UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CODAT - CONTROL DATAPLANE TESTBED

DIEGO FRAZATTO PEDROSO

ORIENTADOR: PROF^{O}. DR. HERMES SENGER

São Carlos – SP

Fevereiro/2019

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CODAT - CONTROL DATAPLANE TESTBED

DIEGO FRAZATTO PEDROSO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Redes e Sistemas Distribuídos

Orientador: Profº. Dr. Hermes Senger

São Carlos – SP

Fevereiro/2019

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de Dissertação de Mestrado do(a) candidato(a) **Diego Frazatto Pedroso**, realizada em **25 de fevereiro de 2019.**

Prof(a). Dr(a). Ricardo Menotti
(UFSCar)

Prof(a). Dr(a). Hermes Senger
(UFSCar)

Prof(a). Dr(a). César A. Cavalheiro Marcondes
(ITA)

Certifico que a defesa realizou-se com a participação à distância do membro Hermes Senger, depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da banca examinadora redigido neste relatório de defesa.

Prof(a). Dr(a). Ricardo Menotti
Presidente da Comissão Examinadora
(UFSCar)

Dedico o resultado desse trabalho à minha família, orientador, professores e amigos.

AGRADECIMENTOS

Agradeço primeiramente ao meu orientador Profº Dr. Cesar Marcondes por todo apoio nessa jornada, a todos os professores da Faculdade de Tecnologia de Ourinhos (FATEC), em especial aos meus amigos e eternos professores Emerson Barea (Tocha) e Alex Marino, a todos os professores do Departamento de Computação da Universidade Federal de São Carlos (UFSCar), em especial ao meu coorientador Profº Dr. Paulo Matias, a todos meus amigos do laboratório Asgard e por fim, mas não menos importante, aos meus amigos da república Vegas de São Carlos.

RESUMO

Os *testbeds* são plataformas fundamentais para a criação e validação de novas tecnologias e arquiteturas de Internet do futuro. Eles possibilitam um controle mais preciso, escalável e próximo das características reais de uma rede como o ambiente da Internet. Desse modo, estes *testbeds* permitem a validação de arquiteturas elaboradas a partir do zero, com novos conceitos baseados em novas tecnologias e voltados principalmente para resolver todas as implicações de segurança e desempenho que atravancam este modelo atual da Internet. Para preencher esta lacuna, este trabalho tem como objetivo propor um ambiente onde um experimento tenha a possibilidade de controlar o plano de dados em redes, utilizando uma API que reduz o tempo de criação e instanciação do experimento, sem restrições funcionais ao usuário, visando proporcionar um ambiente flexível para a experimentação de partes significativas operando na Internet.

Palavras-chave: Plano de Dados, Testbed, OMF, SDN, Internet do Futuro.

ABSTRACT

Testbeds are fundamental platforms for the creation and validation of new technologies and future Internet architectures. They enable a more precise, scalable and close control of the real characteristics of a network like the Internet environment. In this way, these testbeds allow the validation of architectures elaborated from scratch with new concepts based on new technologies and focused mainly to solve all the security and performance implications that clutter this current model of Internet. To fill this gap, this work aims to propose an environment where the experiment has the possibility to control the data plan in networks, using an API that reduces the time of creation and instantiation of the experiment, without functional restrictions to the user, aiming to provide a flexible environment for experimenting with significant parts operating on the Internet.

Keywords: Data Plane, Testbed, OMF, SDN, Future Internet

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1

INTRODUÇÃO E MOTIVAÇÃO

Este capítulo apresenta o contexto ao qual o trabalho está inserido e a introdução ao tema tratado. A Seção 1.1 contextualiza o tema principal; a Seção 1.2 trata das motivações e objetivos; e a Seção 1.3 descreve a estrutura deste documento.

1.1 Contexto

A rede mundial de computadores foi projetada em 1969, nos Estados Unidos, e tinha como função interligar laboratórios de pesquisa, seguindo modelos e tecnologias que perpetuam até os dias atuais. A Internet foi projetada baseando-se no principio de comunicação fim a fim (SALTZER; REED; CLARK, 1984), que estabelece que todo processamento complexo deve ser feito nos sistemas finais, e o núcleo da rede apenas execute tarefas simples. O trabalho do núcleo da rede é transmitir pacotes da maneira mais eficiente e flexível possível. Todo o restante deve ser feito nas bordas (sistemas finais) (KAMIENSKI et al., 2005). A priori, o único tipo de informação a ser processada pelo núcleo de rede seriam as tabelas de roteamento.

O projeto inicial da Internet suportava um número pequeno de usuários conectados, esse número cresceu e vem crescendo de forma exponencial com o passar dos anos. Uma das realizações mais notáveis da Internet não é necessariamente o que ela é capaz de fazer hoje, mas o fato de ter assumido as dimensões atuais, comparada aos seus propósitos iniciais. Ela iniciou com objetivos bem modestos, não foi projetada para ser utilizada por milhões de pessoas no mundo inteiro. O conjunto de princípios que balizou o seu aparecimento e que hoje suporta a sua evolução é o grande responsável por isso (KAMIENSKI; SADOK, 2000).

Para atender novos requisitos inúmeras tecnologias foram implementadas, como por exem-

plo, a tecnologia NAT[1] (Network Address Translation) que mapeia vários hospedeiros privados para um endereço IP[2] *(Internet Protocol)* exposto publicamente, fazendo com que o problema da limitação de endereços IPV4[3] disponíveis, seja retardada, porém não totalmente sanada. Além disso, a simplicidade do modelo TCP/IP[4] implica em uma rede sem inteligência, o que permite a rápida evolução das aplicações e o grande crescimento da rede (MOREIRA et al., 2009).

1.2 Motivação e Objetivos

Todas estas limitações são consequências da ossificação do projeto inicial da internet, que não permite modificações no núcleo da rede (MOREIRA et al., 2009). Em decorrência disso, novas arquiteturas não são desenvolvidas, e os problemas aumentam na proporção do crescimento da rede. Vale ressaltar que os processos de padronização de novas tecnologias são burocráticos e demandam muito tempo.

A inovação de tecnologias é muito prejudicada em decorrência da difícil reprodução de ambientes verossímeis de experimentação em larga escala, como resultado de todas essas questões elencadas acima, alguns novos paradigmas de arquitetura de Internet têm sido construídos e testados tanto pela indústria como pela acadêmia, com a meta de construir um novo modelo de Internet, chamado de Internet do Futuro (IF).

Os *testbeds* como o FIBRE [5] *(Future Internet BRazilian environment for Experimentation)* (CIUFFO et al., 2016), são plataformas fundamentais para a criação e validação de novas tecnologias e arquiteturas de IF. Visto que, oferecem um ambiente totalmente controlável, escalável e próximo das características reais de uma rede como a Internet. Estes *testbeds* permitem a validação de arquiteturas elaboradas a partir do zero *(from-scratch)*, com novos conceitos, baseada em novas tecnologias e principalmente, visando solucionar as implicações de segurança e desempenho que obstam o modelo atual de Internet.

Para a consolidação de determinada arquitetura são necessárias intensas experimentações, testes em larga escala, simulações que extenuem a estrutura em prova perante uma situação real. (GAO et al., 2014) faz uma abordagem e analisa as tendências das principais arquiteturas utilizadas pela comunidade acadêmica e esses trabalhos apresentam uma grande variedade de redes

[1] https://tools.ietf.org/html/rfc2663
[2] https://tools.ietf.org/html/rfc791
[3] https://tools.ietf.org/html/rfc791
[4] https://tools.ietf.org/html/rfc1180
[5] https://fibre.org.br

de experimentação, com diversas características diferentes entre si, porém as contribuições com *testbeds* que controlam o plano de dados são mínimas, visto que não há disponibilização de ambientes que permitam ao experimentador o controle do plano de dados. Buscando preencher essa lacuna, esse trabalho tem 3 objetivos principais: (1) Propor um ambiente do tipo PaaS *(*Platform as a Service) cujo experimento oferece a possibilidade de controlar o plano de dados, juntamente a reprogramação de FPGA (*Field-Programmable Gate Array*) visando proporcionar uma ambiente de grande representatividade de partes da Internet. (3) Desenvolver uma metodologia de experimentação automática controlando o planos de dados em *bare-metal*, que garanta mais controle para o usuário e menor latência no acesso aos dados, já que evita etapas de processamento via camadas de *software*, focada em aplicações sensíveis a desempenho e velocidade. (3) Promover uma IaaS *(Infrastructure as a Service)* que controle todo o ambiente utilizando sistemas de automação, controle, provisionamento, tolerância a falhas e monitoramento.

1.3 Organização do trabalho

O restante desse trabalho esta organizado da seguinte forma. O Capítulo 2 detalha o atual estado da arte referente a redes de experimentação através de uma revisão da literatura, juntamente com os trabalhos relacionados. O Capítulo 3 apresenta os conceitos dos planos de controle e dados, e esclarece as diferenças entre tecnologias e protocolos de rede. O Capítulo 4 apresenta o protótipo do *testbed*, descreve a arquitetura, os componentes e exemplos das funcionalidades do ponto de vista do experimentador. O Capítulo 5 apresenta em detalhes as tecnologias e métodos para as operações e automações no testebed. O Capitulo 6 apresenta os resultados gerados a partir de experimentos de diversas naturezas. Por fim, o Capítulo 7 conclui e propõe os trabalhos futuros.

Capítulo 2

TRABALHOS CORRELATOS

Apesar da existência de vários *testbeds* ao redor do mundo, nenhum deles trata especificamente do controle do plano de dados, e não permite esse controle ao experimentador.

As pesquisas em ambientes que utilizam *testbeds* para gerar resultados tem despontado para dois horizontes (MARCONDES et al., 2012): no primeiro deles os pesquisadores idealizam a Internet *from-scratch* e apresentam arquiteturas cujo modelo de internet atual é totalmente reformulado, enquanto o segundo apresentam arquiteturas que implementam melhorias e fomentam a Internet vigente.

Nesse sentido, com o objetivo de sintetizar os trabalhos realizados relacionados a temática de redes de experimentação em plano de dados e componentes de *Software Defined network* (SDN), e também reunir todas as técnicas, métodos e tecnologias, apresenta-se uma revisão sistemática que tem por objetivo caracterizar o atual estado da arte do tema proposto. Desta forma, a organização do restante deste Capítulo compreende a Seção 2.1 que elucida os princípios norteadores da Revisão Sistemática (RS); seguido pela Seção 2.2 que descreve o protocolo de revisão; sucedida pela Seção 2.3 com a execução do protocolo; na Seção 2.4 a descrição da seleção e extração dos artigos e finalizada pela Seção 2.5 contendo a síntese da RS.

2.1 Revisão Sistemática

A Revisão Sistemática (RS) faz referência a um tipo de revisão da literatura que visa estabelecer um processo formal a fim de conduzir esse tipo de investigação (ZAMBONI et al., 2010). As RSs são particularmente úteis para integrar e complementar as informações de uma coleção de estudos.

Com esse tipo de revisão é possível reunir uma coleção de trabalhos e extrair uma síntese

das evidências relacionadas a uma estratégia de uma intervenção específica, mediante aplicações de técnicas e metodologias explícitos e sistematizados de busca, com apreciação crítica da informação selecionada (RF, 2007).

Para realização desta RS foram construídas *strings* de busca formadas pela combinação dos sinônimos das palavras-chaves identificados. Essas strings foram submetidas às máquinas de busca relacionadas. Em seguida, realizou-se a leitura dos *abstracts* dos trabalhos recuperados das fontes de busca. Na sequência, uma vez encontrados materiais relevantes para o presente estudo, estes foram selecionados para posterior leitura detalhada. O material carente de unanimidade, referente a relevância, foi estabelecido critério de inclusão ou exclusão pelos revisores e autores.

Os materiais retornados pelas máquinas de busca obedeceram o seguinte processo: (I) identificação de estudos relevantes, (II) exclusão dos estudos com base no título,(III) exclusão dos estudos com base no *abstract*. Por fim, (IV) leitura detalhada do material.

2.2 Protocolo da Revisão Sistemática

Utilizou-se para a revisão sistemática o *software* Start[1]. Essa ferramenta auxilia a montagem de protocolo da Revisão Sistemática (RS).

O objetivo da revisão é identificar as principais redes de experimentação para Internet do Futuro, destacando os aspectos mais importantes, suas particularidades e ressaltando os pontos positivos e negativos de cada ambiente. A população da RS é definida por toda publicação que tem relação com redes de experimentação para Internet do Futuro em redes de computadores. Serão beneficiados com essa revisão profissionais da área de computação, com foco em redes de computadores e a Internet, desenvolvedores de aplicações relacionadas a Internet e comunicação de dados.

Os critérios para seleção de máquinas de buscam obedeceram as seguintes premissas:

1. Conter as principais conferências de redes de computadores e computação.
2. As fontes precisam ser constantemente atualizadas, e devem conter periódicos e/ou estudos em ambas as línguas definidas no protocolo.
3. A *string* de busca foi definida com base nas palavras-chaves, com pode ser observado abaixo. Não foi estipulado um intervalo entre os anos de publicação dos trabalhos, visto

[1] http://lapes.dc.ufscar.br/tools/start_tool

que o tema é relativamente novo, portanto grande partes dos estudos é da última década.

A máquina de busca deve ser fácil de operar e manusear, e deve retornar documentos relevantes a área de estudo da revisão. Com base nos critérios da seleção das fontes de busca, foram selecionadas quatro bases de dados e respectivas strings de busca estão devidamente representados na Tabela 2.1.

Máquina de Busca	String de Busca
ACM	((”Redes de Experimentacão” OR Testbed) OR (”Data Plane Control” OR ”Future Internet”) OR P4 AND ”Software Defined Networking”)
GOOGLE ACADEMIC	((”Redes de Experimentação”OR Testbed) OR (”Data Plane Control” OR ”Future Internet”) OR P4 AND ”Software Defined Networking”)
IEEE	((”Redes de Experimentação”OR Testbed) OR (”Data Plane Control” OR ”Future Internet”) OR P4 AND ”Software Defined Networking”)
SCOPUS	((”Redes de Experimentação”OR Testbed) OR (”Data Plane Control” OR ”Future Internet”) OR P4 AND ”Software Defined Networking”)

Tabela 2.1: Tabela com relação de máquinas de busca e *strings* de busca.

Para estabelecer os interesses da pesquisa, definiu-se no protocolo alguns critérios para seleção e exclusão dos trabalhos retornados das máquinas de buscas, utilizando a *strings* de pesquisa.

Critérios de Inclusão e Exclusão: (I) O artigo retornado pela fonte de busca é pertinente com o tema proposto na revisão. (E) O artigo retornado pela fonte de busca não é pertinente com o tema proposto na revisão. (E) O *Full Paper* não está disponível na íntegra para leitura.

2.3 Execução do Protocolo

Conforme foram definidos na fase do protocolo, diversos trabalhos foram recuperados utilizando a *string* nas máquinas de busca. Todos os trabalhos foram importados através da ferramenta, no formato PDF, o que facilitou a síntese, análise quantitativa e qualitativa dos dados. Na primeira fase de execução apenas informações como título, resumo, palavras-chaves, ano de publicação e referências foram armazenadas na ferramenta Start.

Foram identificados no total, 437 artigos, onde cada base de busca totalizou: ACM (321 artigos), Google Academic (17 artigos), Scopus (42 artigos), IEEE (57 artigos).

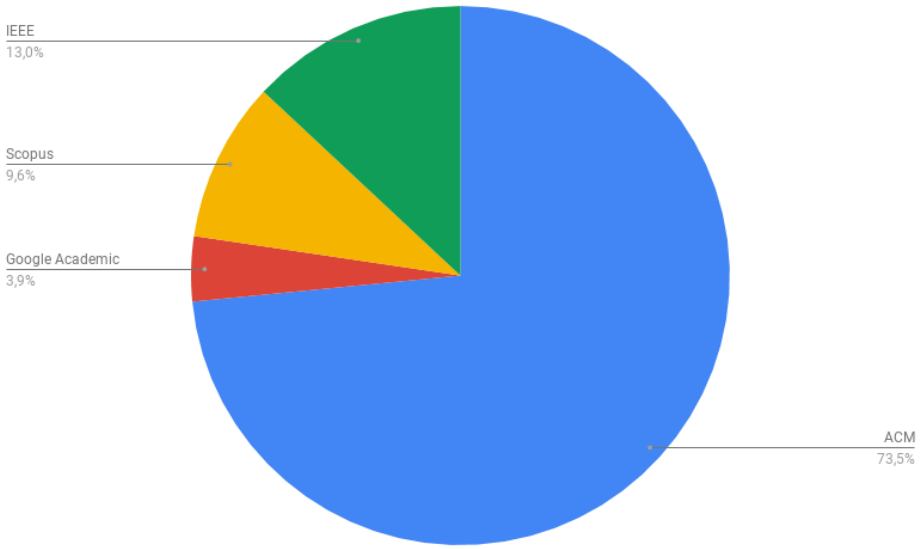

Figura 2.1: Resultados da Execução do Protocolo nas Bases.

2.4 Seleção e Extração

Dos 437 artigos retornados pelas máquinas de buscas, 380 foram aceitos numa primeira seleção com base na leitura do título. Dos 380 trabalhos aceitos no primeiro processo, 101 trabalhos foram aceitos com base dos critérios de inclusão e exclusão, 200 foram rejeitos e 79 era duplicados, ou seja, mais de uma base indexou o mesmo artigo.

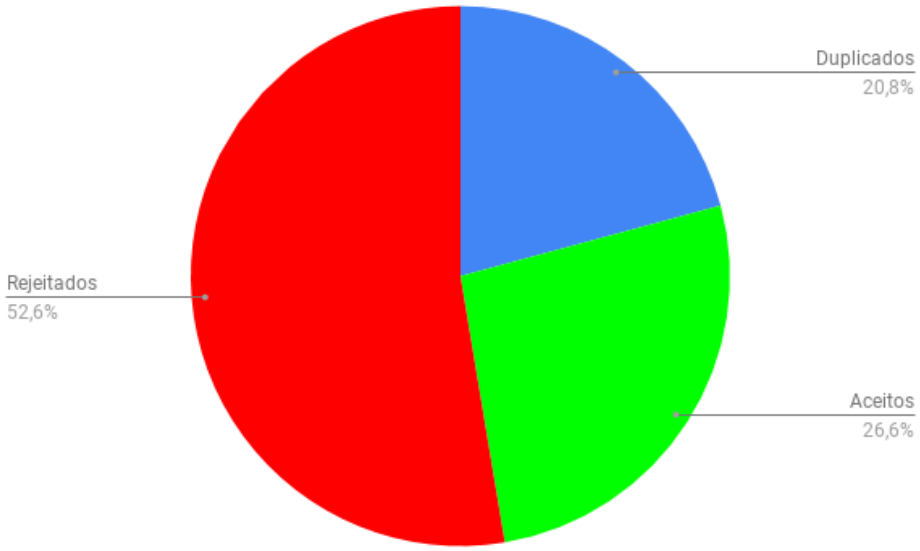

Figura 2.2: Resultados da Seleção dos Artigos.

Os 101 trabalhos aceitos pelo único critério de seleção foram conduzidos para etapa de

extração, onde o processo de leitura do artigo é completo. Destes, apenas 27 atenderam os critérios formalizados e descritos no protocolo da RS. Os demais trabalhos não atenderam a todos os critérios da etapa de extração, e por esse motivo não foram utilizados para a síntese final.

2.5 Resultados e Síntese da Revisão

Com os resultados obtidos na etapa de Seleção e Extração, permitiu-se discutir aspectos específicos de cada artigo, envolvendo métodos, técnicas, parâmetros e tecnologias em ambientes de experimentação para Internet do Futuro. Para finalizar a revisão, a síntese dos artigos foi convertida em tabelas que indexam os campos do protocolo, as tabelas podem ser observadas a seguir.

Nome	Informações do Trabalho	Particularidades ou aplicações para o projeto.	Tipo de Estudo: Qualitativo, Quantitativo ou Estudo de Caso.	O testbed foi implementado usando qual Arcabouço de Controle ?	O ambiente permite experimentação em larga escala ?	O ambiente permite ao usuário a experimentação direta com o Plano de Dados ?	O ambiente permite como recurso, a reprogramação em Hardware?	Inovações e contruibuições para área.
The Design of Real-Time Warning System in Future Internet Environment	Huang, Wun-Yuan, Jen-Wei Hu, and Te-Lung Liu. "The Design of Real-Time Warning System in Future Internet Environment." Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on. IEEE, 2011.	O artigo tem por objetivo ajudar a Taiwan Central Weather Bureau a transmitir mensagens de alerta antecipado de terremoto, utilizando tramissão de mensagens utilizando o Openflowm, otimizando as rotas para pré-alocar os caminhos de transmissão e acelerar o tempo de transmissão.	Estudo de Caso	Openflow	**Não**	**Não**	**Não**	Breve introdução do Openflow e propõe uma solução A fim de minimizar o atraso na transmissão de mensagens. O viés custa um pouco de sobrecarga de transmissão no mecanismo de manutenção automática. Porem e
Testing end-to-end self-management in a wireless future internet environment	Katsikas, Apostolos Kousaridas George, et al. "Testing end-to-end self-management in a wireless future internet environment." The Future Internet Assembly. Springer, Berlin, Heidelberg, 2011.	Utilizando testbeds como Panlab e Self-Net, o autor fez experimentações no autogerenciamento de redes, usando mecanismos do Self-NET.	Estudo de Caso	WiMax, Panlab, Rede Octopus	**Sim**	**Não**	**Não**	O ambiente de rede sem fio Octopus comprovam a melhoria do QoS, e a redução da taxa de erro de pacote.
Multipath routing slice experiments in federated testbeds	Zseby, Tanja, et al. "Multipath routing slice experiments in federated testbeds." The Future Internet Assembly. Springer, Berlin, Heidelberg, 2011.	O autor realiza um experimento Multipath Routing Slice vários testbeds federados, tais como GENI, VINI e FIRE.	Estudo de Casos	GENI, VINI, G-Lab, FIRE, etc..	**Sim**	**Sim**	**Sim**	Vale ressaltar que a federação de várias instalações experimentais pode contribuir para um design melhorado de futuras arquiteturas federadas da Internet. Esse tipo de experimento pode ser explorados para fatias de roteamento de vários caminhos e isso pode formar um novo conceito para arquiteturas de rede.
TRIM: An architecture for transparent IMS-based mobility	Vidal, Ivan, et al. "TRIM: An architecture for transparent IMS-based mobility." Computer Networks 55.7 (2011): 1474-1486.	O artigo apresenta uma arquitetura para a mobilidade transparente baseada em IMS. Suportando a mobilidade em redes IMS de forma transparente para os aplicativos do usuário final, que desconhecem os procedimentos de gerenciamento de transferência executados entre os Nós móveis e a rede.	Estudo de Caso	IMS	**Sim**	**Não**	**Não**	É um testebd testbed que combina tecnologias 3G e IEEE 802.11 juntamente a um núcleo IMS. o TRIM suporta corretamente a mobilidade para o tráfego de usuários UDP e TCP. O TRIM encaminha o tráfego do usuário por meio de um elemento de rede na rede doméstica do nó móvel, que é semelhante ao IP

Tabela 2.2: Sumários dos Artigos Analisados Parte 1 de 8.

Nome	Informações do Trabalho	Particularidades ou aplicações para o projeto.	Tipo de Estudo: Qualitativo, Quantitativo ou Estudo de Caso.	O testbed foi implementado usando qual Arcabouço de Controle ?	O ambiente permite experimentação em larga escala ?	O ambiente permite ao usuário a experimentação direta com o Plano de Dados ?	O ambiente permite como recurso, a reprogramação em Hardware?	Inovações e contruibuições para área.
A use-case on testing adaptive admission control and resource allocation algorithms on the federated environment of Panlab	Tranoris, Christos, Pierpaolo Giacomin, and Spyros Denazis. "A use-case on testing adaptive admission control and resource allocation algorithms on the federated environment of Panlab." The Future Internet Assembly. Springer, Berlin, Heidelberg, 2011.	O trabalho apresenta um caso de uso em que um algoritmo de alocação adaptativa de recursos foi testado utilizando a infraestrutura do Panlab. Simula problemas de um ambiente em produção.	Estudo de Caso	Panlab	**Sim**	**Não**	**Sim**	A estrutura do Panlab fornece os componentes de arquitetura que permitem testar aplicativos próximos a ambientes de produção em um conjunto heterogêneo de recursos. Os algoritmos usados nos experimentos foram testados em um ambiente de melhor esforço com problemas reais de conectividade que não podem ser facilmente executados em ambientes de simulação.
Emerging testing trends and the Panlab enabling infrastructure	Wahle, Sebastian, et al. "Emerging testing trends and the Panlab enabling infrastructure." IEEE Communications Magazine 49.3 (2011): 167-175.	Este artigo aborda vários princípios fundamentais e suas implementações tecnológicas correspondentes, que permitem o provisionamento de grandes testbeds, bem como a implantação de futuras plataformas para Internet do Futuro, utilizando o testbed Panlab	Estudo de Caso	Panlab	**Sim**	**Não**	**Sim**	A camada de adaptação de recursos (RAL) do PTM implementa mecanismos que auxiliam na descoberta e no monitoramento de recursos, gerando eventos de mudança específicos para comunicar o status do recurso,
Architectures for the future networks and the next generation Internet: A survey	Paul, Subharthi, Jianli Pan, and Raj Jain. "Architectures for the future networks and the next generation Internet: A survey." Computer Communications 34.1 (2011): 2-42.	O artigo fornece um breve descrição de inúmeros projetos de pesquisa relacionados a redes de experimentação. Fornece também uma visão geral destes projetos do ponto de vista de arquitetura e do usuário.	Estudo de Caso		**Sim**	**Não**	**Sim**	
Federating future internet testbeds: an adaptor-based approach	Nam, Ki-Hyuk, et al. "Federating future internet testbeds: an adaptor-based approach." Proceedings of the 6th International Conference on Future Internet Technologies. ACM, 2011.	O artigo apresenta um modelo federado apra experimentos baseado em slices. O autor implementa um adaptador chamado FirstProNET para interagração com as demais federações.	Estudo de Caso Qualitativo	ProtoGENI	**Sim**	**Não**	**Sim**	O adaptador é implementado em uma NetFPGA. A intercace GUI classifica e disponibiliza os recursos anunciados pela distância e posição das coordenadas GPS do celular do usuário.

Tabela 2.3: Sumários dos Artigos Analisados Parte 2 de 8.

Nome	Informações do Trabalho	Particularidades ou aplicações para o projeto.	Tipo de Estudo: Qualitativo, Quantitativo ou Estudo de Caso.	O testbed foi implementado usando qual Arcabouço de Controle ?	O ambiente permite experimentação em larga escala ?	O ambiente permite ao usuário a experimentação direta com o Plano de Dados ?	O ambiente permite como recurso, a reprogramação em Hardware?	Inovações e contruibuições para área.
Virtual federated network operations on future internet	Kim, Dongkyun, Joo-Bum Kim, and Jin-Hyung Park. "Virtual federated network operations on future internet." Proceedings of the 6th International Conference on Future Internet Technologies. ACM, 2011.	O setup permite gerenciar suas próprias redes virtuais para seus experimentos e aplicações, tudo isso em redes federadas.	Qualitativo	GENI	**Sim**	**Não**	**Sim**	É uma rede virtual livre de congestionamento dedicada, garantindo um desempenho muito alto, sem perda de pacotes. Isso é resultado de um pipeline customizado para processamento de pacotes e tolerância a falhas.
Understanding design and control for media-centric service composition experiments	Han, Sang Woo, and JongWon Kim. "Understanding design and control for media-centric service composition experiments." Proceedings of the 6th International Conference on Future Internet Technologies. ACM, 2011.	O trabalho consiste num processo para integrar serviços centrados em mídia e empregar o conceito de adaptação baseado em feedback e no SOC (computação orientada a serviços). A fim de desenvolver experimentos de composição de serviços de mídia em testes de Internet do Futuro	Estudo de Caso	GIST	**Sim**	**Não**	**Não**	Os usuários utilizam linguagens de descrição semântica para especificar recursos testbed e mapear o conteúdo com recursos de computação ou rede.
Testbed Evaluation of P2P Shared Cache Architecture	Yamamoto, Shu, and Akihiro Nakao. "Testbed evaluation of P2P shared cache architecture." Proceedings of the 6th International Conference on Future Internet Technologies. ACM, 2011.	O autor implementa um roteador de cache compartilhado baseado no Kernal do sistema operacional Linux. A proposta é reduzir em 30% a compressão de tráfego para fluxos do BitTorrent.	Estudo de Caso	JGN	**Sim**	**Não**	**Não**	Arquitetura baseada no ambiente do PlanetLab. O CoreLab é a rede de sobreposição que aprimora a estrutura do PlanetLab e acopla uma KVM (Máquina virtual baseada em kernel) como um monitor de máquina virtual hospedado.
Mobilityfirst future internet architecture project	Seskar, Ivan, et al. "Mobilityfirst future internet architecture project." Proceedings of the 7th Asian Internet Engineering Conference. ACM, 2011.	O Auto apresenta a arquitetura de rede MobilityFirst, destina-se a abordar diretamente os desafios do acesso sem fio e da mobilidade em escala		OMF, OpenStack	**Sim**	**Não**	**Não**	Os principais componentes do protocolo são: Separação da nomemclatura do endereçamento, nomes de autocertificação em chave pública, serviço global de resolução de nomes (Global DNS) e roteamento tolerantes a atrasos. O protocolo também é capaz de escolher automaticamente um ou várias rotas do tipo multi-homed disponíveis.

Tabela 2.4: Sumários dos Artigos Analisados Parte 3 de 8.

Nome	Informações do Trabalho	Particularidades ou aplicações para o projeto.	Tipo de Estudo: Qualitativo, Quantitativo ou Estudo de Caso.	O testbed foi implementado usando qual Arcabouço de Controle ?	O ambiente permite experimentação em larga escala ?	O ambiente permite ao usuário a experimentação direta com o Plano de Dados ?	O ambiente permite como recurso, a reprogramação em Hardware?	Inovações e contruibuições para área.
Embedding Virtual Topologies in Networked Clouds	Xin, Yufeng, et al. "Embedding virtual topologies in networked clouds." Proceedings of the 6th international conference on future internet technologies. ACM, 2011.	O autor faz uma incorporação de rede virtual em um ambiente em nuvem. No qual vários sites são de provedor de nuvem são conectados, e para cada solicitação, o sistema planeja uma incorporação de baixo custo.		ORCABEN	**Sim**	**Não**	**Não**	Embora complexo, o sistema de incorporações orquesta e otimiza as solicitações distribuídas dos provedores.
P4: Programming protocol-independent packet processors	Bosshart, Pat, et al. "P4: Programming protocol-independent packet processors." ACM SIGCOMM Computer Communication Review 44.3 (2014): 87-95.	O artigo propoe uma linguagem de programação para equipamentos de rede em alto nível. A linguagem permite a reprogramação do pipeline de equipamentos de rede virtuais em nivel de kernel. Permitindo ao usuário o controle do plano de dados com uma API fácil e de grande abstração de níveis inferiores de programação, como o kernel.	Qualitativo e Quantitativo	Não se Aplica	**Sim**	**Sim**	**Não**	É mais genérico que o Openflow, e nesse sentido a contribuição para área é maior. Permite fazer mais que combinação de tabelas match e action, como o openflow.
Testbed fibre: Passado, presente e perspectivas	Ciuffo, Leandro, et al. "Testbed fibre: Passado, presente e perspectivas." Anais do WPEIF 2016 Workshop de Pesquisa Experimental da Internet do Futuro. sn, 2016.	O FIBRE é uma plataforma que permite criação e validação de novas tecnologias e arquiteturas de Internet do futuro. Oferecem um ambiente controlável, escalável e próximo das características reais de uma rede como a internet. Permitem também a validação de arquiteturas elaboradas a partir do zero (from-scratch), com novos conceitos, baseada em novas tecnologias e, principalmente, visando solucionar todas as implicações de segurança e desempenho que atravancam o modelo atual da internet. Permite experimentação em larga escala, e permite a gerencia do plano de controle, utilizando Openflow, rede de sensores, máquinas virtuais instanciadas em ambientes locais ou remotos (experimentos federados).	OMF e OCF	OMF e OCF	**Sim**	**Não**	**Não**	Atualmente utiliza o framework de controle OCF, está em processo de migração para OMF, que permite uma gama maior de experimentos para os usuários.

Tabela 2.5: Sumários dos Artigos Analisados Parte 4 de 8.

Nome	Informações do Trabalho	Particularidades ou aplicações para o projeto.	Tipo de Estudo: Qualitativo, Quantitativo ou Estudo de Caso.	O testbed foi implementado usando qual Arcabouço de Controle ?	O ambiente permite experimentação em larga escala ?	O ambiente permite ao usuário a experimentação direta com o Plano de Dados ?	O ambiente permite como recurso, a reprogramação em Hardware?	Inovações e contruibuições para área.
Estado da Arte de Sistemas de Controle e Monitoramento de Infraestruturas para Experimentação de Redes de Comunicão	Marcondes, Cesar Augusto Cavalheiro, et al. "Estado da Arte de Sistemas de Controle e Monito-ramento de Infraestruturas para Experimentação de Redes de Comunicação." Minicursos do SBRC, Ouro Preto, MG (2012): 3-7.	As pesquisas em ambientes que utilizam testbeds para gerar resultados tem despontado para dois horizontes, no primeiro deles, os pesquisadores idealizam a internet (from-scratch), que apresenta arquiteturas onde o modelo de internet atual é totalmente reformulado, e o segundo, apresenta as arquiteturas que implementam melhorias e fomentam a arquitetura vigente.	Estudo de Caso e Qualitativo	OMF, OCF e Outros	**Não**	**Não**	**Não**	Caracteriza o atual estado da arte no tema proposto. Apresenta muitos estudos relevantes e de diversas naturezes oriundas do tema central (testbeds).
Internet do futuro: Um novo horizonte Futuro	Moreira, Marcelo DD, et al. "Internet do futuro: Um novo horizonte." Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2009 (2009): 1-59.	O artigo faz abordagem das princiais redes de experimentação utilizadas por pesquisadores. Evidência uma série de problemas que atravancam o modelo atual de internet, causando assim uma grande ossificação , isso dificulta modificações funcionais em equipamentos de rede de toda a internet.	Qualitativo e Quantitativo	Outros	**Sim**	**Não**	**Sim**	O trabalho faz uma abordagem Top-Down da internet, descreve uma série de testbeds e suas particularidades.
Pan-European testbed and experimental facility federation architecture refinement and implementation	Wahle, Sebastian, et al. "Pan-European testbed and experimental facility federation–architecture refinement and implementation." International Journal of Communication Networks and Distributed Systems 5.1-2 (2010): 67-87.	A testbed Panlab é um dos projetos do European Union 6th Framework Programme, e tem como finalidade prover uma plataforma de redes que interconecta diversos laboratórios, oferecendo um serviço fim-a-fim, coordenado e centralizado.	Estudo de Caso	Outros	**Sim**	**Não**	**Sim**	Um das inovações é uma API que possiblita ao usuário o acesso aos recursos em operação durante o experimento. Esse recurso aumenta as possibilidades e recursos do usuário com seu experimento.

Tabela 2.6: Sumários dos Artigos Analisados Parte 5 de 8.

Nome	Informações do Trabalho	Particularidades ou aplicações para o projeto.	Tipo de Estudo: Qualitativo, Quantitativo ou Estudo de Caso.	O testbed foi implementado usando qual Arcabouço de Controle ?	O ambiente permite experimentação em larga escala ?	O ambiente permite ao usuário a experimentação direta com o Plano de Dados ?	O ambiente permite como recurso, a reprogramação em Hardware?	Inovações e contruibuições para área.
OMF: a control and management framework for networking testbeds	Rakotoarivelo, Thierry, et al. "OMF: a control and management framework for networking testbeds." ACM SIGOPS Operating Systems Review 43.4 (2010): 54-59.	No contexto de testbeds existem softwares para controle de experimentação, como o OCF, que gerencia componentes de múltiplas infraestruturas virtualizadas em cloud. Uma plataforma interessante para experimentação em plano de dados é o OMF, que proporciona amplas possibilidades ao experimentador para criação de um experimento mais seguro, visto que o usuário não precisa ter direitos de administrador no ambiente, permitido executar apenas alguns comandos pré-definidos no experimento, possibilitando controle, auditoria e replicações.	Qualitativo	OMF	**Sim**	**Não**	**Sim**	
Planetlab: an overlay testbed for broad-coverage services	Chun, Brent, et al. "Planetlab: an overlay testbed for broad-coverage services." ACM SIGCOMM Computer Communication Review 33.3 (2003): 3-12.	O PlanetLab permite que serviços múltiplos sejam executados de forma compartilhada, e o utilizador consegue realizar experimentos reais usando protocolos de internet já implementados, como por exemplo, o P2P.	Qualitativo e Quantitativo	Outros	**Sim**	**Não**	**Não**	
Netfpga: Processamento de pacotes em hardware	Goulart, Pablo, et al. "Netfpga: Processamento de pacotes em hardware." Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2015 (2015): 1-3.	O artigo apresenta na prática como funciona a arquitetura FPGA. Demonstrando as funcionalidades da NetFPGA 1G, o autor demostra através de um experimento com um firewall, como funciona o processamento de pacotes em hardware, desde a sintetização até a operação em rede.	Estudo de Caso	Outros	**Não**	**Não**	**Sim**	Tem exemplos práticos e comandos de como operar a placa NetFPGA, porém na versão já descontinuada do Fedora 13. Também aplica os projetos de referências em experimentos que podem ser replicados por usuários do testbed.

Tabela 2.7: Sumários dos Artigos Analisados Parte 6 de 8.

Nome	Informações do Trabalho	Particularidades ou aplicações para o projeto.	Tipo de Estudo: Qualitativo, Quantitativo ou Estudo de Caso.	O testbed foi implementado usando qual Arcabouço de Controle ?	O ambiente permite experimentação em larga escala ?	O ambiente permite ao usuário a experimentação direta com o Plano de Dados ?	O ambiente permite como recurso, a reprogramação em Hardware?	Inovações e contruibuições para área.
Future internet research and experimentation : The FIRE initiative	Gavras, Anastasius, et al. "Future internet research and experimentation: the FIRE initiative." ACM SIGCOMM Computer Communication Review 37.3 (2007): 89-92.	O projeto FIRE é focado em explorar potenciais novos recursos, bem como explorar novas arquiteturas e protocolos de rede,com o objetivo de prover soluções para o futuro da internet. A iniciativa FORGE é uma plataforma que permitem que alunos do ensino superior ao redor do mundo realizem experimentos em IF, demostrou que as testbeds podem ter um impacto valioso na experiência de aprendizagem para educadores e estudantes.	Qualitativo	OCF	**Sim**	**Não**	**Não**	O artigo tem maior foco em ambientes experimentais para ensino e educação.
GENI: A federated testbed for innovative network experiments	Berman, Mark, et al. "GENI: A federated testbed for innovative network experiments." Computer Networks 61 (2014): 5-23.	A GENI é o principal projeto para experimentação em Internet do Futuro, fornecendo um ambiente virtual para redes e sistemas distribuídos de pesquisa em larga escala.	Estudo de Caso	OMF, OpenStack	**Sim**	**Sim**	**Sim**	O projeto GENI permite aos seus utilizadores conectar recursos de computação usando redes de camada 2 em topologias mais adequadas aos seus experimentos, instalar softwares personalizados e até mesmo sistemas operacionais completos, além de executar seus próprios protocolos de camada 3.
Techniques and research trends of network testbed	Gao, Haihui, et al. "Techniques and Research Trends of Network Testbed." Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), 2014 Tenth International Conference on. IEEE, 2014.	O artigo faz uma discussão sobre as principais redes de experimentação, e cita as tendências, tecnologias e vulnerabilidades de segurança. O autor realiza testes de desempenho e segurança nos ambientes de experimentação.	Estudo de Caso	Outros	**Sim**	**Não**	**Não**	As interações dos ambientes com novas tecnologias são validas no sentido de ampliar o mix de serviços para o usuário. Assim o experimentador tem um número maior de recursos para compor o seu experimento.

Tabela 2.8: Sumários dos Artigos Analisados Parte 7 de 8.

Nome	Informações do Trabalho	Particularidades ou aplicações para o projeto.	Tipo de Estudo: Qualitativo, Quantitativo ou Estudo de Caso.	O testbed foi implementado usando qual Arcabouço de Controle ?	O ambiente permite experimentação em larga escala ?	O ambiente permite ao usuário a experimentação direta com o Plano de Dados ?	O ambiente permite como recurso, a reprogramação em Hardware?	Inovações e contruibuições para área.
Orchestration and Reconfiguration Control Architecture ORCA- a 5G Experimental Environment	Kazaz, T., Liu, W., Jiao, X., Moerman, I., Seskar, I., Paisana, F., ... & Danneberg, M. (2017). Orchestration and Reconfiguration Control Architecture.	O ORCA é um ambiente para experimentação em redes 5G que utiliza gerenciamento distribuído e seguro de recursos em domínios federados. É um projeto desenvolvido pelo NICL Lab da Duke University, está integrado com o Openstack, e com o xCAT, para suportar o provisionamento de nó baremetal, basicamente atua como um serviço de provisionamento de redes dinâmicas multicamadas.	Estudo de Caso	ORCABEN	**Sim**	**Não**	**Sim**	
Assessing Locator/Identifier Separation Protocol interworking performance through RIPE Atlas	Li, Yue, and Luigi Iannone. "Assessing Locator/Identifier Separation Protocol interworking performance through RIPE Atlas." Computer Networks 132 (2018): 118-128.	O trabalho é focado no problema de escalabilidade do BGP. A tecnologia LISP divide o espaço de endereçamento IP convencional em duas categorias: identificador e localizador. Onde a primeira é usada para identificação e o Locator é usado para roteamento	Estudo de Caso	LISP	**Não**	**Não**	**Não**	
Future internet architecture and testbeds	Huang, T., Yu, F. R., Xie, G., & Liu, Y. (2017). Future internet architecture and testbeds. China Communications, 14(10), iii-iv.	O artigo evidencia a importancia das redes de experimentação para o surgimento de novas tecnologias. Apesar das análises e simulações serem ferramentas importantes no estudo o comportamento do tráfego de dados, é essencial que novas ideias de pesquisa sejam validados em sistemas reais.	Estudo de Caso	Outros	**Sim**	**Não se Aplica**	**Não se Aplica**	

Tabela 2.9: Sumários dos Artigos Analisados Parte 8 de 8.

O PlanetLab[2] é uma rede de pesquisa global que permite que vários serviços múltiplos sejam executados de forma compartilhada. Desde o início de 2003, mais de mil pesquisadores das principais instituições acadêmicas e laboratórios de pesquisa industrial usaram o PlanetLab para desenvolver novas tecnologias para armazenamento distribuído, mapeamento de rede, sistemas P2P (*peer-to-peer*), tabelas *hash* distribuídas e processamento de consultas.

Figura 2.3: Mapa de distribuição PlanetLab. Fonte: https://www.planet-lab.org/status (2017).

O PlanetLab permite que serviços múltiplos sejam executados de forma simultânea, os autores (CHUN et al., 2003) apresentam a implementação inicial da plataforma, incluindo os mecanismos utilizados para implementar a virtualização dos serviços.

O PlanetLab possui 1353 nós em 717 sites, conforme ilustrado na Figura 2.3, as ilhas são distribuídas em várias localidades ao redor do mundo. A arquitetura da plataforma foi projetada para prover ao experimentador condições similares à realidade da Internet. Uma das funcionalidades do Planetlab é fornecer um forte isolamento dos *slices*. As máquinas virtuais possuem uma imagem padrão que é pré configurada na instanciação dos experimentos.

Os usuários do PlanetLab fornecem uma quantidade de nós para a rede da plataforma, e em contraprestação o *testbed* fornece seus recursos de modo que o usuário possa implantar e realizar experimentos em larga escala.

A GENI[3] *(Global Environment for Network Inovation)* é o principal projeto para realização de experimentos relacionados a Internet do Futuro, fornecendo um ambiente virtual para redes e sistemas distribuídos de pesquisa em larga escala. O projeto GENI permite aos seus expe-

[2]https://www.planet-lab.org
[3]http://www.geni.net

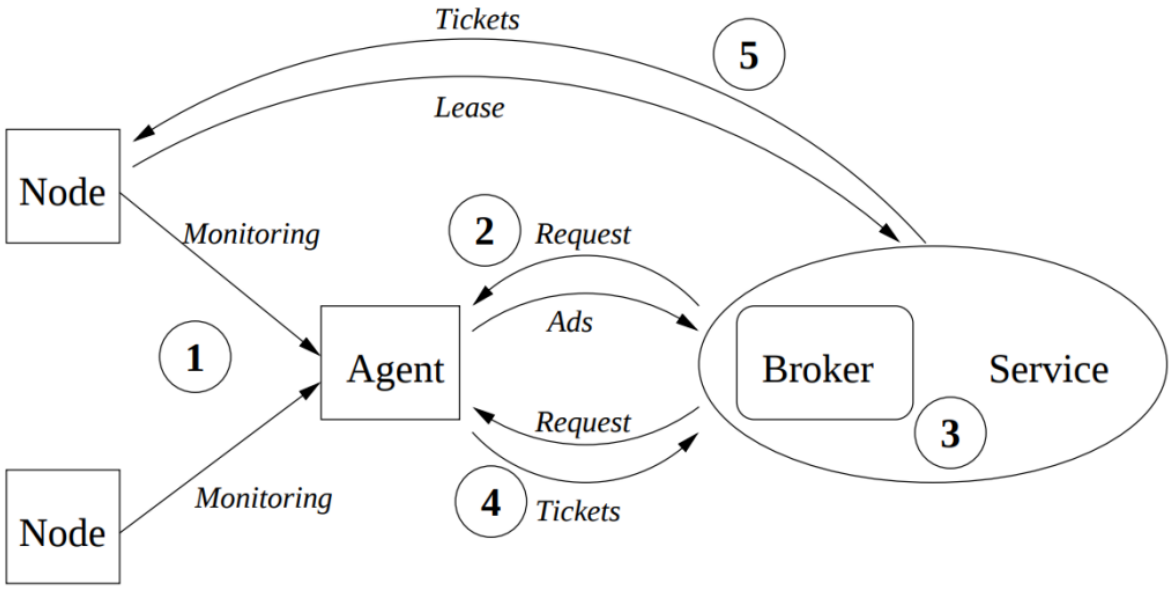

Figura 2.4: Arquitetura de Experimentação do Testbed GENI.
(CHUN et al., 2003)

rimentadores conectar recursos de computação usando redes de camada 2 em topologias mais adequadas às suas experiências, instalar *softwares* personalizados ou mesmo sistemas operacionais (BERMAN et al., 2014) e também executar seus próprios protocolos de camada 3. Esse ambiente é utilizado para experimentação em larga escala.

A arquitetura dos serviços do ambiente GENI pode ser observado na Figura 2.5, a área superior demarcada em azul, chamada de *Clearinghouse*, possui todos os cadastros e recursos da GENI, tais como *slices* e registros para fins de auditoria.

Os *Aggregates* são os agregados, ou entidades independentes, que possuem componentes que serão incluídos em experimentos e controlados pelo plano de controle. Cada uma dessas entidades possuir um controlador genérico (*Aggregate Manager*), e também podem possuir algum tipo de gerenciador específico, a função do *Broker* é gerenciar os recursos agregados, visando facilitar a alocação de recursos para o usuário.

As entidades associadas tem a função de administrar e gerenciar os recursos do experimento do usuário. A função do arcabouço de controle OMF[4] *(Control Monitoring Framework)* é realizar a integração entre todas essas entidades, bem como as funcionalidades de protocolos de controle básicos (MARCONDES et al., 2012).

O *testbed* GENI ainda conta com uma série de outros ambientes de experimentação que são integrados a sua plataforma, tais como a GEMINI (SIVARAMAKRISHNAN et al., 2010), o

[4]http://www.fibre-ict.eu/index.php/cmf/omf

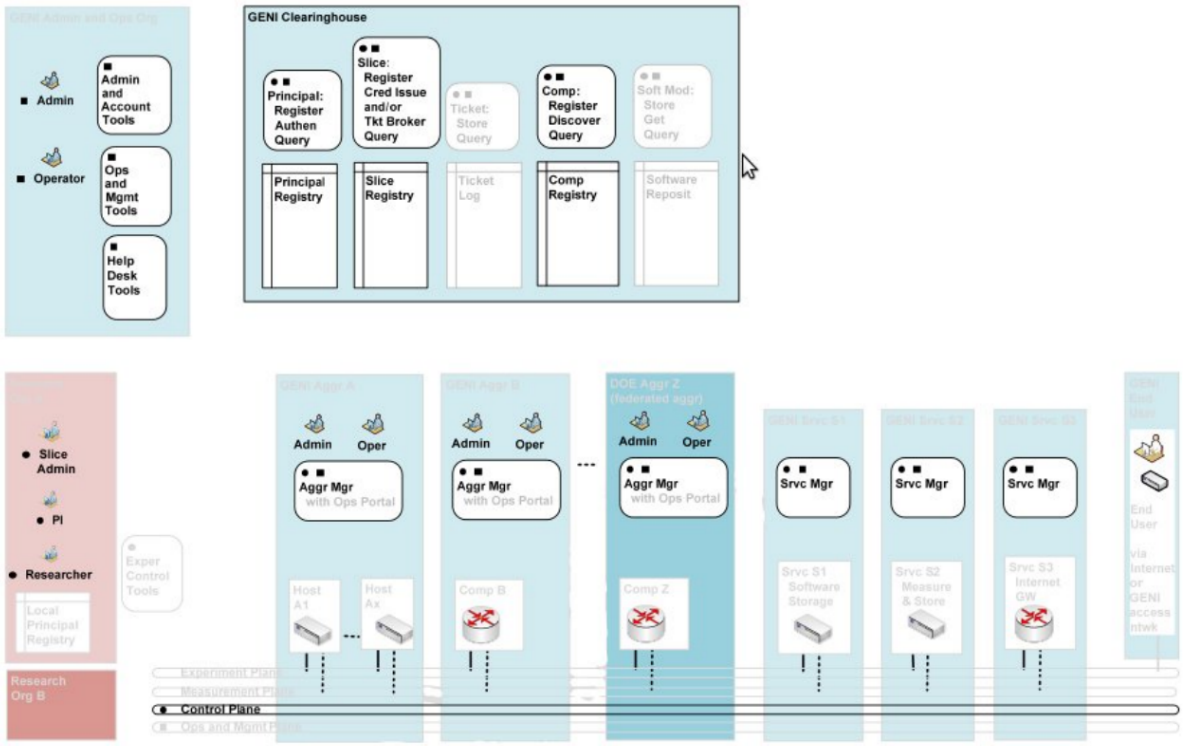

Figura 2.5: Visão Geral dos Serviços do *Tested* GENI. (MARCONDES et al., 2012)

proejto GIMI[5] e o projeto GIMS (THOMAS et al., 2012).

O Panlab[6] (*Pan-European Laboratory*) é um dos projetos do *European Union 6th Framework Programme*, e tem como finalidade prover uma plataforma de redes que interconecta diversos laboratórios (MARCONDES et al., 2012), oferecendo um serviço fim-a-fim, coordenado e centralizado.

O *Panlab Office* faz o intermédio entre os parceiros e clientes do Panlab. Ele atende, opera e coordena processos legais e operacionais, bem como provisionamento das infraestruturas que serão utilizadas nos experimentos.

Como o nome sugere, o *Panlab Customer* é cliente ou usuário que faz uso da plataforma. O usuário tem acesso a infraestrutura e as funcionalidades. Ele recebem suporte do *Panlab Office* para implementar no *pool* da plataforma novos produtos ou serviços.

O ORCA[7] (*Orchestration and Reconfiguration Control Architecture)* é um ambiente *open source* para experimentação em redes 5G (LIU et al., 2017) que utiliza gerenciamento distribuído e seguro de recursos em domínios federados. É um projeto desenvolvido pelo NICL

[5]http://gimi.ecs.umass.edu
[6]http://www.panlab.net
[7]https://www.orca-project.eu/testbeds/

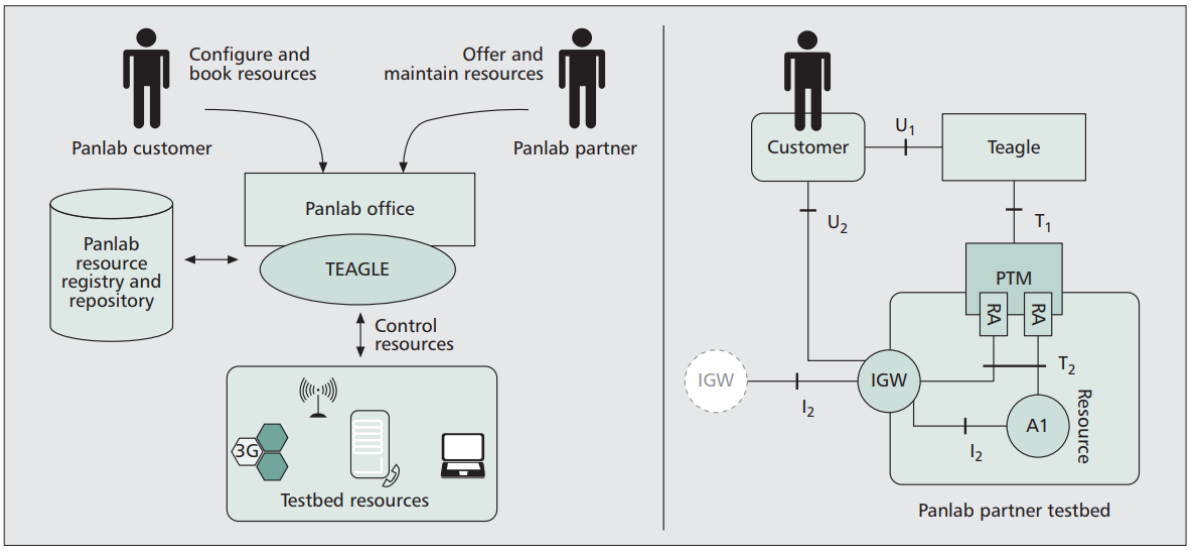

Figura 2.6: Arquitetura e Funções do *Testbed* Panlab (WAHLE et al., 2011)

Lab da Duke University, está integrado com o Openstack[8] e com o xCAT[9], para suportar o provisionamento de nó *baremetal*, basicamente atua como um serviço de provisionamento de redes dinâmicas multicamadas.

O *testbed* ORBIT[10] *(Open-Access Research Testbed for Next-Generation Wireless Networks)* permite emulações de rede sem fio de duas camadas, foi projetada para obter uma experimentação reprodutível, utilizando o *framework* de controle OMF para orquestração do experimento, também conta com uma rede de máquinas SandBox, utilizadas para depuração dos experimentos.

O projeto FIRE[11] *(Future Internet Research and Experimentation)* é focado em explorar potenciais novos recursos (GAVRAS et al., 2007), bem como explorar novas arquiteturas e protocolos de rede, com o objetivo de prover soluções para o futuro da Internet.

A iniciativa FORGE[12] *(Forging Online Education through FIRE)* é uma plataforma que permite que alunos do ensino superior, ao redor do mundo, realizem experimentos em IF, (MIKROYANNIDIS et al., 2016) demostrou que os *testbeds* podem ter um impacto valioso na experiência de aprendizagem para educadores e estudantes.

A plataforma BPFabric (JOUET; PEZAROS, 2017) foi implementada em *software* sobre uma interface *socket raw* do Linux, e utilizando o DPDK. Funciona de forma análoga a arquitetura PISCES (SHAHBAZ et al., 2016a), porém a diferença é que utiliza instruções eBPF *exten-*

[8]https://www.openstack.org
[9]https://xcat.org
[10]http://www.orbit-lab.org
[11]https://www.ict-fire.eu
[12]https://www.openeducationeuropa.eu/en/project/for

ded Berkeley Packet Filter, proporcionando uma interface bruta na camada de enlace, fazendo com que pacotes brutos da mesma camada sejam enviados e recebidos pelo sistema operacional. A arquitetura permite que o plano de controle especifique imediatamente o *pipeline* de processamento de pacotes do plano de dados dos equipamentos de rede, bem como para consultar e manipular o estado da rede diretamente.

Autores como (PACÍFICO et al., 2018), (PEDROSO et al.,) e (IBANEZ et al., 2019) implementam um modelo utilizando um roteador SDN implementado em hardware com objetivo de possibilitar a utilização de novos campos e protocolos, sendo que o ultimo trabalho também aborda todo o *workflow* e os **pipelines** de execução do P4 em uma placa programável. O projeto P4FPGA (WANG et al., 2017) também permite a criação de novas arquiteturas a partir de um *hardware* programável.

O *testbed* FIBRE[13] é um ambiente de experimentação em redes para pesquisa em larga escala, com foco em experimentos que utilizam SDN (*Software-Defined Networking*). A infraestrutura fornecida pelo FIBRE (SALMITO et al., 2014) pode permitir que pesquisadores avaliem algoritmos, técnicas e abordagens inovadoras, que contribuam para novas arquiteturas e que possam ser usadas em modelos para a Internet do futuro. A plataforma permite ao experimentador operar o plano de controle, utilizando o Openflow[14]. O ambiente do FIBRE proporciona grande flexibilidade na criação e escalabilidade dos experimentos (FERREIRA et al., 2017), uma vez que os recursos virtuais podem ser alocados conforme a necessidade do experimentador.

Nota-se que as principais redes de experimentação permitem muita flexibilidade ao usuário, provendo controle de aplicações em vários níveis. Porém poucos ambientes fornecem o controle do plano de dados ao experimentador, que é o foco deste trabalho.

[13] https://fibre.org.br
[14] https://tools.ietf.org/html/rfc7426

Capítulo 3

PLANO DE CONTROLE E PLANO DE DADOS

Esse capítulo aborda os conceitos fundamentais do plano de controle e do plano de dados, que são a base deste trabalho. Contextualiza o Openflow, a linguagem P4 e a arquitetura PISA, explicando suas semelhanças e diferenças.

As redes de computadores são compostas de diversos dispositivos que operam sobre protocolos e arquiteturas de rede. O conceito de redes programáveis foi proposto com o objetivo de facilitar evoluções em implementações e soluções.

Historicamente, o modelo de arquitetura de roteamento pode ser definido pela Figura 3.1. Nota-se que o modelo é basicamente formado por 2 camadas, o *software* de controle e o **hardware** que realiza o encaminhamento dos pacotes entrantes no dispositivo. A integração do administrador do sistema com o equipamento é através de interfaces de configuração tais como SNMP[1], CLI (*Command-Line Interface* Web. Essa restrição limita o uso dos dispositivos às funcionalidades programadas pelo fabricante (NASCIMENTO et al., 2011).

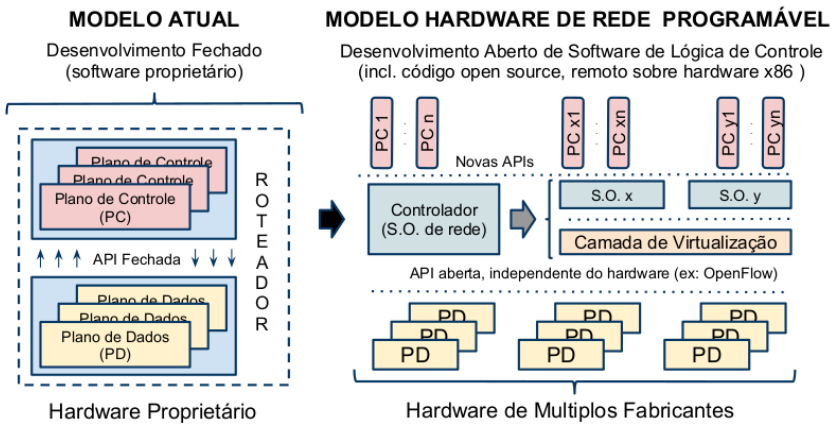

Figura 3.1: Arquitetura Clássica de Roteamento de Pacotes (NASCIMENTO et al., 2011)

.

[1]https://tools.ietf.org/html/rfc1157

Com o passar do tempo, observou-se a necessidade de maior flexibilidade em equipamentos de rede. Roteadores e *switchs* implementam arquiteturas fechadas, que são executadas em hardwares proprietários (ROTHENBERG et al., 2010).

Historicamente, as redes de computadores possuem o plano de controle vinculado ao plano de dados, ou seja, essa estreita ligação dificulta a depuração de configurações e a gerência de comportamento de equipamentos de rede (FEAMSTER; REXFORD; ZEGURA, 2014).

As SDNs são marcadas pela separação do plano de controle do plano de dados. São definidas como uma arquitetura gerenciável, emergente, e dinâmica, separando o controle da rede e as funções de encaminhamento (GARCIA et al., 2018).

Com o objetivo de melhor organizar o restante deste Capítulo segregou-se na Seção 3.1 onde se descreve especifidades do Plano de Controle; seguida pela Seção 3.2 que descreve o Plano de Dados e por fim na Seção 3.3 onde são relacionadas as principais diferenças entre o OpenFlow e P4.

3.1 Plano de Controle

Conceitualmente, um equipamento de rede, tal como um *switch* ou roteador pode ser dividido em dois planos, que são eles de controle e de dados conforme ilustrado na Figura 3.2. O plano de controle é responsável pela execução de códigos dos protocolos de roteamento, tais como as rotas.

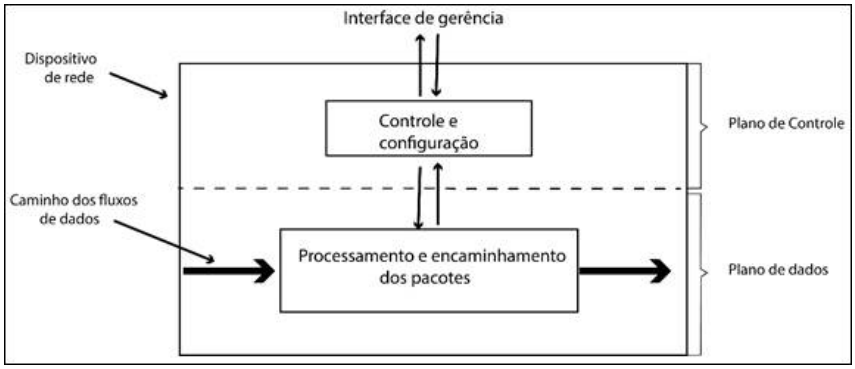

Figura 3.2: Plano de Controle e Plano de dados. (COMER, 2000)

.

A separação desses planos é benéfica no ponto de vista de inovação e evolução, uma vez que serviços podem ser implementados com mais facilidade, diminuindo o custo operacional das redes (COSTA et al., 2016).

3.1.1 OpenFlow

O Openflow é um protocolo de comunicação base para a criação de uma SDN. Foi criado em 2007 em parceira das Universidades da Califórnia e Stanford. Esse protocolo permite a manipulação e o encaminhamento de pacotes de dispositivos de rede, como roteadores e *switchs*. As regras e ações instaladas nos equipamentos são funções de um elemento externo chamado controlador, que pode ser implementado em um *hardware* comum, conforme ilustra a Figura 3.3.

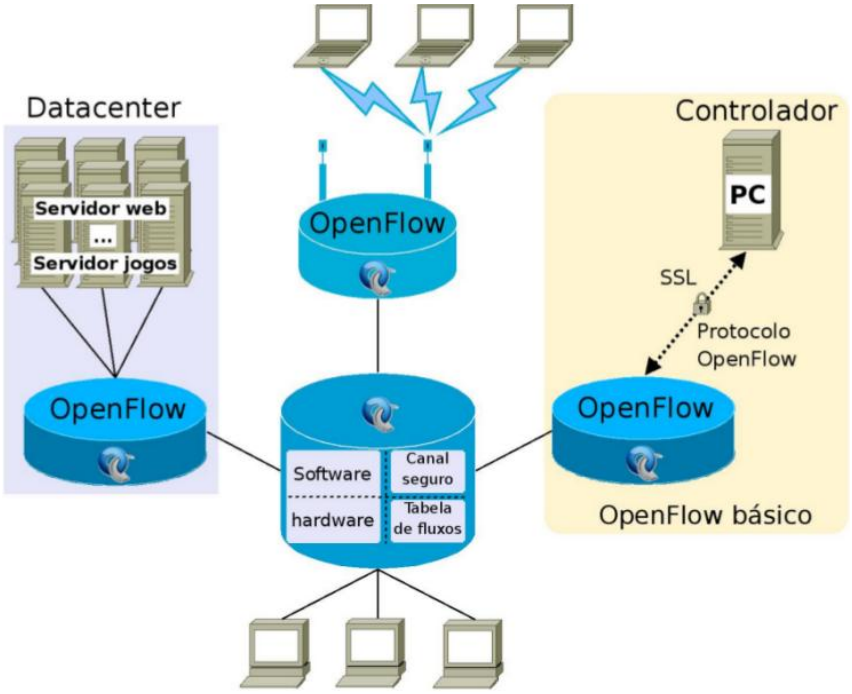

Figura 3.3: Exemplo de uma rede com Openflow. (BOSSHART et al., 2014)

.

O protocolo Openflow trabalha com o conceito de fluxo. Um fluxo pode ser definido como a combinação de campos do cabeçalho do pacote que será processado pelo dispositivo (ROTHENBERG et al., 2010). A contribuição mais importante do Openflow é capacidade de generalização do plano de dados.

Os equipamentos de rede que implementam o protocolo Openflow utilizam um controlador externo, confiando a ele as funções de encaminhamento de pacotes, os tipos híbridos suportam comandos de controladores juntamente ás operações e protocolos atuais (NUNES et al., 2014).

Data	Versão	Campos e Protocolos
Dez/2009	1.0	12 Campos - (Ethernet, TCP, IPV4)
Fev/2011	1.1	15 Campos - (MPLS, inter table metadata)
Dez/2011	1.2	36 Campos - (ARP. ICMP, IPV6)
Jun/2012	1.3	40 Campos
Out/2013	1.4	41 Campos
Dez/2014	1.5	44 Campos

Tabela 3.1: Comparativo de suporte de versões do Openflow. Adaptado de (BOSSHART et al., 2014).

Hoje, o Openflow nos permite adicionar e excluir entradas de encaminhamento para cerca de 50 tipos diferentes de cabeçalhos, na Tabela 3.1 têm-se um panorama da evolução no decorrer dos anos do número de campos e protocolos. Os fabricantes podem informar ao plano de controle quais cabeçalhos eles suportam usando o padrão TTP *(Table Type Patterns)*. Nota-se então que o Openflow controla o comportamento do equipamento de rede, mas sim preenche um conjunto de tabelas conhecida.

Esse conjunto de campos aumentou a flexibilidade de especificação de tabelas de *match* e *action*, porém o protocolo Openflow ainda não oferece a flexibilidade suficiente para adicionar novos cabeçalhos e especificar ações após um *flow matching* (GARCIA et al., 2018).

3.2 Plano de Dados

O plano de dados é responsável pela parte de processamento e encaminhamento de pacotes, conforme já ilustrado na Figura 3.2. O P4, abreviação em inglês para *(Programming Protocol independent Packet Processors)*, é uma linguagem de domínio de alto nível que permite programar processadores de pacotes independentes de protocolo (BOSSHART et al., 2014). A linguagem permite que a recuperação do plano de dados e a realização de funções otimizadas para um rede específica (GARCIA et al., 2018), na Figura 3.4 é possível observar o esquema de atuação do mesmo. O P4 surgiu com três principais objetivos, são eles:

(1) Reconfigurabilidade. Os programadores devem ser capazes de mudar a forma como os equipamentos de rede, redefinindo a análise e o processamento de pacotes.

(2) Independência de protocolo. Desanexar qualquer vínculo com formato de pacotes específicos. Em vez disso, o controlador deve ser capaz de especificar:

a) um analisador para extrair campos do cabeçalho do pacote com tipos e nomes específicos.

b) uma coleção de tabelas de correspondência juntamente com as ações que processam esses cabeçalhos.

(3) Independência do equipamento alvo. Desta forma o programador não precisa conhecer os detalhes da arquitetura subjacente. O compilador deve levar em conta as capacidades do equipamento de rede ao gerar um elemento de baixo nível independente de alvo.

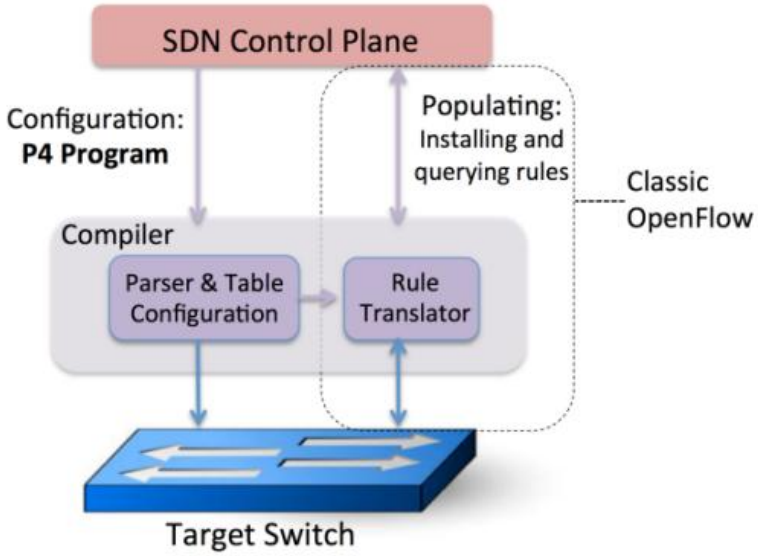

Figura 3.4: Exemplo de atuação do P4. (ROTHENBERG et al., 2010).

A linguagem é muito particular em experimentos de determinada natureza, como por exemplo o INT (KIM et al., 2015) *(In-band Network Telemetry)* que realiza uma abstração que permite que os pacotes de dados consultem o estado do equipamento de rede, como tamanho da fila, latência e atrasos no *link*.

Esse tipo de experimento lida com uma ordem de grandeza tão pequena que só é possível medir utilizando a linguagem P4. Cada comutador virtual de rede em L2 nesse exemplo possui um canal de controle que permite que o controlador P4 realize ações de inserção e exclusão, e modifique entradas nas tabelas de ação de correspondência. Essas APIs são geradas automaticamente pelo compilador P4 e fornecem *hooks* para executar tarefas em tempo de execução, como adicionar novas rotas.

A linguagem P4 torna a implementação de novos protocolos de redes mais simples. Por exemplo, uma implementação em P4 (PATRA; ROTHENBERG; PONGRÁCZ, 2016) para suportar uma rede de virtualização por sobreposição (VXLAN) requer poucas linhas de código.

Desta forma os usuários podem não só desenvolver novos protocolos, mas também remover protocolos não utilizados, otimizando os recursos do *hardware* para outros finalidades. Em suma, os protocolos de rede P4 podem ser escritos e implantados por operadores de rede e não por fabricantes de dispositivos de rede.

O experimento realizado por (SILVA et al., 2018) sobre identificação de fluxos em redes programáveis apresenta resultados obtidos nos testes experimentais de um protótipo P4, que se mostra significativamente mais eficiente em relação a uma solução equivalente implementada utilizando o protocolo OpenFlow. Principalmente no que diz respeito a identificação e reação de grandes fluxos (elefantes) em menos de 0.5 ms.

O P4 possibilita a utilização de protocolos ou campos nas tabelas de *match* e *action* definidos dinamicamente. O trabalho de (PACÍFICO et al., 2018) implementa uma arquitetura com eBPF em *hardware*, utilizando uma placa NetFPGA 1G. Esse modelo proposto evita recompilar ou reiniciar o equipamento de rede quando o usuário altera, em tempo de execução, a forma como os fluxos devem ser processados no *pipeline* de execução.

Notou-se então que os equipamentos de rede operam sempre com os mesmo protocolos e tecnologias, tais como IPv4, *Ethernet*, *Access Control Lists* (ACLs), etc. Se fosse possível definir uma interface padrão aberta para preencher as tabelas de encaminhamento nesses equipamentos de rede, seria possível criar planos de controle para controlar comutadores de uma variedade de equipamentos de rede diferentes.

3.3 Diferenças entre OpenFlow e P4

A linguagem P4 aborda um escopo mais genérico do que o OpenFlow. Com o surgimento do OpenFlow, em 2007, surgiu a necessidade de que os plano de controle de *software* controlassem remotamente vários *switchs* diferentes. O desafio era definir uma interface padrão para preencher as tabelas de encaminhamento. Deste modo seria possível criar planos de controle para gerenciar equipamentos de uma variedade de fabricantes diferentes e na Figura 3.5 pode ser observado um esquema de funcionamento do casamento P4 & Openflow.

Atualmente muitas redes internas, em sua maioria *data centers* são construídas usando planos de controle internos, são exemplos dessas arquiteturas de rede (MONSANTO et al., 2013) e (KIM et al., 2015). Essas redes programáveis alavancaram interfaces abertas para o plano de dados em parte ou no todo. Como já mencionado anteriormente, o OpenFlow não controla o comportamento de um equipamento de rede. A principal questão em relação ao protocolo é se ele seria capaz de suportar mais e mais cabeçalhos.

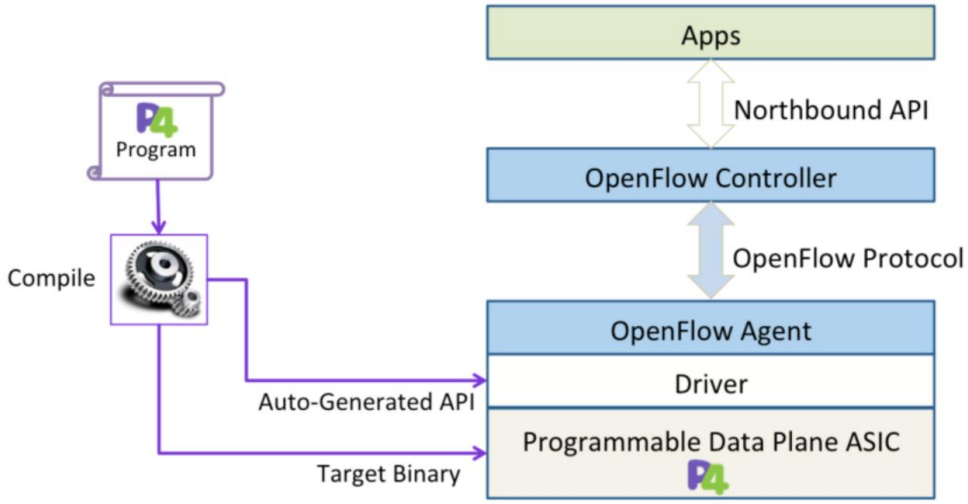

Figura 3.5: Atuação do P4 e do Openflow no Plano de Controle. Fonte: https://p4.org/p4/clarifying-the-differences-between-p4-and-openflow.htm

No passado, circuitos integrados de aplicação especifica só podiam processar pacotes a uma taxa de um décimo ou 1 centésimo da taxa de processamento de ASICs *(Application Specific Integrated Circuits)* de função fixa. O problema é que os *chips* comutadores não são programáveis, se fossem não existiria a necessidade de um protocolo fixo com o Openflow. Atualmente, existem chips de comutação reconfiguráveis como o FPGA, que processam pacotes com a mesma rapidez que os comutadores de função fixa mais rápidos.

Surgiu então a arquitetura PISA *(Protocol Independent Switch Architecture)*. O comutador PISCES é uma versão modificada do OVS com *parse*, *match* e *action* gerados pelo compilador P4 (PACÍFICO et al., 2018). Porém é preciso recompilar o código fonte do PISCES a cada alteração no código P4.

Vale ressaltar que a arquitetura PISCES não foi projetada para ser executada sobre um *hardware*, já o P4FPGA (WANG et al., 2017) foi instrumentado especialmente para FPGAs. A arquitetura é composta basicamente de 3 pilares, o otimizador, o gerador de código e o sistema *runtime*. O primeiro elemento proporciona paralelismo e visa aumentar a vazão e diminuir a latência. O sistema gerador de código gera um *pipeline* customizado para processar os pacotes entrantes. E o sistema *runtime* abstrai as funcionalidades do *hardware*.

Nesse sentido, um dos objetivos do presente trabalho é estender a arquitetura PISA, criando um ambiente de experimentação que permita processar qualquer tipo de pacote entrante, possibilitando e facilitando a criação de novas tecnologias e protocolos para a Internet do Futuro. O próximo capítulo apresenta o protótipo desse ambiente, bem como a sua arquitetura e todas as suas funcionalidades.

Capítulo 4

ARQUITETURA E PROTÓTIPO DE TESTBED

Este capítulo aborda os métodos que foram utilizados para estruturação de todo o testbed. A seguir serão apresentados todos os componentes, desde especificações de hardwares até os *softwares* utilizados para gestão e operação do ambiente de experimentação. Na Seção 4.1 consta o ciclo de vida do experimento; na Seção 4.2 o protótipo de *testbed*; na Seção 4.3 uma descrição detalhada da placa NetFPGA; na Seção 4.4 a descrição da máquina *WhiteBox*; na Seção 4.5 o detalhamento do *switch* Openflow e por fim na Seção 4.6 o detalhamento da biblioteca projetada para o experimento.

4.1 Ciclo de Vida do Experimento

A seguir são descritas as fases que compõe o ciclo de vida de um experimento. Basicamente o ciclo de vida do experimento consiste em 3 etapas, configuração, execução e resultados. Serão descritas cada uma das fases em detalhe tomando como base o ciclo de experimento ilustrado na Figura 4.1.

4.1.1 Fase de Configuração

Na primeira fase, a **Descoberta de Recursos** é responsável pelas permissões que o usuário possui para gerir os recursos do *testbed*. Quanto mais alto é o nível de privilégio, maior é o número de recursos que o usuário tem disponível para operar. Não é possível multiplexar as placas FPGAs, visto que o *hardware* é reconfigurado, sendo possível apenas que um único usuário faça uso destes recursos por experimento.

Na fase de **Descrição do Experimento** o usuário configura e orquestra como será o seu experimento e pode fazer isso de duas formas, na primeira delas, de forma mais simples, o usuário

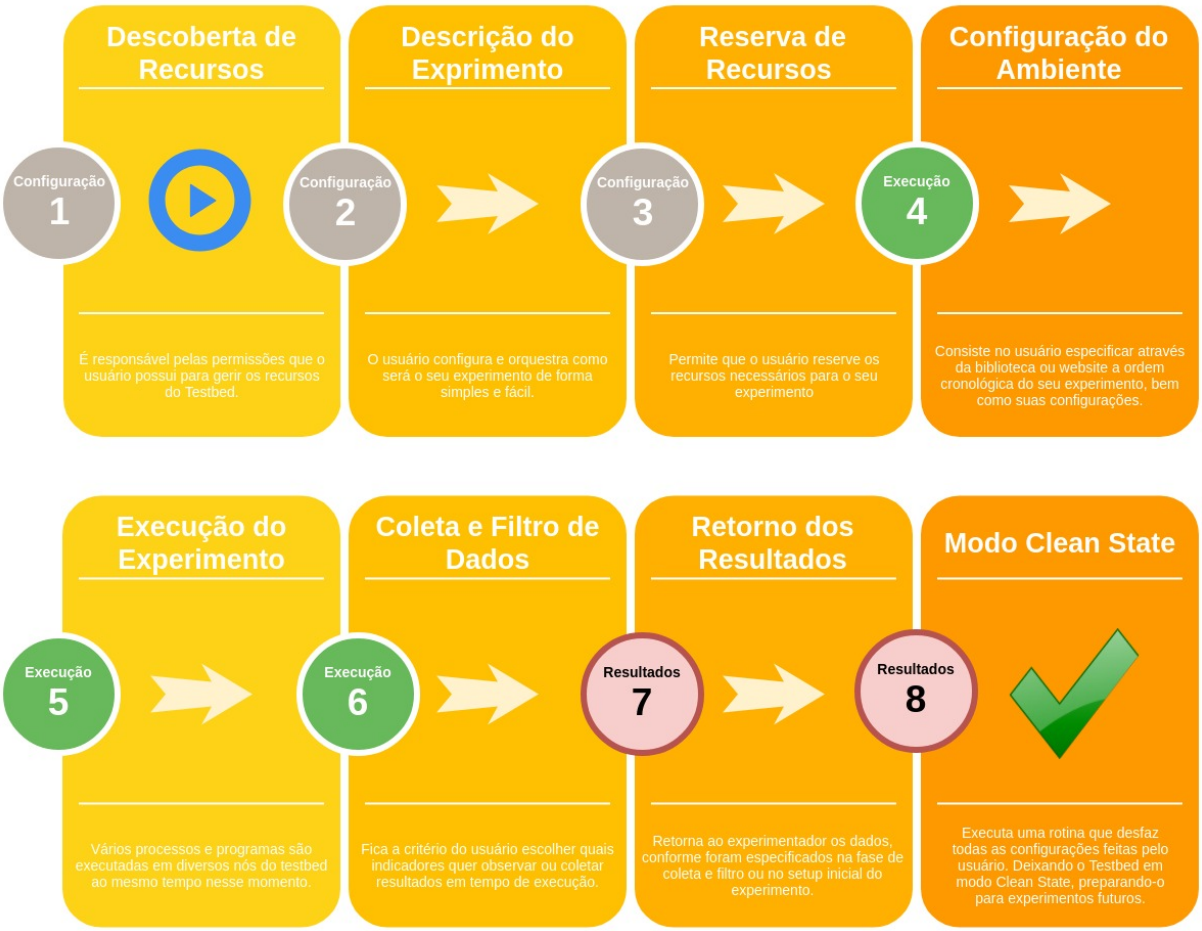

Figura 4.1: Ciclo de Vida do Experimento.

pode utilizar a plataforma gráfica web, e arquitetar seu experimento. Essa plataforma permite que o usuário configure o seu experimento de uma forma simples e fácil, quais máquinas deseja utilizar e o qual tipo de topologia. Assim como descarregar seus arquivos de reprogramação e receber os resultados pela mesma interface. O usuário também tem a opção de programar manualmente o descritor de experimentos ED (*Experiment Description*) usando a linguagem de programação Ruby.

A **Reserva de Recursos** permite que o usuário reserve os recursos necessários para o seu experimento, lembrando que ele não é capaz de reservar recursos aos quais não tem permissão, ou que já foram agendados para outro experimento que ocorrerá simultaneamente ao seu. Alocações de recursos permitem que os recursos reservados sejam disponibilizados para os experimentos em horários especificados, definidos pelo usuário.

4.1.2 Fase de Execução

A fase **Configuração do Ambiente** consiste no usuário especificar (através da biblioteca ou *website*) a ordem cronológica do seu experimento, bem como suas configurações, tais como definir quais interfaces físicas vai utilizar, definir os endereços IP de cada interface, configurar uma ou mais VLANs, etc. Como mencionado anteriormente, esse *setup* pode ser feito pela interface *web*, ou usando ama biblioteca de controle de recursos, como a que apresentaremos neste trabalho. De qualquer forma será gerado o mesmo tipo de *script* ED, ao qual o OMF é capaz de interpretar e executar.

Durante a **Execução do Experimento**, vários processos e programas são executadas em diversos nós do *testbed* ao mesmo tempo. Fica a critério do usuário escolher quais indicadores quer observar ou coletar resultados, isso acontece na fase de **Coleta e Filtro de dados**. Em exemplo desta etapa: o experimentador deseja ver um registrador de determinada NetFPGA ou o fluxo de dados em determinada interface de uma máquina Whitebox.

4.1.3 Fase de Resultados

Por fim, a fase de **Retorno dos Resultados** retorna ao experimentador os dados, conforme foram especificados na fase de coleta e filtro.

O **Modo Padrão de Operação**, permite que uma rotina desfaça todas as configurações feitas pelo usuário. Deixando o *testbed* em modo *Clean State*,e já prepara o ambiente para experimentos futuros.

4.2 Protótipo

Foi desenvolvido dentro do contexto do projeto FIBRE, um protótipo de *testbed* que permite experimentação do plano de dados de uma rede, bem como reprogramação de NetFPGA. Para isso utilizou-se um conjunto de equipamento, dentre eles, 9 máquinas com placas FPGA, 5 máquinas Whitebox, 2 *switchs* Openflow e 1 servidor de controle e orquestração. Descreveremos a seguir cada um desses componentes.

No contexto de *testbeds* existem muitos arcabouços para controle e gestão de ambientes de experimentação, como o OCF (*Ofelia Control Framework*) ou o Openstack. Uma plataforma interessante para realizar experimentos em plano de dados é o OMF (*Control Monitoring Framework*), que proporciona amplas possibilidades ao experimentador para criação de um experi-

mento, e torna o processo experimental mais seguro, visto que o usuário não tem permissão total (superusuário) das máquinas, assim, o usuário pode executar apenas alguns comandos que são disponibilizados para execução do experimento, tornando o experimento passivo de auditorias e futuras replicações. A plataforma possui uma API *(Application Programming Interface)* que permite e simplifica a criação de controladores, essa interface abstrai a cama de comunicação, possibilitando reprogramações em *hardwares* de qualquer natureza.

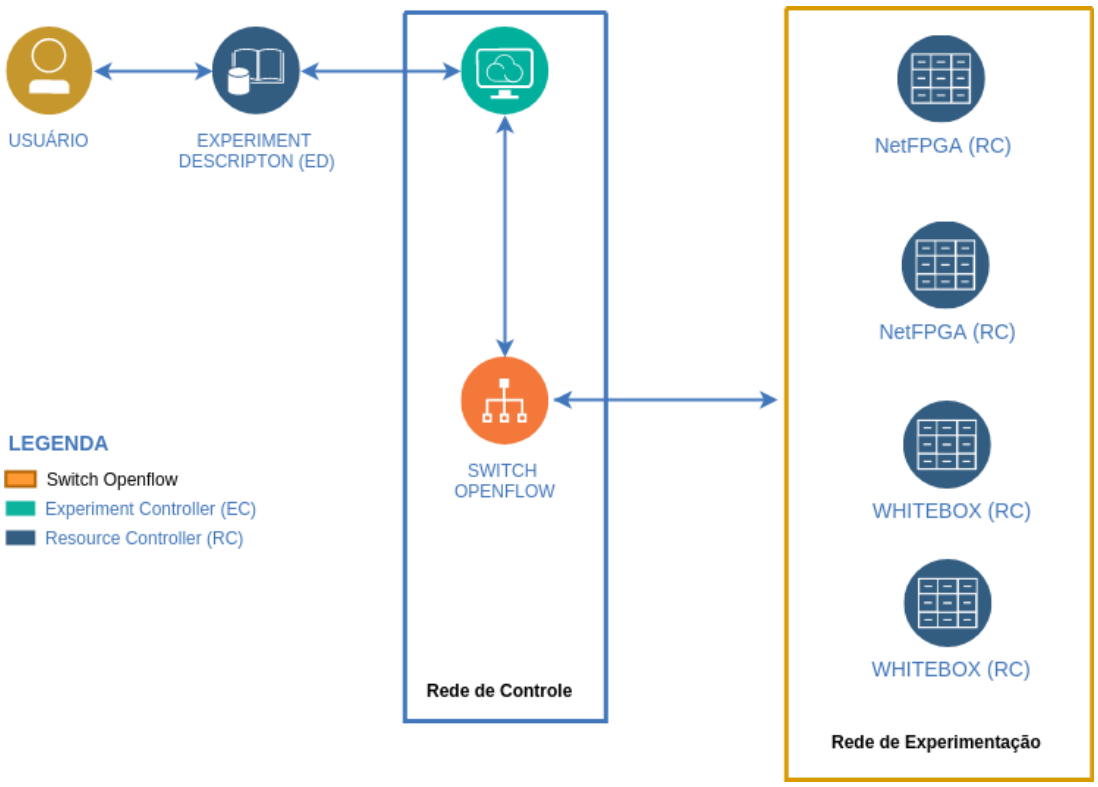

Figura 4.2: Arquitetura do Arcabouço de Controle.

O *testbed* utiliza arcabouço de controle OMF na versão 6 para gerência e orquestração dos experimentos. O *software* foi desenvolvido pelo equipe NICTA *(National ICT Australia)*, e foi escrito na linguagem Ruby, baseado em XMPP (*Extensible Messaging and Presence Protocol*). Esse tipo de *framework* é focado em oferecer facilidades para realizar registro, autenticação e autorização de usuários, descoberta dos recursos disponíveis, reserva e liberação dos recursos (SILVESTRE; CARDOSO; CORRÊA, 2012), desta forma a experimentação propriamente dita fica a cargo do experimentador.

Para controlar de maneira única os componentes do plano de dados e permitir reprogramação e controles das máquinas, foram desenvolvidos componentes de softwares baseados em OMF que encapsulam função de baixo nível. Todas as máquinas possuem um módulo controlador do OMF chamado RC (*Resource Controller*), esse módulo permite operar ações oriundas do ED (*Experiment Description*), e que são comandadas pelo EC (*Experiment Controller*). O OMF

permite que o usuário escreva um ED, que por sua vez, é submetido ao EC, que é responsável pelo controle do projeto em nome do usuário.

O EC emite solicitações no plano de gerenciamento para configurar os recursos conforme especificado no ED. Uma vez que os pré-requisitos da experiência são atendidos, o EC envia diretivas ao RCs associados a cada recurso, os RCs são todos os recursos disponíveis no *testbed*. Esses recursos vão desde máquinas NetFPGAs até o switchs Openflow. A configuração dos recursos também é feita pelo RC, seguindo as diretrizes que foram criadas pelo experimentador em seu ED.

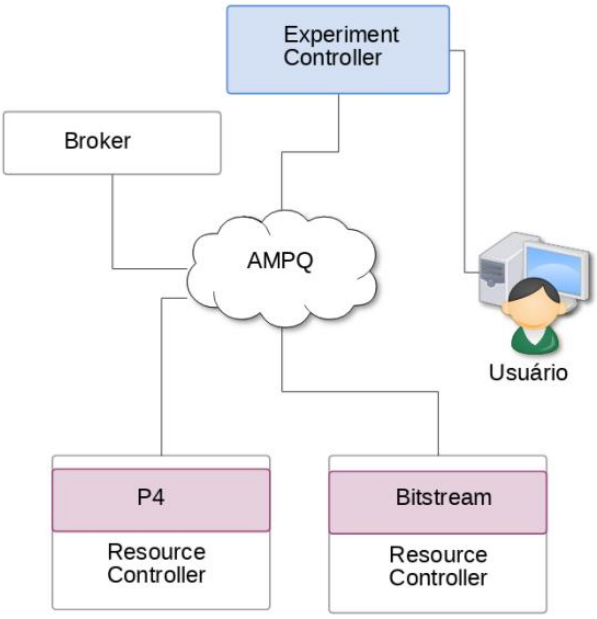

Figura 4.3: Arquitetura de Controle do OMF.

O *software* de mensageria RabbitMQ é utilizado para criar canais e filas de comunicação entre os RCs, ele se comunica através do protocolo AMQP, e conecta todos os nós do testbed, basicamente ele atua como um *middleware* orientado a mensagens. O Broker tem o papel de conferir as permissões que os usuários tem para alocar determinados recursos para experimentação.

4.3 Placa NetFPGA

O primeiro modelo de placas programáveis criado pela Xilinx foi a NetFPGA 1G, O projeto surgiu em 2006, em parceria com alunos de doutorado da Univerdade de Stanford. Com quatro portas *ethernet* de 1Gb cada, a placa possui um processador FPGA Xilinx Virtex-II Pro 50 com dois núcleos PowerPC, para processamento dos pacotes, 53.136 elementos lógicos e ciclo de relógio de 8ns (125 MHz). A placa utiliza um barramento PCI ou PCI *Express* para

comunicação e programação do FPGA.

A arquitetura FPGA (*Field-Programmable Gate Array*) ou arranjo de portas programável, é uma união de memória e processadores de sinais com quatro portas *ethernet* de 1Gbps. A *testbed* conta com 9 servidores com placas FPGA integradas. Os servidores contém 2 HDDs (*Hard Drive Disk*) de 500GB cada, com um mecanismo de tolerância a falhas e automação, que será abordado no capítulo 6, ainda conta com um processador Intel Core 2 Quad 2.66GHz, com 4 núcleos e 8GB de memória RAM DDR3 com 1334 MHz. A documentação oficial do conjunto de *drivers* e programas da NetFPGA 1G utiliza o sistema operacional Fedora, na versão 13, para sua operação.

Pelo fato de ser uma distribuição antiga, e já descontinuada, todos os *drivers* e *sofwares* foram adaptados para uma distribuição mais recente, do CentOs 7, na arquitetura 64 bits. A portabilidade do conjunto de *softwares* foi possível visto que ambos os sistemas originam da distribuição Red Hat Linux.

A placa NetFPGA dispõe de dois bancos de memórias SRAM *Cypress* (Static Random Access Memory) ou memória estática de acesso aleatório, possuindo um espaço total de 4608KiB. A memória SRAM pode realizar operações de I/O (Input/Output) por ciclo de *clock*, e retorna os dados em três ciclos de clock. A placa NetFPGA dispõe ainda de dois *slots* de memória DRAM Micron (Dynamic random access memory) ou memória de acesso aleatório dinâmico, totalizando 64MiB de memória. Esse tipo de memória trabalha de forma assíncrona, portanto necessita de uma atualização contínua dos dados. A memória DRAM é ideal para aplicações que precisam de uma quantidade maior de memória, como armazenamento temporário de pacotes, e também permite montar uma hierarquia de memória da NetFPGA (GOULART et al., 2015).

A NetFPGA evolui sua tecnologia para modelos mais potentes, mais velozes e com maior capacidade de memória, porém com uma arquitetura de hardware similar, que obedece os mesmos conceitos da placa NetFPGA 1G. As versões atuais contam com interfaces de 10 Gbps e até 100 Gbps. O objetivo desse projeto é explorar e utilizar os recursos da NeswitchOftFPGA 1G, as técnicas, conceitos e métodos utilizados na NetFPGA 1GB podem ser estendidos e aplicados aos demais modelos de placas NetFPGAs disponíveis no mercado.

O *software* ISE[1] da Xilinx permite sintetizar os projetos feitos em Verylog, de modo que a saída seja um arquivo do tipo *bitstream* e, desta forma, possa reprogramar o FPGA. O *software* também tem como objetivo para lidar com as limitações de *hardware* da plataforma (BILAR GUILHERME, 2016). Um conjunto de interfaces e abstrações contidos no software gerenciam

[1] https://www.xilinx.com/products/design-tools/ise-design-suite/

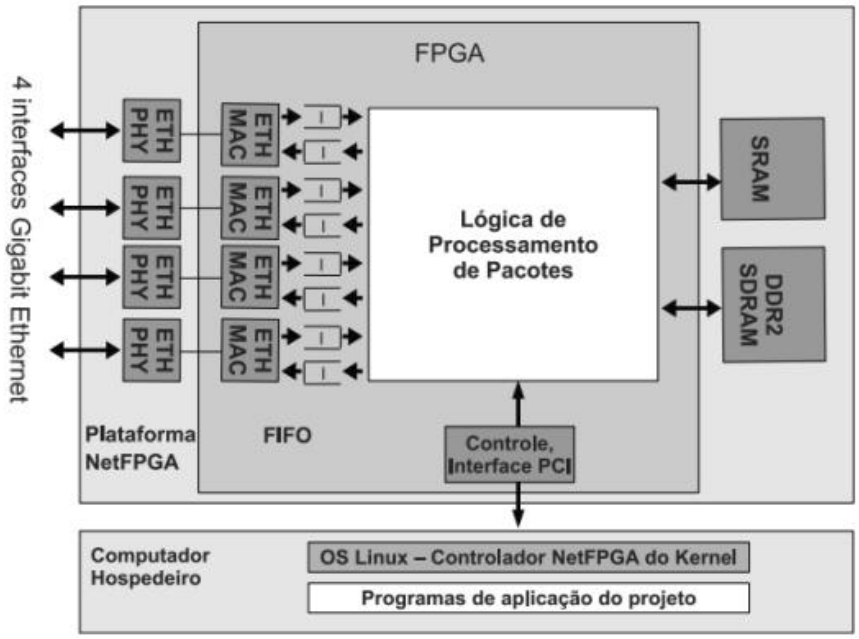

Figura 4.4: Componentes da Placa NetFPGA. Fonte: (GOULART et al., 2015).

a manipulação de recursos e possibilitam a comunicação entre diferentes módulos. O sistema é compatível com as distribuições Windows e Linux, porém é recomendado o uso da segunda arquitetura.

A placa NetFPGA pode ser configurada de acordo com os projetos de referência, esses projetos fazem com que a placa funcione como um equipamento de rede, tais como roteador, *switch* ou uma placa de rede *ethernet*. Esses projetos dão suporte para o desenvolvimento de outros projetos mais complexos.

A base dos projetos de referência é o chamado reference_nic, que nada mais é do que uma placa de rede *ethernet*. As 4 interfaces físicas da placa NetFPGA são mapeadas pelas distribuições Linux como **nf2cx**, onde a variável x varia de 0 até 3, e tem a mesma função de outras interfaces de rede comuns do sistema, tais como eth0, wlan0, etc.

Os pacotes que chegam da rede podem ser recebidos pela interface *ethernet* da placa FPGA, ou ainda, pelo barramento PCI, que conecta a placa NetFPGA na placa mãe. Então, o **reference_nic** instancia oito módulos **rx_queue** para gerenciar o recebimento dos pacotes nas interfaces. Os módulos **rx_queue** e **tx_queue** fazem a interface entre a placa FPGA, o controlador MAC e o controlador PCI (GOULART et al., 2015).

Dentro de cada máquina NetFPGA foi desenvolvido um módulo controlador do OMF chamado RC, esse módulo permite operar ações oriundas do ED, e que são comandadas pelo EC. O módulo presente na NetFPGA permite que o experimentador faça *upload* de um arquivo do tipo

bitstream, que é capaz de reprogramar a placa FPGA a cargo do usuário. Essa funcionalidade dá total flexibilidade, uma vez que pode ser utilizada para desenvolvimento de protótipos de pesquisa, implementação e testes de novos protocolos ou sistemas de rede.

Essa *feature* garante mais controle para o usuário e menor latência no acesso aos dados, já que evita etapas de processamento via camadas de *software*. Para aplicações sensíveis a desempenho e velocidade, o *bare-metal* pode trazer ganhos em relação aos servidores virtualizados.

4.4 Máquina Whitebox

A Whitebox serve como um componente para regular o fluxo de dados de acordo com o controlador SDN *(Software Defined Networks)*. Ao contrário do OVS *(Open Virtual Switch)* convencional, a Whitebox é apenas um *switch* cru e vazio que não tem inteligência. Para poder operar, as máquinas Whitebox requerem um software de comutação virtual que possa ser incorporado diretamente.

A Whitebox trabalha como um componente para regular o fluxo de dados de acordo com o controlador SDN. Ao contrário OVS comum, a Whitebox é apenas um *switch* sem um controlador. Para funcionar como switch inteligente, requerem um software de comutação que possa ser configurado pelo controlador SDN, (MANGGALA; TANWIDJAJA et al., 2015).

As máquinas possuem todas suas placas de rede *ethernet* mapeadas em um *switch* OVS, com aceleração via *Data Plane Development Kit* (DPDK). Os avanços recentes em interfaces de rede permitem o aumento desempenho, devido ao processamento de pacotes de baixa latência usando o DPDK, essa estrutura permite que as aplicações recebam dados diretamente das interfaces de rede (HWANG; RAMAKRISHNAN; WOOD, 2015), eliminando em grande partes os gargalos do processamento comum de pacotes no nível do sistema operacional baseado em interrupção.

O *software* que recebe o arquivo compilado da linguagem P4 é uma extensão do projeto PISCES *(Programmable Protocol-Independent Software Switch)* (SHAHBAZ et al., 2016b), esse tipo de arquitetura possibilita especificar como os pacotes serão processados e encaminhados, através de um domínio específico de alto nível.

Isso permite de forma rápida, modificar o comportamento de encaminhamento dos pacotes desses *switchs* virtuais. Atualmente a modificação nas arquiteturas desses equipamentos exigem conhecimento profundo no código do *switch* e ampla experiência em desing de protocolos de rede e Internet. O Projeto PISCES permite grande otimização grande, são cerca de 40 vezes

mais concisos (SHAHBAZ et al., 2016b) do que os programas equivalentes em código nativo para a troca de software.

Além disso, as Whitebox também devem receber ordens do controlador SDN (MANGGALA; TANWIDJAJA et al., 2015) para poderem funcionar. Essa é a diferença mais fundamental entre as Whitebox e os *switchs* convencionais.

O *testbed* dispõe 5 máquinas Whitebox, que possuem 8 placas de rede *ethernet* Gigabit, 1 disco SSD 256GB, 8GB de memória RAM DDR3 HyperX 1866MHz, com processador Intel Atom CPU C2758 2.40GHz com 6 núcleos e placa mãe A1SRM-LN7F-2758. O sistema operacional utilizado é o Ubuntu 17.04, na arquitetura 64 bits.

Todas as máquinas Whitebox contém um controlador RC, que tem função semelhante ao controlador as máquinas NetFPGAs, ele permite que o experimentador faça *upload* de um arquivo com extensão P4. A linguagem de programação P4 permite controlar o *dataplane* do *switch* virtual contido na máquina Whitebox. Desta forma, o experimentador pode programar e controlar o *pipeline* do OVS. Vale ressaltar que não existe nenhuma camada de virtualização onde os comandos dos controladores (RCs) são executados, ou seja, o experimento ocorre em *bare-metal*.

Para aplicações sensíveis a desempenho e velocidade, o *bare-metal* pode trazer ganhos em relação a instâncias virtualizadas.

4.5 Switch Openflow

O arcabouço de controle OMF trata todo hardware conectado ao *testbed* como recurso. Os equipamentos de rede funcionam da forma análoga, e para que seja colocado em operação também é necessário um controlador (RC) presente e configurado, com módulos que gerenciam os experimentos.

O ambiente possibilita ao usuário fazer experimentos usando o protocolo Openflow, o *hardware* utilizado para isso é o *switch* Datacom DM4100, de 48 portas. Esse mesmo equipamento é responsável pela comunicação entre todas as máquinas do ambiente.

Fica a cargo do experimentador determinar qual será a topologia lógica de seu experimento. A topologia física foi arquitetada de modo que fosse possível para o experimentador montar qualquer topologia lógica. O módulo do RC responsável por essa configuração e comunicação com o *swtich* openflow fica dentro do EC, visto que o *switch* Openflow não tem um sistema operacional capaz de interpretar os códigos do OMF. Então os comandos de configuração de

topologia são enviados, via telnet do EC para o Pronto, como por exemplo, configurar uma VLAN (Virtual Local Area Network) entre duas máquinas, ou uma regra Openflow.

4.6 Biblioteca Netfpga-OMF

Como parte do projeto, e de modo a diminuir o tempo de instanciação dos experimentos, foi projetada uma biblioteca que permite encapsular funções de baixo nível para instanciação de experimentos. Deste modo, o usuário apenas invoca as funções no seu descritor de experimento, e informa os valores ou arquivos como parâmetro juntamente com a chamada da função. Abaixo temos um exemplo de uma função configurada nos controladores das máquinas NetFPGAs, essa função faz com que os *gates* da placa FPGA seja reprogramados, através de um arquivo binário do tipo *bitstream*.

```
1  # Configure the :upload_bit
2
3  configure :upload_bit do |netfpga1, value|
4    info 'Configure(upload_bit) called'
5    #value = netfpga1.execute_cmd("/root/ls -l >> /root/SAIDA.txt ")
6    netfpga1.execute_cmd("sshpass -p '\$passwd' scp \
7        ec-controller@#{netfpga1.property.ip_ec}:#{value} \
8        #{netfpga1.property.bit_name} /tmp").delete("\n")
9  value
10 end
11
12 # Configure the :upload_conf
13
14 configure :upload_conf do |netfpga1, value|
15   info 'Configure(upload_conf) called'
16   netfpga1.execute_cmd("sshpass -p '\$passwd' \
17   scp ec-controller@#{netfpga1.property.ip_ec}: \
18   #{value}/#{netfpga1.property.conf_name} /tmp").delete("\n")
19 value
20 end
21
22 # Configure the :run_bit
23
24 configure :run_bit do |netfpga1, value|
25   info 'Configure(run_bit) called'
26   netfpga1.execute_cmd("/usr/local/sbin/cpci_reprogram.pl --all").delete("\n")
27   netfpga1.execute_cmd("nf_download \
28   /tmp/#{netfpga1.property.bit_name}").delete("\n")
29   netfpga1.execute_cmd("/root/netfpga/projects/selftest/sw/selftest -n >> \
30   }/tmp/result_netfpga1.txt").delete("\n")
```

```
31      netfpga1.execute_cmd\
32      ("sshpass -p '\$password' scp /tmp/return_iperf.txt \
33      ec-controller@#{netfpga1.property.ip_ec}:#{value}").delete("\n")
34      value
35    end
36  end
```

Um dos objetivos do trabalho é diminuir o tempo de criação e instanciação do experimento, aumentando a expressividade e sem restrições funcionais ao usuário. O primeiro passo é a reprogramação da placa. Para que outro arquivo de mesma espécie possa ser reconfigurado. O segundo passo é descarregar o arquivo que irá programá-la.

O usuário também pode optar por enviar configurações que serão feitas após o arquivo *bitstream* reprogramar a placa. Um exemplo é o projeto do roteador de referência, onde após a reprogramação da placa, é preciso inserir as rotas nas tabelas de roteamento, esses parâmetros podem ser enviados através da função *upload_conf* que já está encapsulada na função *configure* que será exemplificada nos Resultados Preliminares do Capítulo 5. Nota-se que com poucas linhas de código, é possível que o usuário reprograme uma máquina NetFPGA com um arquivo do tipo *bitstream* de sua escolha, utilizando a biblioteca NetFPGA-OMF. Abaixo temos a função que encapsula o código em Ruby.

```
1  {reprogram} netfpga1 -> selftest.bit;
```

Capítulo 5

DETALHAMENTO DOS MÉTODOS E FERRAMENTAS

Neste capítulo vamos apresentar os sistemas de automação, controle, provisionamento, tolerância a falhas e monitoramento que foram implementados no *testbed*. Cada uma destas técnicas, métodos e ferramentas será descrita a seguir.

Para que os elementos reprogramáveis de plano de dados baseados em FPGA e PISA operem adequadamente no *testbed* é necessário a instalação e configuração de um conjunto de *softwares*, dentre eles sistemas operacionais, pacotes, aplicativos e suas configurações. Para facilitar o provisionamento desse conjunto de softwares em caso de falhas, foram estudados e analisados algumas ferramentas de automação como Apache Ant[1], Puppet[2] e Ansible [3].

O Ansible é um *framework* de automação e configuração de sistemas que trabalha com a arquitetura cliente-servidor. No arcabouço escolhido não há necessidade de agentes instalados nos clientes, portanto não existem pacotes circulando, e a rede não fica sobrecarregada quando o Ansible não está em execução. É disponibilizado em duas versões, a primeira, chamada de *Ansible Core*, e o *Ansible Tower*, ambas são *opensource*, sendo a última, uma versão comercial.

O Ansible pode usar o protocolo NetConf[4] para comunicação com equipamentos de rede, tais como roteadores e *switchs*. A ferramenta então estabelece uma conexão segura entre as máquinas via SSH[5], para garantir a confidencialidade dos dados que serão transmitidos. Alguns módulos permitem o envio de comandos para a CLI (*Command-Line Interface*) dos equipamentos como roteadores e *switchs*. A plataforma utiliza arquivos de configuração em formato

[1] https://ant.apache.org
[2] https://puppet.com
[3] https://www.ansible.com
[4] https://tools.ietf.org/html/rfc6241
[5] https://tools.ietf.org/html/rfc4253

YAML[6], e utiliza o formato Jinja2[7] para *templates* de configuração

Essas características sustentam a escolha do Ansible como ferramenta de apoio ao provisionamento a automação do conjuntos de *softwares* do *testbed*.

O Ansible permite que esses conjunto de pacotes e aplicativos necessários para operar a *testbed* sejam instalados simultaneamente em todas as máquinas, independente do sistema operacional. Para tanto, foram desenvolvidos *playbooks*, que funcionam como um *cookbook*, onde vários conjuntos de tarefas são executadas em ordem cronológica, fazendo com que os processos de reestruturação e instalação sejam feitos num intervalo de tempo menor.

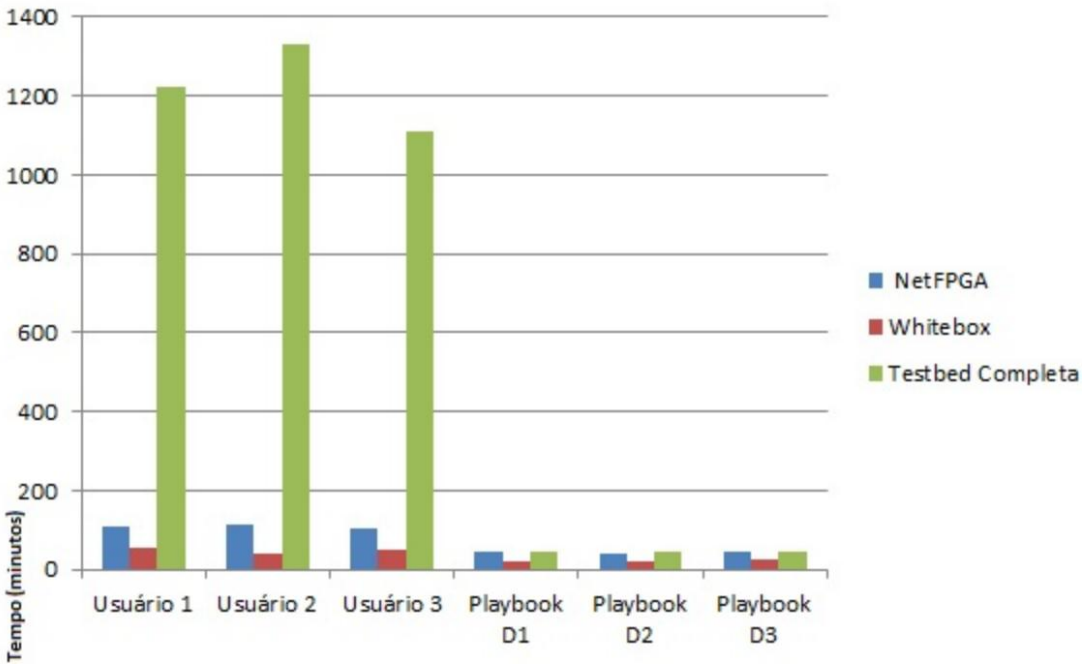

Figura 5.1: Comparativo de Tempo de Provisionamento do *testbed*.

Na Figura 5.1 segue um comparativo de tempo de instalação, onde três usuários realizam a instalação manual do conjunto de programas. Cada usuário repete por três vezes, em dias diferentes, a instalação, isso foi feito para que a velocidade da Internet no momento da instalação não influencie nos resultados do comparativo. O resultado da tabela é o valor da média de tempo dos três dias de experimento de cada usuário.

Os *playbooks* também foram executados em três dias diferentes, e assim como dos usuários, o valor da tabela é a média de tempo dos três dias de instalação, seguidos pelo desvio padrão. O valor de tempo informado é referente a uma máquina de cada tipo.

O Ansible Galaxy faz referência ao site do Galaxy conforme visualizado na Figura 5.2,

[6]http://yaml.org/spec/history/2001-05-26.html
[7]http://jinja.pocoo.org

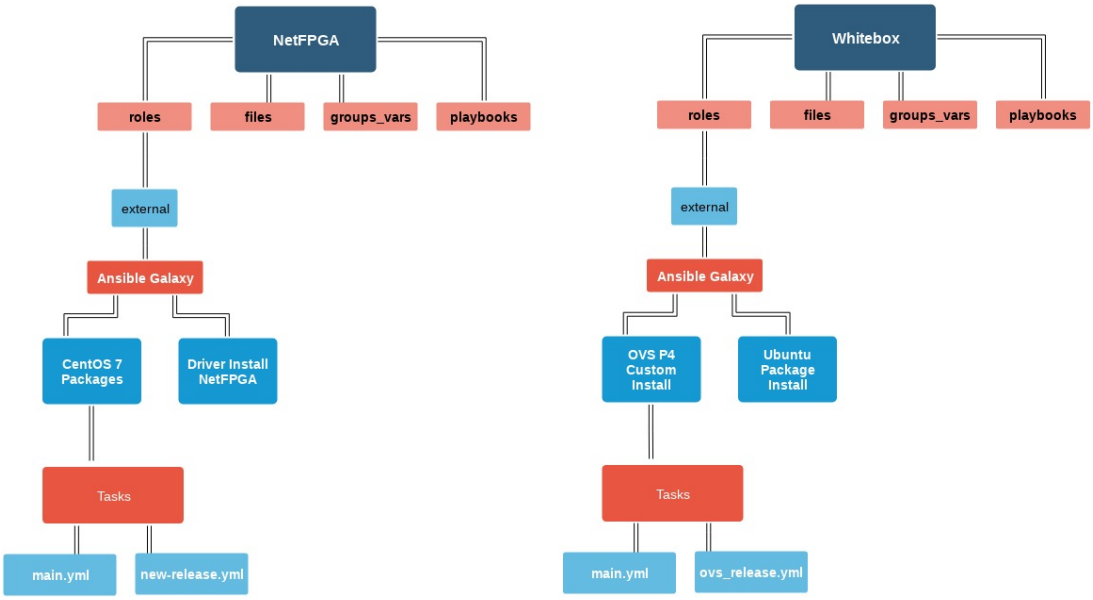

Figura 5.2: Diagrama do Funcionamento do Ansible Galaxy nos Playbooks.

no qual os usuários podem compartilhar funções e a uma ferramenta de linha de comando que permite instalar, criar e gerenciar funções. A função *role* permite o compartilhamento e a reutilização de tarefas Ansible. O objetivo é que os *playbooks* não tenham mais *task*, somente *roles*. Que seriam centralizadas e reutilizadas para as mesmas operações nas diferentes máquinas do *testbed*.

A estrutura de armazenamento dos *playbooks* faz com que as *roles* sejam armazenadas externamente em repositórios individuais no GitHub. O *download* das *roles* para os *playbooks* dos projetos ocorrerão via Ansible Galaxy. A escolha da *task* desejada será realizada por meio de variáveis.

Nota-se, conforme visualizado na Tabela 5.1, grande vantagem ao utilizar o Ansible para instalação e provisionamento, pelo fato de todas as instalações serem feitas simultaneamente, tanto nas máquinas NetFPGAs, quanto nas máquinas *Whitebox*. O tempo médio aproximado de instalação de todo o *testbed* seria de 44 minutos. Enquanto que o tempo total médio e estimado gasto para instalação manual seria de 1222 minutos. A estimativa de tempo de instalação é feita com base no tempo médio da instalação dos usuários, multiplicando cada tempo pelo número de máquinas presentes de cada tipo no *testbed*. Os valores informados podem variar de acordo com a conexão com a Internet.

Instalação Conjunto Sotwares	NetFPGA (Média de Tempo em Minutos)	Whitebox (Média de Tempo em Minutos)	Total Instalação Testbed
Usuário	110 (sd:8)	55 (sd:6)	1222 *
Playbook Ansible	45 (sd:2)	23 (sd:1)	45 *

Tabela 5.1: Tempo de Provisionamento do *testbed*.

5.1 Gestão e Desenvolvimento dos Códigos dos Controladores

O fluxo de trabalho Git pode ser representado pela estrutura da Figura 5.3.

Branch Master: Responsável por manter todo o ambiente de produção atual.

Branch Development: Responsável por manter o ambiente de desenvolvimento.

Branch Hotfix: Responsável por corrigir erros emergenciais no ambiente de produção.

Branch Feature: Responsável por manter testes individuais dos desenvolvedores, evitando *mergers* com *bugs* para o ambiente de desenvolvimento.

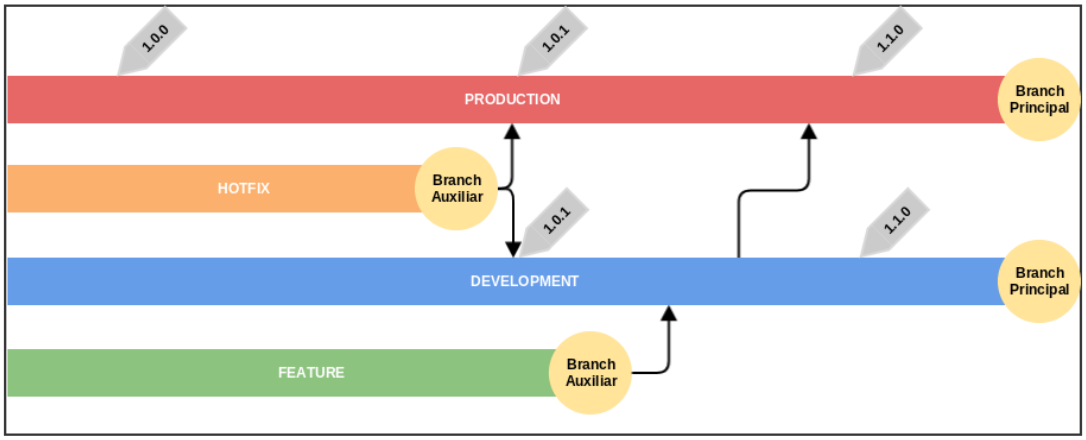

Figura 5.3: Estrutura de Versionamento dos Códigos dos Controladores do Testbed.

A estrutura obedece um versionamento semântico estruturado: X.Y.Z (Maior.Menor.Correção).

Maior - Alterações incompatíveis com as versões anteriores da API devem ser incrementadas nesse campo.

Menor - Adições e alterações compatíveis com as versões anteriores da API devem ser alteradas nesse campo.

Correção -Falhas *(bugs fixes)* que não afetam o funcionamento da API, devem ser incrementadas nesse campo.

5.2 Tolerância a Falhas

O *testbed* conta com um sistema de redundância de discos HDDs nas suas máquinas. As máquinas NetFPGAs estão configuradas em RAID 1 (*Redundant Array of Independent Disks*), onde o conteúdos dos discos é distribuído igualmente entre eles. Isso torna o sistema resiliente

e aumenta a tolerância a falhas, visto que, quando houver alguma falha em um dos discos, prontamente o sistema passa a utilizar com o outro, mantendo a operação do *testbed* sempre em funcionamento.

De forma análoga, também é possível fazer a instalação do sistema operacional via KVM Raritan, basta que uma mídia esteja conectada ao dispositivo de controle, e esta, contenha a nova distribuição a ser instalada. Então, basta referenciar no BIOS (Basic Input/Output System) o caminho para a nova mídia de instalação.

Para gerencia e controle, o *testbed* dispõe de um KVM (*Kernel-based Virtual Machine*) Raritan DLX-116, com 16 portas *ethernet* para controle remoto, equipado com mídia virtual para permitir tarefas remotas, como instalação de *software*, inicialização remota e diagnósticos. A rede de gerência opera em uma rede diferente da rede de experimentação, haja visto que se houver uma falha no *link* principal de Internet, o controle remoto via Raritan permaneceria disponível. A disposição física do aparelho na topologia pode ser observada na figura abaixo.

O KVM utiliza o bloqueio de repetição de senha e o padrão avançado de criptografia AES[8]. É possível a integração junto ao Active Directory, usando LDAP[9] ou RADIUS[10].

Todas as máquinas possuem uma interface IPMI (*Intelligent Platform Management Interface)* para gerência, o elemento de *hardware* responsável por essa tecnologia é o BMC (*Baseboard Management Controller).* Essa interface de rede inteligente opera independente ao sistema operacional, podendo controlar até as configurações do BIOS, o sistema de monitoramento IPMI funciona mesmo na ausência do sistema operacional.

5.3 Sistema de Monitoramento

O sistema operacional também pode ser instalado remotamente, caso haja necessidade, e pode ser feitos de duas formas, a primeira delas seria usando o *framework* de *deploy* Cobbler, que é um *software* que permite a rápida configuração e instalação de sistemas operacionais Linux, utilizando o ambiente PXE (Preboot Execution Environment).

No contexto de soluções para monitoramento, existem muitos arcabouços para controle e monitoramento de ambientes de experimentação. Dentre as diversas ferramentas de código aberto disponíveis, foram estudadas e implementadas algumas dessas tecnologias para compor o sistemas de monitoramento do *testbed*.

[8]https://tools.ietf.org/html/rfc3565
[9]https://tools.ietf.org/html/rfc4511
[10]https://tools.ietf.org/html/rfc2865

Após o estudo foi composta então, uma *stack* completa de monitoramento e armazenamento que utiliza banco de dados *time series*, que proporciona ao gestor da rede maiores possibilidades para análise e tratamento dos dados, visto que oferecem a possibilidade de armazenar informações históricas a respeito de um determinado objeto, podendo correlacionar eventos e extrair informações numa série de dados temporais.

A seguir serão descritas as ferramentas que foram analisadas e implementadas, juntamente a topologia com a *stack* completa de controle e monitoramento escolhida para gestão do ambiente. A pilha de monitoramento da Elastic[11] permite extrair dados de forma confiável e segura de diversas fontes de dados. Essa pilha é constituída de um conjunto de ferramentas *open source*, que compõem uma solução completa de monitoramento. O monitoramento começa com os clientes Beats e Logstash, o primeiro coletam as métricas dos nós do *cluster*, enquanto o segundo funciona como um canal dinâmico de coleta com um ecossistema de *plug-ins* extensível para personalizar métricas especificas.

O Elasticsearch é uma ferramenta de pesquisa e análise de dados distribuída e especialmente projetava para escalabilidade horizontal. O Kibana é o *front-end* da aplicação de monitoramento. Ele permite a visualização dos dados coletados pelos clientes, que estão armazenados no Elasticsearch.

Como solução de monitoramento, além da ferramenta Beats apresenta acima, foram estudadas mais 2 tecnologias que tem a mesma função, atuar como cliente de coleta de métricas nos *hosts*. O Node Exporter é um cliente coletor que funciona de forma análoga aos demais já citados, porém ele tem integração nativa com o banco de dados do Prometheus[12]. A desvantagem do Note Exporter é que ele não possui interface gráfica de monitoramento e acompanhamento das operações.

O Netdata foi a solução para monitoramento nos clientes escolhido para compor a *stack* do testbed, devido a sua fácil integração com o banco de dados do Prometheus, e principalmente pelo baixo consumo dos recursos da máquina hospedeira, a Figura 5.4 demonstra o *dashboard* do mesmo.

Nos teste de performance, o Netdata coletava mais de 5.000 métricas consumindo apenas 1% de 1 *core* das máquinas. A interface gráfica nativa também é um ponto positivo da ferramenta, permitindo ao administrador de redes verificar a saúde da máquina e verificar eventuais anomalias sem precisar ter acesso físico a máquina.

[11] https://www.elastic.co/blog/elastic-stack-6-5-0-released
[12] https://prometheus.io/

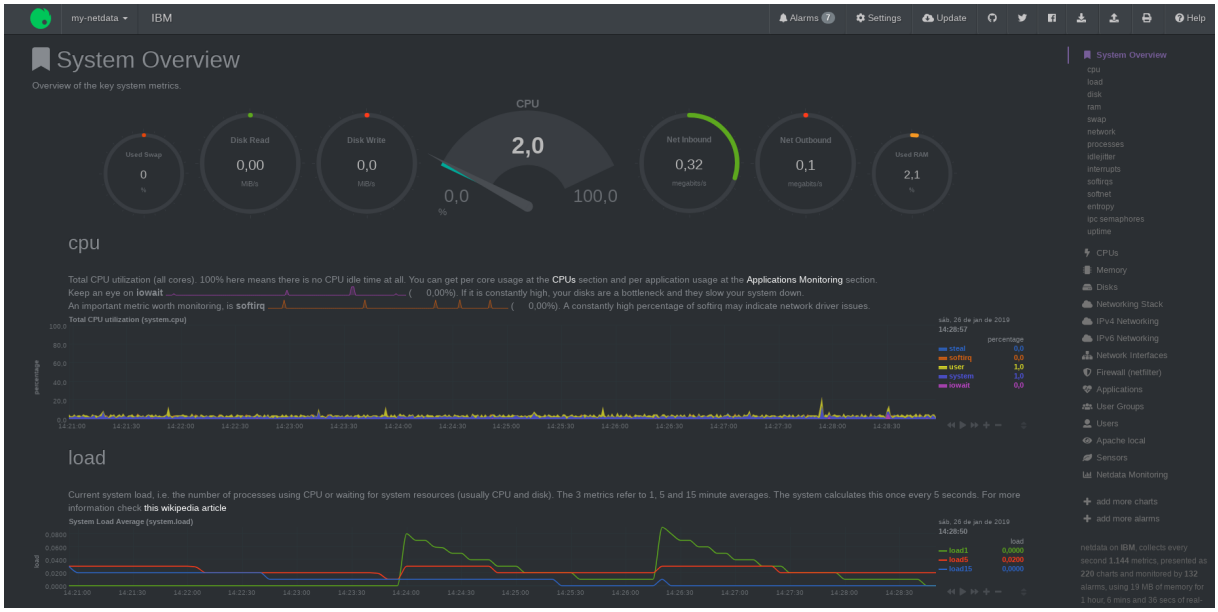

Figura 5.4: *Dashboard* de Monitoramento da Ferramenta Netdata

O Prometheus[13] consiste em um conjunto de ferramentas para monitoramento e alertas. Originalmente foi desenvolvido e criado no SoundCloud. O Prometheus consiste em um modelo de dados multidimensional com dados em séries temporais identificados por pares de nome de métrica e chave/valor. As consultas são feitas através da linguagem PromQL, permitindo flexibilidade para alavancar essa dimensionalidade dos banco temporal sem depender do armazenamento distribuído.

Figura 5.5: *Dashboard* de Monitoramento do *Cluster Whitebox*.

O Grafana[14] é uma solução para monitoramento de código aberto para gestão de métricas de diversas fontes de dados. A Figura 5.5 mostra o *dashboard* do *cluster* de máquinas *Whitebox*.

[13] https://prometheus.io/
[14] https://grafana.com/

A figura 5.6 mostra o *dashboard* do *cluster* de máquinas NetFPGAs. Os gráficos coletam as métricas em tempo real.

Figura 5.6: *Dashboard* de Monitoramento do *Cluster* NetFPGA.

O Grafana permite exibir métricas de diversas ferramentas através de uma interface mais atrativa. O banco *time series* do Prometheus armazena os dados que são coletados do cliente Netdata, através do Grafana é possível fazer *querys* nesses bancos e gerar gráficos através de *logs* e métricas. A ferramenta também possui um sistema de alerta que pode ser integrado com diversos aplicativos de mensagem, como o Telegram.

É possível criar alertas orientados a eventos, de modo que quando as métricas ultrapassem os limites estabelecidos, mensagem são automaticamente enviadas por *bots* para os canais de comunicação definidos. Trabalhos como (BRATTSTROM; MORREALE, 2017) evidenciam a eficiência e a integração de todas estas ferramentas de monitoramento de sistemas. Com esse tipo de sistema de alerta instantâneo, o administrador de redes tem um maior controle do sistema e tem informações consistentes que ajudam na tomada de decisões.

Capítulo 6

RESULTADOS

Neste capítulo serão demonstrados três experimentos de diferentes naturezas realizados no *testbed*. Nossos resultados indicam que a arquitetura proposta, implementada e testada atende vários requisitos considerados importantes para implementar novas arquiteturas de IF. O *testbed* proporciona para os usuários a possibilidade de realizar demonstrações práticas de como os sistemas de rede operam. A seguir são descritos os métodos utilizados para arquitetar este experimentos. Após a descrição são apresentados os gráficos com as métricas que os experimentos retornam aos usuários. O restante deste Capítulo foi segmentado descrevendo os resultados seguindo na Seção 6.1 referentes aos experimentos com P4 e FPGA; na Seção 6.2 os experimentos de roteamento com NetFPGA; na Seção 6.3 os experimentos com balanceamento de carga e por fim na Seção 6.4 os experimentos com deploy.

6.1 Experimento Utilizando NetFPGA e *Switch* P4

Utilizou-se um experimento para demostrar as funcionalidades do *testbed*. Instanciou-se um servidor gerador de tráfego em uma máquina do tipo NetFPGA, com *throughput* total 1Gbps. Esse servidor enviou os dados que seriam consumidos pelo cliente do outro lado da topologia. Os pacotes enviados eram encaminhados através de dois *switches* P4.

As portas físicas das máquinas eram mapeadas em um único *switch* virtual, que por sua vez, tiveram seus controladores programados na linguagem P4, dessa forma, foi possível controlar como seria o fluxo dos dados.

Nota-se que com poucas linhas de código é possível instanciar um experimento utilizando a plataforma para experimentação apresentada neste trabalho. O Algoritmo 1 apresenta como funcionam as chamadas das funções dos controladores e também a simplicidade para *deploy*

usando o *testbed*.

Algoritmo 1: Código Simplificado do Experimento Teste.

Entrada: *Setup* em forma de ED do experimento

Saída: O resultado é um arquivo de texto contendo as métricas especificadas no EC

```
1  início
2      init (netfpga7 - netfpga9) and (whitebox5, whitebox3);
3      set vlan 35 [netfpga7, netfpga9, netfpga8, whitebox5, whitebox3];
4      connect [whitebox5 on netfpga7 (nf2c0)] and [netfpga7 (nf2c1) on whitebox3];
5      (...)
6      reprogram [netfpga7] ← firewall.bit;
7      reprogram [whitebox5, whitebox3] ← controller.p4;
8      (...)
9      configure interface from netfpga8 [nf2c0 100.100.100.1/24];
10     configure interface from netfpga9 [nf2c0 100.100.100.5/24];
11     (...)
12     start iperf in [netfpga8] parameters -u -m 1000 -t 30 -c dest 100.100.100.5;
13     (...)
14     result = (log_iperf_.txt + dump_reg_netfpga.txt + ovs_log.txt + (...) );
15 fim
```

A Figura 6.1 representa a arquitetura lógica arquitetada de acordo com as métricas do Algoritmo 1.

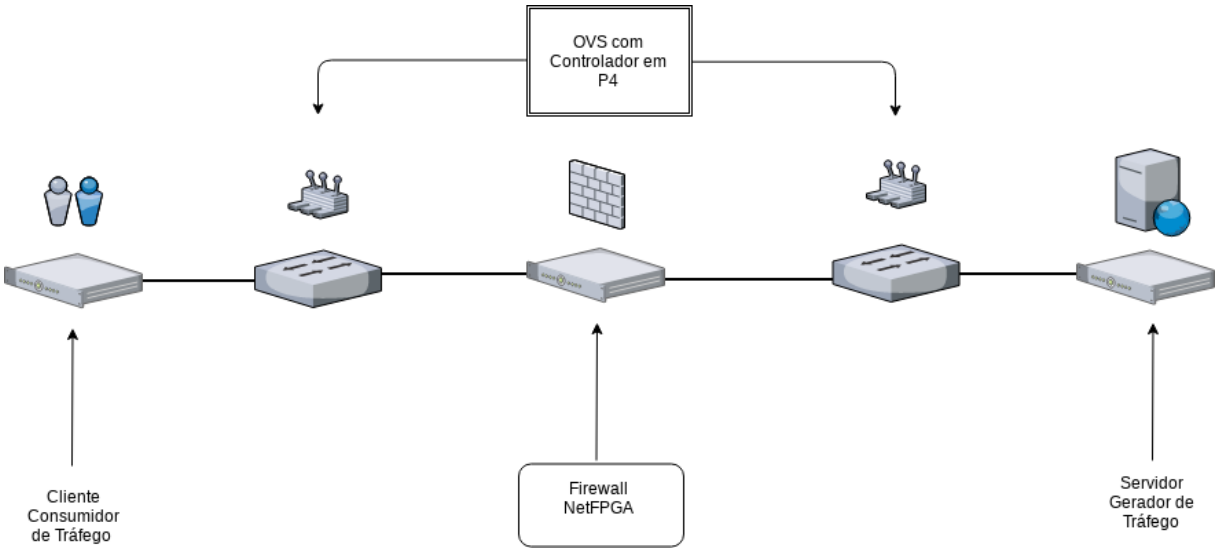

Figura 6.1: Topologia Lógica do Experimento Utilizando *Switch P4*.

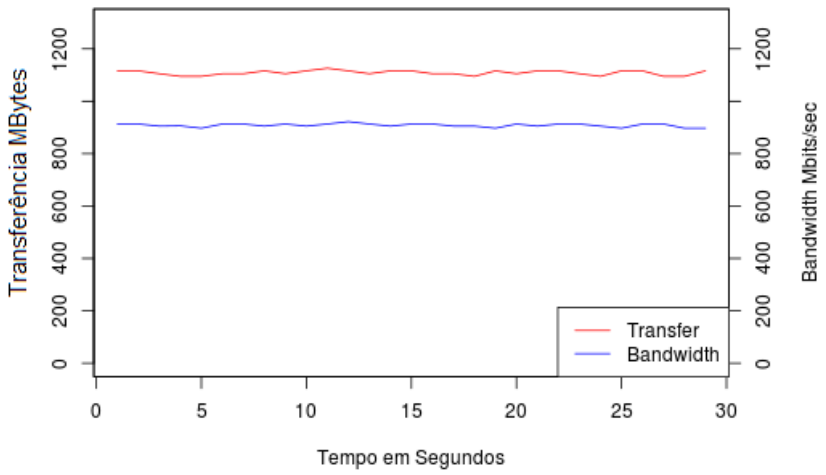

Figura 6.2: Gráfico da Vazão Durante o Experimento entre o Cliente e o *Switch P4.*

A Figura 6.2 mostra a vazão entre o cliente e o *switch* P4. Nota-se que essa vazão é constante e quase não sofre oscilações. Neste exemplo foi utilizado um *switch* P4 de referência[1] que apenas comuta pacotes em camada 2.

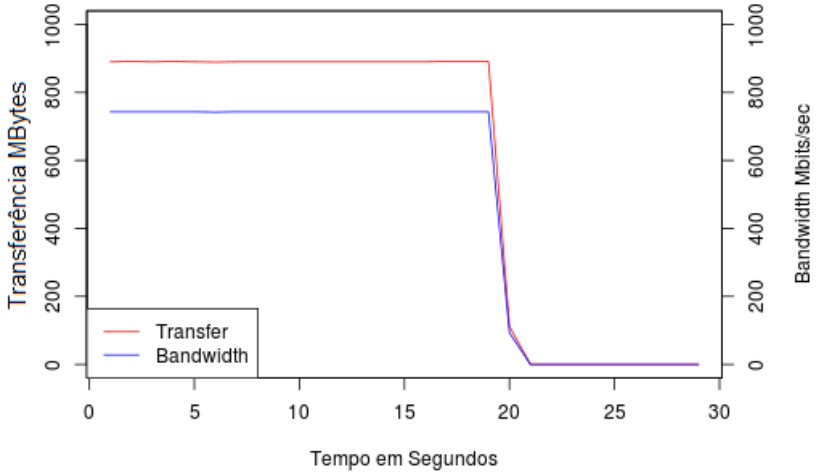

Figura 6.3: Gráfico da Vazão entre o Servidor e o Consumidor de Dados passando pelo *Firewall* NetFPGA.

O *firewall* NetFPGA processava os pacotes em nível de *hardware*, e bloqueava todos os pacotes com destino à porta 5151, conforme pode ser visualizado na Figura 6.3. Nota-se nos resultados uma grande queda no tráfego nos intervalo de tempo em que o *firewall* é instanciado,

[1] https://github.com/P4-vSwitch/vagrant

visto que todo o fluxo total gerado pelo servidor de tráfego tinha como destino a porta obstruída. Esse bloqueio ocorre de forma instantânea, pois o processamento ocorre no *hardware*, aumentado assim a efetividade e tempo de resposta.

6.2 Experimento Utilizando Roteamento com NetFPGA

O código abaixo é um exemplo de um experimento, onde o usuário realiza um experimento de roteamento utilizando placas NetFPGAs. Com poucas linhas de código é possível instanciar um experimento que pode ser facilmente replicado.

Algoritmo 2: Trecho de Código para Experimento com Roteamento Usando Máquina NetFPGA.

```
1  início
2      init (netfpga3 - netfpga9)
3      set vlan 43 [netfpga3 - netfpga9];
4      connect [netfpga3 on netfpga4 (nf2c0)] and [netfpga9 (nf2c1) on netfpga8];
5      (...)
6      reprogram [netfpga4 - netfpga8] ← router.bit;
7      reprogram [netfpga3, netfpga9] ← traffic_gen.bit;
8      configure router [netfpga4 - netfpga8] ← config.c;
9      (...)
10     configure interface from netfpga3 [nf2c0 10.0.0.1/24];
11     configure interface from netfpga9 [nf2c0 200.0.0.2/24];
12     (...)
13     start iperf-server in [netfpga9] parameters –server -U
14     start iperf-client in [netfpga3] parameters -u -m 1000 to 200.0.0.2
15     (...)
16 fim
```

O processo experimental começa iniciando as máquinas *netfpga3* até *netfpga9*, depois essas máquinas são configuradas na mesma VLAN *(*Virtual Local Area Network). A topologia lógica pode ser observada na Figura 6.4.

O próximo passo é a reprogramação das NetFPGAs. Quando o usuário faz o *upload* o arquivo (.bit), que neste caso fará com que a placa opere como um roteador. O arquivo **config.c** tem as configurações das rotas que serão enviadas para todos os roteados da topologia. Cada Máquina pode receber uma rota diferente das demais. No nosso experimento teste faremos a configuração de três rotas, sendo uma padrão para todas, e duas rotas alternativas, caso a

comunicação não ocorra pela rota padrão, o tráfego é transferido para as rotas alternativas.

A seguir são configuradas as interfaces de rede das máquinas (cliente e servidor de tráfego). Após essas configurações um servidor gerador e consumidor de tráfego são instanciados nas máquinas que ficam nos extremos da topologia.

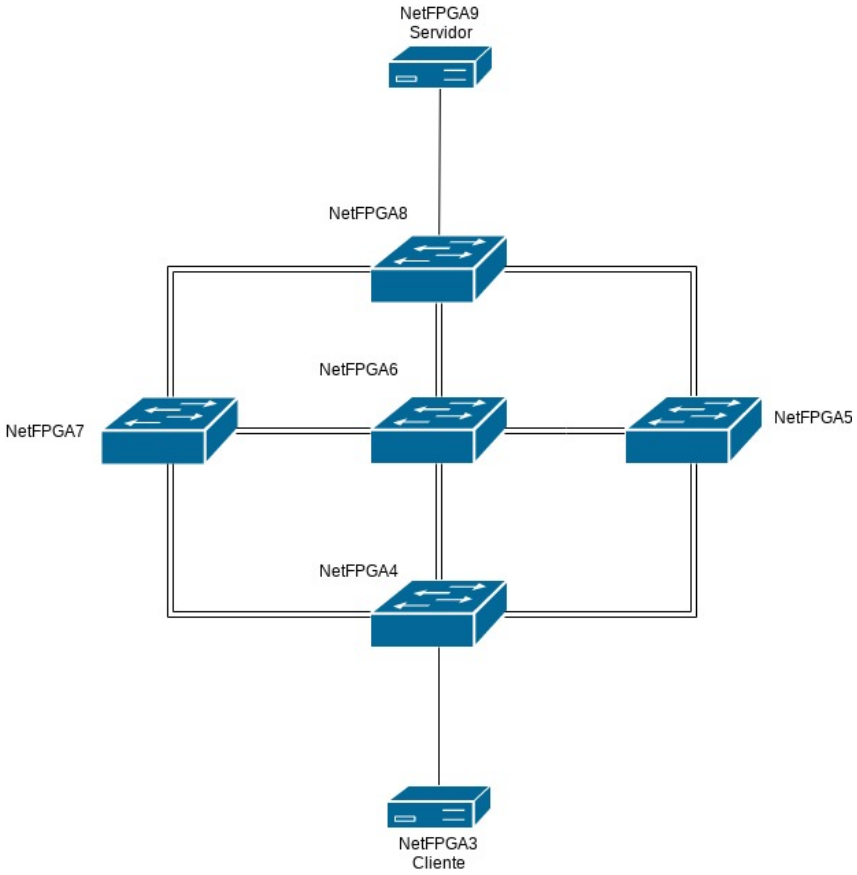

Figura 6.4: Topologia Lógica do Experimento de Roteamento usando Máquinas FPGAs.

Como já foi mencionado, cada roteador NetFPGA foi configurado com três rotas estáticas para encaminhamento dos pacotes, sendo uma padrão e duas alternativas. A rota padrão da NetFPGA4 é enviar pacotes para a NetFPGA7. Em caso de falha no envio, a primeira rota alternativa é para a NetFPGA6, e por fim, para a NetFPGA5. As NetFPGAs 5, 6 e 7 tem rota padrão para a NetFPGA8, que está diretamente conectada ao nosso servidor de dados.

Neste experimento, após 10 segundos de transmissão de dados pela rota padrão da NetFPGA4, a interface da NetFPGA7 é desativada, forçando a transmissão de pacotes pela primeira rota alternativa de tramissão configurada. Novamente após 10 segundos de transmissão a interface da NetFPGA6 é desativada, forçando o envio pela única rota restante, a NetFPGA5.

Ao final do processo, o usuário recebe um arquivo com os *logs* do servidor e do cliente, contendo as métricas de rede referentes ao experimento.

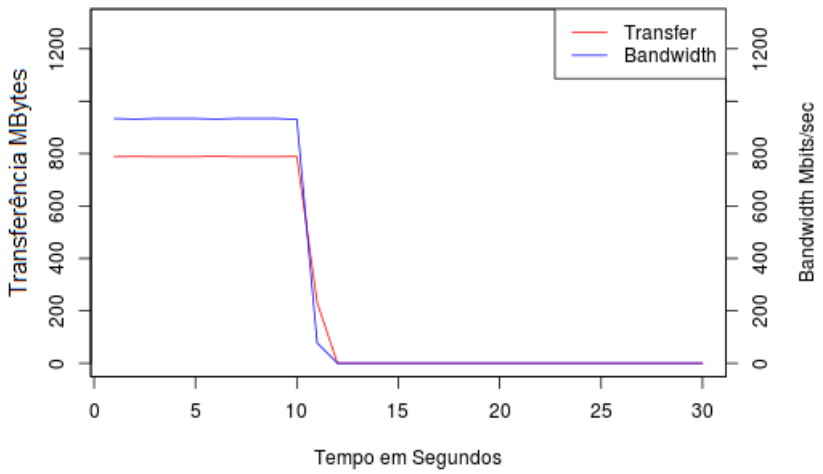

Figura 6.5: Vazão de Dados pela Rota Padrão.

Como já mencionado, a rota padrão para encaminhamento de pacotes em camada 3 do cliente que envia dados para a NetFPGA4 é a NetFPGA 7. Observando-se a Figura 6.5 nota-se queda total do tráfego após 10 segundos de experimento. Isso ocorre pois a interface do roteador que encaminhava os pacotes foi desativada. Forçando assim o desvio de dados para outro caminho.

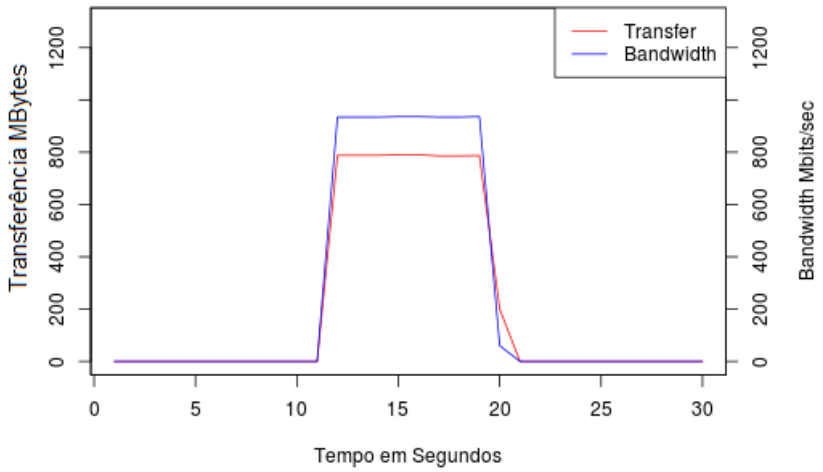

Figura 6.6: Gráfico da Vazão de Dados pela Rota Alternativa.

Após o desligando da interface do roteador NetFPGA7, o tráfego é direcionado para a rota alternativa que encaminha os pacotes com origem do cliente para a NetFPGA6. Nota-se que nos primeiros 10 segundos nenhum dado foi transmitido. A transmissão pela rota alternativa

passando pela NetFPGA6 durou 10 segundos, logo após esse tempo a interface do roteador foi desativada, forçando a pausa da transmissão de dados, conforme ilustra a Figura 6.6. A convergência de rotas foi então, para a ultima máquina configurada nos caminhos possíveis.

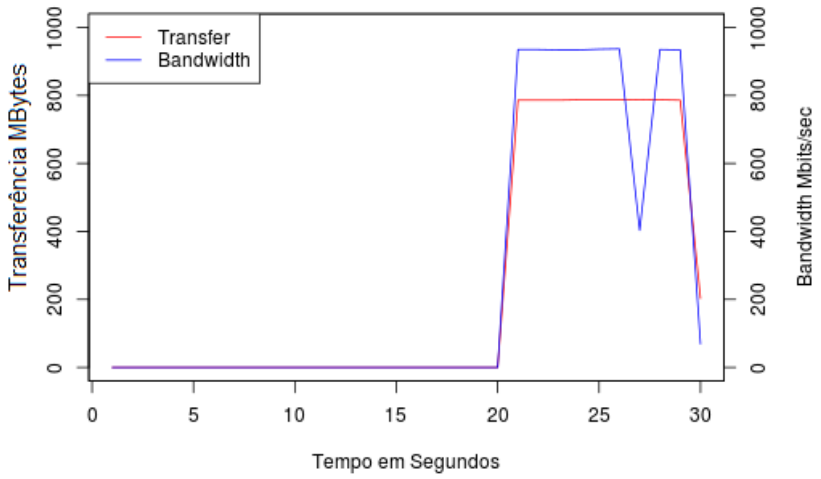

Figura 6.7: Gráfico da Vazão de Dados pela Rota Alternativa 2.

Conforme ilustra a Figura 6.7 a NetFPGA5 não comutou pacotes durante os primeiros 20 segundos do experimento. Somente após desativar as duas outras rotas estáticas com prioridade superior, o roteador começou a encaminhar os pacotes para seu próximo roteador, até o final do período de transmissão estipulado no experimento.

O tempo de convergência de uma rota estática para outra é quase imediato, visto que os caminhos são configurados diretamente na placa FPGA. Eliminando a consulta ao sistema operacional, essa operação diminui o tempo na ordem de mili-segundos, fazendo com que quase não haja percepção quando as rotas mudam. O uso de uma NetFPGA para comutação de pacotes em camada 3 diminui o tempo de convergência de rotas e diminuindo a latência.

6.3 Experimento Utilizando Balanceador de Carga TCP em P4

Este experimento agrega um pouco de todas as funções de experimentação disponíveis no *testbed*. Como já foi descrito nos experimentos anteriores, são passiveis de experimentação elementos de diversas naturezes, alguns exemplo foram o uso de *firewall* implementando em *hardware* utilizando a NetFPGA e um *pipeline* programável utilizando um *switch* escrito em

P4. A Figura 6.8 demonstra um experimento que agrega os componentes já conhecidos e apresentados até agora e adiciona um balanceador de carga TCP escrito em P4 entre um *firewall* em NetFPGA e um *switch* escrito em P4.

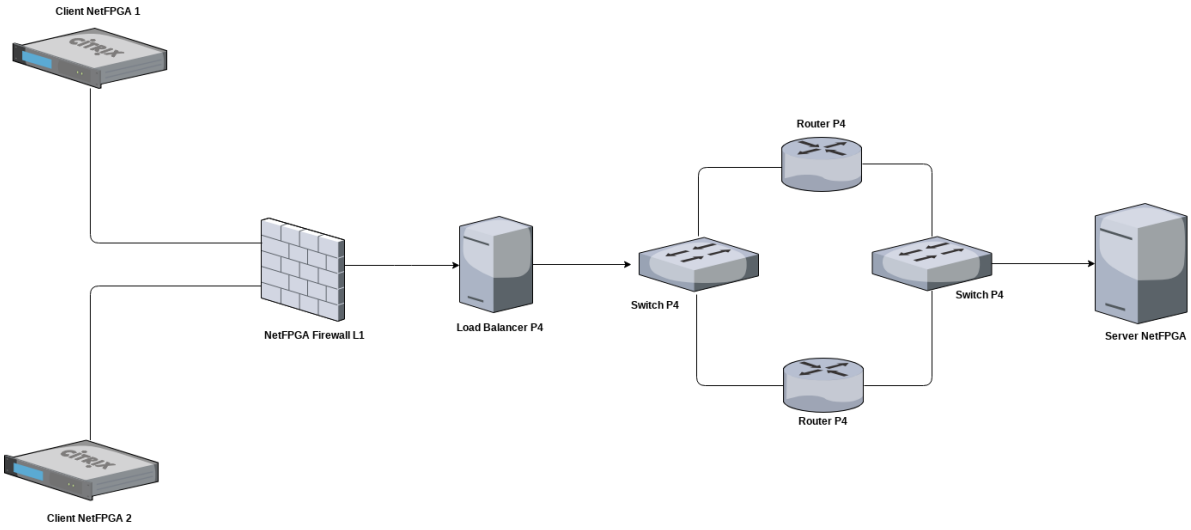

Figura 6.8: Topologia Lógica do Experimento com Balanceador de Carga em P4.

O viés desse novo elemento é proporcionar que o usuário modifique o algoritmo de encaminhamento de requisições do balanceador. Neste caso o balanceador implementa um algoritmo aleatório para encaminhamento de pacotes, no lugar do clássico **round-robin** presente na maioria dos balanceadores de carga.

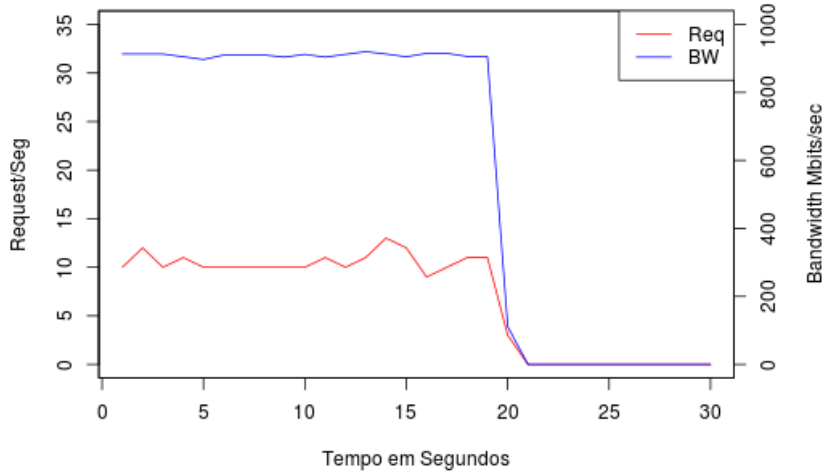

Figura 6.9: Gráfico da Vazão de Dados do *Load Balancer (Front-End)* e das Requisições por segundo

O Figura 6.9 mostra a quantidade de requisições que o balanceador recebe dos clientes, passando pelo *firewall*. Com 20 segundos de tramissão de dados o *firewall* em NetFPGA é

instanciado e começa bloquear todo o fluxo de dados.

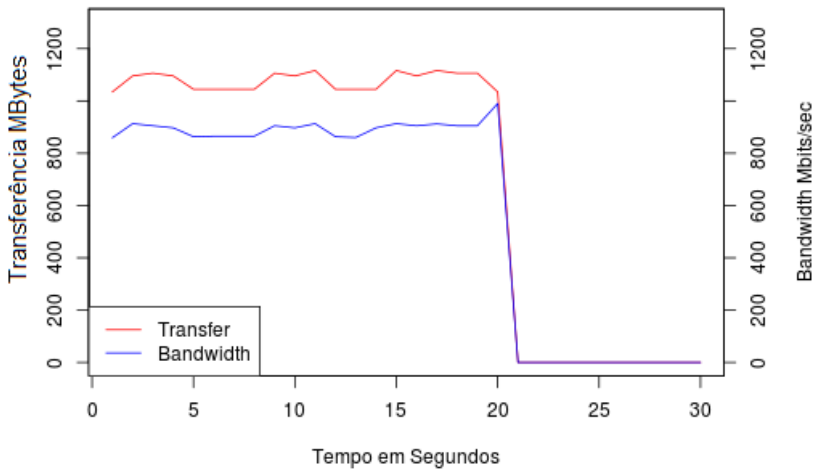

Figura 6.10: Gráfico da Vazão de Dados do *Load Balancer (Back-End)* para o *Switch* P4

A Figura 6.10 mostra a vazão entre o *back-end* e o *switch* P4. Até o 20º segundo a transmissão de dados é estável, e alcança um *throughput* médio de 1Gbps. Isso mostra que o algoritmo de encaminhamento aleatório do balanceador em P4 do balanceador não limitou o envio de pacotes, mantendo a transmissão na ordem da capacidade máxima das interfaces de rede dos equipamentos. Quando o *firewall* é instanciado após 20 segundos de experimento, os pacotes são bloqueados de forma instantânea. Desta forma o *front-end* do balanceador não recebe mais pacotes dos clientes, que por sua vez também não comuta mais os pacotes para o *back-end*, fazendo com que pacotes não sejam mais transmitidos até o final do experimentos, aos 30 segundos.

6.4 Servidor *Deploy* para Síntese de Projeto em Verilog

Para que os elementos reprogramáveis de plano de dados baseados em FPGA e PISA operem adequadamente no testbed é necessário que os respectivos projetos sejam compilados.

Ocorre que, o processo de compilação dos projetos utilizando um *hardware* comum ou qualquer uma das máquinas do *testbed* é muito demorado. Portanto, criou-se um controlador para era direcionar uma *thread* para um servidor com um *hardware* dedicado a esse tipo de função, de modo que este realizaria a compilação em um intervalo de tempo menor do que as máquinas do ambiente de experimentação.

Essa etapa consiste em transferir as tarefas de sintetização de projetos em P4 ou Verilog para uma máquina com *hardware* dedicado para essa função. Isso engloba mais uma função no *testbed*, e não gera sobrecarga em experimentos que estão acontecendo em paralelo.

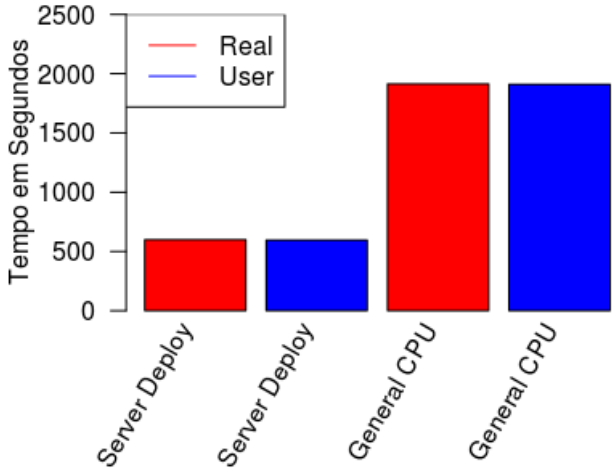

Figura 6.11: Gráfico Comparativo de Tempo de Compilação de Projeto em Verilog

A Figura 6.11 exibe os resultados das execuções. Nota-se que o *Server Deploy* executa a compilação do projeto aproximadamente três vezes mais rápido que uma máquina de uso comum.

Porém a eficiência deste controlador está totalmente condicionada ao poder computacional da máquina que fará a compilação do projeto Verilog. Ou seja, caso o *hardware* não seja dedicado a esse tipo de função, os resultados não tem essa disparidade.

Após esse processo, o arquivo binário compilado seria entregue ao experimentador, e estaria pronto para ser descarregado na plataforma de testes. Este processo aumenta a gama de possibilidades para o experimentador, visto que oferece um recurso a mais ao usuário. Ajudando a reduzir ainda mais, o tempo de instanciação dos experimentos, que é um dos objetivos desse trabalho.

Capítulo 7

CONCLUSÕES E TRABALHOS FUTUROS

7.1 Conclusões

A metodologia desenvolvida, de experimentação automática em planos de dados, permite que os tempos de instanciação, construção e execução de experimentos seja minimizado. Dessa forma, os pesquisadores podem gastar mais tempo com o núcleo cientifico da pesquisa, ao invés de ter que configurar manualmente e criar *scripts* para os seus experimentos. Isso facilita também a reprodutibilidade da pesquisa, que é um tema de bastante relevância nos dias atuais, uma vez que a programabilidade do plano de dados está evoluindo rapidamente com linguagens emergentes como o P4.

A base das contribuições foi feita a partir de ferramentas como OMF, com o qual estendemos para suporte a FPGA e P4. E também, nossa contribuição em melhorar toda a infraestrutura robusta do *testbed* e do ciclo de vida de controle do experimento. Esses avanços possibilitarão a criação de experimentos de protótipos, nessas tecnologias, de maneira rápida que podem resultar em redes mais baratas e flexíveis, mantendo a compatibilidade com redes legadas. Com a linguagem P4 os protocolos de rede P4 podem ser escritos e implantados por operadores de rede e não por fabricantes de dispositivos de rede.

Os benefícios da utilização de linguagens de descrição para experimentação são variados. Conforme já comentamos, o experimento é auditável, passível de replicação. Sua descrição é formal em código que pode ser compartilhado. Além disso, o ambiente seguro proposto em nossa arquitetura permite que aspectos do experimento tenham modo reativo, onde os resultados podem alterar o próprio fluxo experimento. Nota-se que a metodologia de experimentação automática de planos de dados em *bare-metal* garante mais controle para o usuário e menor latência no acesso aos dados, já que evita etapas de processamento via camadas de *software*,

focada em aplicações sensíveis a desempenho e velocidade. Isso é feito através de funções pré-definidas nos controladores acessíveis pelo experimentador, sem conceder acesso de superusuário *(root)* nas máquinas.

O sistema de monitoramento implementado proporciona ao gestor da rede maiores possibilidades para análise e tratamento dos dados, visto que oferecem a possibilidade de armazenar informações históricas a respeito de um determinado objeto, podendo correlacionar eventos e extrair informações numa série de dados temporais. Com esse tipo de sistema de alerta instantâneo, o administrador de redes tem um maior controle do sistema e tem informações consistentes que ajudam na tomada de decisões.

Os resultados experimentais do protótipo demonstram que os objetivos foram alcançados, como, por exemplo, os tempos de implantação automática são superiores a uma implantação manual. Além disso, os experimentos para coleta de resultados foram feito para demonstrar que a complexidade de cenário pode ser gerenciada pela linguagem de experimentos. Escolhemos cenários contendo vários elementos e várias tecnologias e coletando dados de forma automática, como gerenciar grupos de *firewalls* e roteadores.

7.2 Trabalhos Futuros

Na sequencia desse trabalho, embora desenvolver com linguagem de descrição de experimentos seja algo formal, e produtivo, entretanto, a curva de aprendizado pode ser alta para o experimentador. O experimentador para começar a ser produtivo, minimamente, precisa saber programar em linguagem Ruby. Portanto, uma ideia é o desenvolvimento de uma interface *web*, onde o experimentador possa configurar seu ambiente de experimentação usando apenas recursos gráficos e de fácil interação com o usuário. Para usuários leigos, isso pode acelerar ainda mais, da criação visual até o seu mapeamento em script de descrição do experimento, que seria gerado por essa plataforma. Esse desenvolvimento poderia estar sendo integrado ao portal do projeto FIBRE, pois as contribuições desenvolvidas tem facilidade de integração com os arcabouços do FIBRE.

Outra atividade futura pode ser a continuação e refinamento do sistema de monitoramento apresentado nesse trabalho. Sendo esse esforço replicado em todas as ilhas do *testbed* FIBRE ao redor do Brasil. Desse modo, as monitorações poderiam usar bancos de dados *time series* distribuídos e armazenarem, tanto as condições do *testbed*, quanto os resultados dos experimento, por um período longo de tempo. Eventualmente, tais dados poderiam ser processados

com ferramentas de processamento distribuído como Apache Storm[1] e Spark[2].

Finalmente, durante a pesquisa, nós verificamos que os projetos de planos de dados, em geral, não permitem a multiplexação de diferentes experimentadores no mesmo recurso. Uma placa netFPGA é reprogramada somente por um usuário, ou um projeto P4 reprograma o dispositivo de uma maneira atômica e exclusiva. Desse modo, detectamos que as *testbeds* de planos de dados podem ficar sub-utilizadas. Uma ideia para resolver esse problema é criar um multiplexador lógico, a ser acoplado à cada design de experimentador. Desta forma, os pacotes podem ser processados por diferentes lógicas dentro do plano de dados. Ao escrever o seu *design* FPGA, um experimentador estará somente ciente da sua contribuição, porém ao ser carregado no equipamento, a linguagem de descrição de experimentos introduziria esse multiplexador na frente do *design*. Com esta modificação, um próximo experimentador poderá criar seu *design* FPGA, sem saber do primeiro, e a linguagem de descrição de experimentos integrará este novo *design* FPGA com o do outro experimentador em tempo de execução. Em resumo, o multiplexador fará uma combinação de todos os *designs* de todos os experimentadores concorrentes, e montará um único projeto que será reprogramado na placa. E para desambiguação, a multiplexação poderá ser feita acondicionada às interfaces físicas das placas. Portanto cada *design* será redirecionado para sua porta de entrada e saída, respectivamente, sem comprometer o desempenho do outro sendo testado.

7.3 Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brasil (CAPES) - Código de Financiamento 001". Os autores também agradecem ao apoio financeiro da Rede Nacional de Pesquisa e Ensino (RNP).

[1] http://storm.apache.org/
[2] https://spark.apache.org/

REFERÊNCIAS

BERMAN, M. et al. Geni: A federated testbed for innovative network experiments. *Computer Networks*, v. 61, p. 5 – 23, 2014. ISSN 1389-1286. Special issue on Future Internet Testbeds – Part I. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S1389128613004507⟩.

BILAR GUILHERME, M. R. M. C. Linguagem de domínio especifico para geração de firewalls modulares em hardware. *1º Workshop de Teses e Dissertações - Departamento de Computação UFSCAR*, p. 1–2, 2016.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, ACM, v. 44, n. 3, p. 87–95, 2014.

BRATTSTROM, M.; MORREALE, P. Scalable agentless cloud network monitoring. In: IEEE. *Cyber Security and Cloud Computing (CSCloud), 2017 IEEE 4th International Conference on*. [S.l.], 2017. p. 171–176.

CHUN, B. et al. Planetlab: An overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 33, n. 3, p. 3–12, jul. 2003. ISSN 0146-4833. Disponível em: ⟨http://doi.acm.org/10.1145/956993.956995⟩.

CIUFFO, L. et al. Testbed fibre: Passado, presente e perspectivas. In: SN. *Anais do WPEIF 2016 Workshop de Pesquisa Experimental da Internet do Futuro*. [S.l.], 2016. p. 3–6.

COMER, D. E. *Internetworking with TCP/IP: Principles, Protocols, and Architecture, Vol. 1*. [S.l.]: Prentice Hall Upper Saddle River, 2000. 101-104 p.

COSTA, L. C. et al. Avaliaç ao de desempenho de planos de dados openflow. 2016.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, ACM, v. 44, n. 2, p. 87–98, 2014.

FERREIRA, F. A. et al. Novagenesis no ambiente fibre: Primeiras impressoes. *Sociedade Brasileira de Computação - SBRC*, p. 6, 2017.

GAO, H. et al. Techniques and research trends of network testbed. In: *2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. [S.l.: s.n.], 2014. p. 537–541.

GARCIA, L. F. U. et al. Introdução a linguagem p4-teoria e prática. *Minicursos do SBRC 2018 - Campos do Jordão - SP*, v. 1, p. 2–5, 2018.

GAVRAS, A. et al. Future internet research and experimentation: the fire initiative. *ACM SIGCOMM Computer Communication Review*, ACM, v. 37, n. 3, p. 89–92, 2007.

GOULART, P. et al. Netfpga: Processamento de pacotes em hardware. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2015*, p. 1–3, 2015.

HWANG, J.; RAMAKRISHNAN, K. K.; WOOD, T. Netvm: High performance and flexible networking using virtualization on commodity platforms. *IEEE Transactions on Network and Service Management*, v. 12, n. 1, p. 34–47, March 2015. ISSN 1932-4537.

IBANEZ, S. et al. The p4-¿ netfpga workflow for line-rate packet processing. In: ACM. *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. [S.l.], 2019. p. 1–9.

JOUET, S.; PEZAROS, D. P. Bpfabric: Data plane programmability for software defined networks. In: IEEE PRESS. *Proceedings of the Symposium on Architectures for Networking and Communications Systems*. [S.l.], 2017. p. 38–48.

KAMIENSKI, C. et al. Arquiteturas de rede para a próxima geraçao da internet. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2005*, v. 9, n. 3, p. 1–50, 2005.

KAMIENSKI, C. A.; SADOK, D. Qualidade de serviço na internet. *Minicursos do Simpósio Brasileiro de Redes de Computadores, SBRC, Belo Horizonte/MG*, p. 1–5, 2000.

KIM, C. et al. In-band network telemetry via programmable dataplanes. In: *ACM SIGCOMM*. [S.l.: s.n.], 2015.

LIU, W. et al. Orchestration and reconfiguration control architecture orca- a 5g experimental environment. *Conference: EuCNC, At Oulu Finland*, June 2017.

MANGGALA, A. W.; TANWIDJAJA, A. et al. Performance analysis of white box switch on software defined networking using open vswitch. In: IEEE. *Wireless and Telematics (ICWT), 2015 1st International Conference on*. [S.l.], 2015. p. 1–7.

MARCONDES, C. A. C. et al. Estado da arte de sistemas de controle e monito-ramento de infraestruturas para experimentação de redes de comunicação. *Minicursos da Sociedade Brasileira de Computação - SBRC, Ouro Preto, MG*, p. 3–7, 2012.

MIKROYANNIDIS, A. et al. Forge : an elearning framework for remote laboratory experimentation on fire testbed infrastructure. In: *Building the future internet through FIRE*. [S.l.: s.n.], 2016. p. 521–559. ISBN 978-87-93519-12-1.

MONSANTO, C. et al. Composing software-defined networks. In: *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2013. (nsdi'13), p. 1–14. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=2482626.2482629⟩.

MOREIRA, M. D. et al. Internet do futuro: Um novo horizonte. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC*, v. 2009, p. 1–59, 2009.

NASCIMENTO, M. R. et al. Routeflow: Roteamento commodity sobre redes programáveis. *XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos-SBRC*, 2011.

NUNES, B. A. A. et al. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys Tutorials*, v. 16, n. 3, p. 1617–1634, Third 2014. ISSN 1553-877X.

PACÍFICO, R. D. et al. Roteador sdn em hardware independente de protocolo com análise, casamento e ações dinâmicas. In: *Simpósio Brasileiro de Redes de Computadores (SBRC)*. [S.l.: s.n.], 2018. v. 36.

PATRA, P. G.; ROTHENBERG, C. E.; PONGRÁCZ, G. Macsad: Multi-architecture compiler system for abstract dataplanes (aka partnering p4 with odp). In: ACM. *Proceedings of the 2016 ACM SIGCOMM Conference*. [S.l.], 2016. p. 623–624.

PEDROSO, D. et al. Simulação de ambientes de redes utilizando a testbed fibre-aplicações na pesquisa e no ensino.

RF, S. Estudos de revisão sistemática: um guia para síntese criteriosa da evidência científica. *Revista brasileira de fisioterapia*, SciELO Brasil, v. 11, n. 1, p. 83–89, 2007.

ROTHENBERG, C. E. et al. Openflow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. *Cad. CPqD Tecnologia, Campinas*, v. 7, n. 1, p. 65–76, 2010.

SALMITO, T. et al. FIBRE - An International Testbed for Future Internet Experimentation. In: *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2014*. Florianopolis, Brazil: [s.n.], 2014. p. p. 969. Disponível em: ⟨https://hal.archives-ouvertes.fr/hal-00990392⟩.

SALTZER, J. H.; REED, D. P.; CLARK, D. D. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, ACM, v. 2, n. 4, p. 277–288, 1984.

SHAHBAZ, M. et al. Pisces: A programmable, protocol-independent software switch. In: ACM. *Proceedings of the 2016 ACM SIGCOMM Conference*. [S.l.], 2016. p. 525–538.

SHAHBAZ, M. et al. Pisces: A programmable, protocol-independent software switch. In: *Proceedings of the 2016 ACM SIGCOMM Conference*. New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 525–538. ISBN 978-1-4503-4193-6. Disponível em: ⟨http://doi.acm.org/10.1145/2934872.2934886⟩.

SILVA, M. V. B. da et al. Identificação de fluxos elefantes em redes de ponto de troca de tráfego com suporteá programabilidade p4. In: SBC. *Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.], 2018.

SILVESTRE, B. O.; CARDOSO, K. V.; CORRÊA, S. L. Controle e monitoramento de experimentos para a internet do futuro usando omf. In: *Simposio Brasileiro de Redes de Computadores*. [S.l.]: SBRC, 2012. v. 1, p. 2–6.

SIVARAMAKRISHNAN, A. et al. Gemini planet imager coronagraph testbed results. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Ground-based and Airborne Instrumentation for Astronomy III*. [S.l.], 2010. v. 7735, p. 773586.

THOMAS, C. et al. A passive measurement system for network testbeds. In: KORAKIS, T.; ZINK, M.; OTT, M. (Ed.). *Testbeds and Research Infrastructure. Development of Networks and Communities*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 130–145.

WAHLE, S. et al. Emerging testing trends and the panlab enabling infrastructure. *IEEE Communications Magazine*, v. 49, n. 3, p. 167–175, March 2011. ISSN 0163-6804.

WANG, H. et al. P4fpga: A rapid prototyping framework for p4. In: ACM. *Proceedings of the Symposium on SDN Research*. [S.l.], 2017. p. 122–135.

ZAMBONI, A. et al. Start uma ferramenta computacional de apoio à revisão sistemática. In: *Proc.: Congresso Brasileiro de Software (CBSoft'10), Salvador, Brazil*. [S.l.: s.n.], 2010.