"port":"51", "
            vlan_range":"3000"},
44 {"port": "2", "vlan_range": "3000"}] }
45   ],
46 }
47
48 Mensagens de reply
49
50 1.  Success Status
51 { "type" : "oscars-reply", "action" : "setup", "version": "1.0",
52   "gri":"testdomain-1.net-123", "status" : "ACTIVE", "err_msg": ""}
53
54 2.  Failure Status - Rolled back
55 { "type" : "oscars-reply", "action" : "setup", "version": "1.0",
56   "gri":"testdomain-1.net-123", "status" : "FAILED",
57   "err_msg": "Failed to connect to switch: 10.10.29.14; Rollack OK"}
58
59 3.  Failure Status - Rollback error
60 { "type" : "oscars-reply", "action" : "setup", "version": "1.0",
61   "gri":"testdomain-1.net-123", "status" : "FAILED",
62   "err_msg": "Failed to connect to switch: 10.10.29.14; Rollback
        error: switch: 10.10.25.14, flows: blah, blah"}
```

Referências

BEACON. Setembro 2012. Disponível em: <https://openflow.stanford.edu/display/Beacon/Home>.

BHANAGE, G. et al. Experimental evaluation of openvz from a testbed deployment perspective. In: MAGEDANZ, T. et al. (Ed.). *Testbeds and Research Infrastructures. Development of Networks and Communities*. [S.l.]: Springer Berlin Heidelberg, 2011, (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, v. 46). p. 103–112. ISBN 978-3-642-17850-4.

BLESS, R.; NICHOLS, K.; WEHRLE, K. *A Lower Effort Per-Domain Behavior (PDB) for Differentiated Services*. IETF, December 2003. RFC 3662 (Informational). (Request for Comments, 3662). Disponível em: <http://www.ietf.org/rfc/rfc3662.txt>.

BOOTE, J. et al. *Dynamic Circuit Network Services and OpenFlow Integration*. 2011. Http://archstone.east.isi.edu/twiki/pub/ARCHSTONE/OpenFlow/sc-2011-srs-internet2-v2.pdf.

CAI, A. L. C. Z.; NG., T. S. E. *Maestro: A System for Scalable OpenFlow Control*. [S.l.], 2010. Disponível em: <http://code.google.com/p/maestro-platform/>.

CASADO, M. et al. Ethane: taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 37, n. 4, p. 1–12, ago. 2007. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1282427.1282382>.

CASADO, M. et al. Sane: a protection architecture for enterprise networks. In: *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*. Berkeley, CA, USA: USENIX Association, 2006. (USENIX-SS'06). Disponível em: <http://dl.acm.org/citation.cfm?id=1267336.1267346>.

CERN. *CERN - the European Organization for Nuclear Research*. 2012. Disponível em: <http://public.web.cern.ch/public/>.

COLLIDER, L. H. *Large Hadron Collider*. 2012. Disponível em: <http://lhc.web.cern.ch/lhc/>.

CONSORTIUM, O. *Linux Software Reference System*. Dezembro 2009. Disponível em: <http://www.openflow.org/wp/downloads/>.

CONSORTIUM, O. *OpenFlow Switch Specification Version 1.0.0*. Dezembro 2009. Disponível em: <http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf>.

CONSORTIUM, O. *OpenFlow Tutorial*. 2012. Disponível em: <http://tinyurl.com/7uu93yk>.

CONSORTIUM, O. *Pantou : OpenFlow 1.0 for OpenWRT*. Setembro 2012. Disponível em: <http://z6.co.uk/openflow>.

CORREA, C. et al. Uma avaliação experimental de soluções de virtualização para o plano de controle de roteamento de redes virtuais. In: *Anais do XXIX Simpósio Brasileiro de Redes de Computadores*. [S.l.: s.n.], 2011.

CPQD. *OpenFlow 1.3 Software Switch*. Novembro 2012. Disponível em: <https://github.com/CPqD/ofsoftswitch13>.

CPQD - Centro de Pesquisa e Desenvolvimento em Telecomunicações. Novembro 2012. Disponível em: <www.cpqd.com.br>.

DANTE. *DANTE Delivery of Advanced Network Technology to Europe*. 2012. Disponível em: <http://www.dante.net/>.

DAVIE, B. et al. *An Expedited Forwarding PHB (Per-Hop Behavior)*. IETF, March 2002. RFC 3246 (Proposed Standard). (Request for Comments, 3246). Disponível em: <http://www.ietf.org/rfc/rfc3246.txt>.

DICE. *Inter-domain Controller (IDC) Protocol Specification*. February 2010. Disponível em: <http://www.controlplane.net/idcp-v1.1/idc-protocol-specification-v1.1-feb092010.pdf>.

DRAGON. *Radio Astronomy: e-VLBI*. Setembro 2012. Disponível em: <http://dragon.maxgigapop.net/twiki/bin/view/DRAGON/E-VLBI>.

ESNET. *ESnet*. 2012. Disponível em: <http://www.es.net/>.

ESNET. *OSCARS 0.5 Architecture*. Setembro 2012. Disponível em: <http://www.es.net/services/virtual-circuits-oscars/documentation/0-5-Architecture/>.

ESNET. *OSCARS Architecture Overview*. Abril 2012. Disponível em: <http://tinyurl.com/98afcd2>.

FARREL, A. *The Internet and Its Protocols: A Comparative Approach*. Morgan Kaufmann Publishers, 2004. (Morgan Kaufmann Series in Networking). ISBN 9781558609136. Disponível em: <http://books.google.com.br/books?id=LtBegQowqFsC>.

FLOODLIGHT. *Floodlight OpenFlow Controller*. Setembro 2012. Disponível em: <http://floodlight.openflowhub.org/>.

FOUNDATION, O. N. *OpenFlow Switch Specification Version 1.3.0*. Junho 2012. Disponível em: <https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf>.

FOUNDATION, O. N. *Software-Defined Networking: The New Norm for Networks*. Abril 2012. Disponível em: <https://www.opennetworking.org/images/stories/downloads/white-papers/wp-sdn-newnorm.pdf>.

GEANT. *GEANT*. 2012. Disponível em: <http://www.geant.net>.

GUDE, N. et al. Nox: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 3, p. 105–110, jul. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1384609.1384625>.

GUOK, C. *ESnet On-Demand Secure Circuits and Advance Reservation System*. [S.l.], 2006.

GUOK, C. et al. A user driven dynamic circuit network implementation. In: *GLOBECOM Workshops, 2008 IEEE*. [S.l.: s.n.], 2008. p. 1 –5.

H3C. *QoS Introduction*. Setembro 2012. Disponível em: <http://tinyurl.com/37vceco>.

HEINANEN, J. et al. *Assured Forwarding PHB Group*. IETF, jun 1999. RFC 2597 (Proposed Standard). (Request for Comments, 2597). Updated by RFC 3260. Disponível em: <http://www.ietf.org/rfc/rfc2597.txt>.

INTERNET2. *OSCARS to OpenFlow NOX Controller API*. Novembro 2011. Disponível em: <http://archstone.east.isi.edu/twiki/pub/ARCHSTONE/OpenFlow/oscars-openflow-v3.doc>.

INTERNET2. *Dynamic Circuit Network*. Setembro 2012. Disponível em: <http://www.internet2.edu/network/dc/>.

INTERNET2. *Internet2*. 2012. Disponível em: <http://www.internet2.edu/>.

ISHIMORI, A. *QoSFlow Tutorial*. Agosto 2012. Disponível em: <http://tinyurl.com/8nwkbjk>.

ISHIMORI, A. et al. Qosflow: Gerenciamento automático da qualidade de serviço em infraestruturas de experimentação baseadas em framework openflow. In: *III Workshop de Pesquisa Experimental da Internet do Futuro - Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2012)*. [S.l.: s.n.], 2012. p. 24–29.

KATZ, D.; KOMPELLA, K.; YEUNG, D. *Traffic Engineering (TE) Extensions to OSPF Version 2*. IETF, September 2003. RFC 3630 (Proposed Standard). (Request for Comments, 3630). Updated by RFC 4203. Disponível em: <http://www.ietf.org/rfc/rfc3630.txt>.

KELLY, R.; GASPARAKIS, J. *Common Functionality in the 2.6 Linux* Network Stack*. April 2010.

KOMPELLA, K.; REKHTER, Y. *OSPF Extensions in Support of Generalized Multi-Protocol Label Switching (GMPLS)*. IETF, October 2005. RFC 4203 (Proposed Standard). (Request for Comments, 4203). Disponível em: <http://www.ietf.org/rfc/rfc4203.txt>.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. New York, NY, USA: ACM, 2010. (Hotnets-IX), p. 19:1–19:6. ISBN 978-1-4503-0409-2. Disponível em: <http://doi.acm.org/10.1145/1868447.1868466>.

LEHMAN, T.; SOBIESKI, J.; JABBARI, B. Dragon: a framework for service provisioning in heterogeneous grid networks. *Comm. Mag.*, IEEE Press, Piscataway, NJ, USA, v. 44, n. 3, p. 84–90, mar. 2006. ISSN 0163-6804. Disponível em: <http://dx.doi.org/10.1109/MCOM.2006.1607870>.

LXC. *lxc Linux Containers*. Setembro 2012. Disponível em: <http://lxc.sourceforge.net/>.

MANDEVILLE, R.; PERSER, J. *RFC 2889: Benchmarking Methodology for LAN Switching Devices*. Agosto 2000. Disponível em: <http://www.ietf.org/rfc/rfc2889.txt>.

MCCAULEY, M. *Introducing POX*. Março 2012. Acessado em Setembro de 2012. Disponível em: <http://www.noxrepo.org/2012/03/introducing-pox/>.

MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>.

MOTA, J. *Controle de tráfego com TC, HTB e Iptables*. Setembro 2012. Disponível em: <http://tinyurl.com/8kuknxw>.

NAOUS, J. et al. Implementing an openflow switch on the netfpga platform. In: *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. New York, NY, USA: ACM, 2008. (ANCS '08), p. 1–9. ISBN 978-1-60558-346-4. Disponível em: <http://doi.acm.org/10.1145/1477942.1477944>.

NASCIMENTO, M. R. et al. Routeflow: Roteamento commodity sobre redes programáveis. In: *Anais do XXIX Simpósio Brasileiro de Redes de Computadores*. [S.l.: s.n.], 2011a.

NDDI. *OESS - Open Exchange Software Suite*. Setembro 2012. Disponível em: <https://code.google.com/p/nddi/>.

NORDUNET. *NORDUnet - Nordic Infrastructure for Research & Education*. 2012. Disponível em: <NORDUnet - Home http://www.nordu.net/ndnweb/home.html>.

NOX-CLASSIC. Setembro 2012. Disponível em: <https://github.com/noxrepo/nox-classic/wiki>.

OPENWRT. *OpenWrt - Wireless Freedom*. Setembro 2012. Disponível em: <https://openwrt.org/>.

OSCARS. *OSCARS Sequence Diagrams*. Setembro 2012. Disponível em: <http://code.google.com/p/oscars-idc/wiki/OSCARSSequenceDiagrams>.

PETTIT, J. et al. *Virtual Switching in an Era of Advanced Edges*. 2010.

RNP. *RNP- Rede Nacional de Ensino e Pesquisa*. 2012. Disponível em: <http://www.rnp.br/>.

ROSEN, E.; VISWANATHAN, A.; CALLON, R. *RFC 3031 : Multiprotocol Label Switching Architecture*. 2001. Disponível em: <http://www.ietf.org/rfc/rfc3031.txt>.

SALMITO, T. *Estudo de popularidade e base de usuários*. Setembro 2012. Disponível em: <http://tinyurl.com/8nb25os>.

SC11. *Dynamic Circuit Network Services and OpenFlow Integration*. Setembro 2012. Disponível em: <http://tinyurl.com/94ak4k2>.

SHERWOOD, R. et al. *FlowVisor: A Network Virtualization Layer*. Outubro 2009. Disponível em: <http://www.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf>.

TANENBAUM, A. *Redes de Computadores*. Campus, 2003. ISBN 9788535211856. Disponível em: <http://books.google.com.ec/books?id=0tjB8FbV590C>.

TREMA. *Trema - Full-Stack OpenFlow Framework in Ruby and C*. Setembro 2012. Disponível em: <http://trema.github.com/trema/>.

VOLLBRECHT, J.; CASHMAN, B.; LAKE, A. *Using the Dynamic Circuit Network: A Brief Tutorial*. 2008. Disponível em: <http://www.internet2.edu/dcresearch/SMM08/dcn-tutorial.pdf>.