UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ANA CAROLINA EWALD ELLER

ENCAMINHAMENTO POR HARDWARE EM REDES DEFINIDAS POR SOFTWARE: AVALIAÇÃO EXPERIMENTAL UTILIZANDO NETFPGA

VITÓRIA
2016

ANA CAROLINA EWALD ELLER

ENCAMINHAMENTO POR HARDWARE EM REDES DEFINIDAS POR SOFTWARE: AVALIAÇÃO EXPERIMENTAL UTILIZANDO NETFPGA

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Engenharia Elétrica.
Orientador: Prof. Dr. Moisés Renato Nunes Ribeiro

VITÓRIA
2016

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Setorial Tecnológica,
Universidade Federal do Espírito Santo, ES, Brasil)

Eller, Ana Carolina Ewald, 1989-

E45e Encaminhamento por hardware em redes definidas por software: avaliação de desempenho / Ana Carolina Ewald Eller. – 2016.

79 f. : il.

Orientador: Moisés Renato Nunes Ribeiro.

Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Redes de computadores. 2. Redes definidas por software (SDN). 3. OpenFlow (Protocolo de rede de computador). I. Ribeiro, Moisés Renato Nunes. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 621.3

ANA CAROLINA EWALD ELLER

ENCAMINHAMENTO POR HARDWARE EM REDES DEFINIDAS POR SOFTWARE: AVALIAÇÃO EXPERIMENTAL UTILIZANDO NETFPGA

Dissertação submetida ao programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para a obtenção do Grau de Mestre em Engenharia Elétrica.

Aprovada em 15 de abril de 2016

COMISSÃO EXAMINADORA

Prof. Dr. Moisés Renato Nunes Ribeiro – UFES
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Gilmar Luiz Vassoler
Instituto Federal do Espírito Santo - IFES

Prof. Dr. Antônio Marcos Alberti
Instituto Nacional de Telecomunicações - INATEL

Dedico este trabalho a minha família e amigos

Agradecimentos

Primeiramente a Deus pela minha vida, por confortar meu coração e por proporcionar a minha chegada até aqui.

Ao meu orientador Moisés por ter me aceitado me orientar, por ter acompanhado meu trabalho e por todas as direções, revisões, ideias e ensinamentos passados ao longo do mestrado.

Aos meus companheiros do NERDS e do LabTel que tornaram os dias no laboratório mais alegres e sempre foram muito colaborativos. Em especial ao Gilmar, pelas ideias, revisões e por toda paciência e disponibilidade em me explicar com detalhes a implementação do seu trabalho de doutorado e sanar todas as minhas dúvidas para que, assim, eu pudesse utilizá-lo como base para desenvolver grande parte deste trabalho.

Por fim, agradeço aos meus pais Wilmar e Alini, aos meus irmãos, Camila e Bernardo pela paciência, apoio, incentivo, carinho e compreensão nos momentos difíceis. Vocês foram fundamentais para que me manter firme, apesar dos dias ruins, me dar forças para não desistir e chegar até aqui!

A todos vocês, muito obrigada!

Resumo

As redes de computadores se tornaram parte essencial da infraestrutura de nossa sociedade, no entanto, o projeto inicial das redes tradicionais não poderia atender os requisitos de todas essas aplicações, pois muitas não foram sequer imaginadas. Como resultado, tem-se um ambiente de redes com pouco suporte e flexibilidade para anteder às novas exigências. Nesse contexto, surgiu, então, um conceito promissor de redes programáveis denominado Redes Definidas por Software (SDN – *Software Defined Network*). SDN separa o plano de controle do plano de dados permitindo aos desenvolvedores programar a rede de acordo com as necessidades de suas aplicações. Além disso, tornou possível a implementação e testes em ambientes reais, inovações tecnológicas para o ambiente de redes sem, para isso, depender dos fabricantes de equipamentos. Este trabalho propõe a utilização de NetFPGAs como ferramenta de encaminhamento em SDN, controlada por um controlador OpenFlow a fim de diminuir o caminho percorrido por um pacote de sua origem ao seu destino. Além disso, propõe-se utilizar a NetFPGA com OpenFlow para uma implementação de múltiplos caminhos com balanceamento de cargas. Será avaliada, ainda, a influência de tabelas de fluxos na comutação de pacotes comparando um protocolo que utiliza tabelas para encaminhamento (OpenFlow) com um que não utiliza tabelas para este fim (KeyFlow).

Palavras-chave: Redes, Redes Definidas por Software, OpenFlow, KeyFlow, NetFPGA

Abstract

Computer networks have become an essential part of the infrastructure of our society, however, the initial design of traditional networks cannot meet the requirements of all these applications because many were not even imagined. As a result, there is a network environment with little support and flexibility to the new requirements. In this context, there was then a promising concept of programmable networks called Defined Networks Software (SDN - Software Defined Network). SDN separates the data plane from the control plane enabling developers to program the network according to their application needs. In addition, it made possible the implementation and testing in real environments, technological innovations for the environment networks without, for that, rely on the equipment manufacturers. This paper proposes the use of NetFPGAs as routing tool in an SDN controlled by an OpenFlow controller to decrease the path a packet from its source to its destination. Furthermore, it is proposed to use NetFPGA OpenFlow for implementing multiple paths with load balancing. It will be evaluated yet, the influence of packet switching in the flow charts comparing a protocol which uses routing tables for (OpenFlow) with one that does not use tables for this purpose (KeyFlow).

Keywords: Networks, Software Defined Networks, OpenFlow, KeyFlow, NetFPGA

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API	Application Programming Interface
CPqD	Centro de Pesquisa e Desenvolvimento em Telecomunicações
CPU	Computer Processor Unit
FPGA	Field-Programmable Gate Array
IP	Internet Protocol
MAC	Media Access Control
MBPS	Mega Bits por Segundo
NAT	Network Address Translation
NERDS	Núcleo de Estudos de Redes Definidas por Software
OF	OpenFlow
PC	Personal Computer
QoS	Quality of Service
RTT	Round Trip Time
SDN	Software Defined Networking
TCAM	Ternary Content Access Memory
TCP	Transport Control Protocol
TI	Tecnologia da Informação
TTL	Time To Live
UDP	User Datagram Protocol
VM	Virtual Machine

Capítulo 1 - Introdução

1.1 Contexto

Desde sua origem, a Internet tem crescido bastante. Estima-se que o número de usuários conectados no ano de 2015 tenha atingido a marca de 3,2 bilhões de pessoas (vide Figura 1.1). Além disso, o uso da Internet, criada inicialmente com objetivos militares, tem sido cada vez mais diversificado. Atualmente, qualquer usuário com acesso à Internet pode utilizar serviços como e-mail, serviços de busca e redes sociais, entre outros [1] [2].

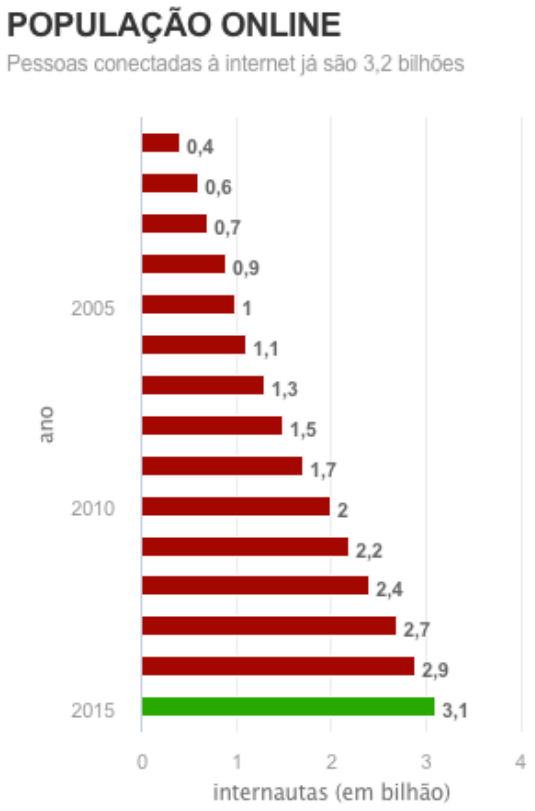

Figura 1.1: Usuários conectados à Internet (fonte: União Internacional das Telecomunicações) [3]

Essa popularização da Internet associada ao aumento da conectividade através de banda larga faz com que, cada vez mais, o processamento e o armazenamento de fotos, vídeos, aplicações entre outros, que antes eram armazenadas e processadas em PCs

(*Personal Computers*), migrem dos PCs para grandes provedores de serviços para serem hospedadas e processadas sob o formato de serviços Web [1]. Hoje, é possível a usuários comuns, por exemplo, acessar, editar, salvar e compartilhar documentos armazenados em aplicações hospedadas e executadas remotamente em *data centers*, tais como Google Drive, Dropbox, Amazon Cloud Drive, entre outros [4] [5].

Os *data centers* são estruturas físicas, que podem ser desde salas a edifícios, projetadas para serem capazes de armazenar e gerenciar equipamentos de redes, servidores e telecomunicação. Eles estão crescendo em tamanho e importância, mostrando-se um componente essencial na Internet atual [6]. Isso faz com que o planejamento adequado do projeto de infraestrutura de um *data center* seja algo crítico, pois este tem que atender requisitos de desempenho, resiliência, escalabilidade entre outros. No entanto, segundo Kim [7], apesar de sua importância, as arquiteturas de *data center* atuais mostram-se insuficientes para atender demandas de computação em nuvem, uma vez que os protocolos de roteamento, encaminhamento e gerenciamento atualmente executados foram projetados, em geral, para redes locais e não são flexíveis o suficiente para suportar novas exigências. Além disso, o processo de pesquisa, implantação de novos protocolos e tecnologias para redes tradicionais torna-se caro, visto que novas propostas, em geral, requerem mudanças em todos os equipamentos de redes instalados.

Nesse contexto, surgiu um conceito promissor de redes que tende a revolucionar o mercado e ser o futuro das aplicações de rede atuais, as Redes Definidas por *Software* (SDN – *Software Defined Networks*) [8]. Uma proposta que separa os planos de dados e de controle, em que a infraestrutura de rede não depende mais de tecnologias dos fornecedores, pois tem como única função realizar encaminhamento dos pacotes. Os equipamentos são controlados por um plano de controle logicamente centralizado, que detêm uma visão geral dos recursos da rede e contém toda a sua inteligência. Assim, é possível aos desenvolvedores definir o comportamento da rede e adicionar funcionalidades de acordo com seus objetivos específicos, sem depender dos demorados ciclos de atualizações dos equipamentos de rede [9].

Atualmente, o protocolo mais utilizado em arquiteturas SDN é o OpenFlow [10]. Este, propicia a um controlador externo uma visão geral da rede e encaminha os pacotes de acordo com regras instaladas em tabelas de fluxos, que são atualizadas com base nos fluxos ativos da rede. Essa estratégia, no entanto, pode comprometer o tempo necessário para encaminhar os pacotes. Visto que os comutadores OpenFlow devem consultar tabelas para realizar o encaminhamento de cada fluxo e conforme o número de fluxos

ativos aumenta, a tabela também irá crescer. Isso pode elevar o tempo necessário para realizar consultas à tabela, comprometendo, então, a latência da rede, que é uma medida fundamental para avaliar o desempenho de uma rede, pois ela mede o tempo necessário para realizar uma determinada ação, por exemplo: o tempo gasto por um pacote de uma origem a um destino qualquer na rede. Conhecer essa métrica é algo essencial para que seja garantida a confiabilidade dos serviços da rede, uma vez que será possível prever o tempo de resposta das aplicações por ela suportadas. Além disso, tem-se a possibilidade de identificar os serviços com altas taxas de latência e estudar soluções a fim de reduzir os valores, pois quanto maior a latência, maior será o tempo de resposta comprometendo a eficiência das aplicações.

Uma possível solução para reduzir a latência é a utilização de um métodos mais simples para encaminhar pacotes, que não dependam de tabelas de fluxos para este fim. Nesse contexto tem-se o KeyFlow [11], método no qual o caminho a ser percorrido pelos pacotes é determinado pelo resto da divisão entre o valor do rótulo do pacote, dado pelo controlador da rede e o valor associado aos comutadores por onde os pacotes passam.

Outra alternativa de encaminhamento em redes sem a necessidade de tabelas de fluxos é o encaminhamento óptico, utilizando chaves ópticas que oferecem a possibilidade de alternar os vizinhos de acordo com a necessidade da rede. Esta alternativa, no entanto, restringe a liberdade de reconfiguração da rede a apenas dois estados, uma vez que é feito apenas chaveamento de caminhos e não um tratamento de fluxos. Por isso, para redes com um número grande de fluxos diferentes é interessante ter uma maior flexibilidade de escolha dos caminhos a ser percorridos pelos pacotes. Esse é o caso da TRIIIAD (**TRI**ple-Layered **I**ntelligent and **I**ntegrated **A**rchitecture for **D**atacenters) [12], uma Rede Definida por Software autônoma que utiliza dispositivos ópticos no encaminhamento de pacotes. Por meio de chaves ópticas o controlador da rede reconfigura a camada de encaminhamento, no entanto, a liberdade de reconfiguração é restrita, impossibilitando ao controlador tratar os fluxos de maneira individual. Neste caso, a utilização de um método com tabelas de fluxos mostra-se interessante pois possibilita a manipulação fina dos fluxos, a fim de proporcionar uma melhor engenharia de tráfego da rede.

1.2 Objetivos deste Trabalho

Esse trabalho tem como primeiro objetivo avaliar experimentalmente os possíveis ganhos obtidos ao substituir o encaminhamento óptico original da TRIIIAD, por um encaminhamento utilizando uma NetFPGA programada com OpenFlow. Espera-se, com isso, aumentar o número de vizinhos dos *hosts*. E consequentemente, minimizar o tráfego de trânsito dos nós intermediários entre a origem e destino dos pacotes. Ainda, deseja-se implementar nesta mesma rede uma melhor distribuição de tráfego, visando o balanceamento das interfaces de rede dos *hosts* quando houver sobrecarga em alguma interface de rede de um determinado nó.

Uma vez que os dispositivos ópticos não possuem tabelas de fluxos para realizar encaminhamento de pacotes, o segundo objetivo é será estudar a influência das tabelas na latência de comutação. Isso será feito comparando um protocolo de encaminhamento que utiliza tabelas para este fim (OpenFlow) e um que não necessita de tabelas de fluxos instaladas em seus comutadores para encaminhar pacotes (KeyFlow). Serão realizados experimentos utilizando NetFPGAs para encaminhamento e geração de pacotes, bem como nos processos de medição.

1.3 Publicações Relacionadas

No ambiente desta dissertação, destaca-se a publicação relacionada:

SALDAÑA CERCÓS, SILVIA ; RAMOS, RAMON M. ; EWALD ELLER, ANA C. ; MARTINELLO, MAGNOS ; RIBEIRO, MOISÉS R. N. ; MANOLOVA FAGERTUN, ANNA ; TAFUR MONROY, IDELFONSO . Design of a stateless low-latency router architecture for green software-defined networking. In: SPIE OPTO, 2015, San Francisco. v. 9388.

1.4 Estrutura

O restante deste trabalho está organizado da seguinte forma:

- O Capítulo 2 apresenta os conceitos de Redes Definidas por *Software*, bem como os conceitos básicos e características das redes de *data centers* atuais. São expostos, também, os conceitos referentes à NetFPGA e possíveis aplicações. Por fim, são mostrados os conceitos referentes à latência de rede e ao OFLOPS, ferramenta *open source* utilizada para avaliação de latência.
- O Capítulo 3 descreve a metodologia utilizada para a avaliação de latência de rede em dispositivos NetFPGA, bem como, para a inserção da NetFPGA no ambiente da TRIIIAD.
- O Capítulo 4 apresenta o ambiente de teste utilizado para a avaliação de latência de comutação dos *switches* KeyFlow e OpenFlow. Além disso expõe e discute os resultados obtidos.
- O Capítulo 5 apresenta os resultados obtidos da implementação da NetFPGA como ferramenta de encaminhamento de auxílio à TRIIIAD, na redução do tráfego de trânsito da rede, bem como no gerenciamento individual de fluxos.
- Finalmente, o Capítulo 6 traz as conclusões e as perspectivas de trabalhos futuros.

Capítulo 2 - Fundamentação Teórica

O objetivo deste Capítulo é apresentar os aspectos teóricos importantes para a compreensão deste trabalho. Serão abordados os principais conceitos de Redes Definidas por Software, Redes de Data Centers, NetFPGA e Latência de Rede.

2.1 Rede Definidas por Software

Mesmo com a evolução das redes de computadores, as modificações já realizadas não estão mais atendendo a demanda de novas aplicações. Além disso, os equipamentos de rede são "caixas pretas", ou seja, os fabricantes os comercializam com implementações integradas baseadas em *software* e hardware proprietário sem que o cliente possa modificá-los [13].

Observando as limitações das redes existentes, a comunidade científica começou a desenvolver novas arquiteturas de modificação do núcleo de rede, fazendo surgir propostas de redes que possam ser programadas sob demanda, ou seja, redes que possam ser flexíveis para atender os requisitos de aplicações atuais e futuras. A esse novo modelo de redes dá-se o nome de Redes Definidas por *Software* (SDN).

A principal característica das Redes Definidas por *Software* (SDN) é que a sua arquitetura tem como principio básico a separação física do plano de controle e do plano de encaminhamento de dados. Nas redes tradicionais, o plano de controle e de dados são implementados e executados no próprio equipamento, impedindo qualquer tomada de decisão que não tenha sido prevista nos protocolos implementados pelo proprietário ou fabricante.

Ao separar os planos de dados e de controle, elimina-se essa restrição, pois os equipamentos de rede terão como única função o encaminhamento de pacotes. A inteligência lógica da rede fica centralizada em controladores, implementações externas contidas em máquinas físicas ou virtuais, que possuem uma visão geral da rede e têm o poder de modificar o comportamento dos equipamentos de rede quando necessário.

A Figura 2.1 apresenta uma visão lógica da arquitetura de Rede Definida por Software. Os equipamentos físicos da rede são visto como uma camada única pelo controlador que centraliza todas as informações e mantem uma visão global da rede [8]. Isso faz com que as aplicações vejam a rede como um único *switch* lógico. Para que

haja comunicação entre a camada de controle (plano de controle) e a infraestrutura utilizada (plano de dados) é preciso que haja um protocolo padronizado que se comunique com ambos os planos. O protocolo mais utilizado atualmente é o OpenFlow, que também será utilizado neste trabalho.

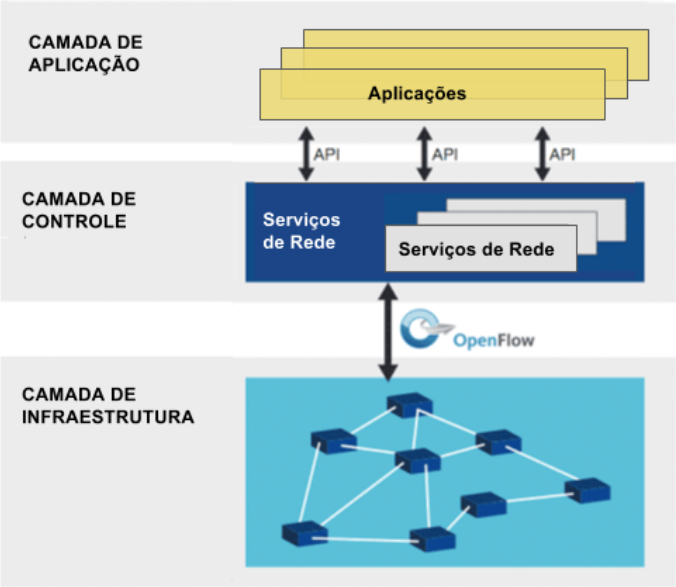

Figura 2.1: Arquitetura de Rede Definida por Software **[14]**

Uma vez que a infraestrutura da rede irá somente encaminhar os pacotes que receber de acordo com as instruções recebidas do plano de controle, pode haver uma redução no custo dos equipamentos utilizados, uma vez que o *hardware* empregado poderá ser mais simples e consequentemente com um custo menor.

2.1.1 Openflow

O OpenFlow é um protocolo de rede de código aberto proposto pela Universidade de Stanford em 2008, seu objetivo inicial é atender à demanda de validação de novas propostas para redes. Pois os pesquisadores, em geral, não possuem uma rede de testes com desempenho próximo ao real e o OpenFlow permite que eles possam testar e validar suas propostas em ambientes de redes reais, sem interferir na rede de produção, uma vez que o tráfego pode ser dividido em fluxos de produção e de pesquisa [10].

Em uma rede OpenFlow, o controlador, por meio de instruções enviadas remotamente aos comutadores, pode modificar completamente o comportamento dos dispositivos de acordo com a necessidade da rede.

O OpenFlow utiliza o conceito de fluxos para identificar o tráfego da rede, sendo um fluxo constituído pela definição dos valores de um ou mais campos dos cabeçalhos dos pacotes processados pelo dispositivo OpenFlow.

Ao receber um fluxo, o equipamento de rede consulta a tabela de fluxos, esta contém as regras definidas pelo controlador da rede que podem ser instaladas de maneira dinâmica ou estática. Após a consulta, é então aplicada aos pacotes recebidos a ação correspondente. Caso não haja uma regra instalada pra determinado fluxo, é enviada uma mensagem ao controlador, este realiza o processamento do pacote e define uma nova regra com a ação necessária para o fluxo em questão.

2.1.1.1 Controlador Openflow

Uma vez que os equipamentos de infraestrutura em SDN tem como única função encaminhar pacotes, o plano de controle é a parte que assume toda a inteligência da rede e condensa todos os conceitos relacionados ao controle de uma rede de computadores convencional.

Assim como em outras arquiteturas, em que existem diversos sistemas operacionais, em SDN também foram desenvolvidos diversos controladores. Cada um com características e objetivos muito variados, que cobrem desde o controle de pequenos protótipos para experimentações acadêmicas, até redes de grande porte. A seguir, serão listados alguns controladores, bem como suas características principais.

Nox [15] é o controlador original do OpenFlow desenvolvido inicialmente pela Nicira e disponibilizado para a comunidade em 2008. A linguagem de programação C++ foi utilizada no desenvolvimento deste controlador, bem como a interface disponível para criar aplicações. O alto desempenho é uma de suas principais características e atualmente, possui suporte à versão 1.0 do OpenFlow. No entanto, existe uma versão modificada pelo CPqD (Centro de Pesquisa e Desenvolvimento em Telecomunicações) que suporta parcialmente a versão 1.3 [16].

A partir do controlador Nox tradicional foi desenvolvido o Pox [17]. Um projeto em Python com uma interface mais elegante, resultando em um controlador mais moderno e simples para controladores SDN. Este controlador normalmente é utilizado como alternativa ao Nox em experimentos de prototipação, pois possui uma interface mais amigável.

OpenDayLight [18] é um projeto *open source* que foi desenvolvido com participação de grandes empresas como Cisco, Citrix, Microsoft, IBM, etc. O objetivo principal da comunidade é acelerar o processo de popularização do uso de SDN's, e como parte do projeto, disponibilizam a plataforma para o desenvolvimento de aplicações. Um dos padrões suportados por este controlador é o OpenFlow, atualmente nas versões 1.0 e 1.3.

Ryu [19] é um controlador *open source* implementado em Python, desenvolvido por um grupo japonês da NTT Lab's. Sua arquitetura é ilustrada na Figura 2.2.

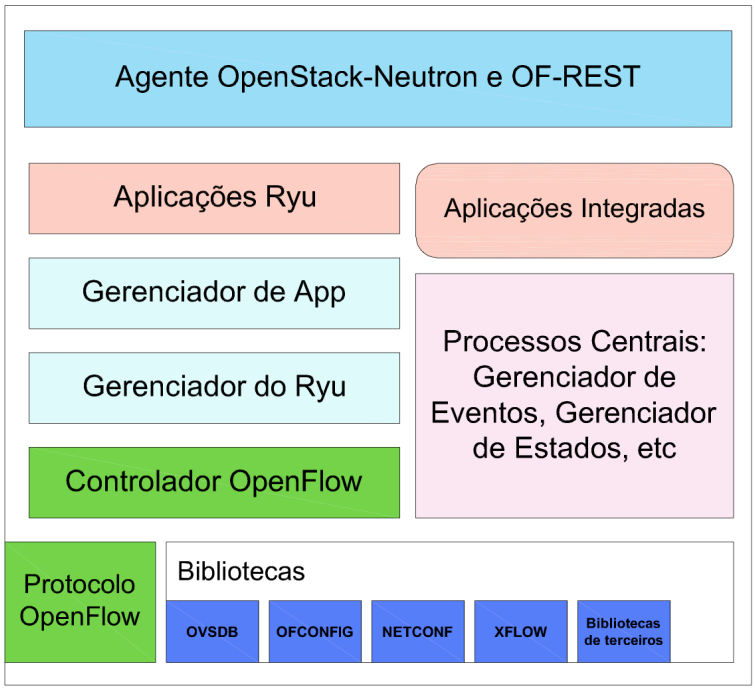

Figura 2.2: Arquitetura do controlador Ryu **[20]**

O Ryu possui muitas bibliotecas capazes de dar suporte a múltiplos protocolos de comunicação entre elementos de controle e encaminhamento de dados, por exemplo DE-Config, Open vSwitch Database Management (OVSDB), NETCONF, XFlow (Netflow e sFlow), utilizados para diferentes operações de processamento de pacotes de rede. Atualmente, possui suporte total até a versão 1.4 do OpenFlow e algumas implementações da versão 1.5. Ele inclui uma biblioteca de codificação e decodificação do protocolo OpenFlow. Além disso, o controlador OpenFlow, que é responsável pela gestão dos comutadores OpenFlow utilizados para configurar fluxos, gerenciar eventos, etc. é um dos principais componentes da arquitetura do Ryu.

O gerenciador Ryu é o executável principal e uma vez em execução, qualquer comutador OpenFlow pode se conectar a ele. Já o gerenciador de App é o componente fundamental para todas as aplicações Ryu, as quais herdam a classe RyuApp do gerenciador de App. O gerenciamento de eventos, por sua vez, bem como de mensagens, de estado em memória, etc. é feito pelo componente de processo central da arquitetura.

Na camada de Interface de Programação de Aplicativos (API), está incluso um *plug-in* de suporte a plataforma OpenStack [21] e à interface REST para operações OpenFlow. A distribuição do Ryu possui várias aplicações, por exemplo *simple_switch*, roteador, *firewall*, etc. As aplicações são entidades de tarefas individuais que implementam várias funcionalidades.

Na Tabela 2.1 está um resumo dos controladores citados, bem como informações adicionais, como a versão do OpenFlow suportado e a linguagem da API.

Tabela 2.1- Resumo das características dos principais Controladores

Controlador	Suporte OpenFlow	Licença	Linguagem API
Nox	1.0	GPLv3	C++
Pox	1.0	GPLv3	Python
OpenDayLigth	1.0 e 1.3	EPL v1.0	Java
Ryu	1.0, 1.1, 1.2, 1.3 e 1.4	Apache 2.0	Python

2.1.1.2 *Switch* Openflow

A maioria dos comutadores e roteadores tradicionais possuem tabelas de fluxos onde são implementados *firewalls*, NAT (tradução de endereços de rede), QoS (qualidade de serviço), entre outros. Enquanto esses equipamentos possuem implementações fechadas com tabelas diferenciadas para cada tipo fornecedor, as instruções do OpenFlow, por outro lado, podem ser implementadas em qualquer comutador, independente do fabricante, pois a tabela de fluxos é definida unicamente pelo protocolo.

O *switch* OpenFlow, também chamado *datapath* representa o seu plano de dados, ele é composto por pelo menos três partes (vide Figura 2.3):

1. Tabela de Fluxos: nesta tabela são instaladas regras que associam ações a cada fluxo, indicando, assim, como o equipamento deve processar os fluxos que chegam na rede;

2. Canal Seguro: este canal conecta o equipamento a um controlador remoto, utilizando uma conexão criptografada, e permite a troca de mensagens e pacotes entre o controlador e o comutador;
3. O Protocolo OpenFlow: o protocolo define um padrão aberto para que haja comunicação entre o controlador e o comutador. É por meio dessa interface de comunicação que é permitido a um controlador externo programar as entradas na tabela de fluxo.

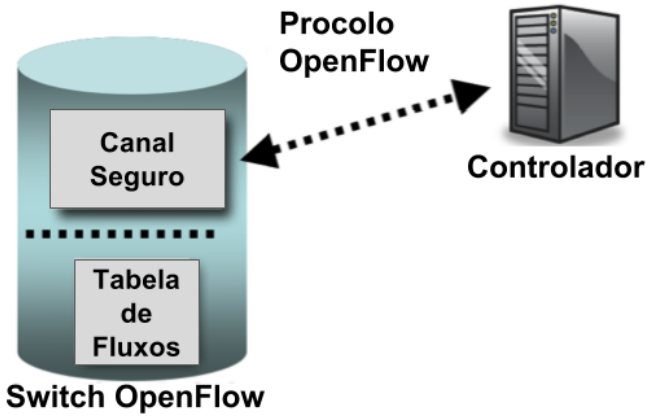

Figura 2.3: Switch OpenFlow **[22]**

Os *switches* OpenFlow podem ser tanto físicos quanto virtuais. Os comutadores físicos são equipamentos embarcados capazes de encaminhar pacotes de rede interligando *hosts* diretamente por meio de suas portas físicas. Já os comutadores virtuais, por sua vez, são *softwares* capazes de encaminhar pacotes utilizando a pilha do sistema operacional do *host* onde são executados. Como exemplo de *switches* físicos comerciais que suportam o OpenFlow, temos o Pica8 [23], Mikrotik RouterBoard, Datacom, e Brocade. Dentre os *switches* virtuais, os mais utilizados são o CPqD Software Switch 13 [24], Click Modular Router [25], Open vSwitch (OvS) [26] e eXtensible OpenFlow DataPath Daemon (xDPd) [27].

Além dos comutadores já citados existem uma plataforma de código aberto que pode ser programada para funcionar com as características de um *switch* OpenFlow, a NetFPGA. Mais detalhes sobre ela serão dados na Subseção 2.1.1.3 a seguir.

2.1.1.3 NetFPGA

A NetFPGA é uma plataforma de código aberto de *hardware* e *software* projetada para pesquisa e ensino. O projeto da plataforma foi desenvolvido por um grupo de

pesquisa de redes de alto desempenho da Universidade de Stanford [28]. Atualmente, as plataformas existentes são: NetFPGA-SUME, NetFPGA-1G-CML, NetFPGA-10G e a NetFPGA-1G.

Utilizando placas NetFPGA é possível a pesquisadores e estudantes desenvolverem protótipos de sistemas de rede de alta velocidade com novos mecanismos, protocolos e arquiteturas. Isso porque, essa plataforma combina processadores, memórias (SRAM e DRAM), todos os recursos lógicos e portas Ethernet necessários para que a placa possa assumir as características de diversos dispositivos de rede como comutadores (OpenFlow ou não), roteadores, placa de rede comum, gerador de tráfego, entre outros.

Mais de 226 trabalhos acadêmicos [29] já utilizaram a NetFPGA como a alternativa para avaliação de protótipos de pesquisa, devido a sua flexibilidade e baixo custo. Dentre esses, estão os trabalhos que utilizam o OpenFlow em NetFPGA. A grande vantagem dela em relação a comutadores baseados em *software* que emulam o funcionamento do OpenFlow é que na NetFPGA o encaminhamento de pacotes é feito em *hardware* e os fluxos são processados imediatamente ao serem recebidos pela placa, permitindo que sua taxa máxima de transmissão seja atingida sem que haja perda de pacotes. Essas características da NetFPGA resultam em um aumento significativo de desempenho em relação a comutadores com OpenFlow emulado e em uma aproximação do ambiente de experimentação da realidade.

Neste trabalho, foi utilizada a NetFPGA 1G, ilustrada na Figura 2.4, que possui quatro portas Ethernet de 1Gbps. O processamento de pacotes é feito pelo FPGA Xilinx Virtex-II Pro 50 presente na placa, ele é capaz de processar os pacotes recebidos pelas portas Ethernet em modo de operação *full duplex*, ou seja, a vazão máxima pode chegar até 8Gbps (1Gbps * 4 portas * 2 (bidirecional)).

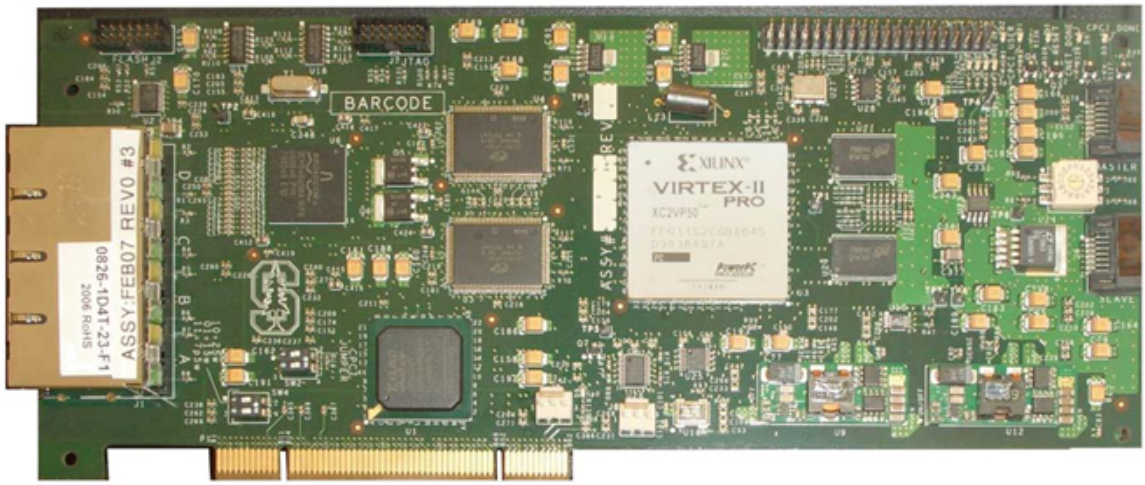

Figura 2.4: Placa NetFPGA 1G [29]

A NetFPGA é instalada em um PC (*host*) através de uma interface PCI e a programação da placa é feita pelo *host*. Quanto às memórias, ela possui 2 bancos de memória SRAM, que podem realizar uma operação de leitura ou escrita por ciclo de clock e retorna os dados lidos em três ciclos, possui também um banco DRAM que funciona de forma assíncrona e necessita de atualização contínua de dados. A Figura 2.5 mostra uma visão geral dos principais componentes da NetFPGA.

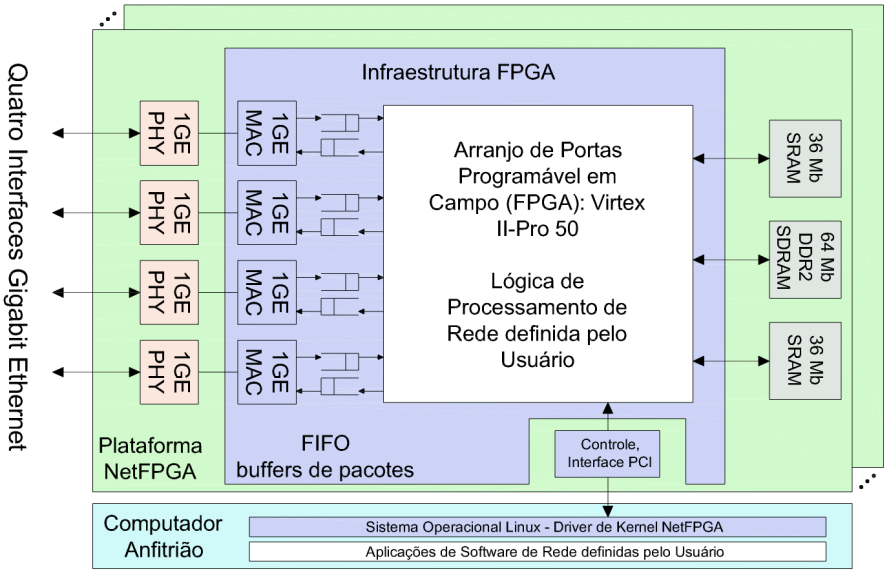

Figura 2.5: Diagrama de Blocos NetFPGA **[30]**

2.2 Redes de *Data Center*

As redes de computadores tiveram seu início a partir das pesquisas militares feitas no período da Guerra Fria, na década de 1960. Essa iniciativa partiu do departamento de defesa americano com o objetivo de descentralizar informações sigilosas armazenadas em pontos específicos. Pois, caso houvesse um ataque a essas bases, todas as informações sigilosas dos EUA poderiam ser reveladas ou perdidas. Além disso, caso os ataques inimigos destruíssem os meios de comunicação convencionais, haveria uma forma de manter a comunicação entre as bases.

Desde então, muito se evoluiu. Nas décadas de 1970 e 1980, as redes passaram a ser utilizadas também para fins acadêmicos e na década de 1990 houve a popularização da Internet. Surgiram os microcomputadores, os meios de comunicação passaram por grandes transformações e houve a diminuição de custos para transmissão de dados.

A fim de suprir as demandas das aplicações e calcular as rotas a serem percorridas pelos pacotes, foram desenvolvidos modelos para os protocolos de comunicação. Esse desenvolvimento, tornou possível uma expansão do número de usuários, bem como a diversificação de métodos de acesso às aplicações de rede. Permitindo o acesso de usuários domésticos por meio de computadores pessoais, celulares e *tablets*, fazendo com que houvesse um crescimento explosivo da Internet e da Web. Atualmente, é possível anexar numerosos arquivos com grande quantidade de dados e enviá-los por e-mail, se comunicar com pessoas do mundo inteiro em tempo real, inclusive por meio de ligações de voz através da Internet.

Entretanto, apesar de todo avanço ocorrido desde a década de 1960, não houveram tantas mudança na infraestrutura de rede, se comparada ao aumento das capacidades dos canais de comunicação e ao maior poder de processamento dos equipamentos [31], [32], [33], [34]. Desde que houve a introdução de *switches* e roteadores como equipamentos encaminhadores de pacotes, muito pouco foi alterado na infraestrutura de rede. Além disso, os fabricantes desses equipamentos os fazem como “caixas pretas”, pois o *firmware* vem encapsulado e fechado no *hardware* do equipamento. O resultado disso é a existência de redes em que o processo de implementação de inovações e modificações para melhoria se torna algo de grande complexidade. E com isso, características necessárias em uma rede como confiabilidade, segurança, baixa latência, flexibilidade e escalabilidade ficam comprometidas.

Nota-se, então que a atual arquitetura de comunicação de dados necessita de melhorias, pois as tecnologias de hoje dificultam bastante o desenvolvimento de aplicações capazes de melhorar as redes de forma significativa.

2.3 Limitações das Redes Definidas por Software

SDNs representam um grande avanço em relação às redes tradicionais, isso ocorre pois permitem a programabilidade da rede deixando-a mais flexível e permitindo melhorias mais rápidas que não dependem de fabricantes de equipamentos.

É preciso, no entanto, uma atenção especial ao tamanho das tabelas de fluxo instaladas no plano de dados. Uma vez que há um crescimento da rede e do número de fluxos, as tabelas tendem a aumentar. Em [35] é feita uma análise de desempenho de comutadores OpenFlow implementados em Linux considerando a ocupação de tabelas de manutenção de estados da rede. É verificado que uma sobrecarga na tabela de fluxos, tem como resultado uma queda no desempenho do sistema. Isso porque operações de busca em tabelas que possuem um grande número de regas terão um número maior de elementos a ser comparados e demandarão mais tempo do que em uma tabela com poucas regras. Deve-se então pensar em formas de conter o crescimento da tabela de fluxos ou formas alternativas à utilização da tabela.

Baseado no Teorema Chinês do Resto [36] foi desenvolvido o KeyFlow [11] que tem como objetivo eliminar as tabelas de fluxo, substituindo-as por uma simples operação de **mod**, que determina o resto de uma divisão. Esse resto da divisão do rótulo dado aos pacotes pelo número do *switch*, corresponde a porta pela qual os pacotes deverão ser encaminhados.

2.4 Estudo de Aplicações em NetFPGA

A fim de verificar a influência que as tabelas de fluxos exercem sobre a latência de comutação de pacotes e como a manipulação dessas tabelas pode ser útil para melhorar a vazão de uma Rede Definida por Software, neste trabalho serão estudadas experimentalmente algumas aplicações em NetFPGA.

A primeira aplicação, utiliza a NetFPGA como ferramenta de geração, transporte de pacotes e medição para avaliar experimentalmente a influência das tabelas de fluxos

na latência de SDNs. Para este fim, serão adotados dois métodos de encaminhamento: o primeiro, que não necessita de tabelas para encaminhar pacotes, é o KeyFlow e o segundo, é o protocolo mais utilizado atualmente em SDN, o OpenFlow, que realiza consultas a tabelas para encaminhar pacotes.

O segundo caso no qual a NetFPGA será inserida, irá comparar os ganhos de vazão que podem ser obtidos ao substituir um método de encaminhamento simples, sem tabela de fluxos, por uma NetFPGA programada com o protocolo OpenFlow. Esse estudo será feito em uma Rede Definida por Software autônoma chamada TRIIIAD [12] que utiliza a computação óptica para encaminhar pacotes.

2.4.1 KeyFlow

KeyFlow [11] é um método de encaminhamento da arquitetura SDN que possibilita a redução da latência de encaminhamento das redes. Isso é possível devido a substituição da pesquisa na tabela de fluxos por uma simples operação de **mod**, que determina o resto de uma divisão. Esse resto, por sua vez, representará a porta de saída dos pacotes.

O mecanismo de encaminhamento utiliza o Esquema de Informação por Chave (KIS) [37] que foi proposto para redes ópticas na criação de topologias overlay em redes OpenFlow. As chaves são os identificadores dos nós da rede, a restrição para determinação das chaves é que os números sejam primos entre si 2 a 2, ou seja, o máximo divisor comum de quaisquer duas chaves selecionadas dentre a lista de chaves dos nós deverá ser igual a 1.

Para determinar a porta de saída dos pacotes que chegam a um comutador, é feita a divisão do rótulo do pacote pela chave do *switch*. O resto dessa divisão é a porta pela qual os pacotes deverão sair. Esse rótulo é gerado utilizando o Teorema Chinês do Resto (TCR) que baseia-se na ideia de que números grandes podem ser representados pela combinação de números pequenos obtidos a partir do resto da divisão desse número por um conjunto de números primos entre si.

A Figura 2.6 mostra um exemplo de uma arquitetura de rede KeyFlow. Nela, um pacote que sai do nó 1 tem como destino o nó 6 passando pelos nós intermediários mi = (4, 3, 5) com portas de saída pi = (1, 1, 0), respectivamente. O caminho percorrido está representado pelas setas laranja.

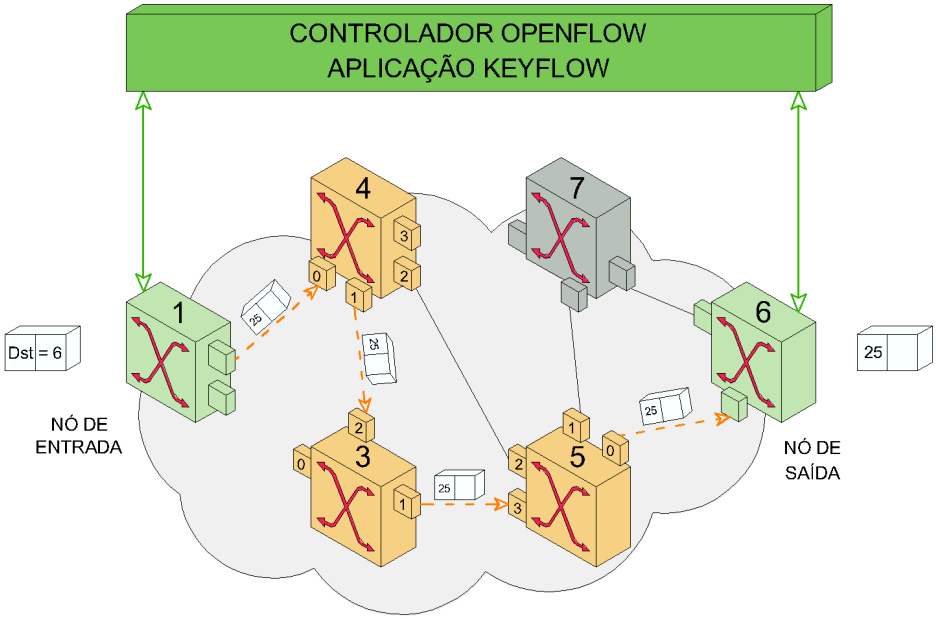

Figura 2.6: Exemplo de arquitetura de rede KeyFlow **[38]**

Para determinar o rótulo do pacote que acaba de chegar, é enviada uma solicitação para o controlador que calcula o rótulo com base no caminho a ser percorrido até o destino. O controlador envia, então, uma regra para os nós de borda com essa informação. A partir disso, será atribuído o mesmo rótulo a todos os pacotes que chegarem tendo o mesmo destino. Já os nós centrais, que são KeyFlow, irão apenas executar a operação de divisão e não possuem tabela de fluxos.

Para o exemplo da Figura 2.6, o rótulo atribuído pelo controlador é 25. Ao chegar ao nó 4 é feita a operação de **mod**, que determina o resto da divisão: (25) mod 4 = 1, isso implica que a porta 1 é a porta de saída. A mesma operação é feita nos nós 3 e 5: (25) mod 3 = 1, (25) mod 5 = 0 e assim, são determinadas as portas de saída de cada nó.

O cálculo do rótulo, feito pelo controlador, é feito da seguinte forma:

Seja:

- Chaves dos nós: mi = (4, 3, 5)
- Portas de saída: pi = (1, 1, 0)

Então:

1. Obtém-se o parâmetro Mi de cada nó dividindo o produto de todos os nós (M) pela chave correspondente a cada nó:

   M = 4*3*5 = 60, logo:

   $M_1 = 60/4 = 15$

   $M_2 = 60/3 = 20$

   $M_3 = 60/5 = 12$

2. Para cada nó, é calculado o valor de Li, sendo Li o produto inverso de Mi, ou seja, Mi*Li = 1, então:
   $L_1 = (M_1^{-1})_{\text{mod m1}} = (15^{-1})_{\text{mod 4}} = 3$
   $L_2 = (M_2^{-1})_{\text{mod m2}} = (20^{-1})_{\text{mod 3}} = 2$
   $L_3 = (M_3^{-1})_{\text{mod m3}} = (12^{-1})_{\text{mod 5}} = 2$
3. Por fim, é calculado o rótulo do pacote (X) que é a soma do produto de Mi, Li e pi de cada nó:
   $X = (L_1*M_1*p_1 + L_2*M_2*p_2 + L_3*M_3*p_3)_{\text{ mod M}}$
   $X = (45 + 40 + 0)_{\text{ mod 60}} = 25$

Esse método faz com que os nós centrais não necessitem ter tabelas de fluxos a eles associadas.

2.4.2 TRIIIAD: Triple-Layered Intelligent and Integrated Architecture for Datacenters

Arquiteturas de redes de *data center* podem ser classificadas como centradas em redes ou centradas em servidores. Arquiteturas centradas em redes são as arquiteturas tradicionais, em que os dispositivos que realizam o encaminhamento dos pacotes da rede são dispositivos externos aos *hosts* como *switches*, roteadores etc. Já arquiteturas centradas em servidores, são os próprios servidores que realizam o encaminhamento dos pacotes gerados.

A TRIIIAD [12] (**TRI**ple-Layered **I**ntelligent and **I**ntegrated **A**rchitecture for **D**atacenters) consiste em uma arquitetura de rede de *data centers* autônoma centrada em servidores composta por três camadas horizontais sobrepostas: uma camada híbrida reconfigurável, uma cada de transporte e uma camada de virtualização. Todas elas são integradas por um plano vertical de controle, gerência e orquestração como pode ser observado na Figura 2.7.

A camada de virtualização representa redes e máquinas virtuais presentes no *data center*. Ela é responsável por permitir a criação e destruição de redes virtuais e máquinas virtuais (VMs), bem como a migração de VMs entre servidores. Provê o compartilhamento dos recursos físicos e torna a arquitetura compatível com as tecnologias de rede mais difundidas, como IPv4 e Ethernet, por exemplo. É essa camada que permite a entrega de produtos/serviços mais flexíveis e customizados para os clientes de *data center*.

A camada de encaminhamento não possui nenhum elemento adicional de rede (como roteadores e *switches*), além dos próprios servidores, tem como principal funcionalidade fornecer um mecanismo eficiente para o encaminhamento de dados por toda a rede. Além disso, é o elo entre a camada de virtualização e a camada híbrida reconfigurável. Ela utiliza o algoritmo XOR que determina o menor caminho a ser percorrido pelos pacotes entre os *hosts* de origem e o destino sem consultas a tabelas de fluxos.

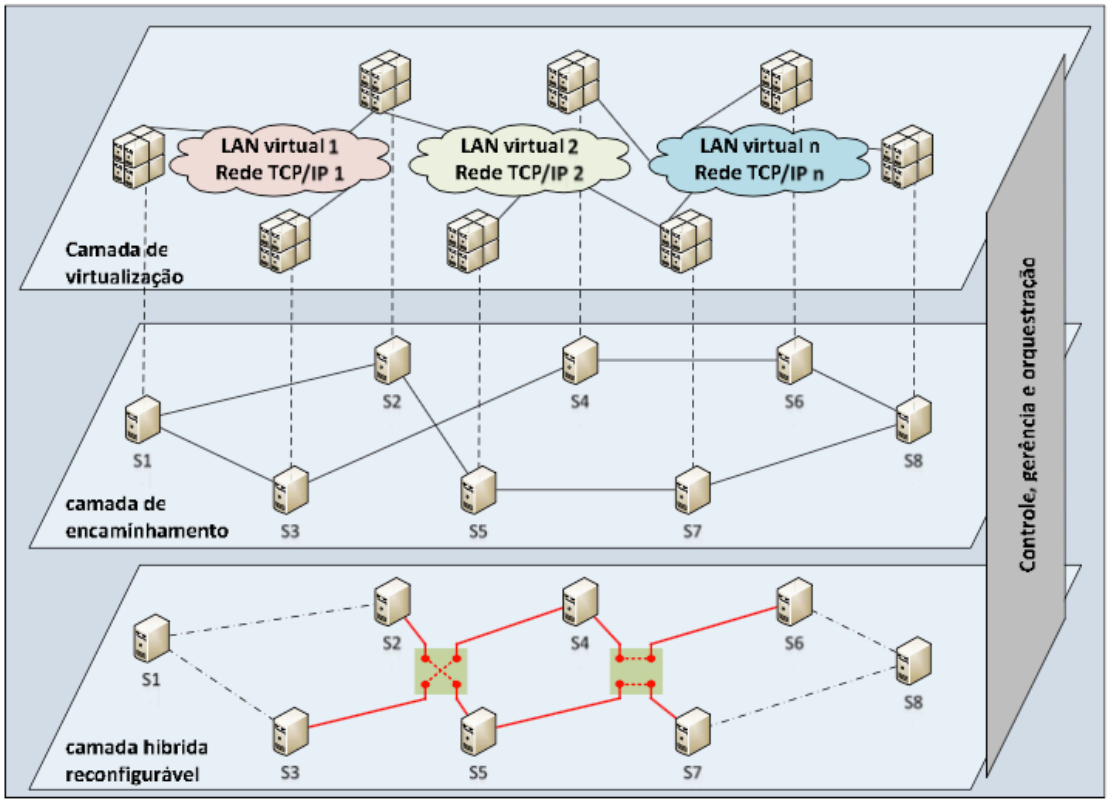

Figura 2.7: Arquitetura TRIIIAD **[12]**

A camada híbrida reconfigurável é composta por enlaces elétricos, enlaces ópticos e comutadores 2x2. Ela tem como principal objetivo a redução do tráfego de trânsito, que é o tráfego encaminhado pelos servidores que estão entre a origem e o destino. Isso é feito reconfigurando os enlaces físicos para que atalhos sejam criados na rede diminuindo, assim, o número de saltos para chegar a um determinado nó.

O plano de controle, gerência e orquestração atua de maneira vertical na arquitetura sendo responsável por alinhar o funcionamento das três camadas. Ele possui desde as funções de alto nível, como a migração de VMs até as de mais baixo, como a manipulação dos enlaces ópticos da camada híbrida para reconfigurar a rede.

2.4.2.1 Reconfiguração Óptica

A camada híbrida reconfigurável possui a propriedade de ser reconfigurada graças à utilização de chaves ópticas [39] que interligam os *hosts*. Ora as chaves podem estar em configuração barra, assim como mostrado na face frontal da figura 2.8a em que o *host* 000 está ligado ao *host* 001 e o *host* 100 está ligado ao *host* 101, ora podem estar em configuração cruzada, assim como mostra a face frontal da figura 2.8b em que o *host* 000 está ligado ao *host* 101 e o *host* 100 está ligado ao host 001.

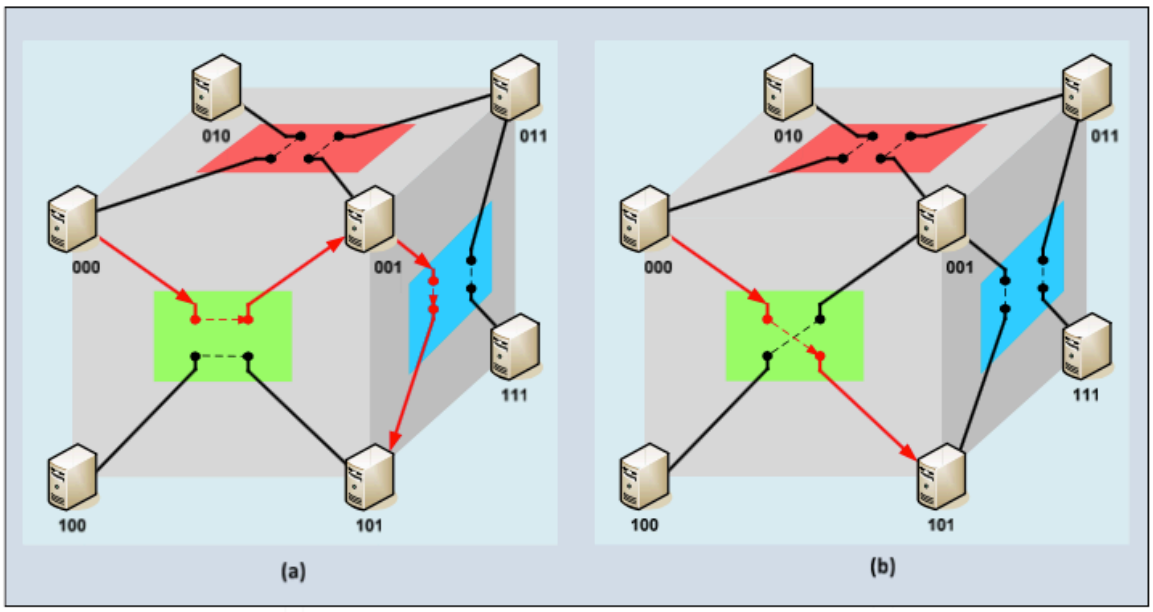

Figura 2.8: Reconfiguração de enlaces ópticos - a) Chaves em barra b) Chaves em cruz **[12]**

Esse mecanismo de reconfiguração dos enlaces ópticos possibilita a redução do tráfego de trânsito gerado em função da arquitetura centrada em servidores. Na Figura 2.8, por exemplo, um tráfego sai do *host* 000 para o *host* 101 inicialmente passando pelo *host* 001. Após o chaveamento, o tráfego de trânsito no *host* 001 não existe mais, uma vez que há uma ligação direta entre os *hosts* 000 e 101. Essa estratégia, no entanto, não é capaz de eliminar o tráfego de trânsito total da rede, apenas reduz o tráfego de trânsito médio.

Outra limitação está em tratar diferentes tráfegos de dados individualmente, uma vez que o método de encaminhamento utilizado não possui tabela de fluxos não há como diferenciá-los e manipulá-los um a um.

2.5 Latência de Rede

Em uma rede, a latência é a medida de quanto tempo um pacote de dados leva para ir de um ponto a outro da rede. Conhecer essa grandeza é fundamental para garantir a confiabilidade e eficiência de serviços de rede que operam suas aplicações em tempo real. Para garantir que as solicitações dos usuários serão entregues conforme o esperado, é necessário aos desenvolvedores conhecer os valores de latência de seus diversos serviços. E com isso, avaliar a eficiência e o estudar melhorias.

A partir dessa necessidade de avaliar o desempenho da rede e suas aplicações, foram criadas plataformas e ferramentas que possibilitam aos desenvolvedores realizar testes na rede e obter os valores correspondentes ao seu desempenho. Dentre os *softwares* e *hardwares* existentes, podemos destacar o pktgen [40], o Anritsu MD1230B Data Quality Analyzer [41] e o OFLOPS [42], esta última será a ferramenta utilizada neste trabalho para avaliar a latência de dispositivos habilitados com KeyFlow ou OpenFlow.

2.5.1 OFLOPS

OFLOPS [42] foi pioneiro como plataforma aberta e genérica para testes OpenFlow. Ele tem como foco principal fornecer um conjunto de testes base para que desenvolvedores possam entender, quantificar os recursos e localizar gargalos de dispositivos OpenFlow habilitados. Sua plataforma unificada permite que sejam obtidas tanto informações relativas aos canais de dados e de controle, como também dados específicos do comutador, como utilização de CPU, ocupação de memória, número de pacotes enviados e recebidos etc.

A Figura 2.9 ilustra o desenho da plataforma, há um canal de controle e múltiplos canais de dados. Além disso, suporta o protocolo SNMP [43] para ler as informações relativas ao comutador citadas anteriormente.

A plataforma oferece uma API orientada a eventos que permite aos desenvolvedores manipular eventos de teste através de módulos programáveis, a fim de implementar e medir funcionalidades customizadas do controlador. Os experimentos são implementados por um conjunto de bibliotecas compartilhadas e carregadas dinamicamente. Além disso, possui implementação para múltiplas tarefas, o que reduz o atraso no processamento de medições.

OFLOPS tem integração com plataformas heterogêneas, desde PCs *commodities* com múltiplas placas de rede até computadores equipados com placas NetFPGA. Isso é possível uma vez que há suporte a vários mecanismos de geração e captura de pacotes.

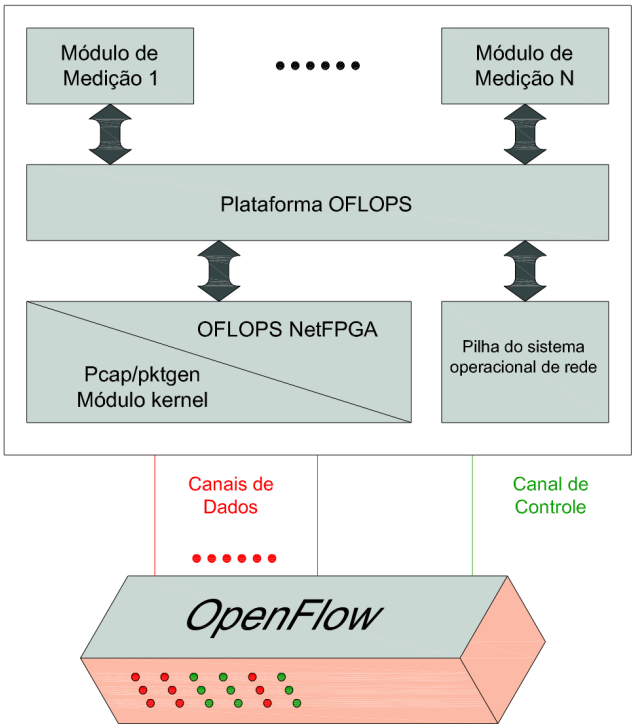

Figura 2.9: Plataforma OFLOPS **[44]**

Para geração de pacotes, o OFLOPS suporta três mecanismos diferentes: modo usuário, modo kernel através do módulo pktgen e através do gerador de tráfego em hardware implementado em NetFPGA. Já para capturar pacotes são suportados a biblioteca pcap (modo kernel) e o modo baseado em hardware implementado em NetFPGA. No entanto, a precisão de microssegundos somente é obtida utilizando placas NetFPGAs para geração e captura de pacotes.

Capítulo 3 - Metodologia

Este capítulo descreve a metodologia utilizada para a avaliação da latência de encaminhamento, utilizando um método que necessita de tabelas de fluxos e um método que não as utiliza para tal. Será também apresentada a metodologia utilizada para a avaliação de vazão na arquitetura TRIIIAD, com a substituição de dispositivos ópticos por uma NetFPGA programada com o protocolo OpenFlow.

3.1 Avaliação de Latência

Conhecer a latência da rede atualmente é um requisito de grande importância para os projetistas e administradores de rede. Pois assim é possível prever o tempo de resposta de uma determinada requisição, planejar melhorias a partir da substituição de dispositivos, detectar falhas e gargalos caso um tempo de resposta já conhecido aumente de maneira significativa, etc.

Este trabalho propõe a substituição das chaves ópticas, localizadas na camada híbrida de encaminhamento da TRIIIAD, por placas NetFPGA programadas com OpenFlow que necessita de tabelas de fluxos para realizar encaminhamento de pacotes. Sendo assim, será feita uma avaliação do quanto as tabelas de fluxos impactam na latência de comutação em dispositivos NetFPGA.

A NetFPGA foi o dispositivo escolhido para este trabalho por ser uma plataforma de código aberto e devido ao seu baixo custo em relação a comutadores OpenFlow habilitados. Além disso, se comparada a *switches* baseados em *software* que emulam o funcionamento do OpenFlow, a NetFPGA possui uma grande vantagem que é o encaminhamento de pacotes ser feito em *hardware* e o processamento dos fluxos ser feito imediatamente quando são recebidos pela placa, permitindo que a taxa máxima do enlace Gigabit Ethernet seja atingida sem que haja perda de pacotes. Essas características da NetFPGA resultam em um aumento significativo de desempenho em relação a comutadores com OpenFlow emulado e em uma aproximação do ambiente de experimentação da realidade.

3.1.1 Avaliação da Influência das Tabelas de Fluxo

Para que se possa avaliar a influência das tabelas de fluxo na latência de comutação de pacotes, são utilizados dois tipos de métodos de encaminhamento: um que necessita realizar consultas a tabelas para encaminhar pacotes e um que encaminha os pacotes tendo como base outro tipo de operação.

Antes de realizar quaisquer medidas de latência, bem como desvio padrão, é necessário obter os valores inerentes ao sistema de medição utilizado e assim, descontá-los dos resultados apresentados pelos métodos de encaminhamento escolhidos. Isso é feito com a utilização de *loopback*, em que as interfaces de entrada e saída de pacotes do sistema de medição são interligadas. Assim, os pacotes não passam por nenhum comutador em seu trajeto, permitindo que as medidas de latência obtidas sejam inerentes unicamente ao sistema de medição. Após essa caracterização, iniciam-se então as medições de latência dos comutadores.

Para o comutador que não necessita de tabelas de fluxos para realizar o encaminhamento dos pacotes, o teste consiste em variar a taxa com a qual os pacotes são enviados e então, coletar os valores de latência e desvio padrão referente a cada taxa.

Para os testes do comutador que realiza consultas a tabela de fluxos, é necessário instalar as regras antes da execução dos testes, pois o objetivo é avaliar a influência do número de regras na latência de encaminhamento dos pacotes e não o tempo de instalação das mesmas.

Os testes são feitos com a tabela cheia e vazia a fim de verificar qual o impacto do número de regras na latência, bem como, se há influência da posição em que a regra a ser utilizada se encontra nas medidas de latência. Para o teste com a tabela vazia é instalada apenas a regra a ser utilizada pelo comutador para realizar o encaminhamento. Já para o caso da tabela cheia, são foram feitos três tipos de testes em que a posição da regra a ser utilizada é variada, ela é colocada no inicio, meio e fim da tabela. A variação do tamanho da tabela é feita a partir da instalação de regras que não são úteis para o comutador, com números de MAC de origem e/ou destino que não correspondem aos utilizados no encaminhamento.

Para a geração de MACs aleatórios foi feito um *script* em Python e a instalação das regras tanto com os MACs falsos, quanto com a regra que contém os MACs correspondentes a origem e destino dos pacotes, são feitas por meio de um *script* em Shell Script. Este, possui a regra a ser utilizada com todos os campos preenchidos e

ainda recebe como entrada os MACs falsos que preenchem as demais regras. Para que haja variação da posição da regra correspondente ao encaminhamento na tabela de regras, é variada a posição desta regra no corpo do *script*. Isso é feito da seguinte forma:

- Primeiro ela é instalada antes das demais ocupando, assim, a primeira posição da tabela;
- Em seguida, é instalado um número aleatório de regras falsas antes da instalação da regra de encaminhamento e o restante após sua instalação, fazendo com que ela ocupe uma posição no meio da tabela;
- E, por último, após a instalação de todas as outras regras, ocupando a última posição da tabela.

Para cada um desses casos é feita a variação da taxa de envio de pacotes e são obtidos os respectivos valores de latência e desvio padrão.

3.1.2 Métodos de Encaminhamento Utilizados

Para os testes sem tabelas de fluxos foi utilizada uma implementação do comutador KeyFlow em NetFPGA que é um projeto desenvolvido no laboratório NERDS (Núcleo de Estudos de Redes Definidas por Software). O *switch* KeyFlow desenvolvido teve sua implementação baseada na implementação do OpenFlow 1.0 para NetFPGA da Universidade de Stanford.

O KeyFlow foi a tecnologia escolhida uma vez que para os testes sem tabelas era necessário um comutador com implementação em NetFPGA que utilizasse outra forma de encaminhamento, que não fosse a pesquisa em tabelas.

Além de ter as características necessárias para ser utilizado como método de encaminhamento, o KeyFlow foi escolhido para os testes por ser uma ferramenta implementada por nosso grupo de pesquisa. Outro fator de escolha está ligado a testes comparativos de latência anteriormente executados por nosso grupo afim de comparar o *switch* KeyFlow e *switch* OpenFlow [45]. Esses testes, no entanto, foram feitos apenas em ambientes emulados, Mininet [46] e agora, tem-se a chance de realizar uma prova de conceito em ambiente real do impacto da tabela de fluxos na latência de comutação.

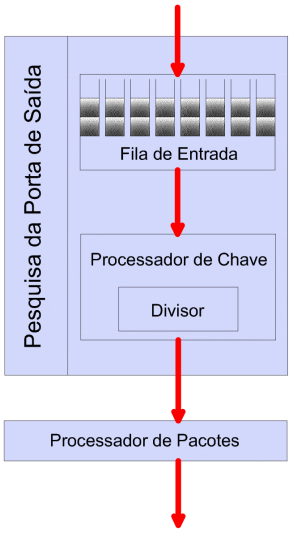

Figura 3.1: Método de operação de encaminhamento KeyFlow **[38]**

A Figura 3.1 mostra um esquemático do método de operação do KeyFlow. Os pacotes chegam ao comutador e entram na fila para serem processados, ocorre a operação de **mod** e os pacotes são então enviados a porta de saída determinada pelo resto da divisão.

Para os testes com tabelas de fluxos optou-se pela implementação em NetFPGA da Universidade de Stanford do comutador OpenFlow 1.0. Essa foi a implementação escolhida pois será utilizada na substituição das chaves ópticas da camada híbrida reconfigurável da TRIIIAD. Logo, deve ser a adotada para a avaliação de latência de rede, para que se possa caracterizá-la.

A Figura 3.2 mostra um esquemático do método de operação do OpenFlow. Os pacotes chegam ao comutador e entram na fila para serem processados. O cabeçalho do pacote é analisado e é feita uma busca dentre as regras completas e regras curinga (regras incompletas, que possuem campos que podem ser preenchidos ou não). Após a busca na tabela de regras, o árbitro verifica se existe uma regra já instalada que possa ser utilizada, caso não exista, uma nova regra é instalada. Os pacotes, então, são encaminhados de acordo com a regra de encaminhamento.

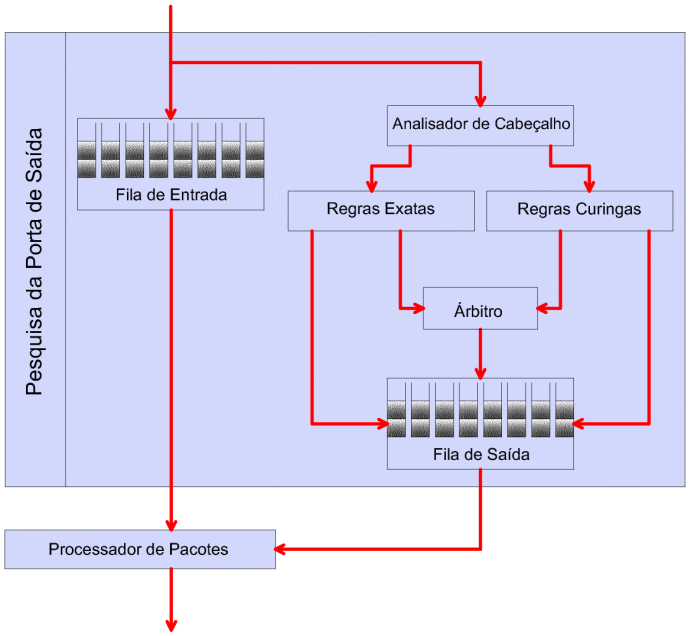

Figura 3.2: Método de operação de encaminhamento OpenFlow **[38]**

A implementação do OpenFlow para NetFPGA possui a tabela de regras subdivida em duas: tabela *hash*, que possui 131.070 entradas de regras exatas e a tabela linear, que possui 100 entradas para regras curingas, ou seja, regras incompletas [45]. Além disso, a NetFPGA agrega mais uma tabela, de acesso rápido, a essa subdivisão que forma o conjunto total de regras. Trata-se da tabela nf2, que possui 32.768 entradas para regras exatas e 24 entradas para regras curingas [47].

Sendo assim, a organização das tabelas para implementação do OpenFlow em NetFPGA é feita da seguinte forma: tabela nf2 (*table*=0, máximo 32.768 entradas exatas e 24 curingas), tabela *hash* (*table*=1, máximo 131.070 entradas exatas) e tabela linear (*table*=2, máximo 100 regras exatas). A busca nas tabelas de regras é feita de forma sequencial dentre as tabelas existentes indo do menor ao maior índice, ou seja, começa pela tabela nf2, caso não seja encontrado um fluxo correspondente ao cabeçalho dos pacotes, a busca passa para a tabela *hash* e por fim, para a tabela linear.

3.1.3 Geração de Quadros e Medições

Os quadros foram gerados pelo gerador de tráfego do OFLOPS. Este possui interfaces geradoras de tráfego que suporta três mecanismos de operação: via modo usuário, via modo *kernel* (pktgen) ou via gerador de tráfego em *hardware* implementado em NetFPGA. No entanto, somente no último caso pode-se alcançar uma

precisão da ordem de microssegundos para medir o tempo de resposta do comutador. A taxa de tráfego gerado é limitada em 1010Mbps, isso ocorre pois estamos trabalhando com NetFPGAs de 1Gbps. Sendo assim, as taxas de envio predefinidas variaram de 10Mbps a 1010Mbps com intervalos de 50Mbps.

Além de utilizar as interfaces geradoras de tráfego do OFLOPS, também é utilizada a plataforma de medições. Esta suporta duas formas de operação: modo *kernel* (pcap) e modo baseado em *hardware* implementado em NetFPGA. Este último foi o módulo utilizado para os testes, pois assim como ocorre com a etapa de geração, somente é possível obter uma precisão de microssegundos, se todo o processo de medição envolvendo a marcação, geração e captura dos quadros é realizada por uma implementação em NetFPGA.

Todas as medidas são feitas com pacotes de tamanho pequeno para que o pacote possa ser enviado por inteiro, sem necessidade de ser subdividido. Isso porque, a necessidade de divisão do pacote, impacta em um aumento da latência, uma vez que todas as subdivisões do pacote deverão chegar ao destino para que a chegada do pacote seja considerada. Para esses testes optou-se por um tamanho de pacotes igual a 150 bytes, essa escolha foi feita tomando como referência uma avaliação experimental do OFLOPS como ferramenta de medida do OpenFlow, feita pelo nosso grupo de pesquisa em [48] que mostrou que pacotes de até 350 bytes não precisam ser subdivididos. Dado que os valores utilizados para os testes variaram de 150 bytes a 1500 bytes, foi escolhido o menor valor possível.

3.1.4 Processo de medição

Foi utilizado o módulo *Action Delay*, disponibilizado pelo OFLOPS, para execução dos testes. Este, mede o impacto da implementação de uma ação sobre o plano de dados de um *switch* que, ao iniciar o teste, envia pacotes para um segundo *switch* conectado a ele. Após 30 segundos, é instalada uma regra no comutador que representa a ação a ser testada (exemplo: troca a porta de saída) e é então gerado um volume de quadros por mais 30 segundos para que sejam executadas as medições. Com isso, são obtidas as informações de *timestamps* do gerador de quadros. Estas, são processadas pelo módulo do OFLOPS e assim, obtém-se os valores de média (latência) e desvio padrão.

Para o módulo utilizado, o valor *default* de tamanho dos pacotes é 1500 bytes. Essa medida, porém, faz com os pacotes tenham que ser fragmentados para que sejam encaminhados impactando, assim, no valor da latência de comutação. Para que isso não ocorresse, optou-se por utilizar em todos os testes pacotes menores, iguais a 150 bytes.

Como o objetivo destes experimentos é avaliar o impacto de múltiplas regras na tabela de fluxos e não o seu tempo de instalação, para os testes com o OpenFlow foram instaladas tanto a regra de encaminhamento a ser utilizada pelo comutador, quanto as demais antes do início da execução do testes.

Todos os testes foram sintetizados por meio de *scripts*, posteriormente armazenados em arquivos e em seguida gerados os gráficos de resultados. Os seguintes passos foram executados para a coleta dos resultados de cada amostra:

Para os experimentos com o KeyFlow:

1. programação das NetFPGAs com os “arquivos.bit” , tanto oflops.bit quanto keyflow.bit;
2. inicialização do ofprotocol para que o comutador KeyFlow se conecte ao OFLOPS;
3. execução do módulo Action Delay que envia os quadros e gera os resultados de latência e desvio padrão;
4. finalização do ofprotocol.

Para os experimentos com o OpenFlow contendo apenas a regra de encaminhamento:

1. programação das NetFPGAs com os “arquivos.bit” , tanto oflops.bit quanto openflow.bit;
2. inicialização do ofdatapath, que inicializa as tabelas de fluxo;
3. inicialização do ofprotocol para que o comutador OpenFlow se conecte ao OFLOPS;
4. instalação da regra de encaminhamento a ser utilizada pelo módulo;
5. execução do módulo *Action Delay* que envia os quadros e gera os resultados de latência e desvio padrão;
6. finalização do *ofdatapath*;
7. finalização do *ofprotocol*.

Para os experimentos com o OpenFlow contendo a regra de encaminhamento e as tabelas cheias (sejam elas tabelas com regras exatas ou curingas):

1. programação das NetFPGAs com os “arquivos.bit” , tanto oflops.bit quanto openflow.bit;
2. inicialização do ofdatapath, que inicializa as tabelas de fluxo;

3. inicialização do ofprotocol para que o comutador OpenFlow se conecte ao OFLOPS;
4. instalação das regra extra (curingas e exatas);
5. instalação da regra de encaminhamento a ser utilizada pelo módulo;
6. execução do módulo *Action Delay* que envia os quadros e gera os resultados de latência e desvio padrão;
7. finalização do *ofdatapath*;
8. finalização do *ofprotocol*.

3.2 Avaliação de Vazão na Arquitetura TRIIIAD

A fim de aumentar a vazão da TRIIIAD e melhorar a distribuição de tráfego, optou-se por abrir mão da simplicidade de um método de encaminhamento sem tabelas de fluxos, as chaves ópticas, e adotar um método que necessita consultar regras instaladas em tabelas. Assim, é necessário avaliar o ganho de vazão e distribuição de tráfego que podem ser obtidos aumentando a complexidade de encaminhamento. Para isso, as chaves ópticas foram substituídas por uma NetFPGA programada com OpenFlow 1.0 e foram realizados três tipos de experimentos:

- NetFPGA operando como chaves ópticas, a fim de verificar a possibilidade da NetFPGA realizar um encaminhamento similar ao óptico;
- NetFPGA contendo as regras instaladas a partir dos MACs das máquinas virtuais, a fim de encaminhar os pacotes pelo menor caminho possível entre os *hosts* de origem e destino;
- E, por último, NetFPGA com controle total dos fluxos, que instala e modifica as regras de cada fluxo individualmente, com o objetivo de realizar uma engenharia de tráfego, a fim de diminuir uma possível sobrecarga do menor caminho com rotas alternativas.

Para cada caso, foram realizados testes por meio de *scripts* que rodavam nas máquinas virtuais da TRIIIAD. Foi utilizado o iperf como ferramenta de geração e recepção de tráfego. Além disso, foi utilizado um processo monitor rodando nos *hosts* físicos a fim de monitorar os usos de CPU, memória e tráfego. A cada 30 segundos esses valores eram reportados ao controlador. Para cada teste foram realizadas 30 repetições, exceto os aleatórios, os quais foram rodados apenas uma vez.

Mais detalhes sobre como as regras foram configuradas e instaladas na NetFPGA serão dados nas subseções a seguir.

3.2.1 NetFPGA Operando como Chaves Ópticas

Foi reproduzido o cenário de comutação óptica com a NetFPGA, que assumiu o mesmo funcionamento das chaves ópticas. Para isso, foram instaladas regras que imitam as configurações barra e cruz das chaves ópticas.

Para a configuração em barra, foram instaladas regras com o seguinte comando: tudo que entra pela porta 1 da NetFPGA sai pela porta 2 e vice versa, interligando as portas 1 e 2 e tudo que entra pela porta 3 da NetFPGA sai pela porta 4 e vice versa, interligando as portas 3 e 4. Para a configuração em cruz, as regras instaladas possuíam o seguinte formato: tudo que entra pela porta 1 da NetFPGA sai pela porta 4 e vice versa, interligando as portas 1 e 4 e tudo que entra pela porta 3 da NetFPGA sai pela porta 2 e vice versa, interligando as portas 2 e 3.

Essas regras foram instaladas de maneira proativa pelo controlador, iniciando sempre com a configuração em barra como default e de acordo com a necessidade da rede, as regras eram modificadas para a configuração em cruz.

3.2.2 NetFPGA Contendo Regras Instaladas a Partir do MAC das Máquinas Virtuais

Com o objetivo de reduzir o número de saltos e eliminar o tráfego de trânsito da rede, todos os *hosts* que estavam ligados à NetFPGA foram interligados diretamente. Para isso, foram feitos os testes com regras curingas, contendo o número MAC de cada maquina virtual no campo de MAC de destino e a porta de destino pela qual os pacotes deveriam sair para percorrer o menor caminho até o destino.

As regras foram instaladas obtendo-se o endereço MAC de cada máquina virtual. Este, foi obtido por meio de consulta ao banco de dados que armazena as informações de cada *host*. Uma vez obtido o MAC, foi calculada a porta de saída da NetFPGA mais próxima ao *host* da VM. Isso é possível uma vez que os MACs das VMs são armazenados da seguinte forma: os 4 primeiros bytes identificam o *host* físico da VM e os outros 2 bytes identificam a VM. Logo, a partir dos 4 primeiros bytes é possível

saber em qual *host* a VM está instalada, e assim é calculada a porta da NetFPGA mais próxima a ele.

Assim como no caso anterior, as regras foram instalas de maneira proativa pelo controlador. Cada vez que uma nova VM era instanciada na rede, uma nova regra era instalada.

3.2.3 NetFPGA com Controle Individual de Fluxos

Por último, foram realizados testes em que os fluxos são tratados de maneira individual, assim é possível realizar um processo de múltiplos caminhos, desviando os fluxos de acordo com a necessidade. Quando uma interface de um *host* estava sobrecarregada, parte do tráfego era desviada por um caminho alternativo quando possível, chegando assim por outra interface do *host*. O fluxograma do algoritmo de tratamento de fluxos desta última etapa pode ser visto na Figura 3.3.

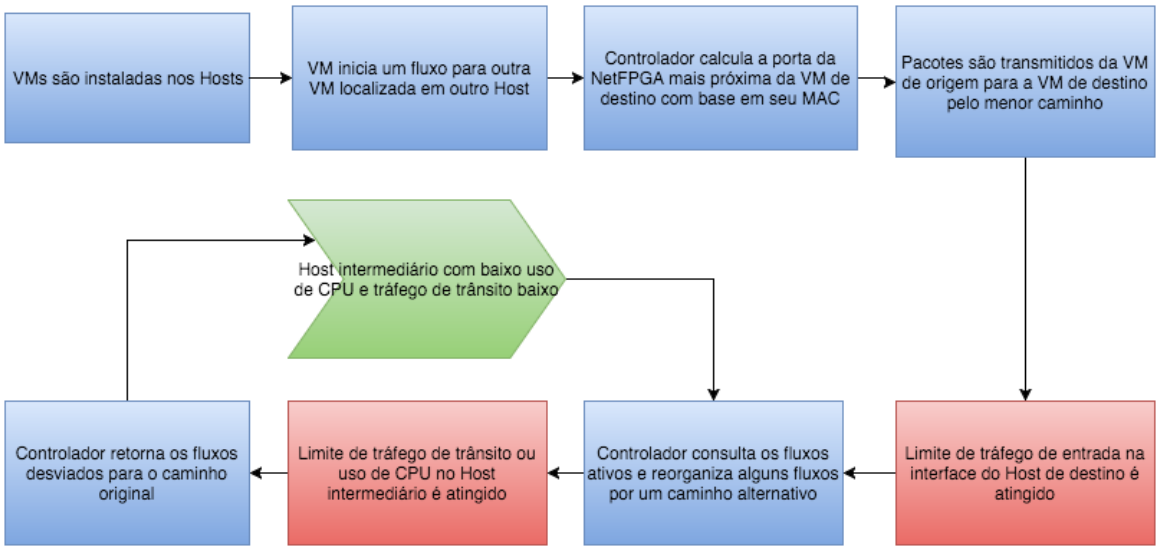

Figura 3.3: Fluxograma do algoritmo para controle individual de fluxos

Para esta última etapa, ao contrário das duas anteriores, as regras foram instaladas na tabela de maneira reativa, pois os fluxos deveriam ser caracterizados individualmente com todos os campos do cabeçalho preenchidos. Além disso, todas as regras possuíam tempo de expiração quando não utilizadas, isso foi feito para que a tabela de fluxos não ficasse muito extensa. Assim, quando chegava à NetFPGA um pacote que não possuía uma regra a ele associada, uma requisição era enviada ao controlador, que extraía todas as informações de seu cabeçalho e instalava uma regra completa na tabela.

Inicialmente, as regras eram instaladas para cada máquina virtual apontando como porta de saída aquela que resultaria no menor caminho. Quando era necessário mudar a rota do fluxo, a porta de saída da regra daquele fluxo era modificada para que este saísse por uma porta diferente e percorresse um caminho alternativo. Se o limite de tráfego de trânsito máximo ou o uso máximo de CPU do *host* intermediário fosse atingido, o controlador retornava os fluxos desviados para o caminho original. Caso fosse possível os fluxos eram desviados novamente para o caminho alternativo, conforme mostra a Figura 3.3.

Visto que estamos trabalhando com NetFPGAs de 1Gbps para execução dos testes, a taxa de tráfego em cada uma das porta poderia ser é limitada em 1010Mbps. No entanto, isso não foi possível pois durante a execução dos testes foi constatado que os oito *desktops* utilizados como nós de computação da TRIIIAD [12] começavam a perder pacotes para taxas acima de 600Mbps. Dado que os testes serão feitos em apenas uma face do cubo e cada nó de computação possui apenas duas interfaces de rede em cada plano de chaveamento, fixou-se a taxa máxima de fluxo por interface em 300Mbps. Outros dois parâmetros definidos para esses testes foram: o limite de tráfego de trânsito igual a 200Mbps e o limite de utilização de CPU igual a 60%.

Capítulo 4 - Ambiente de Avaliação da Latência de Comutação em NetFPGA: KeyFlow versus OpenFlow

O objetivo deste Capítulo é apresentar o ambiente de teste utilizado para a avaliação de latência de comutação dos *switches* KeyFlow e OpenFlow, bem a discussão dos resultados obtidos.

4.1 Ambiente de Testes Utilizado

Para realização dos experimentos foi utilizado o OFLOPS como ferramenta geradora de pacotes. Estes foram enviados ao comutador Keyflow/OpenFlow que, por sua vez, os encaminharam de volta ao OFLOPS para que fossem processados (vide Figura 4.1) e, assim, obtidas as medidas de latência e seu desvio padrão.

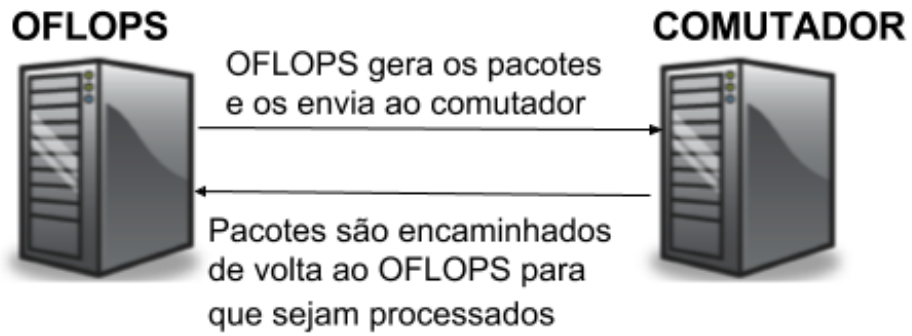

Figura 4.1: Ambiente de medição: OFLOPS gera pacotes que vão para o Comutador e são encaminhados de volta

Para os comutadores KeyFlow/OpenFlow foi utilizada uma NetFPGA de 1Gbps instalada na interface PCI de um computador com processador Dual Core 3GHz, 4GB de RAM, HD 1TB, com sistema operacional Linux CENTOS 5.11 32bits. Foi utilizada a implementação padrão da Universidade de Stanford do OpenFlow 1.0, já o design do *hardware* do KeyFlow é baseado nas implementações de referência desta mesma universidade.

Para o OFLOPS foi utilizada uma NetFPGA de 1Gbps instalada na interface PCI de um computador com processador AMD Athlon IIX4 640 3GHz, 4GB de RAM, HD 1TB, com sistema operacional Linux CENTOS 5.11 32bits.

4.2 Resultados Obtidos

Nesta seção são apresentados e discutidos os resultados obtidos nos experimentos realizados nessa ordem: OFLOPS em *loopback*, testes do KeyFlow, testes do OpenFlow com apenas a regra de encaminhamento a ser utilizada pelo comutador e por fim, testes do OpenFlow com as tabelas cheias de fluxos, dentre eles a regra a ser utilizada pelo comutador. Foi feita apenas uma rodada de testes para cada um dos casos, pois o OFLOPS faz automaticamente a média dos resultados de latência e de desvio padrão de todos os milhares de pacotes encaminhados em cada rodada de teste, garantindo, assim, a confiabilidade dos resultados obtidos.

4.2.1 OFLOPS em *Loopback*

Para quantificar os valores inerentes ao OFLOPS como sistema de medição foi montado um cenário de *loopback*. Isso foi feito ligando diretamente as 2 primeiras portas da NetFPGA, conforme ilustrado na Figura 4.2. Desta forma, os pacotes não passam por nenhum comutador.

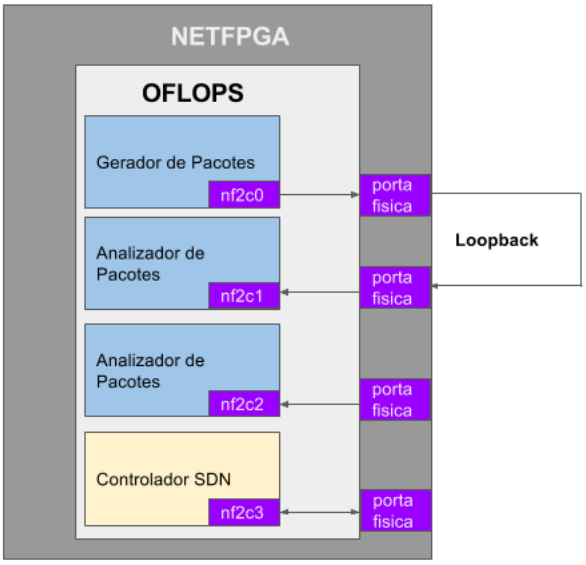

Figura 4.2: Esquemático do OFLOPS em *Loopback*

Observando a Figura 4.3, nota-se que apesar da variação da taxa de dados, a latência permanece constante em 0 microssegundos. Isso se mantém até a taxa de 560Mbps, a partir da qual ocorre um salto na latência para 19 microssegundos e há

também um aumento considerável no desvio padrão. Após este incremento, as medidas de latência e desvio padrão se estabilizam e permanecem constantes.

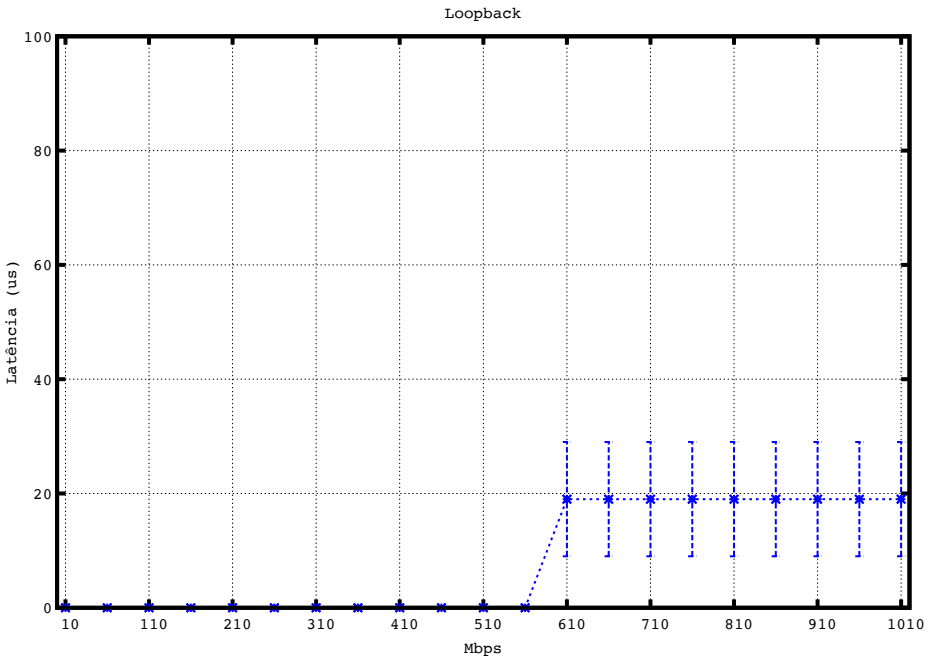

Figura 4.3: Medida de latência – OFLOPS em *Loopback*

Essa latência inerente do OFLOPS para taxas a partir de 560Mbps, provavelmente, se deve a detalhes das interações do gerador de tráfego com a memória SRAM da NetFPGA, conforme discutido em [48].

4.2.2 KeyFlow

Uma vez que o processo de encaminhamento do KeyFlow é feito por meio de uma divisão (a porta de saída é determinada pelo o resto da divisão do rótulo dos pacotes pela chave do comutador), o único teste a ser feito com este comutador é a medida do tempo de ida e volta, *Round-Trip Time* (RTT), dos pacotes de acordo com a variação da taxa de dados.

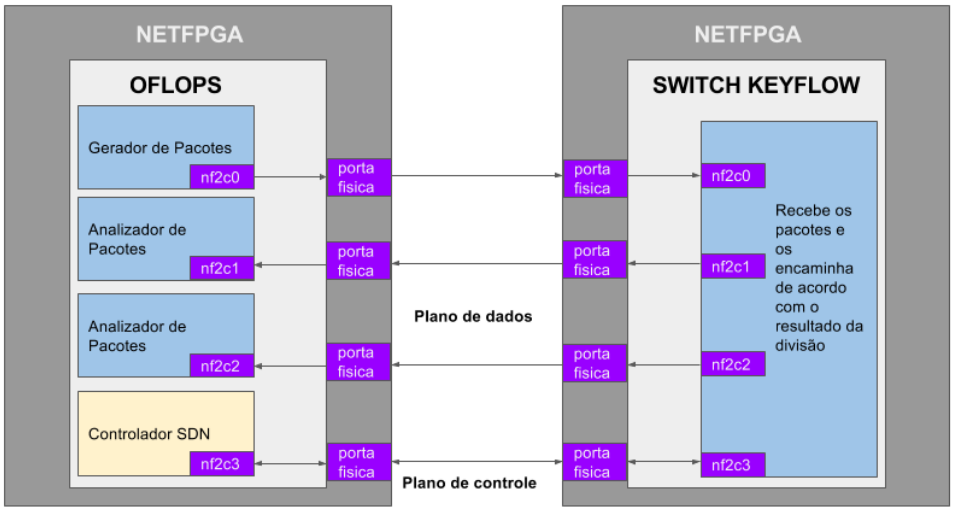

Figura 4.4: Esquemático do teste de medição da latência de encaminhamento do KeyFlow

Para esse teste foram utilizadas 2 NetFPGAs, uma para o OFLOPS outra para o comutador KeyFlow com suas portas interligadas, conforme ilustrado na Figura 4.4. Os pacotes foram gerados e encaminhados pela primeira porta da NetFPGA programada com o OFLOPS chegando na primeira porta da segunda NetFPGA programada com o KeyFlow. Esta, por sua vez, encaminha os pacotes para a porta de saída de acordo com o resto do resultado da divisão do rótulo dos pacotes pela chave, que é definida pelo plano de controle. Os resultados de latência e desvio padrão dessa operação para os diferentes valores de taxa podem ser observados na Figura 4.5.

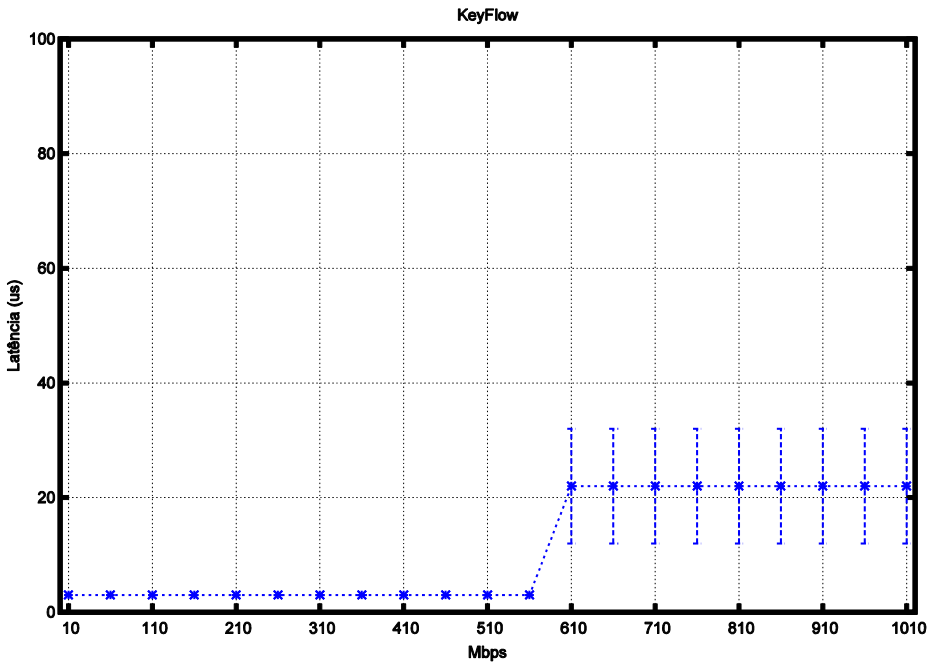

Figura 4.5: Medida de latência – KeyFlow

Observando a Figura 4.5, nota-se um comportamento similar ao da Figura 4.3 Loopback do OFLOPS, os valores de latência permanecem constantes até a taxa de 560Mbps e acima disso há um salto de 19 microssegundos.

Há, porém, um incremento de 3 microssegundos nos valores do KeyFlow em relação a Figura 4.2 para todas as taxas, resultando em uma latência de 3 microssegundos até 560Mbps e 22 microssegundos para taxas maiores que essa. Como trata-se de um incremento constante, pode-se concluir que a operação de divisão necessária para descobrir a porta de saída dos pacotes leva 3 microssegundos para ser executada.

4.2.3 OpenFlow com Apenas a Regra de Encaminhamento

Para os testes do OpenFlow com o menor número de regras possível, foi adicionada apenas a regra de encaminhamento: aquela que é utilizada pelo comutador. Essa regra foi instalada antes do envio dos pacotes para que não fosse adicionado o tempo de instalação da regra aos resultados.

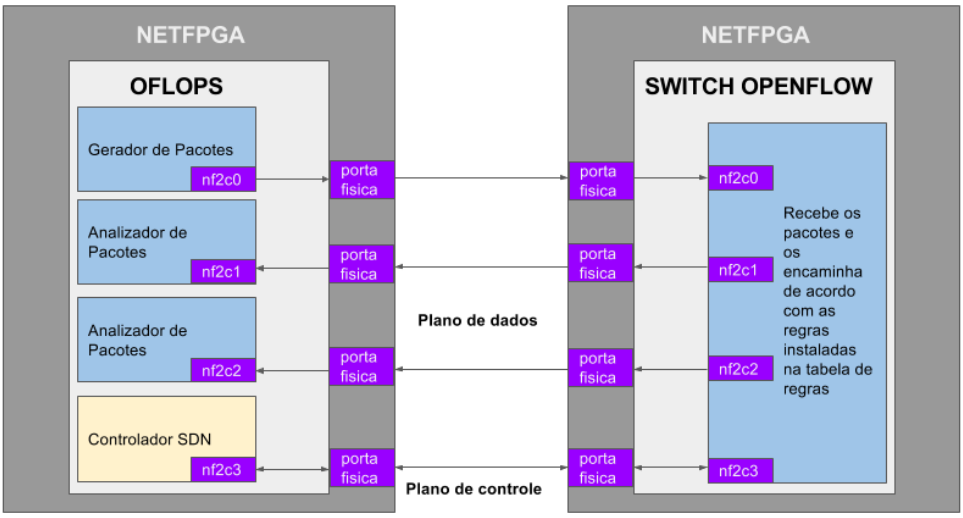

Figura 4.6: Esquemático dos testes de medição da latência de encaminhamento do Openflow

Para este e os demais testes envolvendo o OpenFlow foram utilizadas 2 NetFPGAs, uma para o OFLOPS outra para o comutador OpenFlow com suas portas interligadas, conforme ilustrado na Figura 4.6. Assim como no cenário anterior, os pacotes são gerados e encaminhados pela primeira porta da NetFPGA que está programada com o OFLOPS chegando na primeira porta da segunda NetFPGA que a

partir de agora, é programada com o OpenFlow. Os pacotes são, então, encaminhados de volta para a porta de saída de acordo com as regras instaladas pelo controlador na tabela de regras.

Os resultados de latência e desvio padrão para o teste envolvendo apenas a regra de encaminhamento de acordo com os diferentes valores de taxa podem ser observados na Figura 4.7.

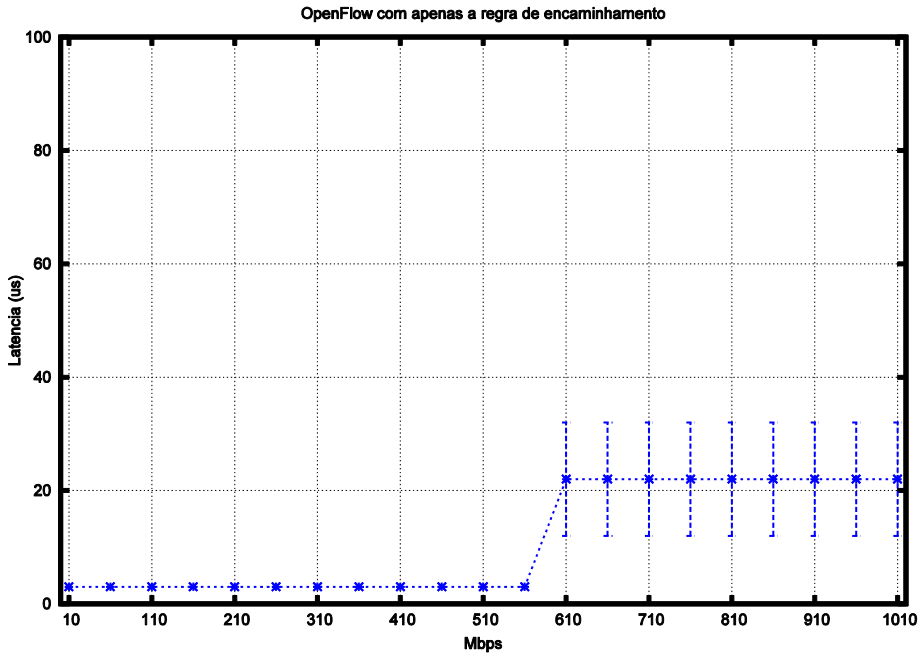

Figura 4.7: Medida de latência - OpenFlow com apenas a regra de encaminhamento

Observando a Figura 4.7, nota-se que se a tabela de regras tiver apenas a regra de encaminhamento, o OpenFlow tem um comportamento similar ao KeyFlow, mantendo a latência para taxas mais baixas em 3 microssegundos e em regime de sobrecarga, em 22 microssegundos. Ou seja, o tempo de consulta de uma tabela contendo apenas a regra a ser utilizada é igual ao tempo necessário para o processo de divisão do KeyFlow.

4.2.4 OpenFlow com Tabelas Cheias

Nas medições realizadas com o OpenFlow com tabelas cheias, foram verificados os valores de latência e desvio padrão para três diferentes posições da regra utilizada pelo comutador para o encaminhamento: quando a regra de encaminhamento está no inicio da tabela cheia, quando a regra de encaminhamento está em uma posição

aleatória no meio da tabela cheia e quando a regra de encaminhamento está no final da tabela cheia.

Uma vez que a tabela de regras do OpenFlow para NetFPGA é subdividida em três tabelas, conforme exposto no capítulo anterior, ao modificarmos a posição da regra, estaremos modificando também a tabela onde a regra está, sendo para os testes: a regra no início da tabela localizada na tabela nf2, a regra no meio da tabela localizada na tabela *hash* e a regra no final da tabela localizada na tabela linear.

O cenário de teste para os testes com a tabela cheia é o mesmo utilizado para o teste com apenas uma regra, ilustrado na Figura 4.6, diferindo apenas no número de regras instaladas e na posição que a regra a ser utilizada ocupa na tabela de fluxos. As tabelas de regras foram preenchidas com regras exatas e curingas.

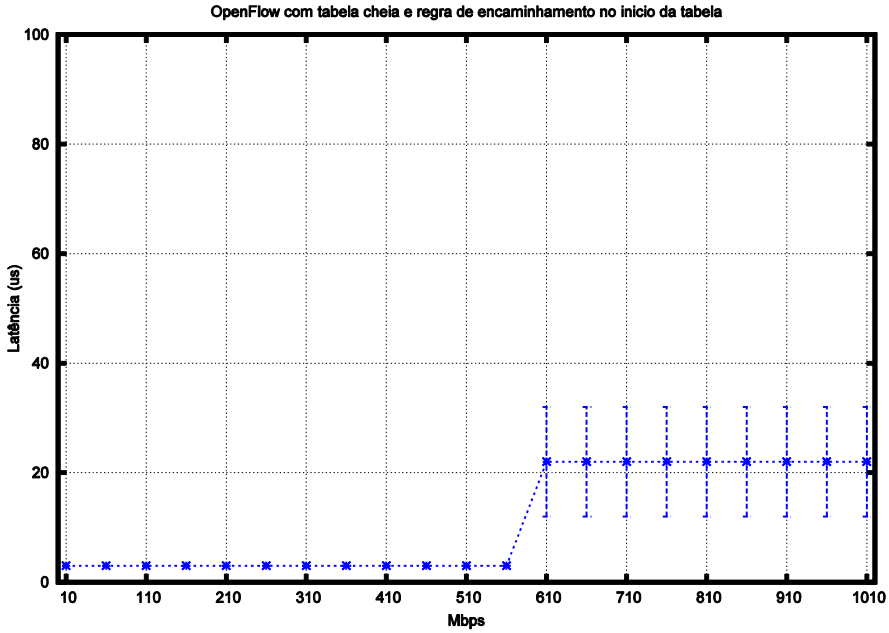

Figura 4.8: Medida de latência - OpenFlow com tabela cheia e regra de encaminhamento no início da tabela

A Figura 4.8 representa os resultados obtidos para a regra de encaminhamento no início da tabela, observa-se que, neste caso, o OpenFlow tem um comportamento semelhante ao apresentado para o caso da tabela estar apenas com esta regra.

Para a regra de encaminhamento no meio da tabela de fluxo, apresentada na Figura 4.9, nota-se que a latência, bem como o desvio padrão, não permanecem constantes e apresentam valores um pouco mais altos que nos casos anteriores. Na latência há um incremento de pelo menos 1 microssegundo para algumas taxas. No desvio padrão, há incrementos substanciais para a faixa de valores em que não há enfileiramento, ou seja, entre 10 Mbps e 560Mbps.

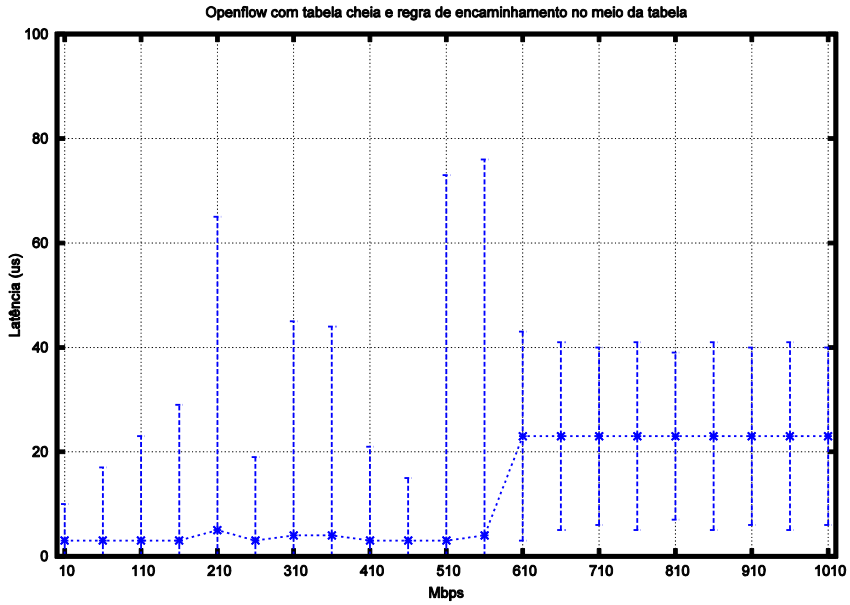

Figura 4.9: Medida de latência - OpenFlow com tabela cheia e regra de encaminhamento no meio da tabela

Em relação aos resultados para a tabela cheia e a regra de encaminhamento localizada no final da tabela, observado na Figura 4.10, nota-se um aumento na escala de aproximadamente 10ˆ6 no valor da latência se comparado aos valores de latência do KeyFlow ou do OpenFlow com tabela vazia. O valor da latência para esses teste varia de 900 milissegundos a 1400 milissegundos.

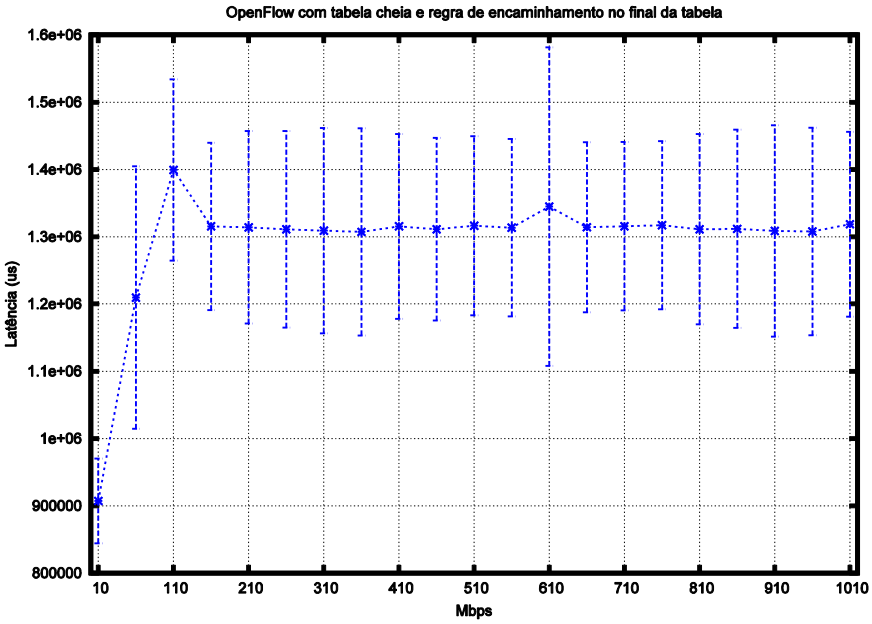

Figura 4.10: Medida de latência - OpenFlow com tabela cheia e regra de encaminhamento no final da tabela

4.3 Comentários Finais

A partir dos resultados obtidos, pode-se concluir que a latência de encaminhamento, bem como o desvio padrão, sofrem maior influência da posição da regra que será utilizada do que do número de regras instaladas. Além disso, diferenças significativas nos valores da latência de comutação ocorrem apenas quando a regra está no final da tabela. Isso ocorre pois, diferente das outras duas tabelas (nf2 e *hash*), a busca na tabela linear é linear, uma vez que essa é uma tabela preferencialmente de regras curingas, ou seja, incompletas. Isso faz com que o controlador não consiga calcular a posição exata da regra que deseja, pois os campos da regra não estão todos preenchidos, e tenha que recorrer a uma busca linear, aumentando o tempo de resposta.

Quando há somente uma regra de encaminhamento ou quando a regra a ser utilizada está localizada no início da tabela, não há uma diferença significativa no RTT do pacote se utilizar um comutador KeyFlow ou se utilizar um comutador OpenFlow. Isso ocorre porque o tempo necessário para realizar a divisão é equivalente ao tempo necessário para executar visitas ao início da tabela no processo de pesquisa. Conforme a tabela de utilização aumenta e a regra a ser utilizada se aproxima do fim da tabela de regras, ou seja, da tabela linear, o comutador KeyFlow supera o comutador OpenFlow.

Capítulo 5 - Engenharia de Tráfego na Camada de Reconfiguração da TRIIIAD

Este capítulo tem como objetivo avaliar os possíveis ganhos da substituição das chaves ópticas, utilizadas na camada híbrida de reconfiguração da TRIIIAD, por NetFPGAs programadas como comutadores OpenFlow. Espera-se que todo o tráfego de trânsito da rede seja eliminado a partir da redução de saltos entre os nós de computação. Além disso, que os fluxos possam ser tratados individualmente, fornecendo maior grau de liberdade a rede e uma possível engenharia de tráfego capaz de desviar fluxos para caminhos alternativos quando necessário.

5.1 Ambiente de testes utilizado

A descrição do ambiente de testes utilizado pode ser encontrada em [12], pois se trata do mesmo ambiente da TRIIIAD, com apenas uma alteração na camada híbrida de reconfiguração: as chaves ópticas foram substituídas por uma NetFPGA de 1Gbps (vide Figura 5.1).

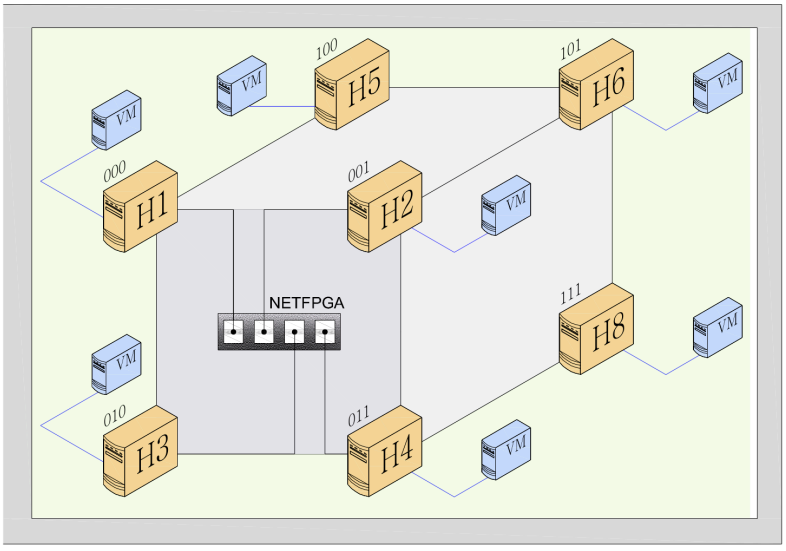

Figura 5.1: Camada híbrida da TRIIIAD com NetFPGA

A NetFPGA foi instalada na interface PCI de um computador com processador Dual Core 3GHz, 4GB de RAM, HD 1TB, com sistema operacional Linux CENTOS

5.11 32*bits* e programada com o protocolo OpenFlow 1.0, controlada remotamente pelo controlador A-SDN da TRIIIAD.

Os testes foram realizados por meio de *scripts* rodando nas maquinas virtuais OpenWRT, que utilizaram o iperf como ferramenta de geração e recepção de tráfego. Nos *hosts* físicos um processo monitor foi utilizado para monitorar os usos de CPU, memória e tráfego. A cada 30 segundos esses valores eram reportados ao controlador.

Para cada teste, foram realizadas 30 repetições, os resultados foram avaliados e processados para gerar os gráficos de resultados, que serão apresentados na Seção 5.2.

5.2 Resultados

Nesta seção são apresentados e discutidos os resultados obtidos nos experimentos realizados na seguinte ordem: NetFPGA operando como chaves ópticas, NetFPGA contendo as regras instaladas de acordo com as máquinas virtuais e, por último, NetFPGA com controle individual dos fluxos.

5.2.1 NetFPGA Operando como Chaves Ópticas

No cenário ilustrado na Figura 5.2, foi implementado o mecanismo de comutação óptica em NetFPGA. O *host* H1 hospeda a VMrx, o *host* H3 hospeda a VM2 e o *host* H4 hospeda a VMtx. O experimento inicia-se na configuração apresentada na Figura 5.2 (a), com as regras de fluxos instaladas de modo a representar as chaves ópticas em estado “barra” e VMtx enviando dois fluxos para H3, sendo o menor para VM2 (tracejado verde) e o maior para VMrx (tracejado azul) em H1. Após 40s as regras de fluxos são modificadas para que represente a comutação com as chaves em estado “cruz”, alterando para o cenário da Figura 5.2 (b), de forma que H1 passa a encaminhar o fluxo menor para VM2 em H3. Reduzindo assim, o tráfego de trânsito total da rede. Os fluxos são acompanhados por mais 30s e então, todo o processo é finalizado e restaurado, permitindo que o experimento seja repetido.

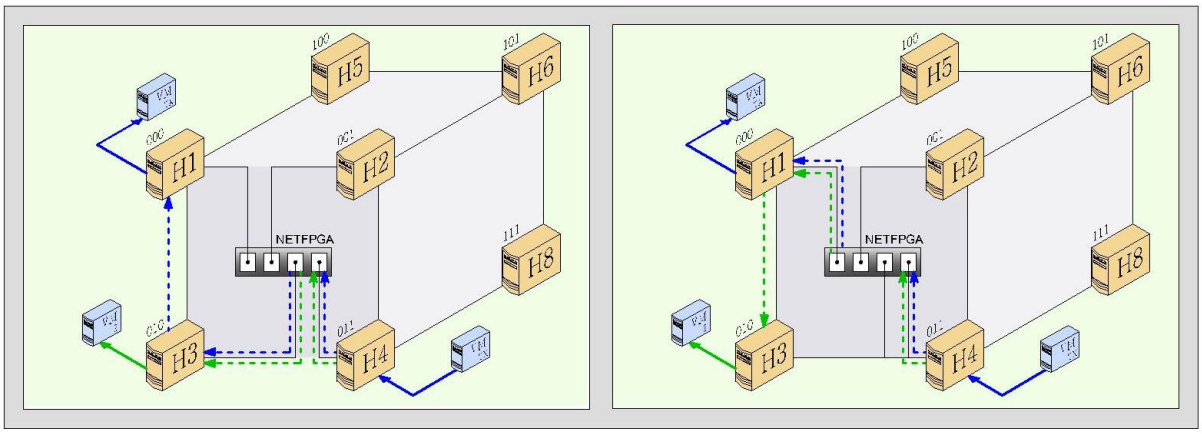

Figura 5.2: NetFPGA operando como chaves ópticas: (a) regras instaladas para operação em estado barra com tráfego de trânsito maior passando por H3; (b) regras instaladas para operação em estado cruz com tráfego de trânsito menor passando por H1

O experimento foi conduzido com o fluxo maior a 350 Mbps e o fluxo menor a 100 Mbps. Esses valores foram escolhidos para não saturar os *hosts*, tanto intermediários quanto os de origem e destino. Foram realizadas trinta repetições e os dados de CPU e tráfego de rede foram coletados e plotados no gráfico da Figura 5.3.

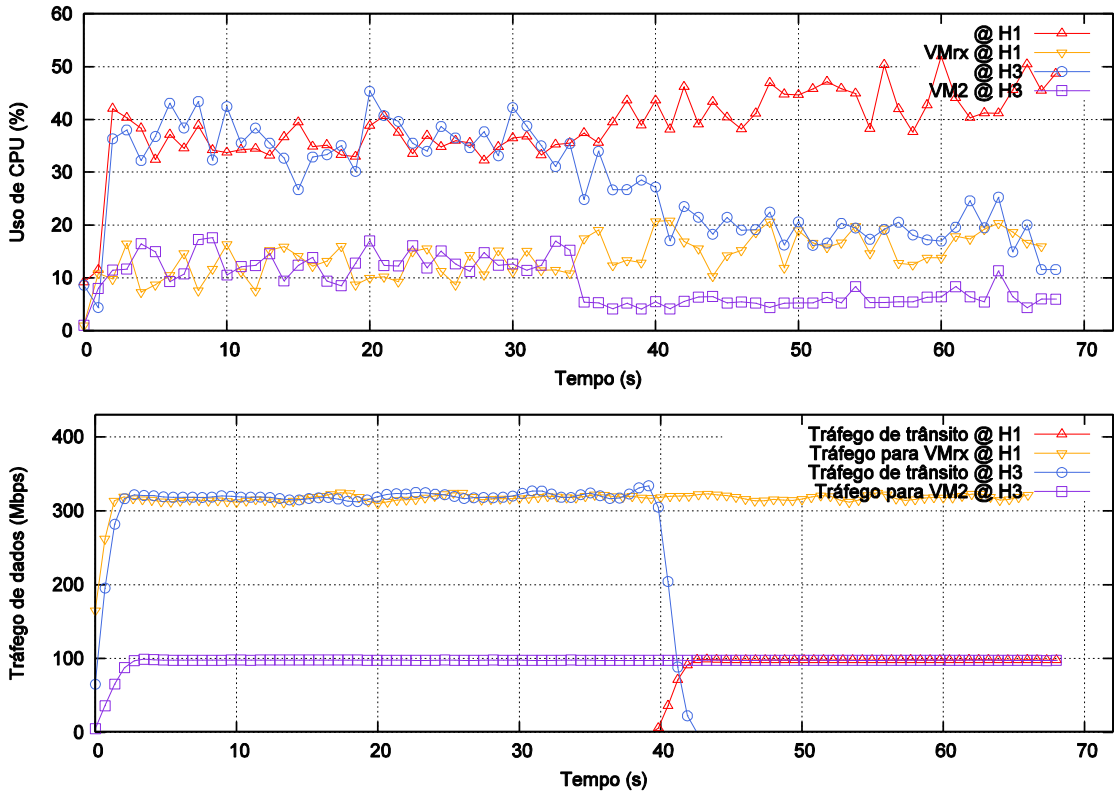

Figura 5.3: Impacto da NetFPGA operando como chaves ópticas no tráfego da rede

O primeiro gráfico da Figura 5.3 apresenta o uso de CPU de H1 e H3 em torno de 35% a 40% até os 40 segundos. H1 está utilizando a CPU apenas para tratar os pacotes encaminhados para a VM que ele hospeda e H3, além de estar tratando os pacotes para a VM que hospeda, ainda encaminha pacotes do fluxo de 350 Mbps para H1.A partir dos 40 segundos, que é quando as regras são modificadas para representar a forma que representa a comutação óptica em estado barra, o uso de CPU de H1 aumenta e H3 diminui. No entanto, o aumento de H1, de mais ou menos 10%, é menor que a diminuição de H3, que é quase 20%. Isso ocorre pois o tráfego de trânsito que passa por H1, de apenas 100 Mbps, é menor que o que passava anteriormente em H3, que era de 350 Mbps. Já o segundo gráfico, mostra que a partir dos 40 segundos há a diminuição do tráfego de trânsito da rede e um ligeiro aumento no tráfego entregue às VMs.

Esse experimento mostra que a NetFPGA pode operar do mesmo modo que as chaves ópticas, ou seja, realizando encaminhamento de pacotes sem muita complexidade redistribuindo os fluxos de modo a reduzir o tráfego de trânsito total da rede e, consequentemente o uso médio de CPU. No entanto, apesar da redução do tráfego de trânsito, não há possiblidade de eliminá-lo totalmente.

5.2.2 NetFPGA Contendo Regras Instaladas a Partir do MAC das Máquinas Virtuais

Para demonstrar como a NetFPGA pode eliminar o tráfego de trânsito total da rede, foi utilizado o mesmo cenário do experimento anterior, conforme ilustrado na Figura 5.4. O *host* H1 hospeda VMrx, o *host* H3 hospeda VM2 e o *host* H4 hospeda VMtx que envia dois fluxos, um fluxo maior para H1, de 350 Mbps e em fluxo menor para H3, de 100 Mbps. O experimento se inicia e a NetFPGA encaminha os fluxos diretamente para os seus destinos, ou seja o fluxo maior vai diretamente para H1 e o menor vai diretamente para H3, isso é feito por meio da instalação de regras curingas contendo no campo MAC destino o MAC de destino correspondente a VM e a porta de saída é preenchida com a porta de saída da NetFPGA correspondente à porta mais próxima do *host* onde está hospedada a VM.

Os fluxos foram acompanhados por 70 segundos, assim como no experimento anterior. Então todo o processo foi finalizado e o cenário restaurado permitindo a repetição do experimento.

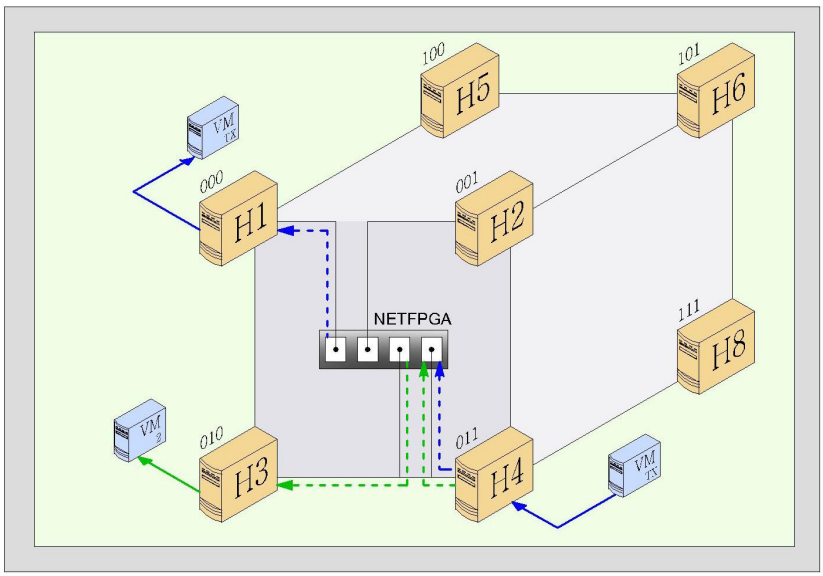

Figura 5.4: NetFPGA contendo regras instaladas a partir do MAC das máquinas virtuais

Foram realizadas trinta repetições e os dados de CPU e tráfego de rede foram coletados e plotados no gráfico da Figura 5.5.

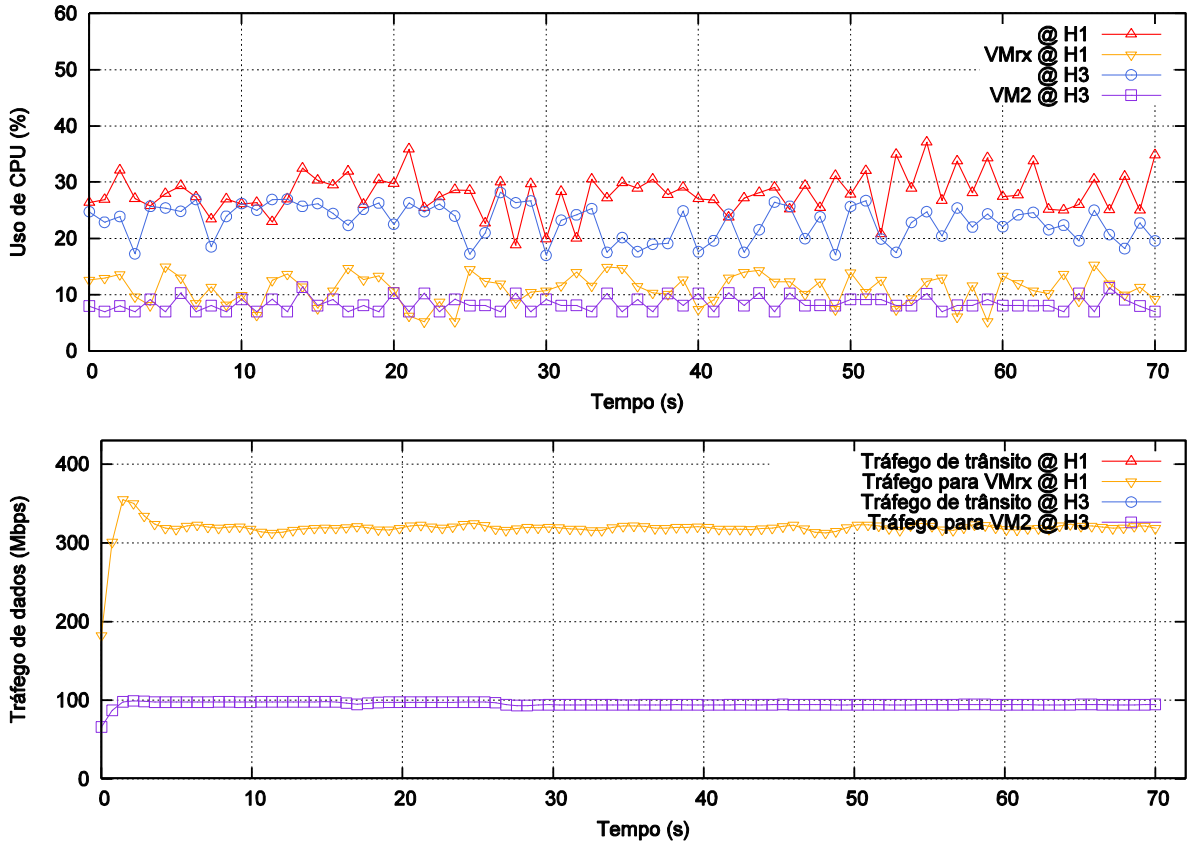

Figura 5.5: Impacto da NetFPGA contendo regras instaladas a partir do MAC das máquinas virtuais

Os gráficos da Figura 5.5 apresentam o uso de CPU, bem como o tráfego da rede constantes durante todo o experimento. O uso de CPU de H1 e H3 permanece em torno de 25% a 30%, esses *hosts* estão utilizando a CPU apenas para tratar os pacotes encaminhados para as VMs que hospedam. O tráfego entregue aos *hosts* permanece constante, uma vez que percorre sempre o mesmo caminho até o destino e não há tráfego de trânsito na rede pois os *hosts* de origem e destino estão ligados diretamente em ambos os casos.

Esse experimento mostra que a NetFPGA pode ser utilizada para encaminhar os fluxos, de modo a eliminar o tráfego de trânsito no plano de chaveamento e, consequentemente diminuir o uso médio de CPU. Essa solução, no entanto, limita o número máximo de VMs que podem ser instaladas nos *hosts* ligados à NetFPGA. A soma da quantidade de VMs deve ser menor ou igual à capacidade da NetFPGA de armazenar regras curingas, uma vez que cada VM possui uma regra curinga a ela associada. O número máximo de regras curingas, conforme dito na seção 3.1.2, é apenas 124 (24 regras curingas na tabela nf2 e 100 regras curingas na tabela linear), dando um número máximo de 124 VMs por face do cubo. Entretanto, esse problema pode ser resolvido na versão 1.3 do OpenFlow, que fornece a possibilidade da instalação de regras com máscaras nos números MAC, ou seja, o MAC de destino pode assumir a seguinte forma: “eth_dst = 00:00:00:01:*”. Assim, as regras curingas poderiam ser instaladas com o endereço MAC de destino identificando apenas o *host* em que a VM está localizada e não a VM em si, o que resultaria em apenas quatro regras de fluxo por plano de chaveamento independente da quantidade de VMs instaladas nos *hosts*.

5.2.3 NetFPGA com Controle Individual de Fluxos

Para demonstrar como a NetFPGA pode auxiliar a rede em um processo de engenharia de tráfego foi montado o cenário da Figura 5.6, em que cinco fluxos diferentes partindo das VMs de H1 e H3, chegam às VMs de H4. Esse processo faz com que a interface de rede de H4 que está ligada a NetFPGA fique sobrecarregada, fazendo assim com que o controlador tenha que desviar alguns fluxos pelo *host* H2 para que cheguem ao *host* H4 por uma interface de rede diferente. Porém, após 180 segundos do início do experimento, o *host* H2 começa a ter tráfego entre suas VMs, fazendo com que o tráfego de trânsito que por ele passa seja reduzido.

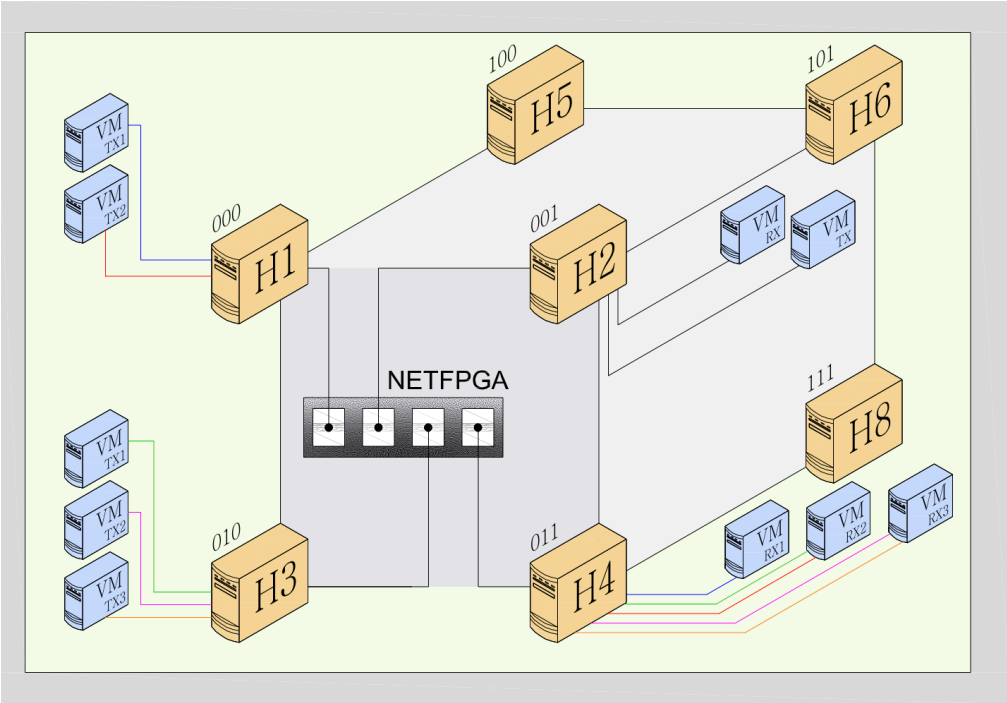

Figura 5.6: Cenário de testes para a NetFPGA com controle individual de fluxos

A Figura 5.7 mostra uma linha do tempo dos fluxos. Os fluxos em azul representam os que foram gerados pelas máquinas virtuais hospedadas no *host* H1, o fluxos em cinza, representam os gerados pelas máquinas virtuais do *host* H4 e o fluxo verde, representa o tráfego interno no *host* H2.

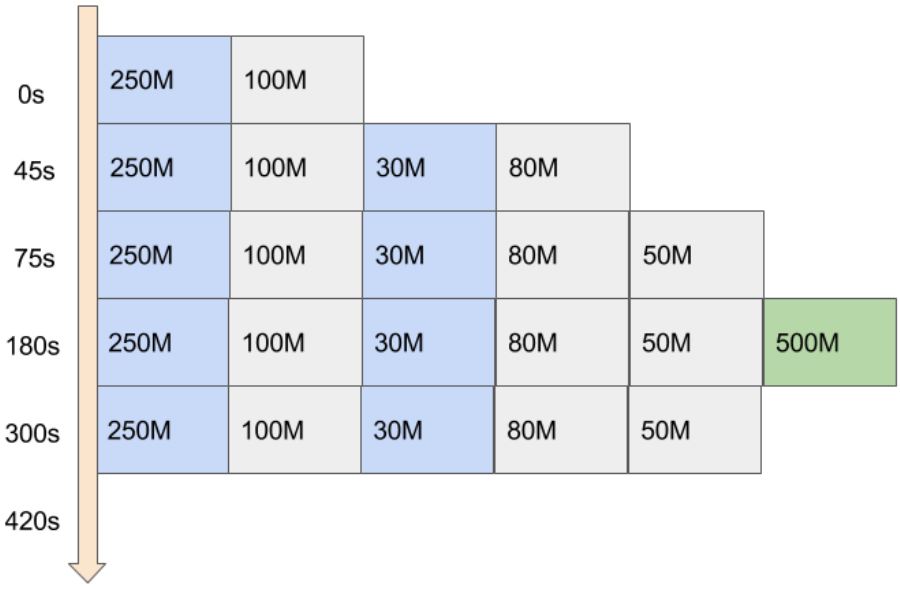

Figura 5.7: Linha do tempo de fluxos

O *host* H1 hospeda as VM1TX1 e VM1TX2, o *host* H2 hospeda as VM2RX e VM2TX, o *host* H3 hospeda as VM3TX1, VM3TX2 e VM3TX3 e o *host* H4 hospeda as VM4RX1, VM4RX2 e VM4RX3.

No inicio do experimento, a VM1TX1 e a VM3TX1 iniciam fluxos para a VM4RX1 e VM4RX2, respectivamente. O fluxo da VM1TX1 para a VM4RX1 é de 250 Mbps e o fluxo da VM3TX1 para a VM4RX2 é igual a 100 Mbps. Aos 45 segundos, dois novos fluxos são iniciados, um fluxo da VM1TX2 para a VM4RX2 de 30 Mbps e um fluxo da VM3TX2 para a VM4RX3 de 80 Mbps. Aos 75 segundos é iniciado um novo fluxo de 50 Mbps partindo da VM3TX3 para a VM4RX3. Por fim, é iniciado um fluxo de 500 Mbps da VM2TX para a VM2RX.

Para esse cenário descrito, foram realizados três tipos de testes: desviando prioritariamente fluxos menores, desviando prioritariamente fluxos maiores e aleatório.

5.2.3.1 Fluxos menores desviados prioritariamente

O comportamento da topologia para a engenharia de tráfego desviando prioritariamente fluxos menores pode ser observado na Figura 5.8.

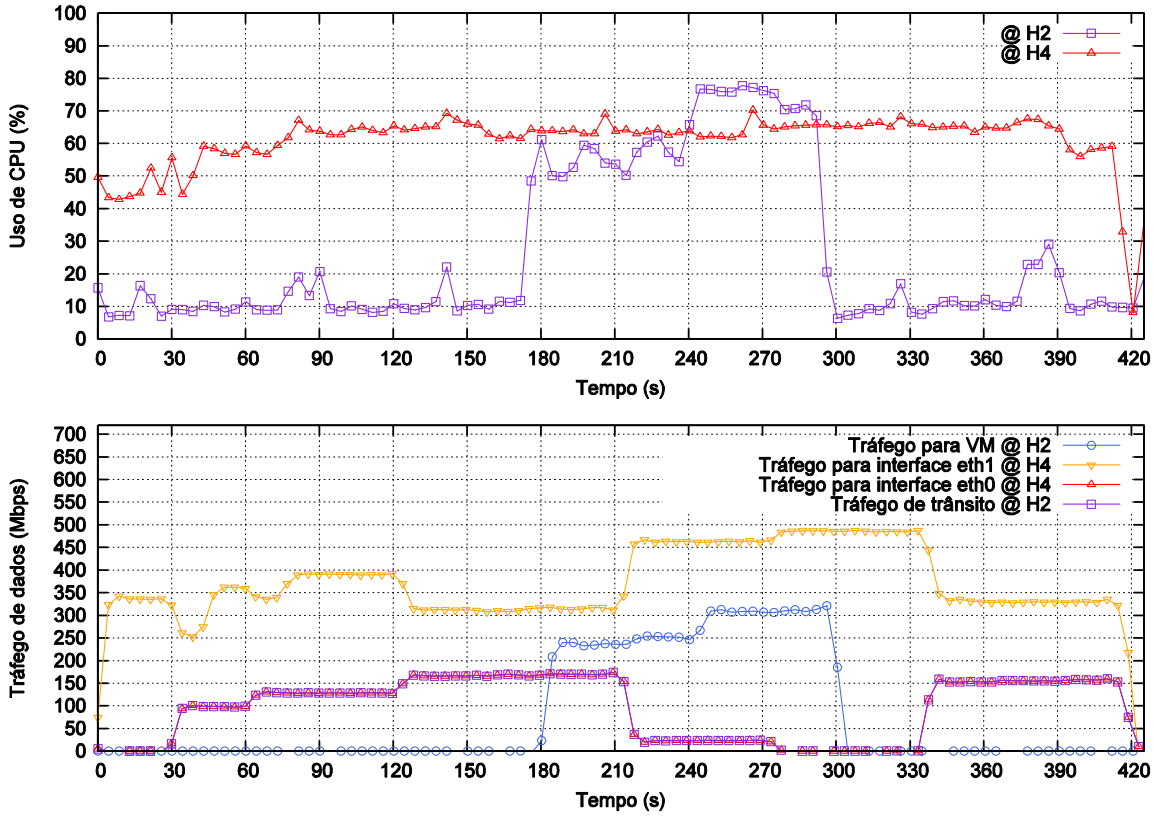

Figura 5.8: Engenharia de tráfego desviando prioritariamente fluxos menores com limite fixo de tráfego de trânsito de 200 Mbps e intervalo de verificação de carga média de 30s

Referente ao gráfico de uso de CPU da Figura 5.8, é importante observar que o uso de CPU de H4 é maior que de H2 em grande parte do tempo, pois H4 é o *host* que recebe todo o tráfego das VMs de H1 e H3 e tem que encaminhá-lo às respectivas VMs nele hospedadas. De 180s a 300s há um aumento significativo no uso de CPU de H2, isso ocorre devido ao tráfego interno gerado pelas VMs VM2TX e VM2RX.

A fim de facilitar o entendimento do gráfico de tráfego de dados da Figura 5.6, sua explicação será realizada dividindo o tempo do experimento nos intervalos: 0s a 45s; 45s a 75s; 75s a 180s; 180s a 270s; 270s a 330s; e 330s a 420s.

No instante 0s iniciam-se dois fluxos, um de 250 Mbps partindo de VM1TX1 para VM4RX1 e um de 100 Mbps partindo de VM3TX1 para VM4RX2, conforme mostra a Figura 5.9 (a). A soma desses dois fluxos excede a taxa máxima de fluxo por interface, que é fixada em 300 Mbps. Então, aos 30s, a porta de saída da regra do fluxo de 100 Mbps é modificada e ele é desviado para um caminho alternativo, que passa pelo *host* H2 que o encaminha ao *host* H4, como mostra a Figura 5.9 (b).

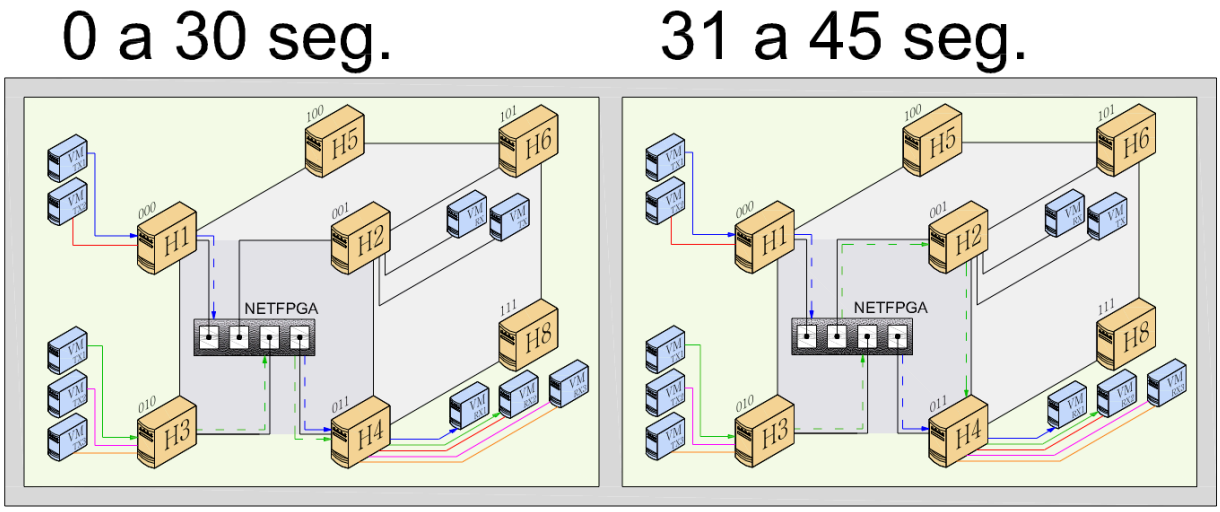

Figura 5.9: Intervalo de 0s a 45s (a) fluxos de VM1TX1 e VM3TX1 iniciam (b) fluxo de VM3TX1 é desviado para o *host* H2

Aos 45s são iniciados dois novos fluxos, um de 30 Mbps partindo de VM1TX2 para VM4RX2 e um de 80 Mbps partindo de VM3TX2 para VM4RX3, conforme mostra a Figura 5.10 (a). Mais uma vez a soma dos fluxos excede a taxa máxima de fluxo por interface. Então, aos 60s, a porta de saída da regra do fluxo de 30 Mbps é modificada e ele é desviado para um caminho alternativo, que passa pelo *host* H2 que o encaminha ao *host* H4, como mostra a Figura 5.10 (b).

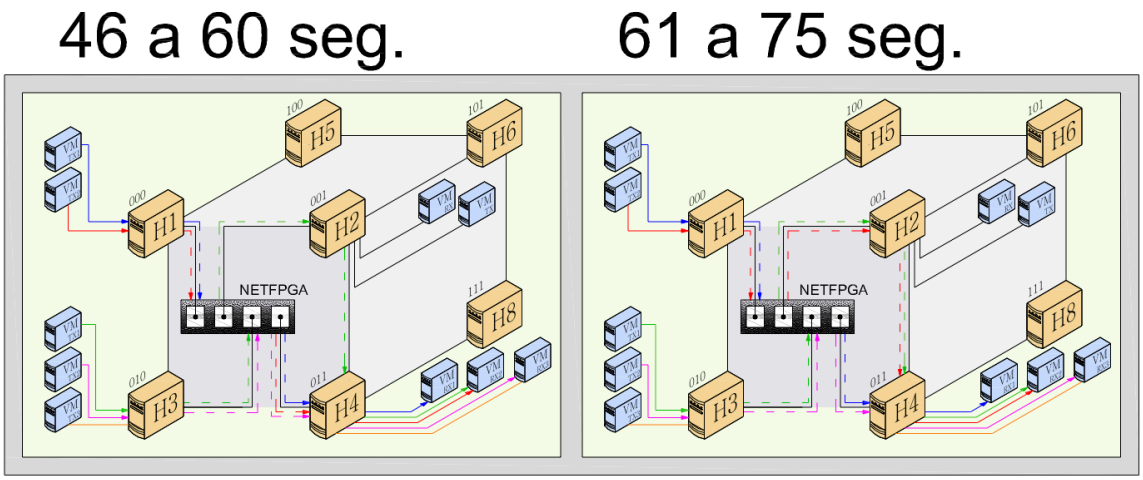

Figura 5.10: Intervalo de 45s a 75s (a) fluxos de VM1TX2 e VM3TX2 iniciam (b) fluxo de VM1TX2 é desviado para o *host* H2

Mesmo desviando o fluxo de 30 Mbps a soma dos fluxos restantes ainda excede o taxa máxima de fluxo por *interface* (250 + 80 = 330), porém o fluxo de 80 Mbps não pode ser desviado para o *host* H2 pois se fosse desviado excederia o limite máximo de tráfico de transito da rede, que é limitado em 200 Mbps.

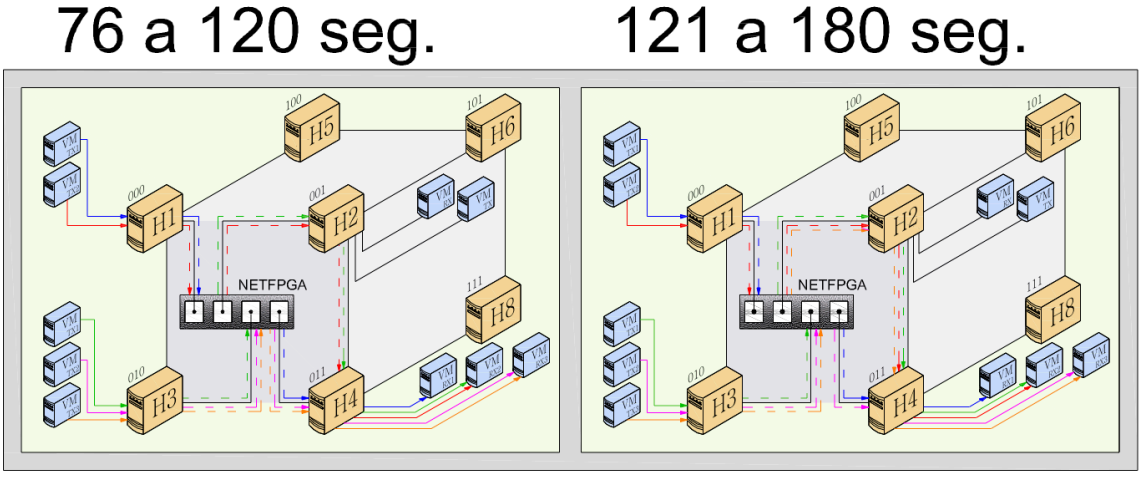

Figura 5.11: Intervalo de 75s a 180s (a) fluxo de VM3TX3 inicia (b) fluxo de VM3TX3 é desviado para o *host* H2

Aos 75s é iniciado um novo fluxo de 50 Mbps partindo de VM3TX3 para VM4RX3, conforme mostra a Figura 5.11 (a). A soma dos fluxos excede a taxa máxima de fluxo por interface. Então, uma vez que o fluxo de 50 Mbps somado aos outros dois fluxos já desviados (100 Mbps e 30 Mbps) é menor que o limite de tráfico de trânsito na rede (200 Mbps), aos 120s, a porta de saída da regra do fluxo de 50 Mbps é modificada

e ele é desviado para um caminho alternativo, que passa pelo *host* H2 que o encaminha ao *host* H4, como mostra a Figura 5.11 (b).

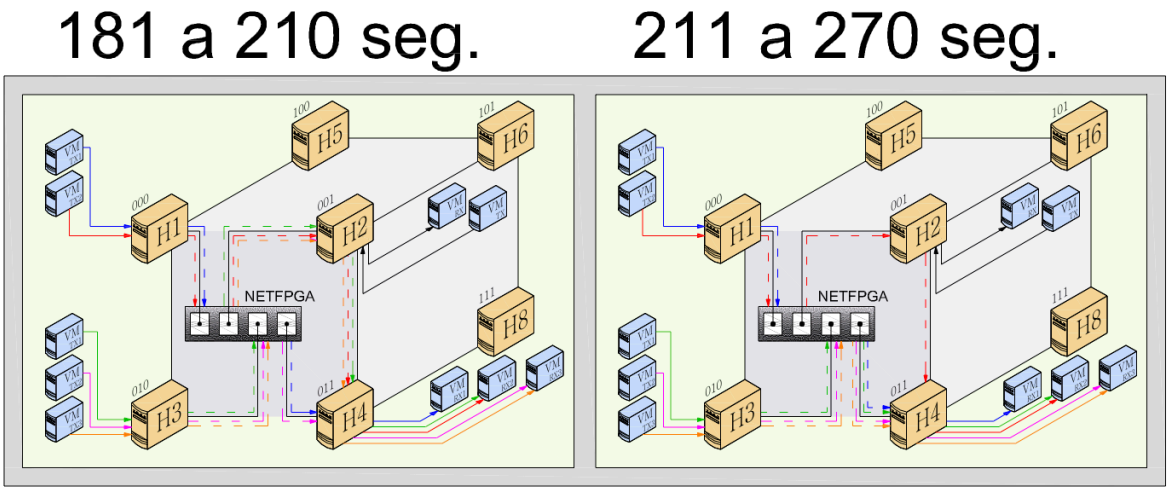

Figura 5.12: Intervalo de 180s a 270s (a) inicia o fluxo de VM2TX para VM2RX (b) fluxo de VM3TX1 e VM3TX3 são desviados para a rota original

Como mostra a Figura 5.12 (a), aos 181s é iniciado um fluxo de 500 Mbps de VM2TX para VM2RX sobrecarregando o *host* H2. Aos 210s, quando o controlador verifica os valores de CPU e tráfego dos *hosts* e nota que H2 está com uso de CPU acima de 60%, as portas de saída dos fluxos VM3TX1 (100 Mbps) e VM3TX3 (50 Mbps) são modificadas para que estes voltem a suas rotas originais, fazendo com que somente o fluxo de VM1TX2 (30 Mbps) permaneça passando por H2 conforme ilustrado na Figura 5.12 (b).

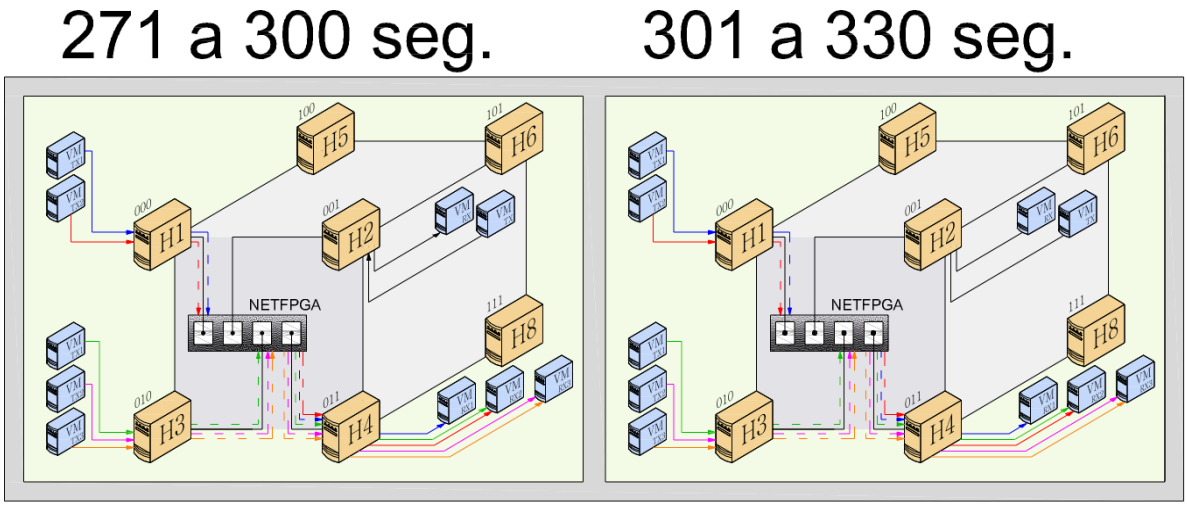

Figura 5.13: Intervalo de 270s a 330s (a) o fluxo de VM1TX2 é desviado para a rota original (b) termina o tráfego entre VM2TX e VM2RX

Conforme mostrado na Figura 5.13 (a), aos 271s o fluxo de VM1TX2 também é desviado de volta, uma vez que o uso de CPU do *host* H2 continua acima dos 60%. Aos 300s, termina o tráfego entre VM2TX e VM2RX, como mostrado na Figura 5.13 (b).

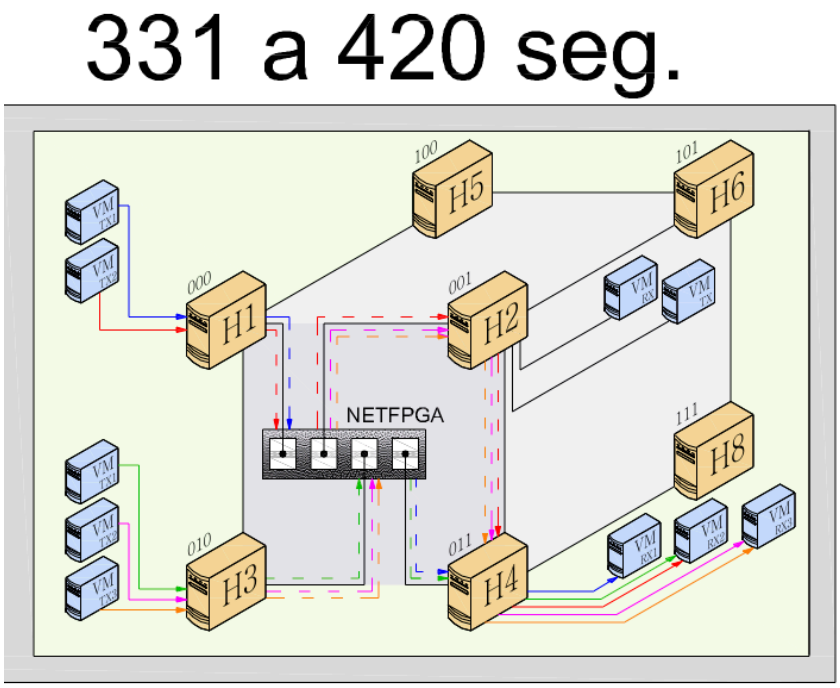

Figura 5.14: Intervalo de 330 a 420 – menores fluxos desviados para o *host* H2

Após o término do tráfego interno do *host* H2, este já pode voltar a receber tráfego de trânsito. Então, conforme mostra a Figura 5.14 aos 331s, os menores fluxos, ou seja VM1TX2 (30 Mbps), VM3TX2 (80 Mpbs) e VM3TX3 (50 Mpbs), têm suas regras modificadas com a porta de saída alterada e são desviados para o *host* H2 fazendo assim com que haja uma melhor distribuição do tráfego que chega às interfaces do *host* H4.

5.2.3.2 Fluxos maiores desviados prioritariamente

O comportamento da topologia para a engenharia de tráfego desviando prioritariamente fluxos maiores pode ser observado na Figura 5.15.

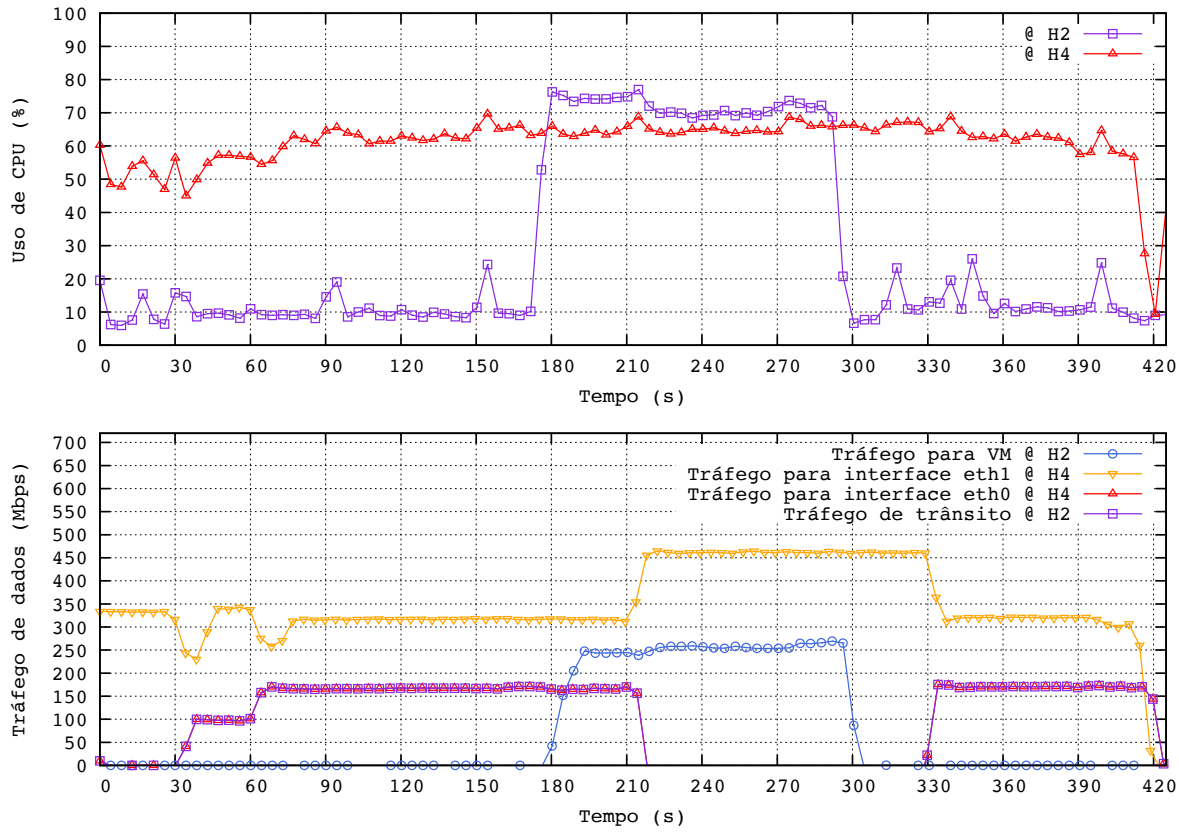

Figura 5.15: Engenharia de tráfego desviando prioritariamente fluxos maiores com limite fixo de tráfego de trânsito de 200 Mbps e intervalo de verificação de carga média de 30s

Assim como ocorre para o caso anterior no gráfico de uso de CPU da Figura 5.14, observa-se que o uso de CPU de H4 é maior que de H2 em grande parte do tempo, pois H4 é o *host* que recebe todo o tráfego das VMs de H1 e H3 e tem que encaminhá-lo às respectivas VMs nele hospedadas. De 180s a 300s há um aumento significativo no uso de CPU de H2, isso ocorre devido ao tráfego interno gerado pelas VMs VM2TX e VM2RX.

Para facilitar o entendimento do gráfico de tráfego de dados da Figura 5.15, sua explicação será realizada dividindo o tempo do experimento nos intervalos: 0s a 45s; 45s a 75s; 75s a 210s; 210s a 300s; e 300s a 420s.

No intervalo de 0s a 30s, iniciam-se dois fluxos, um de 250 Mbps partindo de VM1TX1 para VM4RX1 e um de 100 Mbps partindo de VM3TX1 para VM4RX2, conforme mostra a Figura 5.16 (a). A soma desses dois fluxos excede a taxa máxima de fluxo por interface, que é fixada em 300 Mbps. Então, aos 30s, a porta de saída da regra do fluxo de 100 Mbps é modificada e ele é desviado para um caminho alternativo, que passa pelo *host* H2 que o encaminha ao *host* H4, como mostra a Figura 5.16 (b).

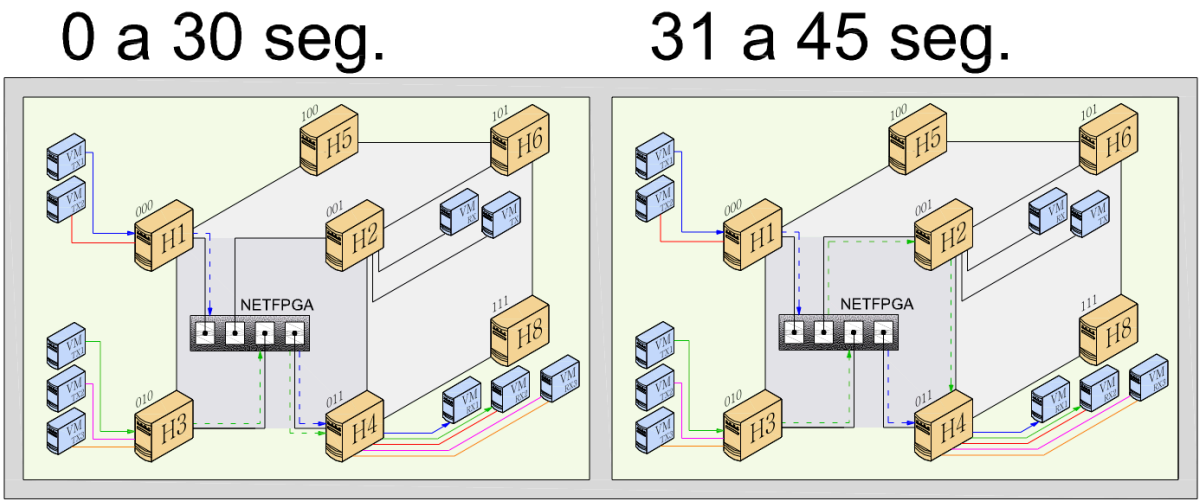

Figura 5.16: Intervalo de 0s a 45s (a) fluxos de VM1TX1 e VM3TX1 iniciam (b) fluxo de VM3TX1 é desviado para o *host* H2

Apesar do fluxo maior ser o fluxo de VM1TX1, ele não pode ser desviado pois ultrapassa o limite máximo de tráfego de trânsito. Por isso, o fluxo desviado foi o de VM3TX1.

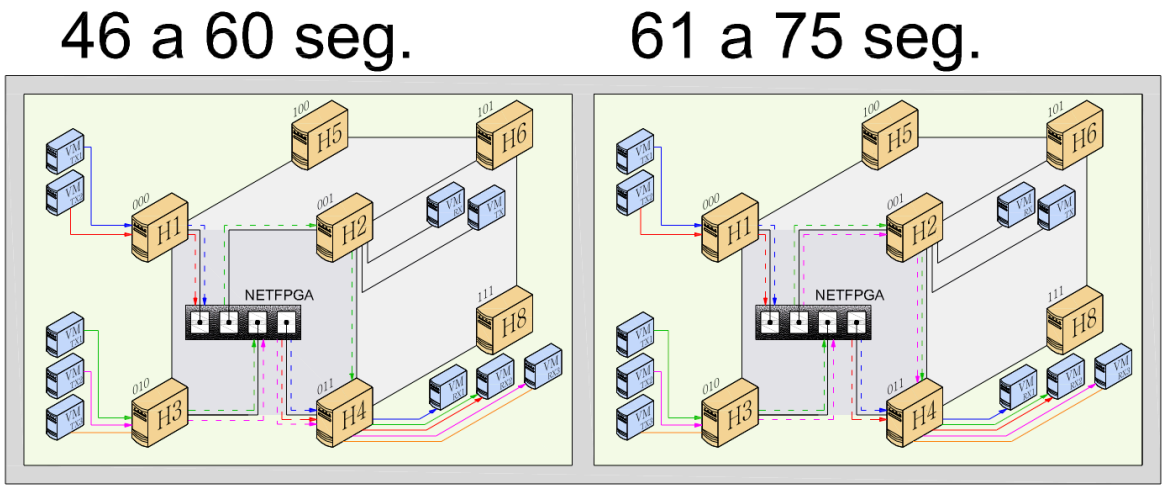

Figura 5.17: Intervalo de 45s a 75s (a) fluxos de VM1TX2 e VM3TX2 iniciam (b) fluxo de VM3TX2 é desviado para o *host* H2

Aos 45s são iniciados dois novos fluxos, um de 30 Mbps partindo de VM1TX2 para VM4RX2 e um de 80 Mbps partindo de VM3TX2 para VM4RX3, conforme mostra a Figura 5.17 (a). Mais uma vez a soma dos fluxos excede a taxa máxima de fluxo por interface. Então, aos 60s, a porta de saída da regra do fluxo de 80 Mbps é modificada e ele é desviado para um caminho alternativo, que passa pelo *host* H2 que o encaminha ao *host* H4, como mostra a Figura 5.17 (b). Como a soma dos fluxos já

desviados é igual a 180 Mbps, não há possibilidade de desviar o fluxo de 30 Mbps pois iria ultrapassar o valor limite de tráfego de trânsito.

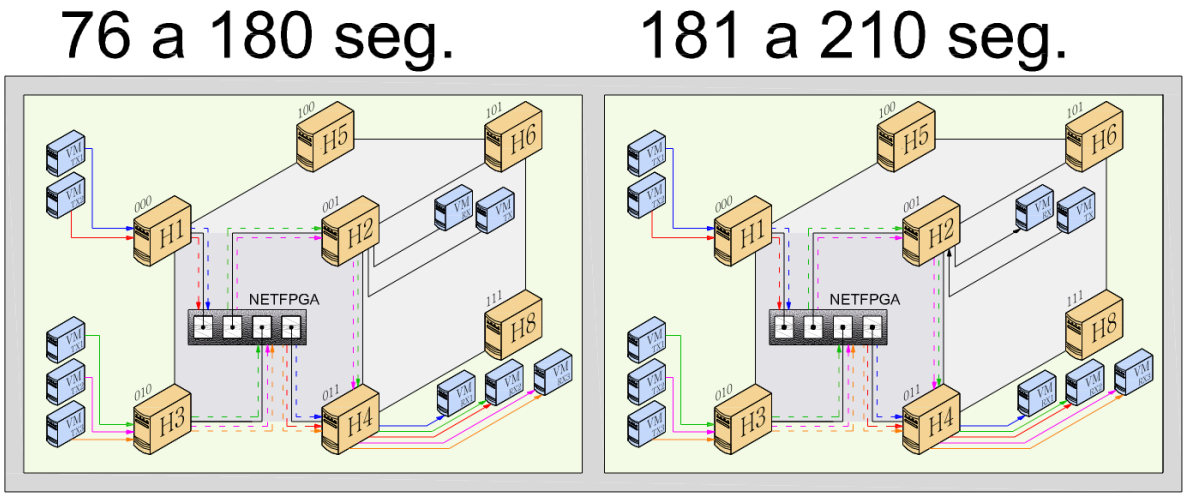

Figura 5.18: Intervalo de 75s a 180s (a) fluxo de VM3TX3 inicia (b) inicia o fluxo de VM2TX para VM2RX

Aos 75s é iniciado um novo fluxo de 50 Mbps partindo de VM3TX3 para VM4RX3, conforme mostra a Figura 5.18 (a). No entanto, esse fluxo não pode ser desviado, pois iria ultrapassar o valor máximo do limite de tráfego de trânsito permitido. Aos 180s, como mostra a Figura 5.18 (b), é iniciado um fluxo de 500 Mbps de VM2TX para VM2RX sobrecarregando o *host* H2.

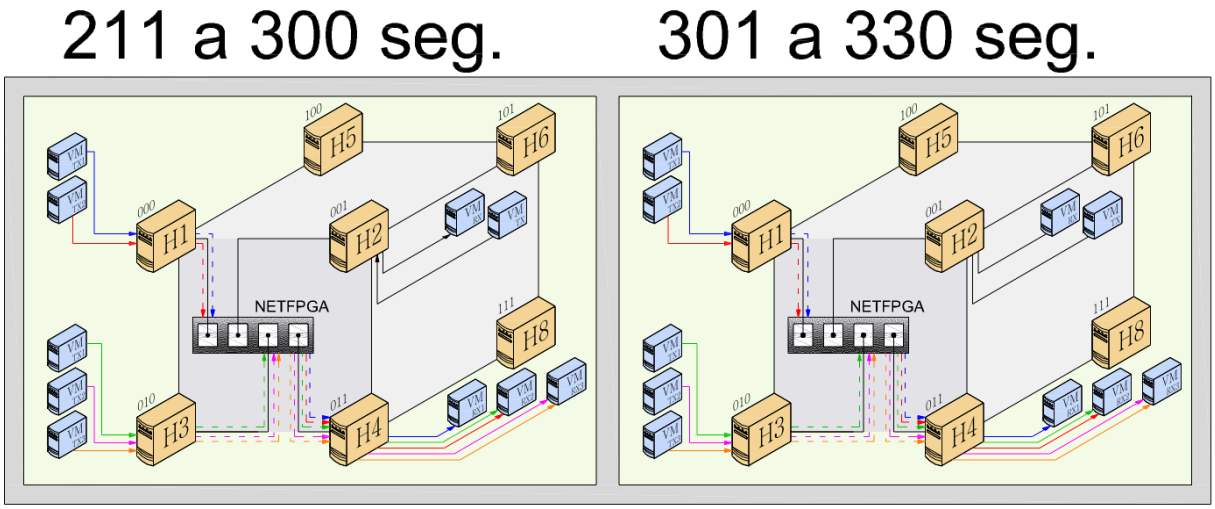

Figura 5.19: Intervalo de 210s a 330s (a) os fluxos de VM3TX1 e VM3TX2 são desviados para a rota original (b) termina o tráfego entre VM2TX e VM2RX

Aos 210s, quando o controlador verifica os valores de CPU e tráfego dos *hosts* e nota que H2 está com uso de CPU acima de 60%, as portas de saída dos dois fluxos

VM3TX1 (100 Mbps) e VM3TX2 (80 Mbps) são modificadas para que estes voltem a suas rotas originais, fazendo com que o *host* H2 fique sem tráfego de trânsito conforme ilustrado na Figura 5.19 (a). Conforme mostra a Figura 5.19 (b), aos 300s o tráfego de VM2TX para VM2RX termina.

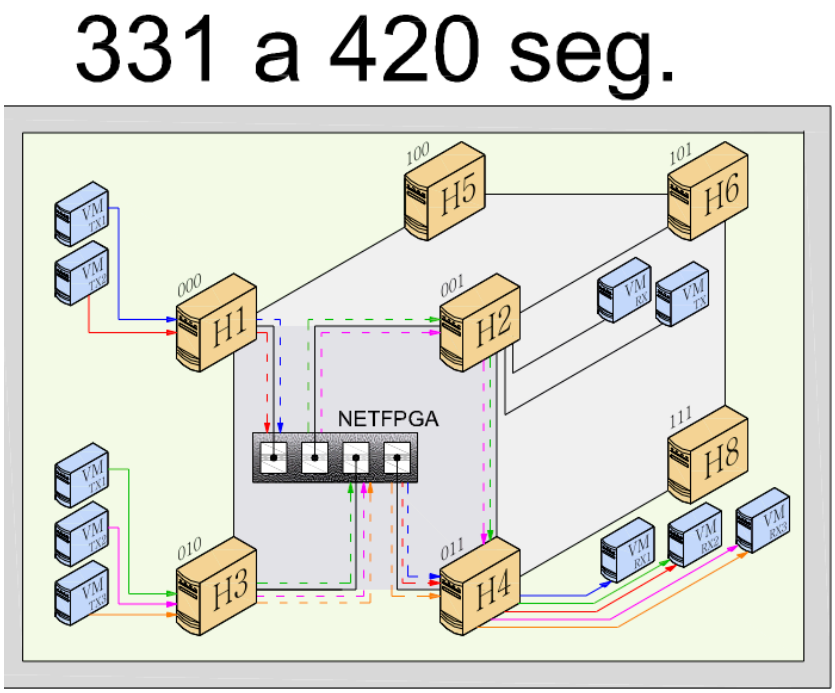

Figura 5.20: Intervalo de 330 a 420 – maiores fluxos desviados para o *host* H2

Finalmente, após o término do tráfego interno do *host* H2, este já pode voltar a receber tráfego de trânsito. Então, conforme mostra a Figura 5.20 aos 330s, os maiores fluxos, ou seja VM3TX1 (100 Mbps) e VM3TX2 (80 Mpbs), têm suas regras modificadas com a porta de saída alterada e são desviados para o *host* H2 fazendo assim com que haja uma melhor distribuição do tráfego que chega às interfaces do *host* H4.

5.2.3.3 Aleatório

Os testes com desvio de fluxos aleatórios foram executados apenas uma vez para a coleta de resultados. Dentre os resultados obtidos, tem-se os das Figuras 5.21 e 5.22.

Para os dois casos, assim como nos anteriores, o uso de CPU do *host* H4 é, em quase todos os intervalos, maior que do *host* H2. Isso ocorre pois H4 é o *host* que recebe todo o tráfego das VMs de H1 e H3 e tem que encaminhá-lo às respectivas VMs

nele hospedadas. De 180s a 300s há um aumento significativo no uso de CPU de H2, isso ocorre devido ao tráfego interno gerado pelas VMs VM2TX e VM2RX.

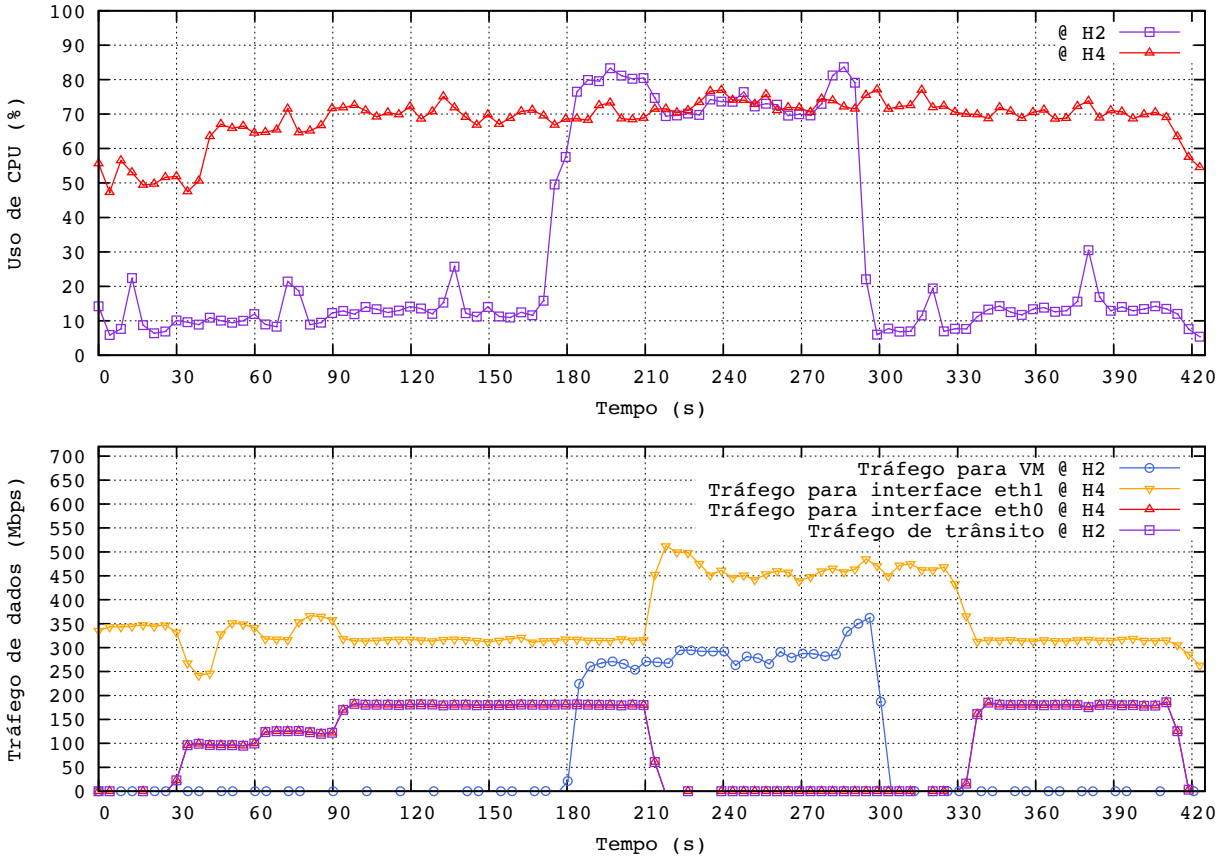

Figura 5.21: Engenharia de tráfego desviando aleatoriamente os fluxos com limite fixo de tráfego de trânsito de 200 Mbps e intervalo de verificação de carga média de 30s

No gráfico de tráfego de dados da Figura 5.21, o fluxo de 100 Mbps começa sendo desviado aos 30s, aos 60s é desviado o fluxo de 30 Mbps e aos 90s é desviado o fluxo de 50 Mbps. Esses fluxos permanecem passando pelo *host* H2 até 210s, quando são desviados de volta a rota de origem. O tráfego interno do *host* H2 vai de 180s a 300s e aos 330s, os fluxos de H4 voltam a ser desviados para H2. São desviados para H2 os mesmos fluxos desviados inicialmente, 100 Mbps, 30 Mbps e 50 Mbps.

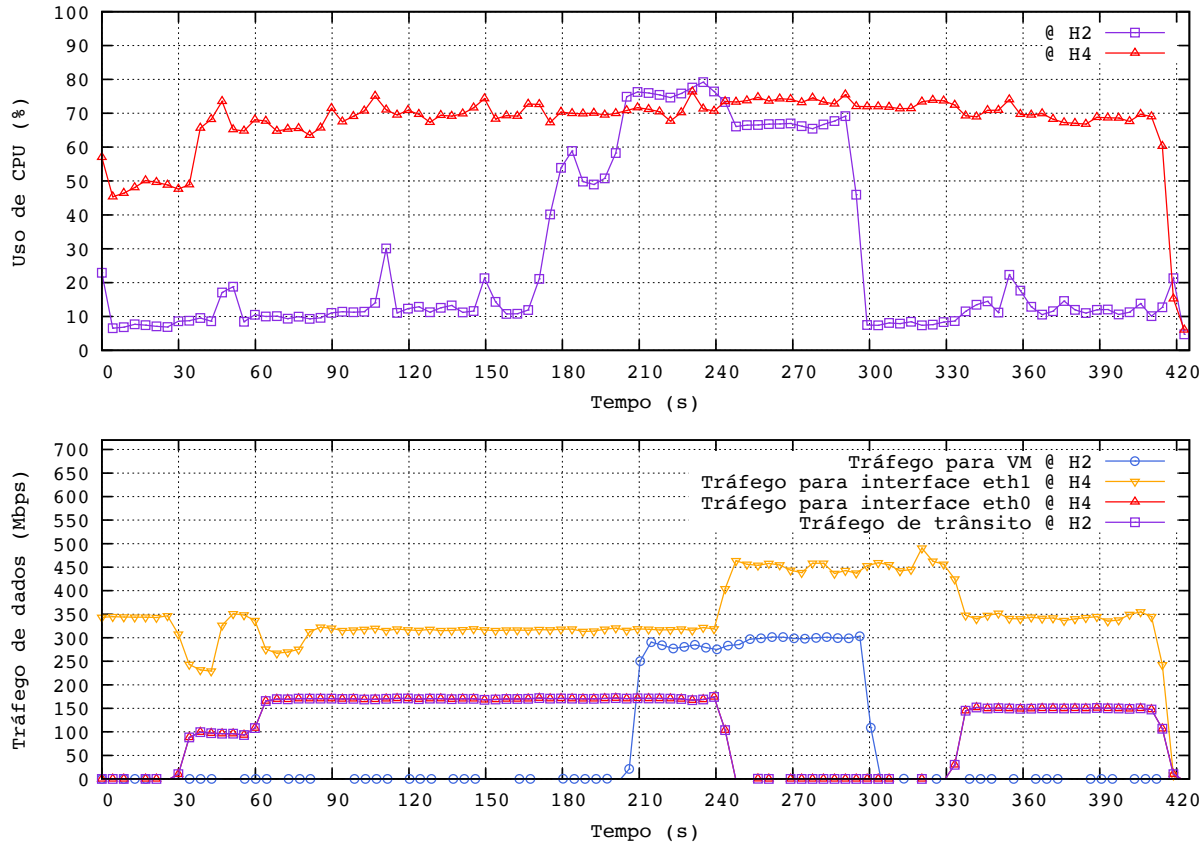

Figura 5.22: Engenharia de tráfego desviando aleatoriamente os fluxos com limite fixo de tráfego de trânsito de 200 Mbps e intervalo de verificação de carga média de 30s

Já no gráfico de tráfego de dados da Figura 5.22, também é desviado inicialmente o fluxo de 100 Mbps aos 30s, porém aos 60s é desviado o fluxo de 80 Mbps fazendo com que mais nenhum outro fluxo possa ser desviado para H2. Esses fluxos permanecem passando pelo *host* H2 até 210s, quando são desviados de volta a rota de origem. O tráfego interno do *host* H2 vai de 180s a 300s e aos 330s, os fluxos de H4 voltam a ser desviados para H2. No entanto, ao contrário do teste anterior, os fluxos desviados para H2 não são exatamente os iniciais. Nessa etapa, são os fluxos de 80 Mbps, 30 Mbps e 50 Mbps.

5.3 Comentários Finais

Neste capítulo pode-se observar que a NetFPGA pode dar ao gerente de uma rede hipercubo vários graus de liberdade de controle de fluxos. Ela pode ser utilizada tanto para assumir as funções de chaves ópticas, como para ampliar o número de vizinhos de um *host*. Essa segunda opção, no entanto, possui uma limitação se utilizada a versão 1.0

do OpenFlow: a soma do número de máquinas virtuais por face do hipercubo não pode ultrapassar 124, que corresponde ao número máximo de regras curinga suportadas pelas tabelas de fluxos. Essa limitação pode ser resolvida com a utilização da versão 1.3 do OpenFlow, que permite a instalação de regras curingas com máscaras no número MAC de destino, assim seria necessário instalar em cada NetFPGA apenas quatro regras por face, correspondendo aos *hosts* a ela ligados.

Além das opções anteriormente citadas, é possível realizar uma gerência de tráfego modificando as regras de fluxos para que os pacotes passem por caminhos alternativos ao menor caminho possibilitando, assim, desafogar uma interface de rede eventualmente sobrecarregada de um determinado *host* de destino. Para esse caso, foram realizados três tipos de testes: desviando os menores fluxos, os maiores fluxos e fluxos aleatórios. Pelos resultados, observa-se que somente no primeiro caso, quando começa um tráfego interno no *host* intermediário, houve a permanência de um fluxo desviado passando pelo caminho alternativo, ou seja, passando pelo *host* intermediário. Isso ocorreu pois a taxa do fluxo que permaneceu é baixa, igual a 30 Mbps, e valor do uso de CPU do *host* intermediário era menor que 60%, o que fez com que o controlador o deixasse por mais dois ciclos, 60 segundos. Quando a porcentagem de uso de CPU desse *host* intermediário aumentou para valores acima de 60%, esse fluxo foi desviado novamente para o menor caminho, assim como os demais fluxos.

Capítulo 6 - Conclusão e Trabalhos Futuros

Este trabalho apresenta uma nova abordagem para a camada híbrida reconfigurável da arquitetura TRIIIAD, em que a comutação óptica é substituída por uma NetFPGA capaz de controlar os fluxos em três níveis diferentes. O primeiro, assim como ocorre na comutação óptica, faz o controle dos fluxos em formatos "barra" ou "cruz", reconfigurando para um formato ou outro de acordo com a necessidade da rede. O segundo, realiza controle com base nas máquinas virtuais de destino a partir da instalação de regras que garantem que o caminho a ser percorrido pelos pacotes será o menor caminho possível, com ligações diretas, utilizando a NetFPGA, entre *hosts* localizados em uma mesma face do hipercubo. E o terceiro, realiza controle por fluxo, com regras inicialmente instaladas para que os pacotes percorram o menor caminho possível. No entanto, além disso é possível realizar uma engenharia de tráfego, por meio da modificação das regras, capaz de desafogar interfaces sobrecarregadas dos *hosts* de destino.

É também feito um estudo da latência de comutação para o caso sem tabela de fluxos e com tabela de fluxos, com a regra a ser utilizada em diferentes posições na tabela. E a partir dos resultados obtidos, pode-se concluir que a latência de encaminhamento, bem como o desvio padrão, sofrem maior influência da posição da regra que será utilizada do que do número de regras instaladas. Além disso, diferenças significativas nos valores da latência de comutação ocorrem apenas quando a regra está no final da tabela, uma vez que para encontrá-la é necessário fazer buscas em todas as regras exatas e em todas as regras curingas da tabela. Esta segunda busca, no entanto é mais custosa, pois é feita de maneira linear, visto que as regras estão incompletas. Quando há somente uma regra de encaminhamento ou quando a regra a ser utilizada está localizada no início da tabela, não há uma diferença significativa no RTT do pacote se utilizar um comutador KeyFlow ou se utilizar um comutador OpenFlow. Isso ocorre porque o tempo necessário para realizar a divisão é equivalente ao tempo necessário para executar visitas ao início da tabela no processo de pesquisa. Conforme a tabela de utilização aumenta e a regra a ser utilizada se aproxima do fim da tabela de regras, o comutador KeyFlow supera o comutador OpenFlow.

Durante o desenvolvimento deste trabalho foram identificadas diversas oportunidades e necessidades de trabalhos futuros, tais como:

- Otimizar o processo de divisão do KeyFlow a fim de diminuir a latência de encaminhamento. Na implementação atual, o tempo demandado para operação de divisão é o mesmo que o requerido para pesquisas na tabela de fluxos com poucas regras.
- Reproduzir cenários de testes em NetFPGA com mais de um salto a fim de verificar a curva de ganho do KeyFlow em relação ao OpenFlow.
- Implementar a versão 1.3, ou superior, do OpenFlow em NetFPGA para que possam ser inseridas regras com máscaras no campo MAC de destino. Isso possibilitaria a redução da tabela de fluxos, diminuindo a latência de pesquisa e aumentando o número de máquinas virtuais suportadas para a implementação da TRIIIAD com NetFPGA contendo regras instaladas a partir do MAC das máquinas virtuais.
- Modificar o algoritmo de encaminhamento para promover a integração da TRIIIAD com NetFPGA contendo regras instaladas a partir do MAC das máquinas virtuais com a TRIIIAD com controle individual de fluxos. Isso possibilitaria a instalação proativa de regras e uma engenharia de tráfego reativa quando necessário.

Referências Bibliográficas

[1] F. L. et al. Verdi, "Novas arquiteturas de data center para cloud computing," *SBRC*, maio 2010, Minicursos.

[2] M. D. D. et al. Moreira. Internet do Futuro: Um Novo Horizonte. [Online]. http://www.gta.ufrj.br/ftp/gta/TechReports/MFCD09.pdf

[3] G1. (2015) G1. [Online]. http://glo.bo/1KxlWeA

[4] Amazon. Amazon. [Online]. http://aws.amazon.com/pt/ec2/

[5] J. DEAN and S. GHEMAWAT, "Mapreduce: simplified data processing on large clusters," in *ACM*, vol. 51, New York, 2008, pp. 107-113.

[6] Cisco, *Cisco Data Center Infrastructure 2.5 Design Guide*. San Jose, CA, 2007.

[7] C. KIM, M. CAESAR, and J. REXFORD, "Floodless in seattle: a scalable ethernet architecture for large enterprises," *SIGCOMM Comput. Commun*, vol. 38, no. 4, pp. 3-14, ago. 2008.

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

[9] M. R. Nascimento, M. R. Salvador, and M. F. Magalhães C. E. Rothenberg, "OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes," in *Cad. CPqD Tecnologia*, July 2010, pp. 1-6.

[10] Nick McKeown et al., "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, 2008.

[11] M., Ribeiro, M. R. N., de Oliveira, R. E. Z., and de Angelis Vitoi, R., Martinello, "KeyFlow: A prototype for evolving SDN toward core network fabric," *IEEE Network*, vol. 28, pp. 12-19, March 2014.

[12] Gilmar L. Vassoler, *TRIIIAD: Uma Arquitetura para Orquestração Autonômica de Redes de Data Center centrado em Servidor*.: Tese de Doutorado, 2015.

[13] Christian Esteve Rothenberg, Marcelo Ribeiro Nascimento, Marcos Rogério Salvador, and Maurício Ferreira Magalhães, "OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes," in *Cad. CPqD Tecnologia*, Campinas, 2011, pp. 65-76.

[14] ONF. Open Networking Foundation. [Online]. https://www.opennetworking.org/sdn-resources/sdn-definition

[15] Natasha Gude et al., "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105-110, 2008.

[16] Elder Leão Fernandes. Nox 1.3 oflib. [Online]. https://github.com/CPqD/nox13oflib

[17] POX. About pox - noxrepo. [Online]. http://www.noxrepo.org/pox/about-pox/

[18] OpenDaylight. Opendaylight sdn controller platform (oscp):overview. [Online]. https://wiki.opendaylight.org/view/OpenDaylight_SDN_Controller_Platform_(OSCP):Overview

[19] P. T. RYU. Ryu sdn framework using openflow 1.3. [Online]. http://osrg.github.io/ryu-book/en/Ryubook.pdf

[20] S Rao. Ryu, a rich featured open source sdn controller supported by ntt labs. [Online]. http://goo.gl/WMbhSU

[21] Antonio Corradi, Mario Fanelli, and Luca Foschini, "Vm consolidation: A real case based on openstack cloud," *Future Generation Computer Systems*, vol. 32, pp. 118-

127, 2014.

[22] *OpenFlow Switch Specification Version 1.0.0.*, 2009.

[23] P. W. B SDN. White box switch os - pica8. [Online]. http://pica8.org/ white-box-switches/white-box-switch-os.php

[24] F CPqD. Openflow 1.3 software switch. [Online]. https://github.com/ CPqD/ofsoftswitch13

[25] E Kohler, R Morris, B Chen, J Jannotti, and M F kaashoek, "The click modular router," *ACM Trans. Comput. Syst*, vol. 18, no. 3, pp. 263-297, 2000.

[26] B., Pettit, J., Koponen, T., Jackson, E. J., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., and Casado, M. Pfaff, "The design and implementation of open vswitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, NSDI'15*, Berkeley, CA, USA, 2015, pp. 117-130.

[27] xDPd. The extensible openflow datapath daemon (xdpd) - bringing innovation into the fast path. [Online]. http://www.xdpd.org

[28] Mckeown. Site oficial do grupo Mckeown. [Online]. http://yuba.stanford.edu/

[29] NetFPGA. Site oficial da NetFPGA. [Online]. http://netfpga.org

[30] NetFPGA. Site oficial Wiki NetFPGA. [Online]. https://github.com/NetFPGA/netfpga/wiki

[31] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," *ACM SIGOPS operating systems review*, vol. 37, p. 29 43, October 2003.

[32] G. DeCandia et al., "Dy- namo: amazon's highly available key-value store," *ACM SIGOPS Operating Systems Review*, vol. 41, p. 205 220, October 2007.

[33] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *CACM*, vol. 51, no. 1, p. 107 113, January 2008.

[34] F. Chang et al., "Bigtable: A distributed storage system for structured data ," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, June 2008.

[35] Bianco, A., Birke, R., Giraudo, L., and Palacin, M, "Openflow switching: Data plane performance," in *IEEE International Conference*, pp. 1-5.

[36] PUC-RIO. [Online]. http://www.matmidia.mat.puc-rio.br/tomlew/MAT1310_11.1/TCR.pdf

[37] H. Wessing, Tech. Univ. Denmark, Lyngby, Denmark COM Res. Center, H. Christiansen, T. Fjelde, and L. Dittmann, "Novel scheme for packet forwarding without header modifications in optical networks," *Lightwave Technology, Journal of*, vol. 20, no. 8, pp. 1277 - 1283, Aug 2002.

[38] Silvia Saldaña Cercós et al., "Design of a stateless low-latency router architecture for green software-defined networking," *SPIE*, vol. 9388, pp. 93880I-93880I-7, February 2014.

[39] Flavio R. de Souza, Pedro P. P. Filho, Moises R. N. Ribeiro Gilmar L. Vassoler, "Hybrid Reconfiguration for Upgrading Datacenter Interconnection Topology," *IEEE Photonics Conference*, pp. 782–783, 2012.

[40] KTH Information and Communication Technology, "Torrents, D.T.: Open Source Traffic Analyzer," *Tese de Mestrado*, 2010.

[41] (2015) Anritsu: Data quality analyzer md1230b. [Online]. https://www.anritsu.com/en-US/test-measurement/products/md1230b

[42] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A.W. Moore, "Oflops: An open framework for openflow switch evaluation," in *Proceedings of the 13th International*

Conference on Passive and Active Measurement, 2012, pp. 85-95.

[43] CISCO. (2015) CISCO. [Online]. http://tools.cisco.com/search/results/en/us/get#q=Simple+Network+Management+Protocol+%28SNMP%29&aus=true

[44] OFLOPS. [Online]. http://www.openflow.org/wk/index.php/Oflops

[45] Rafael Emerick Zape de Oliveira, Rômulo Vitoi, Magnos Martinello, and Moisés Renato Nunes Ribeiro, "KeyFlow: Comutação por Chaves Locais de Fluxos Roteados na Borda via Identificadores Globais," in *SBRC*, 2013.

[46] Lantz, B., Heller, B., and McKeown, N., "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proceedings of the Ninth ACM SIGCOMM Workshop*, New York, USA, 2010, pp. 1-6.

[47] (2010) OpenFlow NetFPGA. [Online]. http://archive.openflow.org/wk/index.php/OpenFlowNetFPGA1_0_0

[48] Pedro P. Piccoli Filho, Moisés R. N. Ribeiro, Magnos Martinello Rafael Emerick Z. de Oliveira, "Parâmetros balizadores para experimentos com comutadores OpenFlow: avaliação experimental baseada em medições de alta precisão," in *XXX Simpósio Brasileiro de Telecomunicações*, Brasilia, 2012, pp. 13-16.

[49] C., Yuan, L., Xiang, D., Dang, Y., Huang, R., Maltz, D., Liu, Z., Wang, V., Pang, B., Chen, H., Lin, Z.W., Kurien, V Guo, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *ACM Conference on Special Interest Group on Data Communication*, New York, USA, 2015, pp. 139–152.