UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

Uma Abordagem para Aprovisionamento de QoS em Redes Definidas por Software baseadas em OpenFlow

Márcio Ricardo Alves Gadelha de Araújo

Recife, 02 de outubro de 2013

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

Uma Abordagem para Aprovisionamento de QoS em Redes Definidas por Software baseadas em OpenFlow

Márcio Ricardo Alves Gadelha de Araújo

Trabalho apresentado ao Programa de GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO do CENTRO DE INFORMÁTICA da UNIVERSIDADE FEDERAL DE PERNAMBUCO como requisito obrigatório parcial para obtenção do grau de Bacharel em ENGENHARIA DA COMPUTAÇÃO.

Orientador: Prof. Dr. Kelvin Lopes Dias

Recife, 02 de outubro de 2013

Dedico este trabalho aos meus pais,
Joselene (in memoriam) e Carlos.

AGRADECIMENTOS

Sou imensamente grato a todas as pessoas que me ajudaram durante essa longa e árdua caminhada. Em especial à minha família, que me deu todo o amor necessário para eu continuar buscando meus sonhos mesmo nos momentos mais difíceis. Agradeço também aos meus amigos, colegas do curso e colegas de trabalho, que me acompanharam e me apoiaram durante essa jornada. E por fim, mas não menos importante, agradeço aos meus professores, que compartilharam as suas experiências, expressaram suas linhas de pensamento e apontaram as direções para o conhecimento, contribuindo para minha formação.

RESUMO

As Redes Definidas por Software (SDNs) já são uma tendência na área de redes de computadores para a Internet do Futuro. Os principais fabricantes de *switches Ethernet* comerciais estão adotando SDN através da implementação do protocolo OpenFlow. O OpenFlow é um padrão aberto que permite implantar novos protocolos de roteamento em uma rede abstraindo componentes da rede. Assim, pesquisadores podem realizar experimentos em redes físicas ou virtuais e distribuir novas aplicações em um ambiente de produção. Neste trabalho, vamos montar uma arquitetura de aprovisionamento de Qualidade de Serviço (QoS) em uma infraestrutura de redes virtuais baseadas em OpenFlow. Para construir o ambiente de rede virtualizado utilizamos o *switch* OpenFlow Pronto e a plataforma *open source* NetFPGA, que permite desenvolver protótipos de dispositivos de rede.

Palavras-chaves: Internet do Futuro, *Software-Defined Networking* (SDN), OpenFlow, NetFPGA, Qualidade de Serviço (*Quality of Service*, QoS).

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO

1.1 Motivação

As Redes Definidas por Software (SDNs) já são uma tendência na área de redes de computadores para a Internet do Futuro. Essa tendência existe devido à real necessidade da programação e de uma maior flexibilidade das redes. As redes locais têm se tornado parte da infraestrutura crítica de ambientes comerciais, residenciais e acadêmicos [1]. No entanto, a grande quantidade de equipamentos e protocolos já consagrados restringe o lançamento de ideias inovadoras, pois geralmente os experimentos precisam ser feitos com tráfego real de produção. Portanto, há um consenso na comunidade acadêmica de que a Internet está "ossificada" [1].

As redes virtuais programáveis permitem que pesquisadores possam experimentar novas arquiteturas e protocolos de redes. Para que os experimentos sejam mais realísticos, algumas grandes plataformas de experimentação (*testbeds*) estão sendo construídas ao redor do mundo. No Brasil, o projeto FIBRE (*Future Internet Testbeds Experimentation Between Brazil and Europe*) consiste em construir uma *testbed* envolvendo várias universidades brasileiras e algumas universidades europeias [2]. Já o projeto GENI (*Global Environment for Network Innovations*) atua nos EUA [3].

O OpenFlow é o principal recurso disponível para os pesquisadores na maioria dos *testbeds*. É através do OpenFlow que é possível modificar o comportamento de dispositivos de rede remotamente. Os principais fabricantes de *switches Ethernet* comerciais estão adotando SDN através da implementação do protocolo OpenFlow [4]. O OpenFlow é um padrão aberto que permite implantar novos protocolos de roteamento em uma rede abstraindo componentes da rede. Assim, pesquisadores podem realizar experimentos em redes físicas ou virtuais e distribuir novas aplicações em um ambiente de produção.

Diante deste cenário de Internet do Futuro, há uma gama de possibilidades a serem exploradas. Uma delas é o fornecimento de Qualidade de Serviço (QoS). Para uma aplicação, oferecer seus serviços com qualidade significa atender às

expectativas do usuário em termos do tempo de resposta e da qualidade [5]. Entretanto, prover QoS fim-a-fim ainda é um dos maiores problemas para o sucesso de determinados serviços nos sistemas heterogêneos de telecomunicações usados [6].

1.2 Objetivo

O principal foco deste trabalho é montar uma arquitetura de aprovisionamento de QoS em uma infraestrutura de redes virtuais baseadas em OpenFlow. Para construir o ambiente de rede virtualizado, utilizamos a plataforma NetFPGA para conceber o *testbed*. A plataforma NetFPGA é *open source* e consiste em um hardware programável FPGA que permite desenvolver protótipos de dispositivos de redes como roteadores e *switches* [7]. Além das placas NetFPGA, também utilizamos um *switch* OpenFlow propriamente dito.

Uma vez com nossa infraestrutura de rede montada, visamos desenvolver uma abordagem de aprovisionamento de QoS acompanhada da realização de experimentos de rede para avaliar a eficácia e eficiência do método implementado. Com isso, a nossa expectativa é de poder contribuir para a comunidade científica com o desenvolvimento deste trabalho.

1.3 Estrutura do documento

O Capítulo 2 provê ao leitor os fundamentos teóricos obtidos a partir da revisão bibliográfica. Os conceitos introduzidos neste capítulo foram utilizados ao longo deste trabalho. No Capítulo 3, a proposta do trabalho é apresentada. O Capítulo 4 detalha o projeto e o desenvolvimento deste trabalho com as principais atividades realizadas e a execução de alguns experimentos. Por fim, o Capítulo 5 conclui este trabalho deixando as considerações finais e as sugestões de possíveis trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Uma visão geral de SDN

Software-Defined Networking (SDN) é o termo utilizado para referenciar a emergente arquitetura de redes de computadores que separa fisicamente o plano de controle do plano de encaminhamento de dados. Segundo a *Open Networking Foundation* (ONF) [9], SDN é uma arquitetura dinâmica, gerenciável, adaptável e com boa relação custo-benefício, tornando-se ideal para as aplicações dinâmicas e com alta largura de banda encontradas atualmente.

Um dos aspectos mais relevantes do SDN é a possibilidade de programação da rede. Esta dissociação entre o plano de controle e o plano de dados tem como função extrair o processo de controle e gerenciamento da rede dos dispositivos de rede. Assim, a rede pode ser tratada como uma entidade lógica ou virtual diretamente programada através de serviços e aplicações, abstraindo os componentes físicos da rede.

A inteligência lógica da rede é centralizada em controladores, os quais mantêm uma visão global da rede. Tais controladores são os softwares responsáveis pelo controle da rede, permitindo que uma rede inteira seja controlada deste único componente. A rede passa a ser divida em camadas, simplificando o projeto e a operação, como pode ser visto na Figura 1 [10].

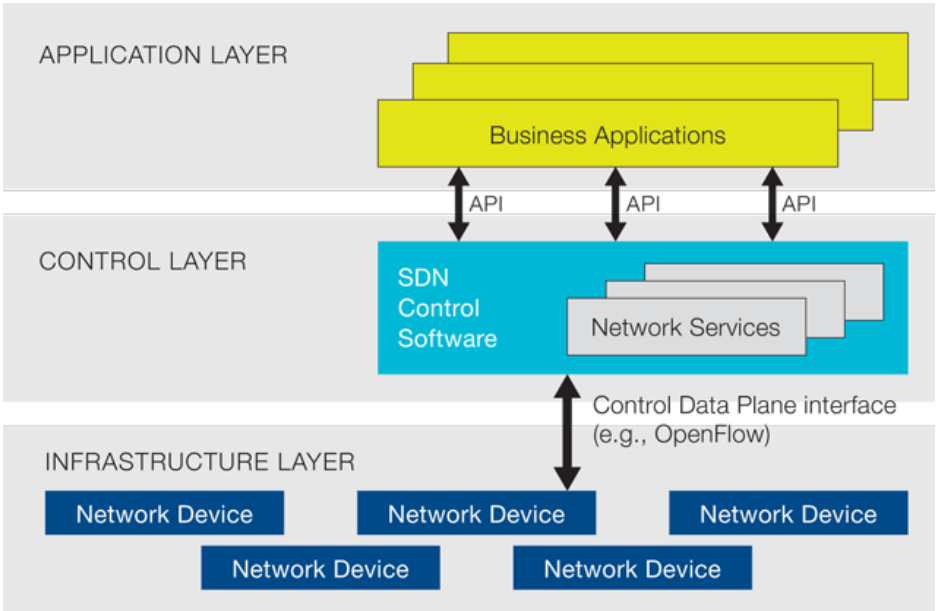

Figura 1. Arquitetura de Rede Definida por Software (SDN) [10].

A comunicação entre a camada de controle e a camada de infraestrutura é feita através de um protocolo padronizado. Deste modo, tanto o controlador como os dispositivos de rede precisam interpretar este protocolo. Neste trabalho, utilizamos o protocolo OpenFlow, porém este não é o único padrão SDN. Grandes grupos corporativos como a *IBM* e a *Intel Corporation* adotaram o OpenFlow como solução de SDN, mas outras empresas como a *Cisco Systems* e a *Juniper Networks* preferiram adotar estratégias diferentes, utilizando suas próprias soluções de SDN [11].

2.2 OpenFlow

O OpenFlow é um protocolo de padrão aberto que permite modificar o comportamento dos dispositivos de rede através de um conjunto de instruções definidos por um controlador remoto. Vários fabricantes estão incluindo o protocolo OpenFlow em seus equipamentos como roteadores, *switches* e *access points*.

Com o OpenFlow, pesquisadores podem criar novos algoritmos e rodar seus experimentos através de um protocolo padronizado sem que os fabricantes tenham que expor a implementação interna dos seus equipamentos. Assim, diferentes *switches* comerciais estão chegando ao mercado com as funcionalidades de um *switch* comum, mas com o recurso de OpenFlow disponível e pronto para ser ativado, esses *switches* são conhecidos como híbridos. Outros modelos ainda podem ser puramente OpenFlow.

Atualmente, o OpenFlow está em constante atualização, tanto de sua documentação, quanto de sua implementação. A versão atual, oficialmente especificada pela *Open Network Foundation (ONF)* [9] é a versão 1.3.2 na data da escrita deste trabalho. A versão 1.4 foi aprovada, mas ainda está sob ratificação. E a versão 1.5 está sendo desenvolvida. No entanto, neste trabalho sempre nos referenciamos a versão 1.0, pois é a versão mais bem difundida do ponto de vista de implementação.

2.2.1 Estrutura de um *switch* OpenFlow

Um *switch* de rede é um dispositivo capaz de analisar o cabeçalho dos pacotes e tomar decisões sobre como encaminhá-los [12]. Dessa forma, a

arquitetura interna de rede do *switch* OpenFlow consiste em dois níveis, exibidos na Figura 2:

- Plano de controle (Path Control): Responsável por tomar decisões sobre como os pacotes serão roteados.
- Plano de dados (Data Path): Responsável por realizar a troca de dados entre portas, em nível de hardware.

Figura 2. Arquitetura interna de um *switch* OpenFlow. Figura adaptada [13].

Neste cenário, normalmente um PC externo é utilizado como um controlador que se comunica com o plano de dados do *switch* através da própria rede utilizando o protocolo OpenFlow. O plano de dados de um *switch* OpenFlow consiste de uma tabela de fluxos (*flow table*) e uma ação associada a cada entrada da tabela (*flow entry*).

Um *switch* OpenFlow é dividido em basicamente três partes (*vide* Figura 3):

- A tabela de fluxos, que indica quais os fluxos que o *switch* deve processar com base nas entradas de fluxo. Além disso, um conjunto de ações deve informar qual a ação deve ser tomada para cada entrada da tabela;
- Um canal seguro, que conecta o *switch* ao controlador através do protocolo OpenFlow, permitindo o envio de comandos e pacotes;
- O protocolo OpenFlow, através do qual o OpenFlow oferece o padrão de comunicação entre o *switch* e o controlador.

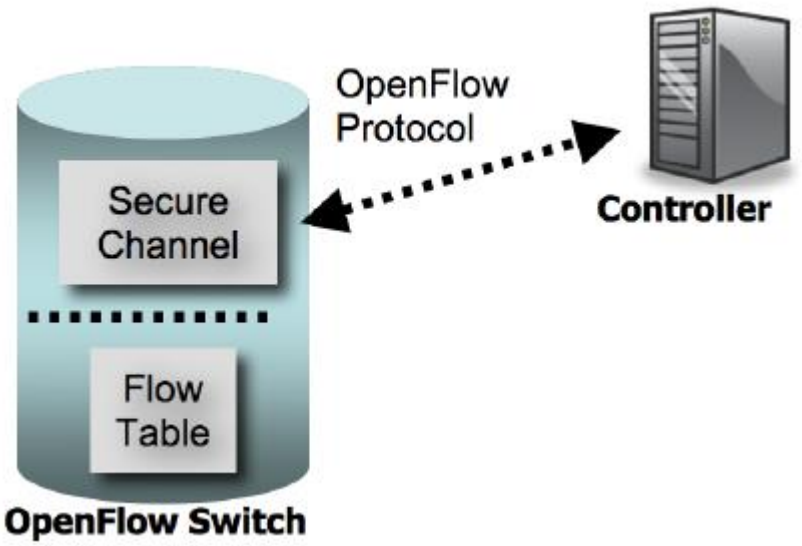

Figura 3. Um *switch* OpenFlow que se comunica com o controlador por um canal seguro [14].

2.2.2 Tabela de Fluxos

Uma tabela de fluxos (*flow table*) consiste de entradas de fluxo (*flow entry*), cujo formato com os principais componentes podem ser vistos na Tabela 1:

- *Match Fields*: Conjunto de campos utilizados para equiparar (*match*) com os pacotes que chegam ao *switch*. Podem ser campos dos cabeçalhos dos datagramas, porta de entrada ou até metadados previamente especificados.
- *Counters*: Existem contadores específicos para tabelas, fluxos, portas ou filas que fornecem informações estatísticas. Por exemplo, o tempo (duração) que uma regra de fluxo foi instalada no *switch*, quantidade de pacotes ou *bytes* recebidos em uma determinada porta, ou por um determinado fluxo.
- *Actions*: Conjunto de instruções que decretam como o *switch* deve manipular os pacotes recebidos que combinaram (*match*) com a regra de fluxo especificada. Caso nenhuma ação seja especificada, o pacote é descartado.

Tabela 1. Formato de uma entrada de fluxo (*flow entry*). Tabela adaptada [14].

Match Fields	*Counters*	*Actions*

A Tabela 2 exibe os campos utilizados para comparar com os datagramas dos fluxos que chegam ao *switch*.

Tabela 2. Campos de cabeçalhos utilizados para equiparar os fluxos (*Match fields*). Tabela adaptada [14].

In Port	*Ethernet*			VLAN		IP				TCP/UDP	
	src	dst	type	id	pcp	src	dst	proto	tos	srcp	dstp

Os seguintes campos constituem a 12-tupla:

- *In Port*: Porta de entrada do *switch*;
- *Ethernet Source*: Endereço MAC de origem;
- *Ethernet Destination*: Endereço MAC de destino;
- *Ethernet Type*: Tipo do quadro (*frame*) *Ethernet*;
- VLAN ID: Número de identificação da VLAN (*Virtual* LAN);
- VLAN PCP (*Priority Code Point*): Nível de prioridade;
- IP *Source*: Endereço IP de origem;
- IP *Destination*: Endereço IP de destino;
- IP *Protocol*: Protocolo IP;
- IP ToS (*Type of Service*): Tipo de serviço;
- TCP/UDP *Source Port*: Porta de origem do protocolo (TCP/UDP);
- TCP/UDP *Destination Port*: Porta de destino do protocolo (TCP/UDP).

As principais ações que um *switch* OpenFlow deve suportar são as seguintes:

- Encaminhar o datagrama pra uma determinada porta;
- Encapsular o datagrama e encaminhar para o controlador;
- Descartar o datagrama.

Outra ação comum é enviar o datagrama realizando o processamento normal, da mesma forma que um *switch* comum faria. Esta ação é essencialmente voltada para os *switches* OpenFlow híbridos.

Além das ações citadas, um *switch* OpenFlow também pode alterar o cabeçalho do fluxo de dados que ele recebe. Desta forma, é possível alterar campos como endereços IP ou MAC, de origem ou destino, e ainda modificar campos

referentes à VLAN. Com isso, também é possível mudar a identificação da VLAN, o nível de prioridade (PCP) da VLAN e ainda colocar ou retirar os campos de VLAN do fluxo, transformando um fluxo normal em um fluxo com VLAN (*tagged*) ou um fluxo com VLAN em um fluxo normal (*untagged*).

2.2.3 Rede OpenFlow

Em uma infraestrutura de rede OpenFlow, podemos ter vários *switches* OpenFlow interligados com a presença de servidores, clientes e pelo menos um controlador ligados aos *switches* OpenFlow. Um *Access Point* também pode usufruir do protocolo OpenFlow.

Os clientes podem acessar os servidores passando por vários *switches* OpenFlow, e em cada um dos *switches*, a tabela de fluxos será analisada em busca da regra de fluxo que trata daquela entrada de fluxo. Caso a regra seja encontrada, as ações predefinidas são tomadas, caso contrário, o fluxo é enviado para o controlador. A Figura 4 exemplifica uma rede OpenFlow.

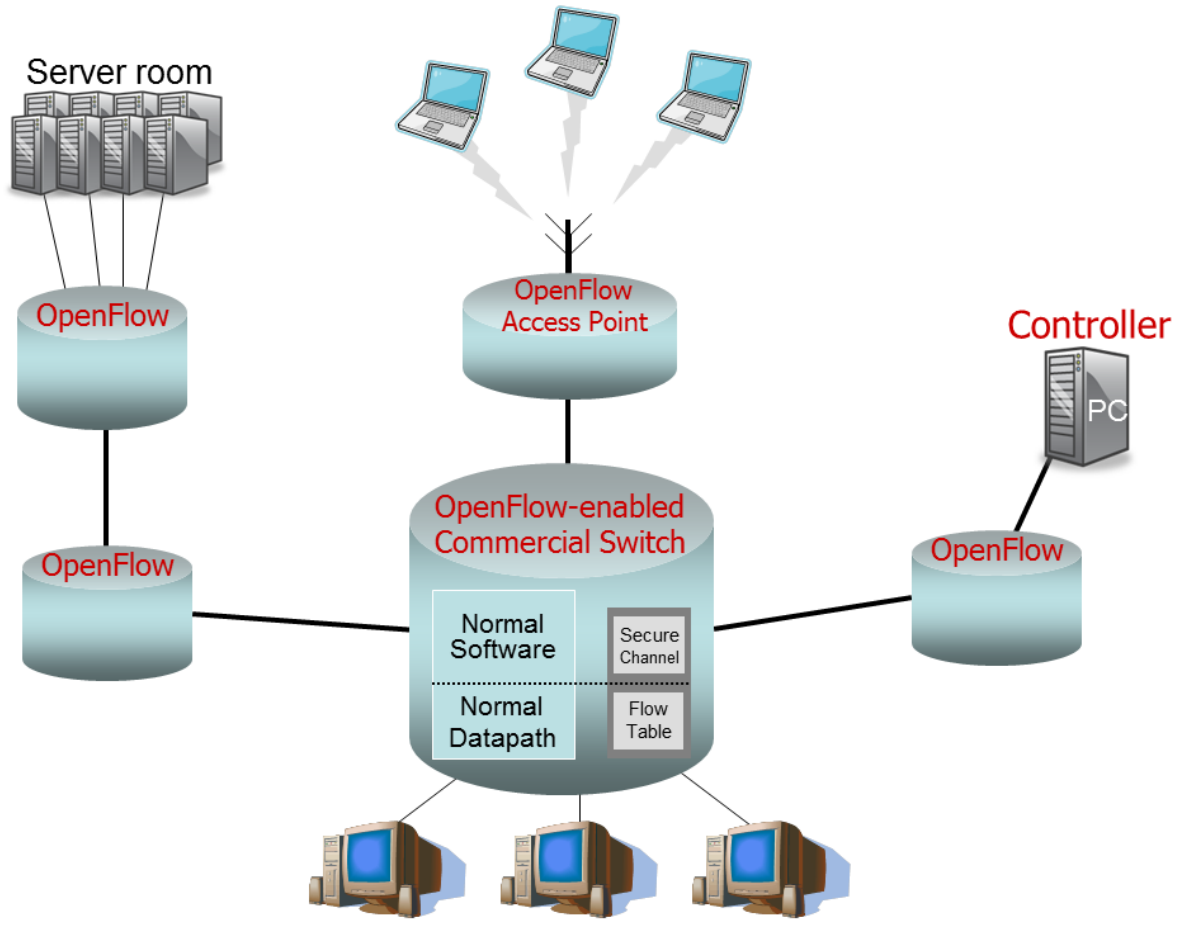

Figura 4. Exemplo de uma rede OpenFlow [1].

2.3 *Switches*

2.3.1 NetFPGA

O NetFPGA é uma plataforma *open source* de hardware e software, projetada para pesquisa, experimentação e ensino. O projeto da plataforma foi desenvolvido pelo grupo de pesquisa de redes de alto desempenho, *The McKeown Group* [15], do departamento de Ciência da Computação da universidade de *Stanford*. E a *Digilent Inc* foi a empresa responsável pela fabricação das placas.

Com o NetFPGA é possível que pesquisadores e estudantes desenvolvam seus próprios protótipos de sistemas de rede de alta velocidade. A plataforma inclui todos os recursos lógicos, memória e portas *Ethernet* para que um *switch*, ou roteador completo possa ser implementado. Além de roteadores e *switches*, outros projetos já foram desenvolvidos para a plataforma NetFPGA, permitindo que a placa tenha as funcionalidades de uma placa de rede comum (*Network Interface Card*, NIC), ou um *switch* OpenFlow 1.0, ou um gerador de tráfego, ou até outro tipo de dispositivo, como um dispositivo de segurança de rede, por exemplo.

Switches baseados em software poderiam ser utilizados para emular o funcionamento de um *switch* OpenFlow, porém uma grande vantagem de se utilizar o NetFPGA ao invés de *switches* baseados em software é que o NetFPGA usa o hardware para encaminhar pacotes, aumentando significativamente o desempenho e aproximando o ambiente de experimentação da realidade. Isto acontece porque o caminho de dados é implementado em hardware, e a placa opera em taxa de linha (*line-rate*), isto é, os fluxos de dados são processados imediatamente ao serem recebidos, permitindo que a taxa máxima de um enlace *Gigabit Ethernet* seja atingida sem haver perda de pacotes.

Atualmente existem dois modelos disponíveis: NetFPGA-1G e NetFPGA-10G. O primeiro disponibiliza quatro interfaces *Gigabit Ethernet* com 1 Gbps e o outro também disponibiliza quatro interfaces de rede, mas com 10 Gbps. O modelo utilizado neste trabalho foi o NetFPGA-1G. As principais especificações técnicas do hardware do NetFPGA-1G estão listadas a seguir, assim como a Figura 5, que exibe uma fotografia desta placa:

- Lógica FPGA
    - *Xilinx Virtex-II Pro 50*
    - 53.136 células lógicas
    - 4.176 *Kbit block RAM*
    - Até 738 *Kbit distributed RAM*
    - 2 x *cores PowerPC*
    - Totalmente programável pelo usuário
- Portas de rede *Gigabit Ethernet*
    - Bloco de conectores do lado esquerdo do PCB com 4 conectores RJ45 externos.
    - Interfaces com padrão Cat5E ou Cat6 com cabos de rede de cobre usando o *chip Broadcom*.
    - Processamento *wire-speed* em todas as portas o tempo todo usando a lógica FPGA
        - 1 *Gbits* * 2 (bidirecional) * 4 (portas) = 8 Gbps de vazão
- Memória Estática de Acesso Aleatório (SRAM)
    - Adequado para armazenar os dados da tabela de encaminhamento
    - *Zero-bus turnaround (ZBT)* sincronizada com a lógica
    - Dois bancos paralelos de 18 *MBit* (2,25 *MByte*) - memórias ZBT
    - Capacidade total: 4,5 *Mbytes*
    - *Cypress*: CY7C1370D-167AXC
- Memória DDR2 Dinâmica de Acesso Aleatório (DDR2 DRAM)
    - 400 MHz de *clock* assíncrono
    - Apropriada para *buffer* de pacotes
    - 25,6 Gbps de taxa de transferência de pico
    - Capacidade total: 64 *Mbytes*
    - *Micron*: MT47H16M16BG-5E
- Fator de forma padrão PCI
    - Placa com *slot* padrão PCI
    - Pode ser usada em um *slot* PCI-X
    - Permite uma reconfiguração rápida do FPGA através do barramento PCI sem utilizar um cabo JTAG

Figura 5. Fotografia de uma placa NetFPGA-1G [7].

A placa NetFPGA deve ser instalada em um PC (*host*) através do *slot* PCI, e toda programação da placa é feita a partir do *host* pelo barramento PCI. A placa contém um chip FPGA (*Field Programmable Gate Array*), um circuito integrado que torna possível que o usuário reprograme todo o hardware da placa. O circuito integrado FPGA é o *Virtex-II Pro 50* produzido pela empresa *Xilinx*.

O diagrama de blocos na Figura 6 exibe os principais componentes da plataforma NetFPGA.

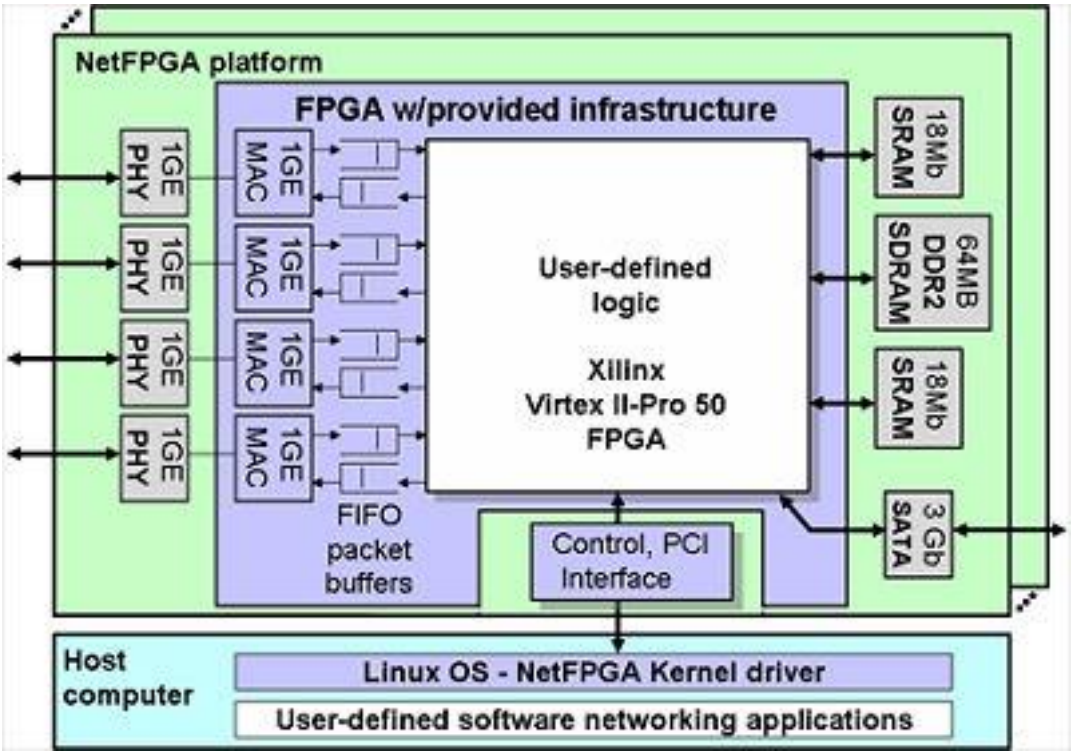

Figura 6. Diagrama de blocos da plataforma NetFPGA [16].

2.3.2 Pronto

Como consequência da fusão das empresas *Pronto Systems* e *Pica8* adveio a empresa *Pica8 Inc.* [8], a qual fornece soluções de plataforma de *switch* e rede abertas (*open switching platform* e *open networking platform*) [17]. Uma das soluções abertas oferecidas pela empresa foi utilizada neste trabalho, o *switch* Pronto 3290, um *switch* OpenFlow com 48 portas *gigabits Ethernet*. A Figura 7 mostra uma fotografia deste modelo.

Figura 7. Fotografia de um Pronto 3290 [18].

O Pronto 3290 é um *switch* de classe empresarial que tem dois modos de operação: nas camadas 2 e 3 (*Layer 2/3*) e com o OpenFlow 1.0. O *switch* pode ser usado na interconexão de clusters de alto desempenho, é escalável, e oferece ao usuário a possibilidade de personalização.

As principais especificações técnicas do *switch* Pronto 3290 estão listadas a seguir:

- Conectores
  - 48 portas 10/100/1000BASE-T RJ45
  - 4 portas 10Gbps SFP+ *uplink*
  - 2 portas de gerenciamento 10/100/1000 BASE-T
  - 1 porta serial de gerenciamento – 115.200 bps
- Dimensões
  - Tamanho (AxLxP): 40,915 (1U) x 423,85 x 274,4 mm
  - Peso: 5,5 kg
- Fonte de Alimentação
  - Tensões nominais de entrada: 110V & 240V

    - Frequência de entrada: 50/60 Hz
    - Consumo máximo de energia: 130W
- LEDs
    - Por porta RJ45: *Speed, Link/Activity*
    - Por porta SFP+: *Status, Rx, Tx, Link*
    - Por dispositivo: *Power, Status*
- Desempenho
    - Modos de encaminhamento: *Store-and-forward*
    - Largura de banda (*Bandwidth)*: 176 Gbps
    - Latência: média de 8 µs para *frames* de 64 *bytes*
    - Buffer de pacotes embutido no chip: 4 MB
    - Tempo médio entre falhas (MTBF): 46410 horas
- Processador de controle
    - *Freescale 8541* – 833Mhz
- Memória do sistema
    - DDR SDRAM 512MB
- *Boot flash size*
    - 32MB
- *Compact flash size*
    - 2GB

2.4 Controlador

O controlador é o principal dispositivo em uma rede definida por software. Ele é responsável por manter todas as regras da rede e distribuir as devidas instruções para os dispositivos da rede [12]. Dessa forma, o controlador centraliza a inteligência da rede, gerenciando as tabelas de fluxo de cada *switch* através da instalação e remoção de regras de fluxo dinamicamente.

Para uma melhor gerência da rede, é possível desenvolver aplicações que além de ter uma visão centralizada da rede, com análises detalhadas, pode também implementar novas funcionalidades para tomar decisões operacionais do sistema. Assim, o controlador atua como uma interface entre as aplicações de controle da rede e a própria rede [19].

Diversos controladores SDN foram desenvolvidos, a Tabela 3 apresenta algumas características dos principais controladores OpenFlow.

Tabela 3. Comparação dos principais controladores SDN.

Nome	Linguagem	Plataforma
NOX	C++	Linux
POX	Python	Windows, Linux e MacOS
Floodlight	Java	Windows, Linux e MacOS
Maestro	Java	Windows, Linux e MacOS
Beacon	Java	Windows, Linux, MacOS e Android
Trema (Helios)	C e Ruby	Linux
SNAC	C++	Linux

A escolha do controlador utilizado no desenvolvimento deste trabalho foi feita baseando-se nas principais características de cada um deles. Utilizamos o controlador Floodlight por ele ser multiplataforma, *open source* e fornecer uma API totalmente baseada na linguagem Java, o que potencialmente aumentaria a produtividade da implementação. Além do Floodlight, os controladores Beacon e Maestro também têm características similares e, portanto também poderiam ter sido utilizados. No entanto, preferimos utilizar o Floodlight por ser de classe empresarial, possuir uma boa documentação, ter uma ampla comunidade, demonstrar estar em um estágio de desenvolvimento mais avançado e fornecer uma interface amigável.

2.4.1 Floodlight

O desenvolvimento do controlador Floodlight surgiu do controlador Beacon e é mantido, testado e apoiado pela *Big Switch Networks* e pela maior comunidade de desenvolvedores do mundo para controladores SDN. Ele foi projetado para ser fácil de configurar, com poucas dependências e apresenta uma interface amigável tanto para o desenvolvedor quanto para o usuário. Distribuído sob a licença Apache, o Floodlight pode ser utilizado praticamente para qualquer propósito [20].

O controlador Floodlight suporta uma ampla gama de *switches* físicos e virtuais podendo ainda manipular redes mistas, OpenFlow e não-OpenFlow simultaneamente. Além de ser também compatível com a ferramenta Mininet, utilizada para simulação de redes virtuais.

O Floodlight oferece um sistema modular, o que facilita bastante para o desenvolvedor estender e melhorar. Além disso, qualquer pessoa pode contribuir com o código, que pode ser facilmente obtido diretamente do GitHub. A Figura 8 apresenta uma arquitetura de rede OpenFlow com o controlador Floodlight e vários *switches* OpenFlow conectados ao controlador.

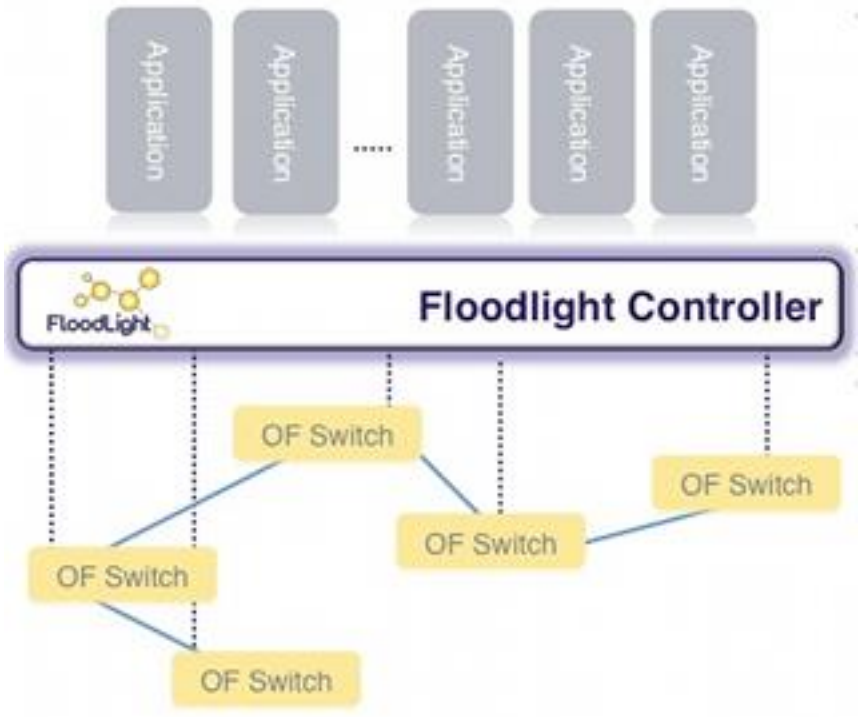

Figura 8. Arquitetura de rede OpenFlow com o controlador Floodlight [21].

Além de ser um controlador OpenFlow, o Floodlight também oferece uma coleção de aplicações desenvolvidas acima do controlador. Estas aplicações fornecem um conjunto de funcionalidades para atender a diferentes necessidades em uma rede OpenFlow. Enquanto que as funcionalidades internas do controlador compreendem um conjunto de serviços e módulos de interesse comum a qualquer aplicação SDN. A Figura 9 ilustra a relação entre o controlador e as aplicações.

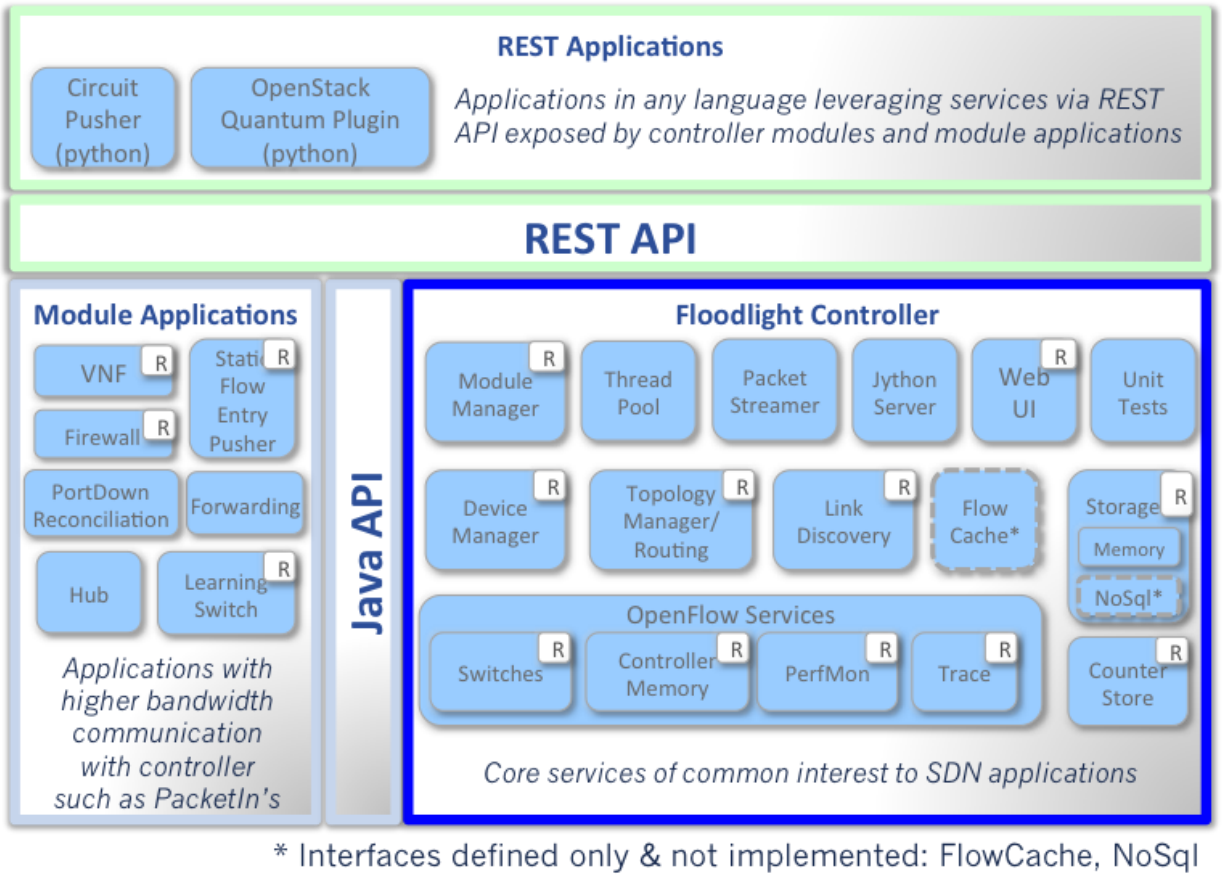

Figura 9. Relação entre o controlador Floodlight, os módulos de aplicação e as aplicações REST [22].

Os módulos de aplicação são implementados em Java e se comunicam com o controlador através de uma API Java. Tais aplicações estão mais inerentes ao controlador e exigem uma comunicação mais intensa com o controlador Floodlight como o *Forwarding* (módulo de encaminhamento de pacotes) e o *Static Flow Entry Pusher* (módulo que instala e remove regras de fluxo em um *switch* específico).

O outro conjunto de aplicações do controlador Floodlight são as aplicações REST (aplicações que trocam mensagens HTTP). Essas podem ser desenvolvidas em qualquer linguagem e se comunicam com o controlador Floodlight e com os módulos de aplicação através de uma API REST.

O controlador Floodlight é executado por padrão na porta 6633 e ainda provê uma interface *web* através da porta 8080 como pode ser visto na Figura 10.

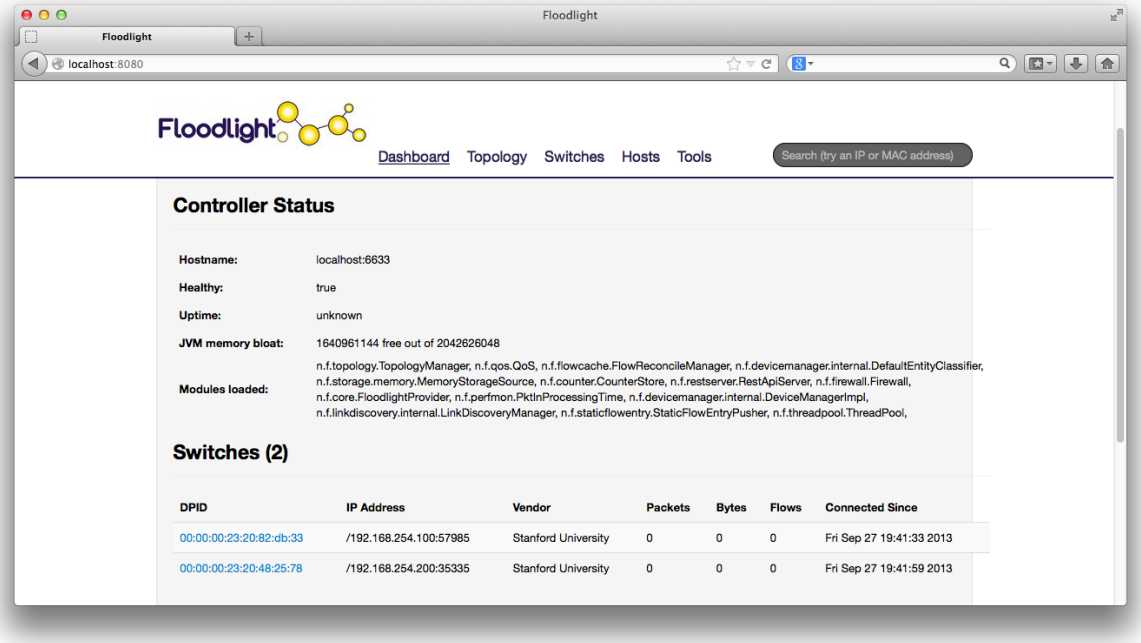

Figura 10. Interface *web* do controlador Floodlight.

No detalhe da Figura anterior, a Figura 11 mostra o status do controlador na interface *web*, e os *switches* controlados. Entre os módulos carregados estão os módulos de aplicação e os módulos internos do Floodlight.

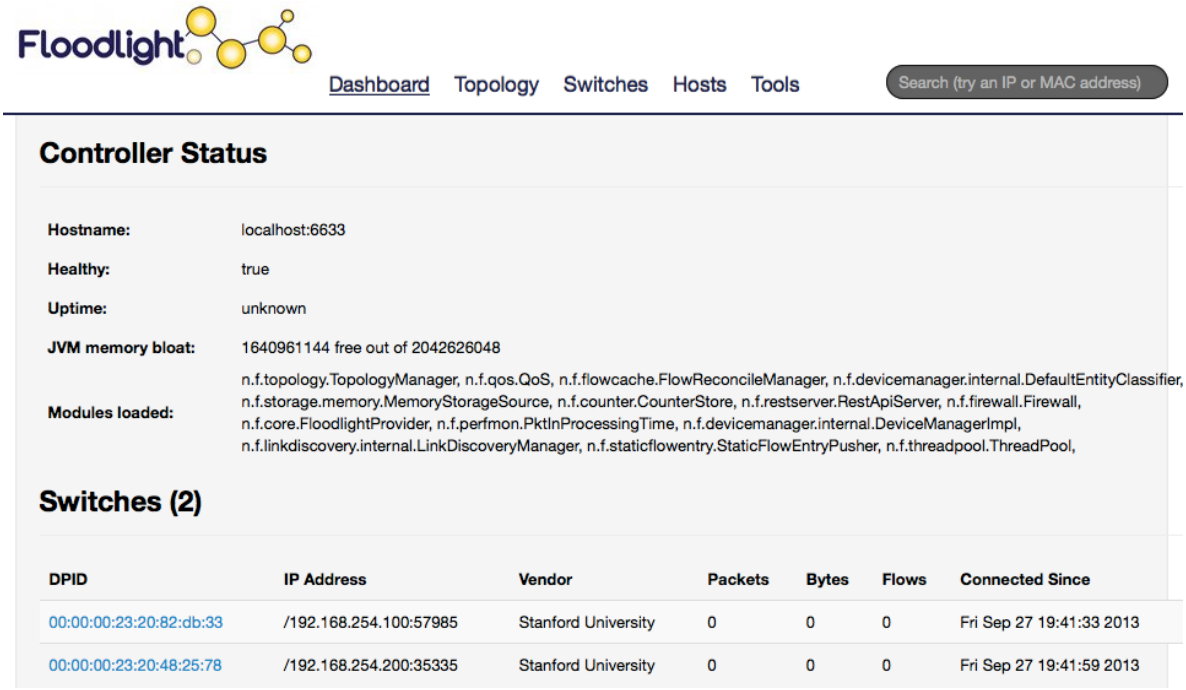

Figura 11. Status do controlador e *switches* OpenFlow NetFPGA controlados.

2.5 Qualidade de Serviço

A implantação de qualidade de serviço (QoS) na rede é essencial para o sucesso de aplicações avançadas, como telemedicina, videoconferência e VoIP (voz sobre IP). Estas aplicações demandam, além de grande largura de banda, um serviço diferenciado [5].

De uma maneira geral, o uso de QoS permite melhorar o fornecimento de um serviço para determinados fluxos de informação, aumentando a prioridade dos fluxos mais importantes em detrimento dos fluxos menos importantes (menor prioridade). Esse nível de importância para cada fluxo em específico é determinado de acordo com as classes de serviços.

O IETF (*Internet Engineering Task Force*) especifica duas propostas de arquitetura para fornecer qualidade de serviço. A arquitetura de serviços integrados (Intserv) e a arquitetura de serviços diferenciados (Diffserv). Intserv, é uma estrutura para fornecer garantias de qualidade de serviço específicas às sessões de aplicações individuais. Já o objetivo da Diffserv é prover a capacidade de manipular diferentes classes de tráfego de modos diferentes dentro da Internet [23]. A arquitetura Diffserv adotou o campo *Type-of-Service* (ToS) referente a um byte da camada 3 (cabeçalho IP) para priorização de tráfego. Os 6 *bits* mais significativos deste campo é chamado de *Differentiated Services Code Point* (DSCP).

As principais métricas para avaliação da qualidade de serviço são:

- **Largura de banda (*bandwidth*)**: Taxa de fluxo de dados suportada pela rede, tipicamente em *bits* por segundo;
- **Latência (atraso)**: O tempo que os dados levam para se deslocar da origem ao destino;
- ***Jitter**:* Variação do atraso de pacotes sucessivos; Influencia diretamente na qualidade de transmissões de tempo real como voz e vídeo;
- **Perda de pacotes**: Taxa de pacotes perdidos em uma transmissão, comumente medida em tráfegos de tempo real como voz e vídeo;
- **Vazão (*Throughput*)**: Taxa média de dados entregues com sucesso ao longo de um canal de comunicação.

2.6 Protocolo 802.1D/Q/P

No contexto de redes *Ethernet*, o protocolo IEEE 802.1P surgiu como uma proposta para priorização de tráfego baseada em classes de serviço [6]. O protocolo 802.1P consiste num campo de três *bits* reconhecido como PCP (*Priority Code Point*) que está contido no cabeçalho 802.1Q.

O cabeçalho 802.1Q é utilizado no contexto de VLANs, ou seja, o cabeçalho é criado apenas em fluxos de dados com a marcação (*tag*) de VLANs. A Figura 12 exibe o quadro *Ethernet* sem VLAN (*untagged*) e o quadro *Ethernet* com a marcação de VLAN (*tagged*).

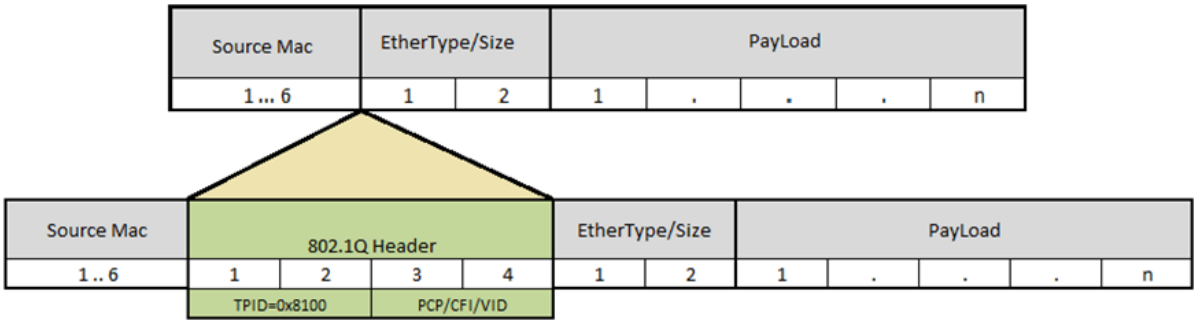

Figura 12. Cabeçalho 802.1Q adicionado ao quadro *Ethernet* [6].

O IEEE 802.1D ainda especifica um valor de prioridade entre 0 e 7 (oito classes) a ser atribuído no campo PCP para diferenciar o tráfego. *Vide* Tabela 4 [6].

Tabela 4. Acrônimos para o tipo de tráfego do IEEE802.1D [6].

Prioridade	Acrônimo	Tipo de tráfego
1	BK	*Background*
2	-	*Spare* (reservado)
0	BE	*Best Effort* (Melhor Esforço)
3	EE	*Excellent Effort* (Excelente Esforço)
4	CL	*Controled Load* (Carga Controlada)
5	VI	*Video* (Vídeo) < 100 ms* de Latência e *Jitter*
6	VO	*Voice* (Voz) < 10 ms* de Latência e *Jitter*
7	NC	*Network Control* (Controle de Rede)

3. PROPOSTA

3.1 Visão geral

De uma forma geral, este trabalho propõe um sistema de aprovisionamento de qualidade de serviço em redes OpenFlow. Assim, nos momentos de pico de tráfego na rede alguns usuários podem ter uma qualidade de serviço diferenciada de acordo com sua classe de serviço.

Neste contexto, construímos um módulo de aplicação para o controlador Floodlight, o qual permite que classes de serviço sejam gerenciadas através da criação e remoção de novas classes. Da mesma forma que outros módulos de aplicação, o módulo de QoS se comunica com o núcleo do controlador através da API Java e fornece uma API REST para gerenciamento. A Figura 13 indica o local em que o módulo de QoS foi desenvolvido na arquitetura do controlador.

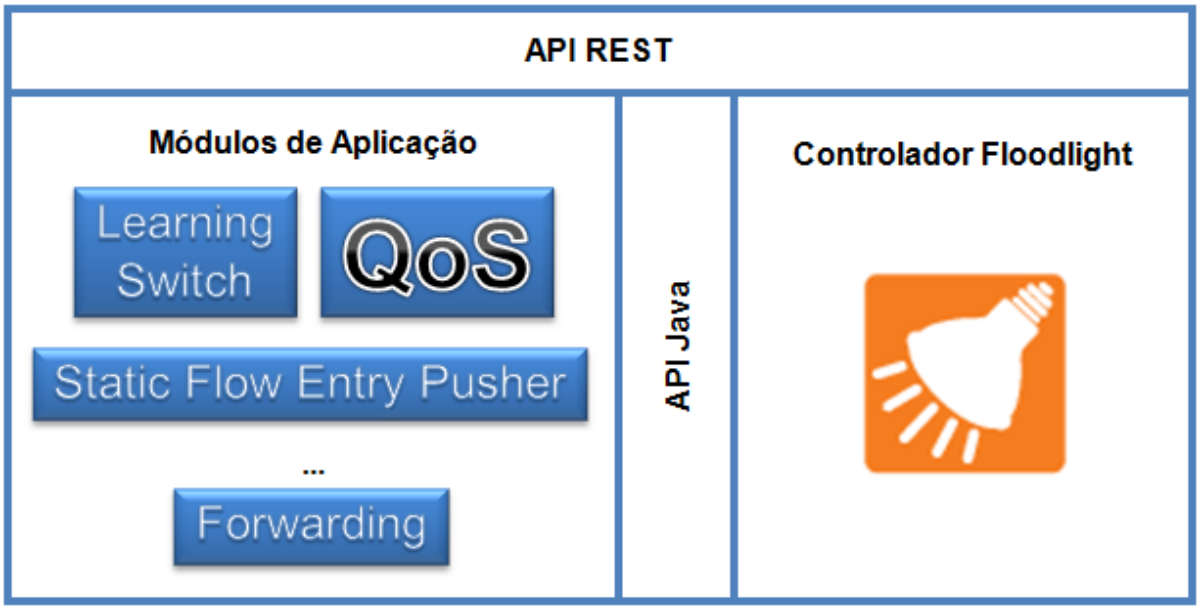

Figura 13. Módulo de QoS na arquitetura do controlador Floodlight.

Sendo assim, para cada classe de serviço deve haver um mapeamento com um nível de prioridade associado. Para diferenciar a prioridade de um tráfego de dados sobre outro, utilizamos o protocolo IEEE 802.1P. Desta forma, para cada classe de serviço é atribuído um nome e um nível de prioridade (PCP), podendo receber valores de 0 (menor prioridade) a 7 (maior prioridade). A Tabela 5 mostra um mapeamento entre as classes de serviço, o PCP e os usuários.

Tabela 5. Mapeamento entre as classes de serviço, o nível de prioridade e os usuários.

PCP	Classe de Serviço	Usuários
0	Melhor esforço	Cliente 1
3	Excelente Esforço	Cliente 2
5	Tráfego Crítico	Cliente 3

A principal vantagem em utilizar o protocolo 802.1P como abordagem de aprovisionamento de QoS é o fato deste protocolo envolver somente a camada 2, permitindo que *switches Ethernet* (*Layer* 2) interpretem o campo PCP utilizado para priorização.

Do ponto de vista de execução, o módulo QoS desenvolvido para o controlador Floodlight instala regras nos *switches* OpenFlow, de modo que os *switches* verificam suas tabelas de fluxo e aplicam os devidos valores ao campo PCP de cada fluxo de dados recebido, de acordo com os parâmetros especificados. Dessa forma, cada fluxo de dados segue na rede com o seu respectivo valor do campo PCP.

Após o valor do campo PCP do fluxo de dados ser atribuído ao primeiro *switch*, os próximos saltos referentes aos próximos *switches* da rede seguem priorizados independentemente deles suportarem o protocolo OpenFlow. No entanto, para que o tráfego de dados de um cliente da rede seja efetivamente priorizado, é preciso utilizar *switches* que implementem a priorização de filas a partir do campo PCP.

E finalmente, também é necessário alocar os usuários da rede em suas respectivas classes de serviço de acordo com parâmetros especificados. Tais parâmetros podem ser o endereço IP, o endereço MAC, a porta conectada ao *switch*, a VLAN pertencente ao cliente, entre outros.

Toda a interação entre o usuário do módulo (gerente ou administrador da rede) e o módulo de QoS é feito através de comandos HTTP da API REST disponibilizada pelo módulo. Entretanto, uma aplicação com interface gráfica poderia ser desenvolvida sobre a API REST como um complemento ao módulo QoS.

3.2 Cenário

Para colocar o módulo de QoS em operação, precisamos de uma topologia de rede local, com clientes, servidores, *switches* OpenFlow, *switches* que implementem a priorização de filas a partir do PCP, e por fim, um controlador Floodlight. Em um cenário ideal, os próprios *switches* OpenFlow também implementam a priorização de filas a partir do PCP. A topologia correspondente a este cenário pode ser visualizada conforme a Figura 14. O fluxo de dados é mostrado em azul, enquanto o fluxo de controle da rede é mostrado em verde e tracejado.

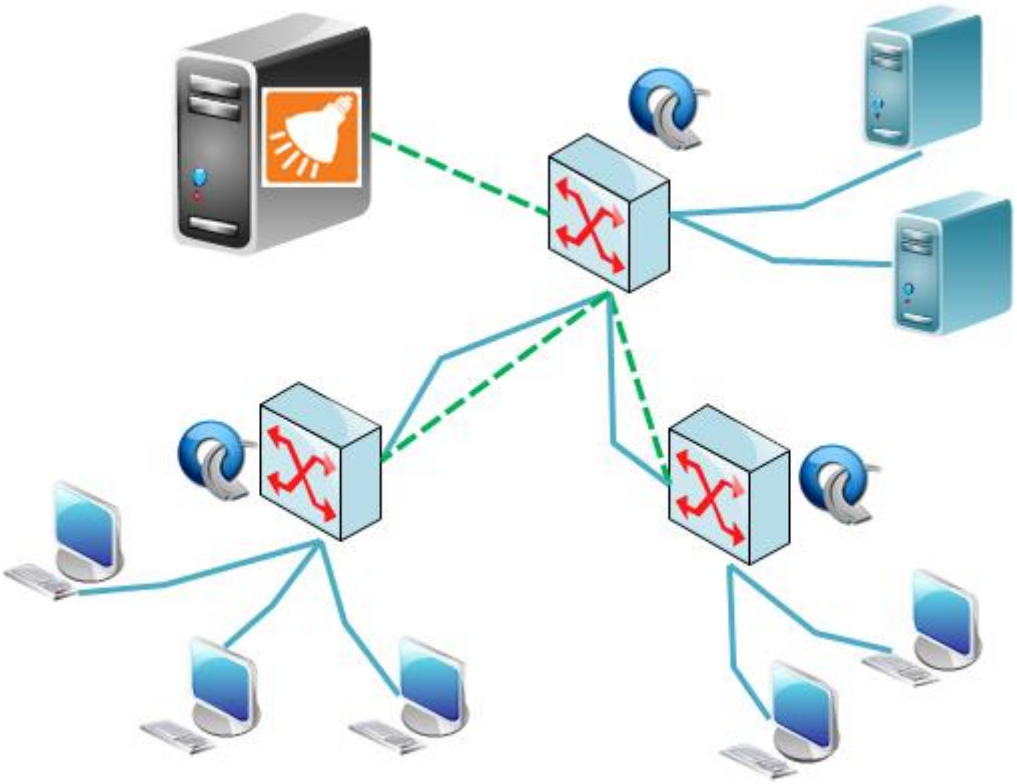

Figura 14. Topologia ideal de operação do módulo de QoS.

Como todo o processo de priorização dos dados é feito na camada 2, esta abordagem de aprovisionamento de QoS se aplica apenas a redes locais, pois em um fluxo de dados entre domínios distintos, o cabeçalho da camada 2 é reconstruído no roteamento.

Outro ponto importante é que precisamos envolver a marcação de VLANs nos fluxos de dados, pois estamos utilizando o protocolo 802.1P, e que por sua vez faz parte do protocolo 802.1Q.

4. PROJETO E IMPLEMENTAÇÃO

A implementação deste projeto foi dividia em três partes: *testbed*, módulo de QoS e experimentos. A primeira parte se refere ao ambiente de experimentação construído, envolvendo detalhes da instalação e configuração da infraestrutura. A segunda trata do desenvolvimento do módulo de QoS propriamente dito, e a última detalha os experimentos realizados com o módulo de QoS no *testbed*.

4.1 Testbed

4.1.1 Visão geral

A infraestrutura criada foi composta por dois clientes, um servidor, dois *switches* OpenFlow NetFPGAs, um *switch* OpenFlow Pronto e um controlador Floodlight. A Figura 15 apresenta a topologia implementada.

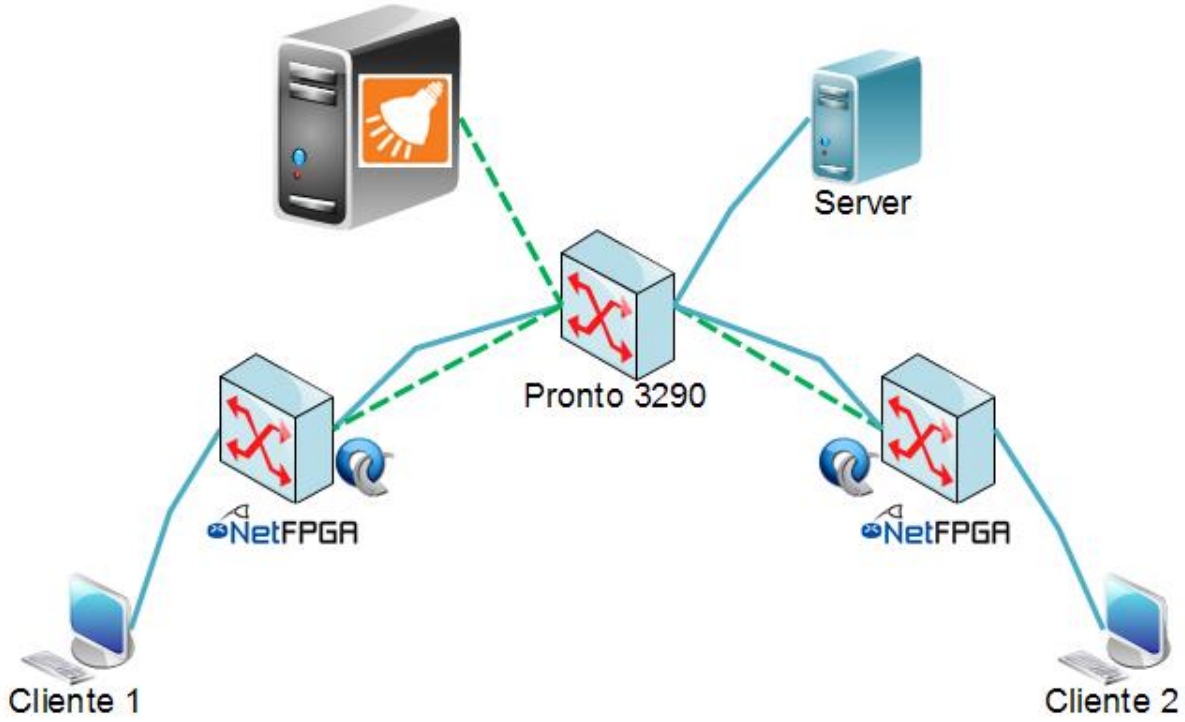

Figura 15. Topologia implementada.

A rede de dados (em azul) foi definida com VLAN 100 (VLAN ID = 100), enquanto que a rede de controle (em verde e tracejado) foi definida com VLAN 254 (VLAN ID = 254). Assim, os clientes e o servidor pertencem a VLAN 100 com IP da rede 192.168.100.0/24 enquanto que os *switches* OpenFlow e o controlador Floodlight pertencem a VLAN 254 com IP da rede 192.168.254.0/24.

Nesta topologia, os *switches* NetFPGAs são ligados diretamente aos clientes, e portanto, são os responsáveis por atribuir o nível de prioridade aos fluxos de dados recebidos dos clientes de acordo com as instruções enviadas pelo controlador Floodlight. Já o *switch* Pronto 3290 é o responsável por implementar as filas de prioridades baseadas no campo PCP.

Perceba que o objetivo desta topologia é priorizar o tráfego no sentido de *upload* de um cliente sobre o outro, ou seja, apenas no envio de dados do cliente para o servidor. Para priorizar o tráfego de *download*, é necessário que os *switches* NetFPGAs também implementem a priorização baseada no campo PCP, no entanto apenas o *switch* Pronto 3290 dispõe deste recurso (ainda que não seja um recurso padrão e tenha sido implementado manualmente).

Desta forma, há uma separação nas funções dos *switches* NetFPGAs do *switch* Pronto. Os *swiches* NetFPGAs são os responsáveis pelo aprovisionamento da qualidade de serviço, repassando todo o tráfego originado dos clientes para o *switch* Pronto com uma alteração no campo PCP do fluxo de dados. E o *switch* Pronto prioriza efetivamente os fluxos de dados baseado no campo PCP previamente definido pelo *switch* NetFPGA.

Uma decisão tomada no decorrer do projeto foi de utilizar apenas um cliente em cada *switch* NetFPGA devido ao fato de cada interface da placa NetFPGA ter largura de banda de 1 Gbps. E como cada cliente também tem uma interface de 1 Gbps, haveria perda de pacotes ainda no primeiro salto (antes que os fluxos chegassem ao *switch* Pronto) nos momentos de stress em dois ou mais clientes simultaneamente.

Como o enlace entre o servidor e o *switch* Pronto também tem uma interface de 1 Gbps, é esperado que se tenha perda de pacotes quando ambos os clientes estiverem enviando dados em suas capacidades máximas, o que implica em uma alta taxa de transferência de dados (próximo dos 2 Gbps). Porém, um dos clientes deve perder mais pacotes do que o outro de acordo com seu nível de prioridade.

Do ponto de vista físico, foram utilizados três servidores de virtualização executando o sistema operacional de virtualização *VMware vSphere 5 Hypervisor*. Dois deles são do modelo IBM System x3650 M3, e o outro é um IBM System x3400

M3. Os modelos idênticos foram instalados igualmente, ambos com uma placa NetFPGA e duas máquinas virtuais, uma para o *switch* NetFPGA e outra para representar um cliente. O controlador Floodlight foi instalado em uma máquina virtual do servidor IBM System x3400 M3. E por fim, um PC *desktop* foi utilizado como servidor. Os servidores IBM System x3650 e IBM System x3400 podem ser vistos respectivamente nas Figuras 16 e 17.

Figura 16. Fotografia de um servidor IBM System x3650 M3 [26].

Figura 17. Fotografia de um servidor IBM System x3400 M3 [27].

4.1.2 Clientes

As duas máquinas virtuais dos clientes foram instaladas de forma idêntica, cada uma em um servidor IBM System x3650 M3. O sistema operacional instalado foi o Linux com a distribuição Ubuntu 12.04.3 LTS em sua versão *Desktop* com arquitetura de 32 *bits*. A quantidade de memória RAM alocada para as máquinas virtuais foi de 2 GB e um disco de 16 GB.

Após a instalação, as máquinas foram configuradas na rede com o IP 192.168.100.1 para o Cliente 1 e 192.168.100.2 para o Cliente 2. A VLAN também foi configurada através do gerenciador de máquinas virtuais do VMware e definida com o valor referente a rede de dados (VLAN ID = 100), para ambas. Além disso, cada máquina virtual foi configurada para utilizar uma interface de rede do servidor físico e foi diretamente conectada via cabo de rede em uma das interfaces de rede disponíveis da sua respectiva placa NetFPGA.

Para finalizar a configuração das máquinas virtuais dos clientes, o programa *iperf* foi instalado para fins de teste. O *iperf* é uma ferramenta para gerar e medir tráfego de rede TCP e UDP entre duas máquinas, além de reportar alguns parâmetros de QoS como vazão, atraso e perda de pacotes. Outros pacotes básicos também foram instalados e uma atualização de todo o sistema incluindo os pacotes já instalados também foi feita.

4.1.3 Servidor

O servidor da rede é uma máquina física do tipo PC *desktop* com 4 GB de memória, 320 GB de disco e processador *Intel Core i5 650*. Tem a distribuição Linux Ubuntu 12.04.2 LTS instalada na versão *Desktop* com arquitetura de 32 *bits*.

Neste caso, a configuração da VLAN foi feita diretamente no sistema operacional da máquina, e não pela interface de gerenciamento do VMware, pois este servidor é uma máquina física, e sua interface de rede foi diretamente conectada ao *switch* Pronto.

A VLAN foi configurada através da instalação do pacote vlan do Ubuntu e do carregamento do módulo 8021q. Como o servidor está na rede dados, recebeu a VLAN 100, e obteve o IP 192.168.100.3.

4.1.4 Máquina do Controlador Floodlight

As configurações da máquina virtual do controlador foram similares às configurações dos clientes com algumas alterações. Uma das diferenças é que ela foi instalada no servidor IBM System x3400 M3. O sistema operacional instalado também foi o Linux com a distribuição Ubuntu 12.04.3 LTS em sua versão *Desktop* com arquitetura de 32 *bits*. A quantidade de memória RAM alocada também foi de 2 GB com um disco de 16 GB.

Diferentemente dos clientes, a VLAN determinada para a máquina virtual do controlador foi a 254 (VLAN ID = 254), pois esta máquina pertence à rede de controle e não há necessidade de estar na mesma VLAN dos clientes, mas sim na mesma VLAN dos *switches* OpenFlow. O IP atribuído para o controlador foi o 192.168.254.1. Uma interface de rede física do servidor foi associada à máquina virtual e ela foi conectada via cabo de rede em uma das interfaces de rede disponíveis do *switch* Pronto, conforme a topologia apresentada.

Os detalhes referentes à implementação do controlador Floodlight propriamente dito serão apresentados no tópico 4.2, o qual trata do projeto e implementação do Módulo de QoS utilizando o controlador Floodlight.

4.1.5 *Switch* OpenFlow NetFPGA

Como já foi mencionado, cada *switch* OpenFlow NetFPGA foi configurado em uma máquina virtual de um servidor de virtualização IBM System x3650 M3 com uma placa NetFPGA. O sistema operacional utilizado foi o Fedora *Core* 13 instalado a partir da imagem disponibilizada na seção *Releases* da Wiki do NetFPGA [16]. A quantidade de memória alocada também foi de 2 GB, já o disco foi de 12 GB, um pouco menor que as outras máquinas virtuais por não precisar armazenar nenhuma informação extra além de executar o *switch* OpenFlow

O *switch* OpenFlow NetFPGA é formado pela placa NetFPGA mais a máquina virtual hospedeira. As interfaces de rede da placa NetFPGA não recebem IP, pois atuam como o *switch* propriamente dito. Enquanto que a interface de rede da máquina virtual recebe um IP para se comunicar com o controlador. Cada *switch* OpenFlow NetFPGA recebeu três cabos de rede, um que conecta o cliente à placa

NetFPGA, outro que conecta a placa NetFPGA ao *switch* OpenFlow Pronto, e por fim, mais um que se liga ao *switch* OpenFlow Pronto, mas vindo da interface de rede da máquina virtual, a qual é associada a uma interface de rede física do servidor.

O IP atribuído a interface de rede da máquina virtual de cada *switch* OpenFlow NetFPGA foi o 192.168.254.100 para o *switch* ligado ao Cliente 1 e 192.168.254.200 para o *switch* ligado ao Cliente 2. Como essas interfaces devem estar na mesma VLAN do controlador, foi atribuída a VLAN 254 (VLAN ID = 254) para ambas.

Depois da instalação de utilitários (como o *Wireshark*, por exemplo) e alguns pacotes básicos e necessários para executar a placa NetFPGA, foram realizados testes operacionais e funcionais com cada uma das placas NetFPGAs conforme a documentação disponível na Wiki NetFPGA [16]. Além disso, também foram instalados os pacotes do OpenFlow.

Para tornar a máquina virtual, com a placa NetFPGA, em um *switch* OpenFlow é necessário fazer o *download* do *bitfile* para a placa NetFPGA. O *bitfile* é um arquivo que contém a implementação de baixo nível a ser instalada na placa NetFPGA. No nosso caso, o *bitfile* é o “openflow_switch.bit”, que contém a implementação do *switch* OpenFlow 1.0 para o NetFPGA. A realização desse passo é feita através do comando *nf_download*. A Figura 18 exibe a execução e a saída deste comando.

Uma vez com o *bitfile* do OpenFlow 1.0 já instalado na placa NetFPGA, deve-se inicializar o *switch* OpenFlow NetFPGA. Para isso, dois comandos do pacote OpenFlow devem ser executados, o *ofdatapath* e o *ofprotocol*. No *ofdatapath* nós especificamos as interfaces de rede do NetFPGA que serão utilizadas no caminho de dados. Já no *ofprotocol*, precisamos especificar o IP do controlador da rede.

Para automatizar o procedimento, criamos um script que executa os três comandos explicados de uma só vez. Contanto, é necessário passar como parâmetro o IP do controlador. A Figura 19 mostra o conteúdo do script e a Figura 20 mostra a saída do script, perceba que o *switch* OpenFlow NetFPGA estabelece uma comunicação com o controlador através do IP 192.168.254.1.

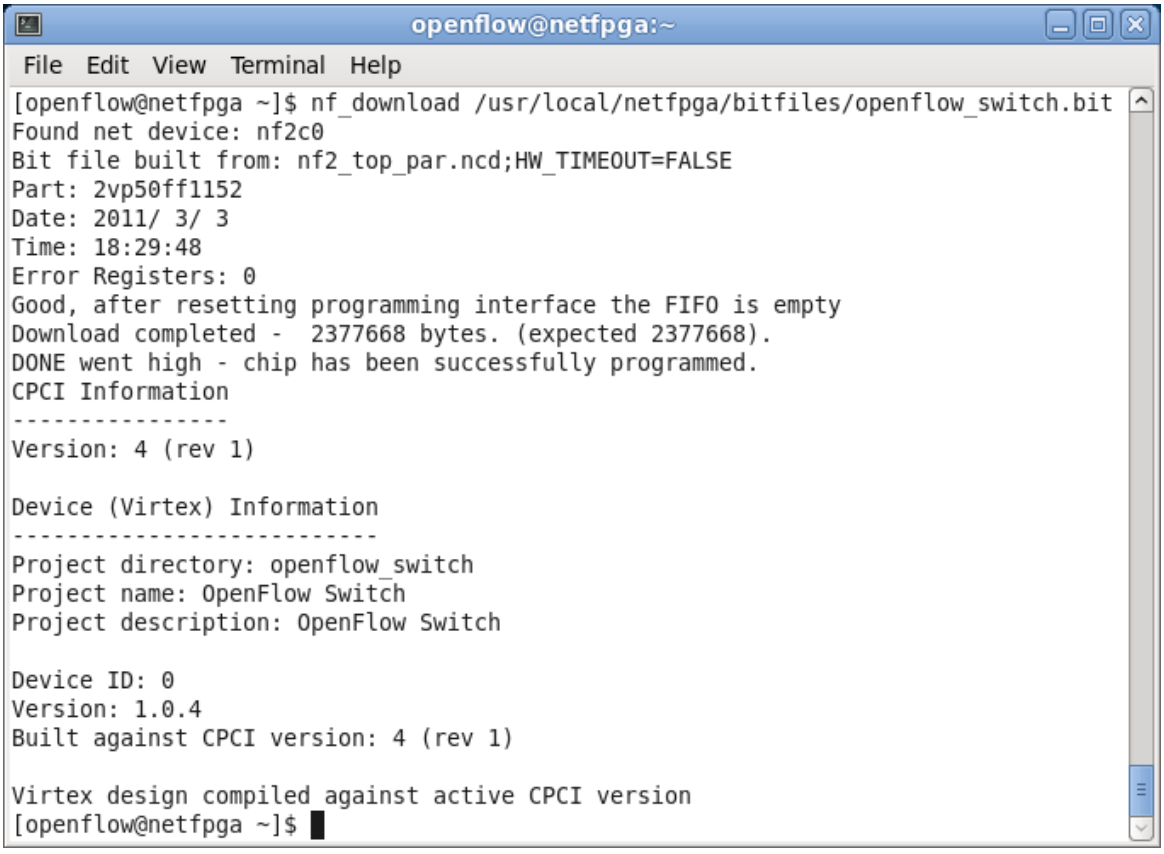

Figura 18. *Download* do *bitfile* "openflow_switch.bit" para uma das placas NetFPGAs.

Figura 19. Script para inicializar um dos *switches* OpenFlow NetFPGAs.

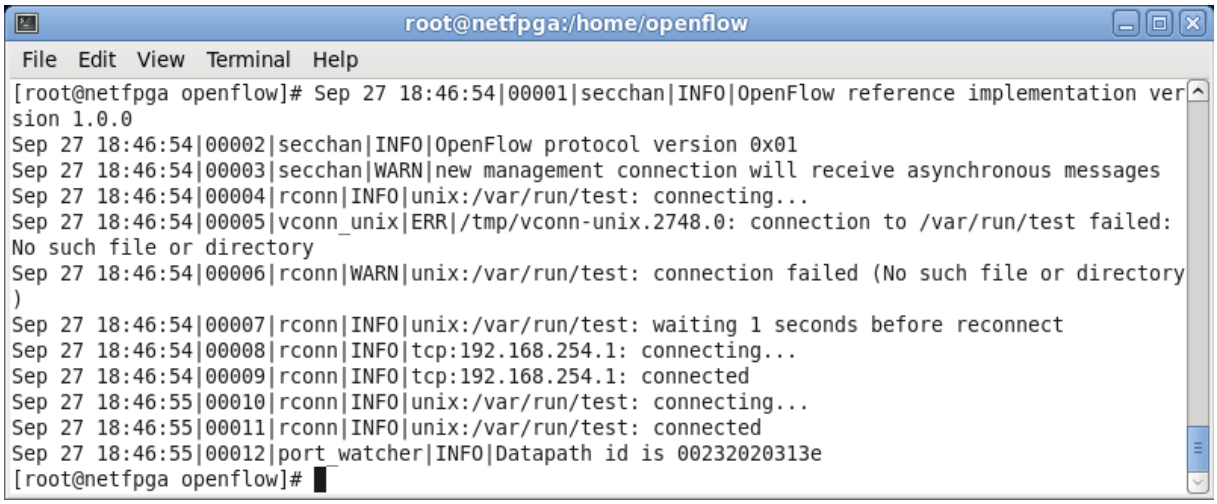

Figura 20. Inicialização de um dos *switches* OpenFlow NetFPGAs.

4.1.6 *Switch* OpenFlow Pronto

A funcionalidade de priorização de filas baseado no protocolo 802.1P foi implementada no projeto Revir [6] utilizando como base o projeto *open source* chamado Indigo [24], o qual faz parte do projeto Floodlight desenvolvido pela *Big Switch Networks*.

O Indigo foi modificado através do IODS (*Open Development System*), uma máquina virtual que contém todo o código fonte do Indigo, o que tornou possível modificar o código, adicionando outras aplicações ou características [6]. A partir do código fonte resultante criou-se uma nova imagem, que foi instalada no *switch* em substituição do Pica8 [17], instalado por padrão.

O Indigo também possui uma interface *web* de configuração. Assim, o gerenciamento pode ser feito a partir de um browser. A Figura 21 mostra uma tela da interface *web* do Indigo modificado para o projeto ReVir [6].

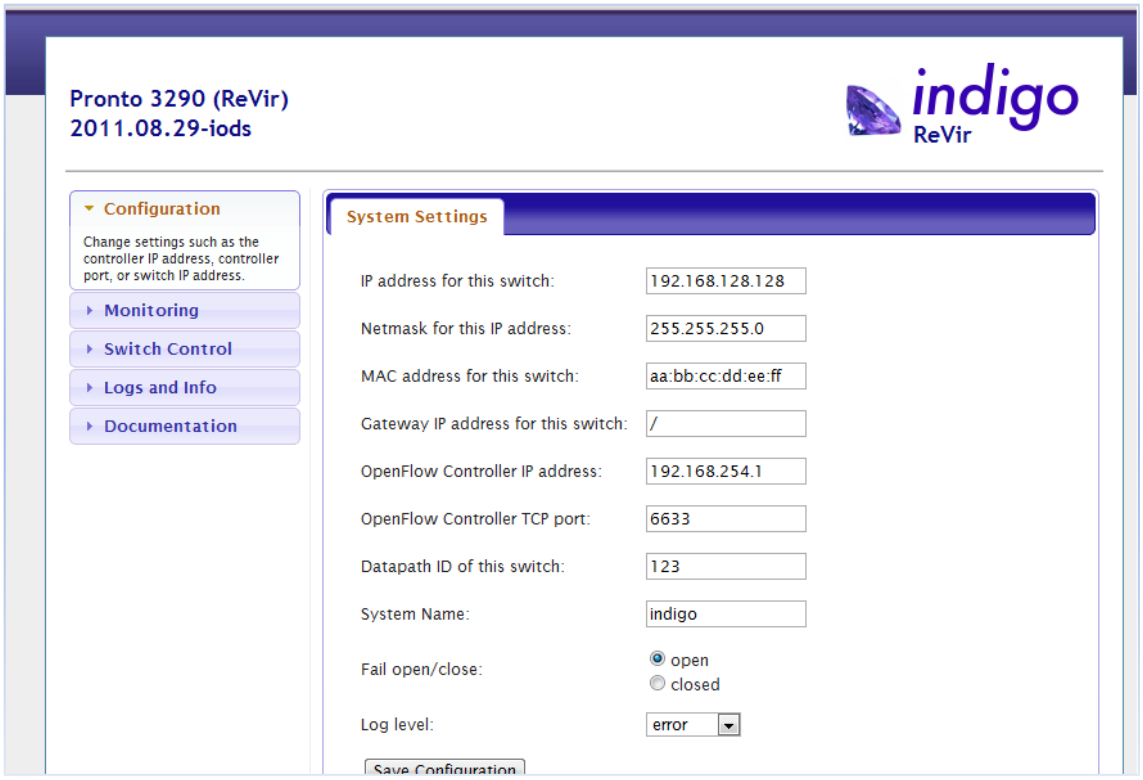

Figura 21. Interface *web* de configuração do Indigo/Revir [6].

Usando a biblioteca PQLib [25], uma robusta fila de prioridades baseada em árvores foi desenvolvida na linguagem C e atua como um escalonador entre os diversos fluxos [6].

4.2 Módulo de QoS

O módulo de QoS foi desenvolvido utilizando o projeto floodlight-qos-beta como base. O projeto floodlight-qos-beta está disponível no GitHub, foi originalmente desenvolvido por Ryan Wallner e faz parte da documentação do controlador Floodlight.

Este módulo foi implementado na linguagem Java, é integrado à interface *web* do Floodlight e disponibiliza uma interface REST para gerenciamento. O foco deste projeto é o aprovisionamento de QoS baseado em filas de prioridade ou no campo ToS/DSCP. No entanto, apenas um modo de priorização pode ser acionado por vez, ou baseado nas filas de prioridade ou no campo ToS.

As Figuras 22 e 23 exibem o módulo de QoS na seção *Tools* da interface *web* do controlador Floodlight.

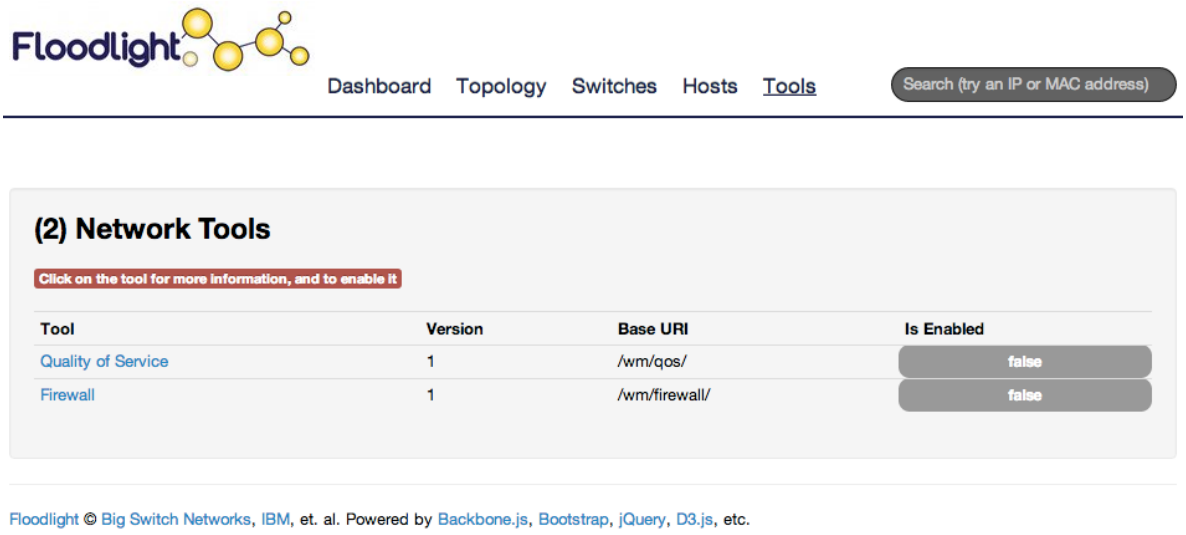

Figura 22. *Network Tools*: Módulos integrados a interface *web* do controlador Floodlight.

Figura 23. Interação do módulo de QoS com a interface *web* do controlador Floodlight.

A priori, a tarefa deste trabalho foi de adaptar o aprovisionamento de QoS baseado no campo ToS para o campo PCP. No entanto, mesmo reaproveitando quase todo o código do projeto floodlight-qos-beta, esta tarefa não foi trivial. Foi necessário entender e modificar quase todas as classes Java do código fonte do projeto, e ainda, manipular outros arquivos responsáveis pela integração do módulo com o *webserver*, envolvendo linguagens como JavaScript e HTML.

Além das alterações relativas à adaptação do campo ToS ao campo PCP, houve mudanças ainda mais complexas devido a dois *bugs* somente identificados durante a execução da funcionalidade de criação de políticas baseadas no campo ToS. Em outras palavras, o projeto original não estava funcional quanto a priorização baseada no campo ToS, a própria documentação do módulo somente exibe exemplos do módulo de aprovisionamento de QoS utilizando filas de prioridade.

Classificamos as duas falhas identificadas como críticas, pois ambas envolvem o fato dos fluxos de dados filtrados pelas políticas definidas não serem encaminhados. Se não há encaminhamento, não há comunicação entre máquinas, inutilizando o serviço. Para que os fluxos de dados sejam encaminhados pelos *switches*, é preciso que uma ação seja criada ordenando o *switch* a encaminhar os pacotes.

A primeira falha identificada foi que somente a ação de definir o campo PCP era realizada após a política ser aplicada. Ou seja, os fluxos eram filtrados devidamente, a prioridade do fluxo era corretamente determinada baseada na classe de serviço, mas os fluxos não eram encaminhados. Para corrigir o problema, adicionamos uma ação para encaminhar os fluxos, além da ação que definia o campo PCP.

No momento em que corrigimos a primeira falha é que identificamos a outra, esta já não era mais no módulo de QoS e sim no módulo *Static Flow Entry Pusher*, um módulo padrão do controlador Floodlight. O encaminhamento só era feito quando especificávamos o número da porta de saída. Mas não funcionava quando especificávamos valores genéricos como *all*, *flood* e *normal*, respectivamente equivalentes a encaminhar para todas as portas, encaminhar para todas as portas exceto a porta de entrada e realizar o encaminhamento normal como de um *switch* comum. Para resolver esse problema um método do módulo *Static Flow Entry*

Pusher foi modificado para suportar os valores genéricos de forma compatível com o módulo de QoS.

No que diz respeito aos serviços e funções disponibilizados pelo módulo de QoS, é possível criar classes de serviço através do nome e do nível de prioridade (valor PCP), além de ativar, desativar e verificar o status (ativado/desativado) do módulo. Toda interação é realizada através da interface REST.

A Tabela 6 exibe os comandos HTTP disponibilizados pela API REST do módulo de QoS.

Tabela 6. API REST do Módulo de QoS. Tabela adaptada [22].

API REST	GET	POST	Descrição
/wm/qos/tool/<op>/json	Sim	N/A	<op>: "*status*", "*enable*" ou "*disable*".
/wm/qos/service/json	Sim	Sim	Cria uma Classe de Serviço (CoS).
/wm/qos/policy/json	Sim	Sim	Cria uma política associada a uma CoS.

A Tabela 7 descreve os campos especificados na criação de uma classe de serviço.

Tabela 7. Campos referentes a criação de uma classe de serviço. Tabela adaptada [22].

Campo	Descrição
Id	Identificador único da classe de serviço.
name	Nome associado a política definida pelo campo PCP (ex: “Melhor Esforço”).
pcp	Valor do campo PCP referente ao nível de prioridade (de 0 a 7).

Por fim, podemos criar políticas para determinados fluxos de dados e associá-los as suas respectivas classes de serviço de acordo com os parâmetros especificados, como a identificação do *switch*, a identificação da VLAN, e dados de origem e/ou destino como IP, MAC e porta.

4.3 Resultados

Para facilitar a execução dos testes e mudanças na arquitetura do *testbed*, todas as portas *Ethernet* das placas NetFPGAs e dos servidores foram espelhadas através de *patch panels*. O rack utilizado no projeto, com todo o cabeamento montado, ligando *switches*, servidores e *patch panels*, é mostrado na Figura 24.

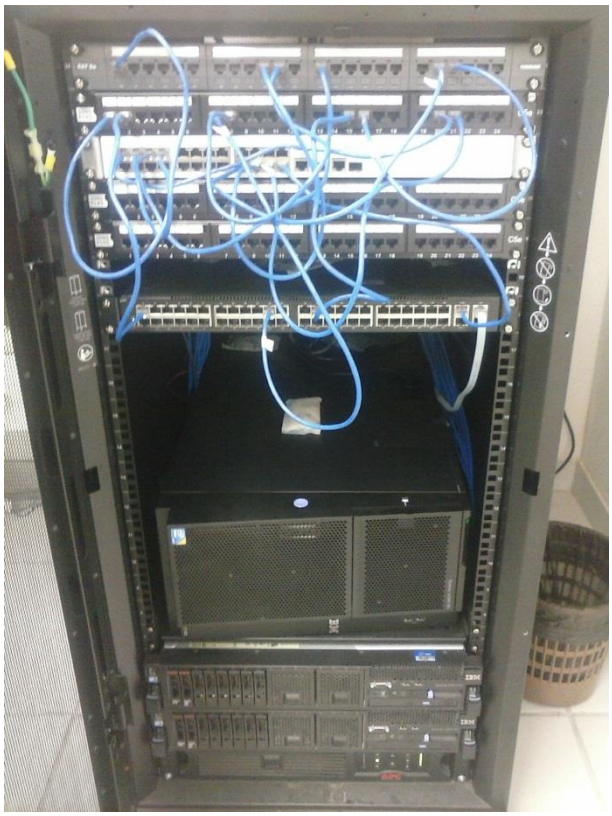

Figura 24. Fotografia do rack com os servidores, *switches* e *patch panels*.

A Figura 25 mostra a console de inicialização do controlador Floodlight. Logo após a inicialização do controlador, os *switches* que estão apontando para o IP do controlador (192.168.254.1) são detectados.

```
mraga-MacBook:floodlight-qos-beta administrador$ ./floodlight.sh
Starting floodlight server ...
INFO [net.floodlightcontroller.core.module.FloodlightModuleLoader:main] Loading default modules
INFO [net.floodlightcontroller.core.internal.Controller:main] Controller role set to null
disabled
INFO [net.floodlightcontroller.core.internal.Controller:main] Listening for switch connections on 0.0.0.0/0.0.0.0:6633
INFO [net.floodlightcontroller.core.internal.Controller:New I/O server worker #1-1] New switch connection from /192.168.254.128:35630
INFO [net.floodlightcontroller.core.internal.Controller:New I/O server worker #1-2] New switch connection from /192.168.254.100:49846
INFO [net.floodlightcontroller.core.internal.Controller:New I/O server worker #1-3] New switch connection from /192.168.254.200:33391
INFO [net.floodlightcontroller.jython.JythonServer:debugserver-main] Starting DebugServer on port 6655
INFO [net.floodlightcontroller.mactracker.MACTracker:New I/O server worker #1-1] MAC Address: 00:00:00:0c:29:ff:ad:28 seen on switch: 291
INFO [net.floodlightcontroller.mactracker.MACTracker:New I/O server worker #1-2] MAC Address: 00:00:00:0c:29:8f:6b:36 seen on switch: 150868475056
INFO [net.floodlightcontroller.mactracker.MACTracker:New I/O server worker #1-3] MAC Address: 00:00:00:0c:29:2f:ab:1a seen on switch: 150865448604
```

Figura 25. Inicialização do controlador Floodlight.

A Figura 26 exibe a interface *web* com os *switches* e *hosts* detectados pelo controlador Floodlight.

Switches (3)

DPID	IP Address	Vendor	Packets	Bytes	Flows	Connected Since
00:00:00:23:20:76:3c:b0	/192.168.254.100:49846	Stanford University	0	0	0	9/28/2013 5:37:54 PM
00:00:00:23:20:48:0e:9c	/192.168.254.200:33391	Stanford University	0	0	0	9/28/2013 5:37:54 PM
00:00:00:00:00:00:01:23	/192.168.254.128:35630	Indigo OpenFlow from Big Switch Networks	0	0	0	9/28/2013 5:37:54 PM

Hosts (3)

MAC Address	IP Address	Switch Port	Last Seen
00:0c:29:ff:ad:28	192.168.100.3	00:00:00:00:00:00:01:23-2	9/28/2013 5:38:38 PM
00:0c:29:2f:ab:1a	192.168.100.2	00:00:00:23:20:48:0e:9c-2	9/28/2013 5:38:38 PM
00:0c:29:8f:6b:36	192.168.100.1	00:00:00:23:20:76:3c:b0-2	9/28/2013 5:38:32 PM

Figura 26. *Switches* e *hosts* detectados pelo controlador Floodlight.

A topologia da rede detectada pelo controlador Floodlight é apresentada na Figura 27 através da seção *Topology* da interface *web*. Os clientes são identificados ligados aos *switches* OpenFlow NetFPGA e o servidor aparece conectado ao *switch* OpenFlow Pronto.

Figura 27. Topologia da rede.

Ativamos o módulo de QoS e adicionamos algumas classes de serviço através da interface API REST utilizando a ferramenta *curl* para enviar os comandos HTTP. Uma vez com as classes de serviço definidas, criamos duas políticas *Switch01* e *Switch02*. A primeira política filtra todos os fluxos da VLAN 100 que chegam ao *switch* OpenFlow NetFPGA com IP 192.168.254.100, e associa tais fluxos ao serviço *Low Priority* (PCP = 1). Similarmente, a outra política filtra os fluxos da VLAN 100 no *switch* OpenFlow NetFPGA com IP 192.168.254.200 e encaminha os fluxos associando-os ao serviço *High Priority* (PCP = 5). A Figura 28 exibe as classes de serviço e as políticas criadas.

Manage QoS

Services

Id	Service	PCP
1468405953	Best Effort	0
486984297	Low Priority	1
784366547	High Priority	5

Policies

Id	Policy	Match Fields	Switches	Enqueue #:#	Service	Priority
1810829772	Switch02	ethernet-type=-1, protocol=-1, ingress-port=-1, ip-dest=null, ip-src=null, tos-bits=-1, vlan-pcp=-1, vlan-id=100, ethsrc=null, ethdst=null, tcpdstport=-1, tcpsrcport=-1,	00:00:00:23:20:48:0e:9c	Enqueue -1:-1	High Priority	32767
252289027	Switch01	ethernet-type=-1, protocol=-1, ingress-port=-1, ip-dest=null, ip-src=null, tos-bits=-1, vlan-pcp=-1, vlan-id=100, ethsrc=null, ethdst=null, tcpdstport=-1, tcpsrcport=-1,	00:00:00:23:20:76:3c:b0	Enqueue -1:-1	Low Priority	32767

Figura 28. Módulo de QoS.

As Figuras 29 e 30 exibem a tabela de fluxos de cada um dos *switches* OpenFlow NetFPGA contendo a regra de fluxo relativa a política criada.

Flows (1)

Cookie	Priority	Match	Action	Packets	Bytes	Age	Timeout
-1999543724	32767	VLAN=100	prio=1, output -5	0	0	35 s	0 s

Figura 29. Tabela de fluxos do *switch* com IP 192.168.254.100.

Flows (1)

Cookie	Priority	Match	Action	Packets	Bytes	Age	Timeout
45035998381019760	32767	VLAN=100	prio=5, output -5	0	0	51 s	0 s

Figura 30. Tabela de fluxos do *switch* com IP 192.168.254.200.

Os testes de fluxo foram realizados com o auxílio da ferramenta *iperf*. Desta forma, ambos os clientes acessaram o servidor simultaneamente enviando pacotes com tráfego UDP.

Os fluxos de dados vindo dos clientes para o servidor foram observados com a ferramenta *Wireshark*. Como o servidor é uma máquina física, e sua implementação de VLAN foi feita no nível do sistema operacional, conseguimos filtrar os pacotes ainda com a marcação da VLAN (*tagged*). Assim, observamos os campos PCP dos fluxos recebidos pelo servidor.

As Figuras 31 e 32 exibem uma tela do programa *Wireshark* sendo executado pelo servidor. O fluxo destacado em cada figura é um pacote UDP originado do Cliente 1 e do Cliente 2, respectivamente.

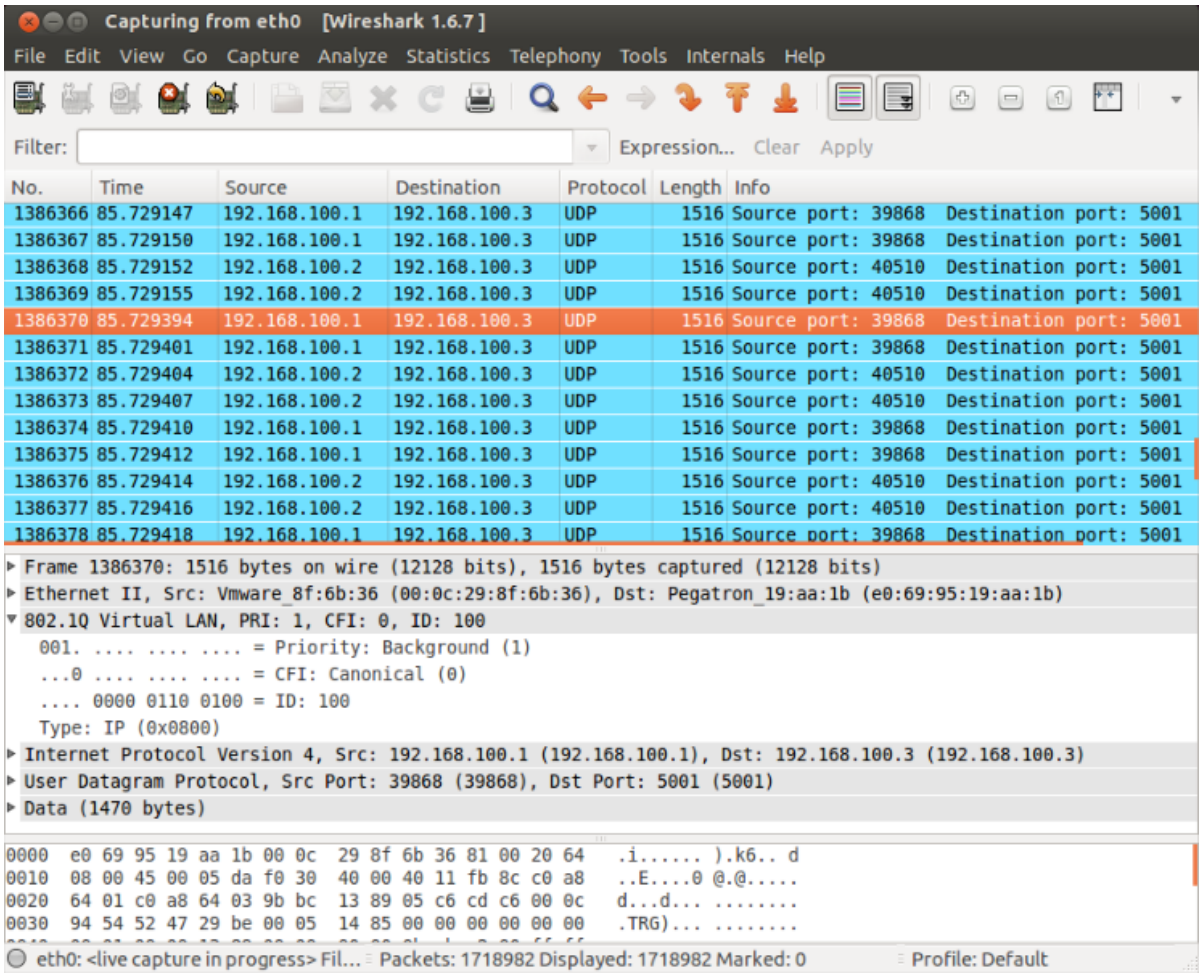

Figura 31. Pacote UDP originado do Cliente 1 (PCP = 1).

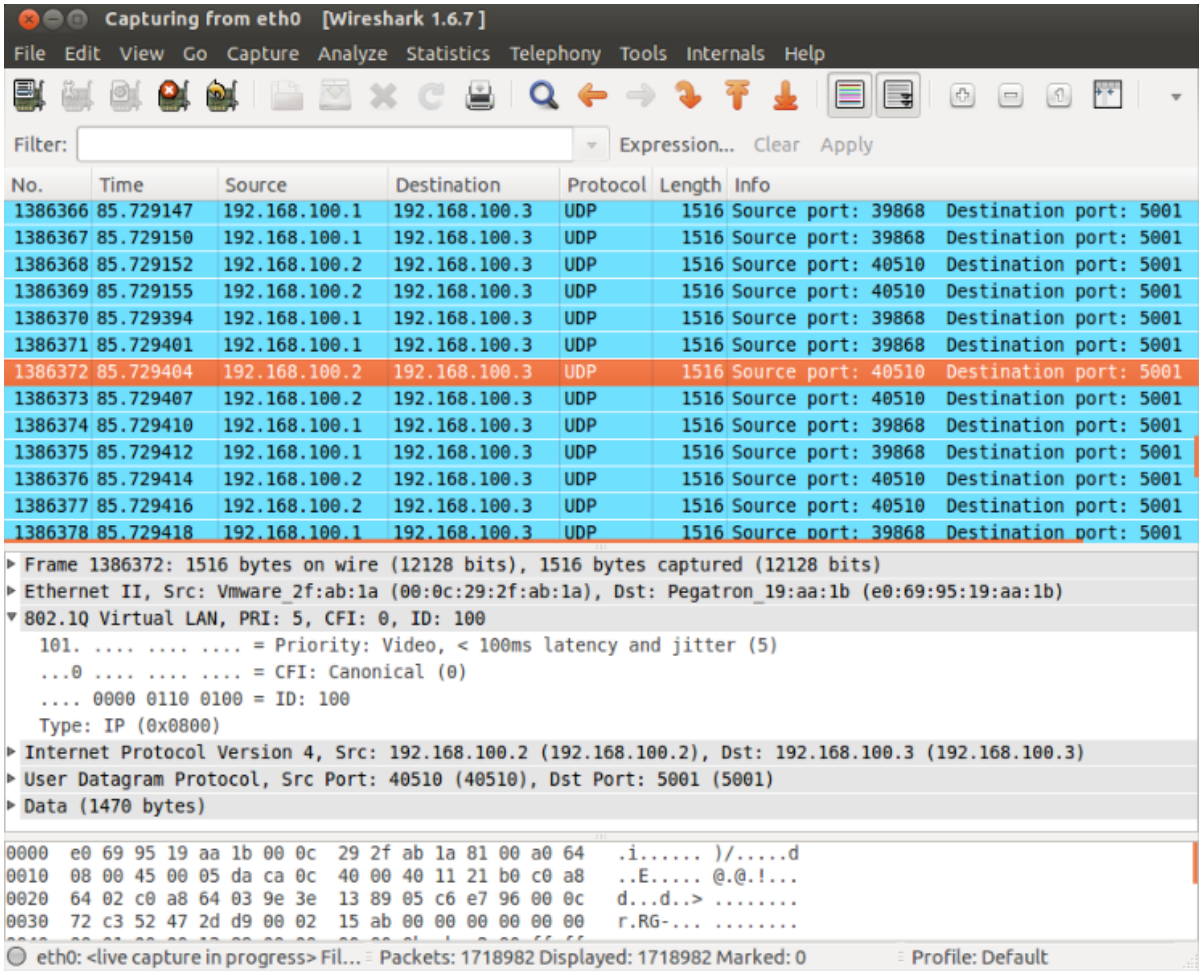

Figura 32. Pacote UDP originado do Cliente 2 (PCP = 5).

Como pode ser visto nas Figuras 31 e 32, o valor referente ao campo PCP no cabeçalho do protocolo 802.1Q foi devidamente determinado conforme a especificação das regras criadas pelo módulo de QoS.

Com relação à priorização do tráfego propriamente dita, verificamos certa instabilidade no *switch* OpenFlow Pronto. Em alguns momentos, o servidor recebia mais fluxos do Cliente 2 com sucesso, em outros, recebia mais fluxos com sucesso do Cliente 1, mesmo o Cliente 1 tendo menor prioridade, não caracterizando uma priorização efetiva. Contudo, a implementação da priorização é feita através do gerenciamento de filas de forma automática pelo *switch* Pronto o que não é o foco deste trabalho.

5. CONCLUSÃO

5.1 Considerações Finais

Com o avanço dos serviços e aplicações de rede, as demandas por uma maior largura de banda são cada vez mais comuns, principalmente em aplicações de tempo real como as de transmissão de voz e vídeo. Mesmo no contexto de redes locais é comum que se queira priorizar o tráfego de determinados clientes da rede, ou até de uma subrede. Unindo a tendência das redes definidas por software da Internet do Futuro à necessidade de QoS, esse trabalho propôs uma abordagem para aprovisionamento de QoS em redes OpenFlow.

O *testbed* referente a toda infraestrutura de rede foi devidamente montado e o módulo do controlador Floodlight para aprovisionamento de QoS através do campo PCP foi implementado. No entanto, o esforço demandado por este trabalho foi bastante extenso, primeiro devido ao tamanho da infraestrutura proposta, envolvendo vários servidores, máquinas físicas, máquinas virtuais, *switch* OpenFlow, placas NetFPGA e todo o cabeamento físico estruturado. Segundo, pela própria implementação do módulo de aprovisionamento de QoS, o que exigiu um certo domínio do controlador Floodlight e envolveu a codificação do sistema na linguagem Java. E por último, para realizar os testes e experimentos do módulo desenvolvido não mais em uma ferramenta de simulação (Mininet) e sim no ambiente construído (*testbed*), o que aumentou a complexidade da execução e análise dos experimentos.

Diversos problemas foram encontrados e enfrentados em várias etapas do desenvolvimento deste trabalho. Houve dúvidas quanto à arquitetura do *testbed*, que sofreu algumas alterações ao longo do projeto. Também houve dificuldades quanto às limitações e problemas de compatibilidade dos equipamentos, os quais foram adaptados de acordo com as necessidades. Foram encontrados *bugs* no controlador Floodlight, na implementação do OpenFlow 1.0 do NetFPGA e no *switch* Pronto. Contudo, o objetivo proposto para este trabalho foi alcançado, e uma das contribuições deste trabalho foi de corrigir a falha encontrada em um dos módulos do controlador Floodlight.

5.2 Trabalhos Futuros

Este trabalho pode servir como base para uma série de outros trabalhos que podem ser realizados a partir deste. Como foi citado no texto, uma aplicação com interface gráfica pode ser construída para utilizar o módulo de QoS desenvolvido, funcionando como um *front-end* para o gerente ou administrador da rede, e também para automatizar os testes e experimentos com o módulo de QoS.

Além de poder ser estendido para um trabalho maior, o desenvolvimento deste trabalho também lançou alguns desafios que precisam ser analisados. Um deles se refere à investigação quanto à priorização do tráfego não ter sido obtida no *switch* Pronto utilizando a implementação de filas baseadas no campo PCP.

Outro desafio envolve a pesquisa de algoritmos de priorização baseados em filas e a implantação desses algoritmos tanto no *switch* Pronto quanto no *switch* NetFPGA. Entretanto, talvez o desafio mais relevante seja a busca por alternativas de aprovisionamento de QoS fim a fim, saindo do escopo de redes locais.

REFERÊNCIAS

[1] *Nick McKeown; Tom Anderson; Hari Balakrishnan; Guru Parulkar; Larry Peterson; Jennifer Rexford; Scott Shenker; Jonathan Turner.* ***OpenFlow: Enabling Innovation in Campus Networks****, ACM SIGCOMM Computer Communication Review, Volume 38 Issue 2, p. 69-74, 2008*.

[2] Site oficial do projeto FIBRE, http://www.fibre-ict.eu/.

[3] Site oficial do projeto GENI, http://www.geni.net/.

[4] Site oficial do OpenFlow, http://www.openflow.org/.

[5] Site oficial da RNP, http://www.rnp.br/.

[6] Diego dos Passos Silva; Allan Borges Pontes; Edson Adriano Maravalho Avelar; Kelvin Lopes Dias. **Uma Arquitetura para o Aprovisionamento de QoS Interdomínios em Redes Virtuais baseadas no OpenFlow**, 31º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, p. 893-906, 2013.

[7] Site oficial do NetFPGA, http://netfpga.org/.

[8] Site oficial da *Pronto Systems*, http://prontosystems.wordpress.com/.

[9] Site oficial da *Open Network Foundation,* https://www.opennetworking.org/

[10] *ONF White Paper.* ***Software-Defined Networking: The New Norm for Networks****. April 13, 2012*.

[11] *CohesiveFT.* ***White Paper: OpenFlow is Software Defined Networking; Software Defined Networking is Not Only OpenFlow****, September 24th, 2012.*

[12] *Guillermo Romero de Tejada Muntaner.* ***Evaluation of OpenFlow Controllers****, October 15, 2012.*

[13] ***OpenFlow Presentation****, 2011.* Disponível em http://archive.openflow.org/documents/OpenFlow_2011.pps (acessado em 25 de setembro de 2013).

[14] ***OpenFlow Switch Specification Version 1.0.0****, December 31, 2009.*

[15] Site oficial do grupo de pesquisa *McKeown*, http://yuba.stanford.edu/

[16] Site oficial da Wiki NetFPGA. https://github.com/NetFPGA/netfpga/wiki

[17] Site oficial da Pica8 Inc, http://www.pica8.com/

[18] Fotografia do *Switch* Pronto 3290, https://macrotronsystems.com/images/p3290-3295_b1.jpg (acessado em 25 de setembro de 2013).

[19] Lucas Rodrigues Costa. **OpenFlow e o Paradigma de Redes Definidas por Software**, Monografia apresentada no Curso de Licenciatura em Computação, UnB, 2013.

[20] Site oficial do Projeto Floodlight, http://www.projectfloodlight.org

[21] Site oficial do Controlador Floodlight, http://floodlight.openflowhub.org/

[22] Site oficial da documentação do Controlador Floodlight, http://docs.projectfloodlight.org/display/floodlightcontroller/Floodlight+Documentation

[23] Kurose, James F.. **Redes de computadores e a Internet: uma abordagem top-down** – 3. Ed. – São Paulo: Pearson Addison Wesley, 2006.

[24] Site oficial do Projeto Indigo. http://www.projectfloodlight.org/indigo/

[25] Site oficial da Biblioteca PQLib, *Priority Queue Library*. http://www.ohloh.net/p/pqlib

[26] Fotografia de um servidor IBM System x3650 M3, http://photos.pcpro.co.uk/images/front_picture_library_PC_Pro/dir_304/it_photo_152440_52.jpg (acessado em 25 de setembro de 2013)

[27] Fotografia de um servidor IBM System x3400. http://www.lojati.com.br/Customer/mercadorias/mer000962001.jpg (acessado em 25 de setembro de 2013)