INSTITUTO FEDERAL DO ESPÍRITO SANTO - IFES
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

EDGARD DA CUNHA PONTES

PROTOTIPAÇÃO E IMPLEMENTAÇÃO DE REDES EXPERIMENTAIS COM WHITEBOXES E RARE/FREERTR

Cachoeiro de Itapemirim
2022

EDGARD DA CUNHA PONTES

PROTOTIPAÇÃO E IMPLEMENTAÇÃO DE REDES EXPERIMENTAIS COM WHITEBOXES E RARE/FREERTR

Trabalho de Conclusão de Curso apresentado à Coordenadoria do Curso de Sistemas de Informação do Instituto Federal do Espírito Santo, Campus Cachoeiro de Itapemirim, como requisito parcial para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: DSc. Rafael Silva Guimarães

Cachoeiro de Itapemirim

2022

(Biblioteca do Campus Cachoeiro de Itapemirim)

P814p Pontes, Edgard da Cunha.

Prototipação e implementação de redes experimentais com whiteboxes e RARE/Freertr / Edgard da Cunha Pontes. - 2023.
64 f. : il. ; 30 cm..

Orientador: Rafael Silva Guimarães

TCC (Graduação) Instituto Federal do Espírito Santo, Campus Cachoeiro de Itapemirim, Sistemas de Informação, 2023.

1. Redes de computadores. 2. Roteadores (redes de computadores). 3. Sistemas de comunicação sem fio. I. Guimarães, Rafael Silva. II.Título III. Instituto Federal do Espírito Santo.

CDD: 004.6

Bibliotecário/a: Renata Lorencini Rizzi CRB6-ES nº 085

MINISTÉRIO DA EDUCAÇÃO
INSTITUTO FEDERAL DO ESPÍRITO SANTO

FOLHA DE APROVAÇÃO-TCC nº 27/2022-CAI-CCSI
Protocolo nº 23151.005309/2022-81

Cachoeiro De Itapemirim-ES, 22 de dezembro de 2022

EDGARD DA CUNHA PONTES

PROTOTIPAÇÃO E IMPLEMENTAÇÃO DE REDES EXPERIMENTAIS COM WHITEBOXES E RARE/FREERTR

Trabalho de Conclusão de Curso apresentado à Coordenadoria de Sistemas de Informação do Instituto Federal do Espírito Santo, como requisito parcial para obtenção de título de Bacharel em Sistemas de Informação

Aprovado em 19 de dezembro de 2022

COMISSÃO EXAMINADORA

D.sc. Rafael Silva Guimarães
Instituto Federal Do Espírito Santo
Orientador

M.Sc. Antonio Izo Júnior
Instituto Federal Do Espírito Santo

M.Sc. Daniel José Ventorim Nunes
Instituto Federal Do Espírito Santo

(Assinado digitalmente em 22/12/2022 11:23)
ANTONIO IZO JUNIOR
PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO-SUBSTITUTO
CAI-CCSI (11.02.18.01.08.02.13)
Matrícula: 3240051

(Assinado digitalmente em 22/12/2022 11:28)
DANIEL JOSE VENTORIM NUNES
PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO
CAI-CCLI (11.02.18.01.08.02.06)
Matrícula: 1918045

(Assinado digitalmente em 22/12/2022 11:23)
RAFAEL SILVA GUIMARAES
PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO
CAI-CCSI (11.02.18.01.08.02.13)
Matrícula: 1919203

DECLARAÇÃO DO AUTOR

Declaro, para fins de pesquisa acadêmica, didática e técnico-científica, que este Trabalho de Conclusão de Curso pode ser parcialmente utilizado, desde que se faça referência à fonte e ao autor.

Cachoeiro de Itapemirim, 19 de dezembro de 2022.

Edgard da Cunha Pontes

Dedico esse trabalho primeiramente a Deus, a minha família e a todos que de alguma forma contribuíram para sua conclusão.

AGRADECIMENTOS

Agradeço a Deus pela minha vida, pela graça, pela oportunidade de concluir mais uma graduação e pelos vários desafios que foram vencidos durante essa jornada.

A minha esposa Paula por todo amor, paciência, carinho e dedicação, e a toda minha família pelo apoio que sempre me deram.

Ao meu orientador na graduação, Prof. Rafael Silva Guimarães, pela amizade, apoio e paciência na condução desse trabalho. Ao Prof. Everson Scherrer Borges, pela parceria, apoio e amizade de sempre, ao Prof. Flavio Izo e a Profa. Susana Brunoro Costa Oliveira, pela orientação dos projetos de diplomação antes e durante o período de pandemia. Ao Prof. Moises Renato Nunes Ribeiro, pela oportunidade de participação no artigo (BORGES et al., 2022b) e o Prof. Magnos Martinello pela orientação no mestrado e oportunidade de participação no artigo (BORGES et al., 2022a). A Profa. Cristina Klippel Dominicini pela oportunidade de participação no projeto de pesquisa (FACTO).

Aos professores do Ifes, pelo tempo, dedicação e pela oportunidade de aprender com eles durante este curso.

Aos companheiros do LEDS no Ifes, por todo apoio e carinho durante as várias horas de experimentos no laboratório.

Aos desenvolvedores do RARE/freeRtr, Csaba Máté e Frédéric Loui, pela atenção e paciência de sempre, ao responder tantas dúvidas.

A empresa Vixphy pela oportunidade de participar dos projetos de pesquisa SAWI (FAPESP) e RENaaS (GEÁNT), por permitir o uso dos equipamentos para realização dos experimentos. E aos companheiros destes projetos pelo apoio de sempre.

"*Man's mind, stretched to a new idea,*
never goes back to its original dimension."
Oliver Wendell Holmes, Jr.

"A mente do homem, estendida a uma nova ideia,
nunca volta à sua dimensão original." (tradução livre)
Oliver Wendell Holmes, Jr.

RESUMO

Neste trabalho, será abordado os requisitos necessários para prototipar e implementar uma rede experimental, comumente chamada de *testbed*, que propicie o acesso aos recursos mais próximos do “estado da arte” em redes experimentais de computadores. Para tal propósito, foi utilizado o paradigma SDN (*Software-Defined Networking*), ou seja, uma rede programável, bem como as exigências necessárias para configurar dois modelos de *whiteboxes* com o Sistema Operacional para roteadores baseado em processos chamado RARE/freeRtr. Também foi proposto um método para experimentação do RARE/freeRtr em ambientes emulados, além de três testes de performance *throughput* para avaliar e analisar o funcionamento das *whiteboxes* nos modos de configuração: Plano de Controle e Plano de Dados.

Palavras-chave: Redes Experimentais, *Testbed*, Redes Programáveis, SDN, *Whitebox*, RARE/freeRtr.

ABSTRACT

In this work, the necessary requirements for prototyping and implementing an experimental network, commonly called a testbed, that provides access to resources closer to the “state of the art” in experimental computer networks will be approached. For this purpose, the SDN (Software-Defined Networking) paradigm was used, that is, a programmable network, as well as the necessary requirements to configure two models of whiteboxes with the Operating System for routers based on processes called RARE/freeRtr. A method for testing RARE/freeRtr in emulated environments was also proposed, as well as three throughput performance tests to evaluate and analyze the functioning of whiteboxes in configuration modes: Control Plane and Data Plane.

Keywords: Experimental Computing Network, Testbed, Programmable Networks, SDN, Whitebox, RARE/freeRtr.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

ForCES – *Forwarding and Control Element Separation*

GbE – *Gigabit Ethernet*

Gbps – Gigabit por segundo

Ifes – Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo

LEDS – Laboratório de Desenvolvimento de Soluções

Mbps – Megabit por segundo

NETCONF – *Network Configuration Protocol*

P4 – *Programming Protocol-independent Packet Processors*

LISTA DE SIGLAS

API – *Application Programming Interface*

DPDK – *Data Plane Development Kit*

DTN – *Data Transfer Node*

HDMI – *High-Definition Multimedia Interface*

IETF – *Internet Engineering Task Force*

IP – *Internet Protocol*

JAR – *Java ARchive*

JDK – *Java Development Kit*

JRE – *Java Runtime Environment*

LLDP – *Link Layer Discovery Protocol*

MAC – *Media Access Control*

NREN – *National Research and Education Network*

OS – *Operating System*

OSI – *Open System Interconnection*

OSPF – *Open Shortest Path First*

PBR – *Policy-Based Routing*

RARE – *Router for Academia Research & Education*

RFC – *Request for Comments*

SDN – *Software-Defined Network*

SO – Sistema Operacional

TCP – *Transmission Control Protocol*

UDP – *User Datagram Protocol*

VM – *Virtual Machine*

VRF – *Virtual Routing Forwarding*

XDP – *Express Data Path / eXpress Data Path*

SUMÁRIO

1 INTRODUÇÃO

Para ter acesso aos recursos mais próximos do “estado da arte” em redes experimentais de computadores, não é necessário implementar uma infraestrutura de grande porte, comumente conhecida como *testbed*. Desta forma, *testbeds* como: FIBRE (SALLENT et al., 2012), GENI (BERMAN et al., 2014) e FABRIC (BALDIN et al., 2019), por exemplo, normalmente são projetos com recursos compartilhados por vários países e/ou instituições. Pois o custo necessário para implementar e manter redes experimentais como essa estrutura é extremamente elevado.

Outro fato que dificulta o desenvolvimento de *testbeds*, é que durante décadas, as empresas e/ou instituições compraram equipamentos de rede de um único *vendor* (fornecedor), como a Cisco. Na maioria dos casos, criando uma forte dependência de um único fornecedor e impossibilitando a substituição destes equipamentos, muitas vezes por causa da incompatibilidade. Uma resposta para essa “ossificação” da rede é a utilização das *whiteboxes* devido a vantagens como independência do software da plataforma de hardware, redução de custo e potencial de trazer inovações. Permitindo, por tanto, a redução do aprisionamento do fornecedor, pois é possível executar qualquer sistema operacional de rede (NOS) em *whiteboxes*, principalmente sistemas operacionais *open source* (código aberto).

O RARE/freeRtr surge como resposta para qual NOS *open source* utilizar em *whiteboxes*, já que ele é um *Router OS Process* (processo do sistema operacional como roteador, na tradução livre) gratuito e de código aberto. Ele é definido como “*swiss army knife*” (ou “canivete suiço” em português) por seus desenvolvedores, devido às suas capacidades intrínsecas de (re)encapsular pacotes em múltiplas faixas de interfaces. Além de uma série de outras ferramentas abordadas neste trabalho. Outra definição importante, apresentada por (BORGES et al., 2022b), o descreve como “*protocoland*” (ou “terra dos protocolos” em tradução livre). Definição essa que tenta expressar uma de suas maiores vantagens, que é a implementação de um portfólio de protocolos imenso e excepcional.

Este trabalho visa responder a dúvida de como ter acesso aos recursos mais recentes na área de redes de computadores, sem a necessidade de implementar um *testbed* de grande porte. Ou seja, uma rede experimental acadêmica e/ou de inovação com custo reduzido de aquisição e manutenção dos equipamentos, além permitir a independência do fornecedor. Vale a pena ressaltar que não faz parte do escopo deste trabalho a comparação de desempenho de hardware e software com outros sistemas operacionais de rede. Restringindo-se a explorar as vantagens e/ou desvantagens da ferramenta RARE/freeRtr em conjunto com as *whiteboxes*.

No **capítulo 2** é apresentada a revisão bibliográfica com a fundamentação teórica utilizada neste trabalho. No **capítulo 3** são apresentados os materiais e métodos demonstrando os procedimentos necessários para configurar as *whiteboxes* para execução dos experimentos. No **capítulo 4** são apresentados os resultados obtidos nos três testes realizados nos modos de configuração de Plano de Controle e de Dados nas *whiteboxes*, além de demonstrar os caminhos mais curto (*shortest path*) e mais longo (*longest path*) na topologia.

1.1 OBJETIVOS

Este trabalho tem como objetivo geral descrever e analisar os procedimentos necessários para prototipar e implementar um *testbed* para redes programáveis, utilizando *whiteboxes* e o RARE/freeRtr.

Os objetivos específicos:

- Prototipar uma topologia com 7 nós, sendo 5 roteadores e 2 DTNs em um experimento emulado com RARE/freeRtr;
- Analisar os resultados alcançados com o experimento emulado, antes de começar a configurar as *whiteboxes*;
- Configurar os DTNs;
- Configurar o RARE/freeRtr nas *whiteboxes*;

- Implantar a configuração do experimento emulado nas *whiteboxes*;
- Executar os testes de *throughput* nos modos de Plano de Controle e de Dados;
- Analisar os resultados alcançados com os experimentos.

1.2 JUSTIFICATIVA

Ao observar que o RARE/freeRtr é uma possível resposta para a necessidade de um ambiente de experimentação de rede que integre protocolos legados em conjunto com novos protocolos emergentes, com ênfase na inovação e utilização de projetos *open source* (código aberto), este trabalho visa comprovar a viabilidade de instalação e configuração dessa ferramenta em *whiteboxes*.

Por sua vez, as *whiteboxes* promovem um passo adiante no objetivo de ser ter uma rede "mais aberta", possibilitando: simplicidade, baixo custo, aumento de performance e menos dependência de fornecedor. A utilização de hardware "genérico", ou *off-the-shelf* hardware (hardware de prateleira, em português), em conjunto com *open source Network Operating Systems* (sistemas operacionais de rede com código aberto), diminui significavelmente as limitações do desenvolvimento e manutenção de um testbed.

2 REFERENCIAL TEÓRICO

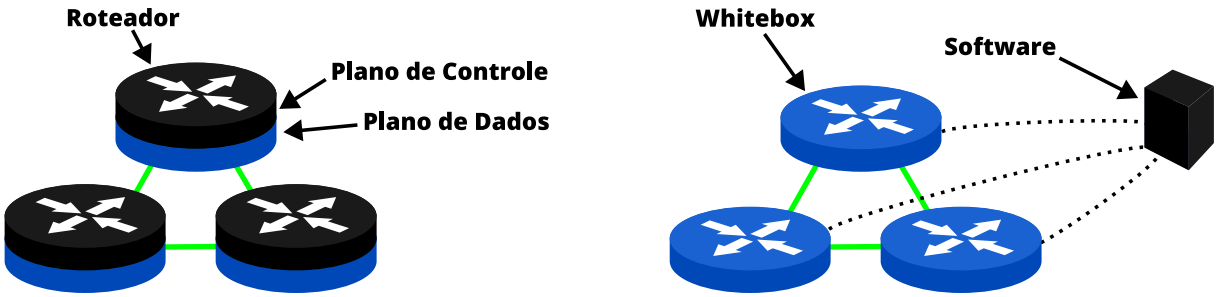
Neste capítulo são apresentados os principais resultados da revisão de literatura relacionados ao tema deste trabalho.

2.1 REDES DEFINIDAS POR SOFTWARE

Redes Definidas por Software (*Software-Defined Networks* em inglês), é uma abstração dos recursos de rede para um sistema virtualizado proposto por (MCKEOWN et al., 2008), separando as responsabilidades de encaminhamento (*forwarding*) e roteamento (*Routing*) dos datagramas, pois ambas as funcionalidades estão na camada de rede (*network layer* ou *layer 3*) do modelo OSI (KUROSE; ROSS, 2020). Diferentemente das camadas de transporte e aplicação, há uma parte da camada de rede em cada *host* e roteador na rede.

Antes que o paradigma SDN fosse amplamente conhecido na comunidade de redes acadêmicas, com o OpenFlow, houveram esforços iniciais da *Internet Engineering Task Force* (IETF), com o *Forwarding and Control Element Separation* (ForCES) na RFC 3746 (YANG et al., 2004). Até mesmo a mudança da abordagem distribuída para centralizada em relação às decisões de roteamento foi considerada antes do SDN com o *Path Computation Element (PCE)-Based Architecture* na RFC 4655 (FARREL; VASSEUR; ASH, 2006).

Figura 1 – Diferença entre Redes Tradicionais e SDN



(a) Rede Tradicional (b) Rede Definida por Software

*Fonte: adaptado de (CASADO; FOSTER; GUHA, 2014)

A principal diferença entre a SDN e a rede tradicional é a infraestrutura: a SDN é baseada em software, enquanto a rede tradicional é baseada em hardware. A camada

de rede pode ser decomposta em duas partes que interagem, o plano de dados e o plano de controle (KUROSE; ROSS, 2020).

A arquitetura da SDN pode ser dividida em cinco partes principais: os planos de gerenciamento, controle e dados e as interfaces para o norte (*Northbound Interface*) e para o sul (*Southbound Interface*), descritas a seguir.

2.1.1 Plano de Controle

Em um nível muito alto, o plano de controle estabelece o conjunto de dados local usado para criar as entradas da tabela de encaminhamento, que por sua vez são usadas pelo plano de dados para encaminhar o tráfego entre as portas de entrada e saída em um dispositivo (NADEAU; GRAY, 2013).

2.1.2 Plano de Dados

O plano de dados fornece modelos e abstrações de programação SDN ricos para gerenciar recursos de hardware (CASADO; FOSTER; GUHA, 2014). Ele lida com datagramas de entrada por meio de uma série de operações de nível de link que coletam o datagrama e executam verificações básicas de sanidade (NADEAU; GRAY, 2013).

2.1.3 *Northbound Interface*

A *Northbound Interface* (interface para o norte, em português), conceitua os detalhes de nível inferior (por exemplo, dados ou funções) usados por, ou no, componente (NADEAU; GRAY, 2013). O que permite que as aplicações utilizadas por administradores de rede efetuem o controle e o monitoramento das funções da rede sem ter que ajustar os detalhes mais finos da comunicação.

2.1.4 *Southbound Interface*

Uma interface que conceitua o oposto de uma interface para o norte. A *Southbound Interface* (interface para o sul, em português) normalmente é desenhada na parte inferior de um diagrama arquitetônico, atuando como uma ponte entre os elementos de controle e encaminhamento de dados. Este é um elemento fundamental para a

separação entre os planos de controle e dados. Exemplos de *southbound interfaces* incluem I2RS, NETCONF ou uma interface de linha de comando.

2.1.5 Plano de Gerenciamento

O plano de gerenciamento, segundo (WANG; MATTA, 2014), deve permitir o monitoramento e controle da rede. Ou seja, a própria camada de gerenciamento não gerencia a rede, mas fornece uma interface programática para softwares de gerenciamento (ou usuário), que por sua vez gerenciam a rede. Exemplos desses softwares incluem: controle de acesso, migração de máquina virtual (VM), seleção de caminho com reconhecimento de tráfego e adaptação de caminho e redirecionamento ou descarte de tráfego de ataque suspeito.

2.2 REDES EXPERIMENTAIS

Uma rede experimental, também conhecida como *testbed*, é uma plataforma para conduzir experimentos rigorosos, transparentes e replicáveis de teorias científicas, ferramentas computacionais e novas tecnologias (RNP, 2022). Além de permitir a inclusão de componentes de software, hardware e rede que, quando combinados, permitem a realização e a coleta dos dados resultantes desses experimentos.

O *testbed* também pode ser caracterizado como um conjunto de programas, dados e documentação de suporte que permitirá aos pesquisadores testar suas novas tecnologias em uma plataforma de software padrão (LINDVALL et al., 2007).

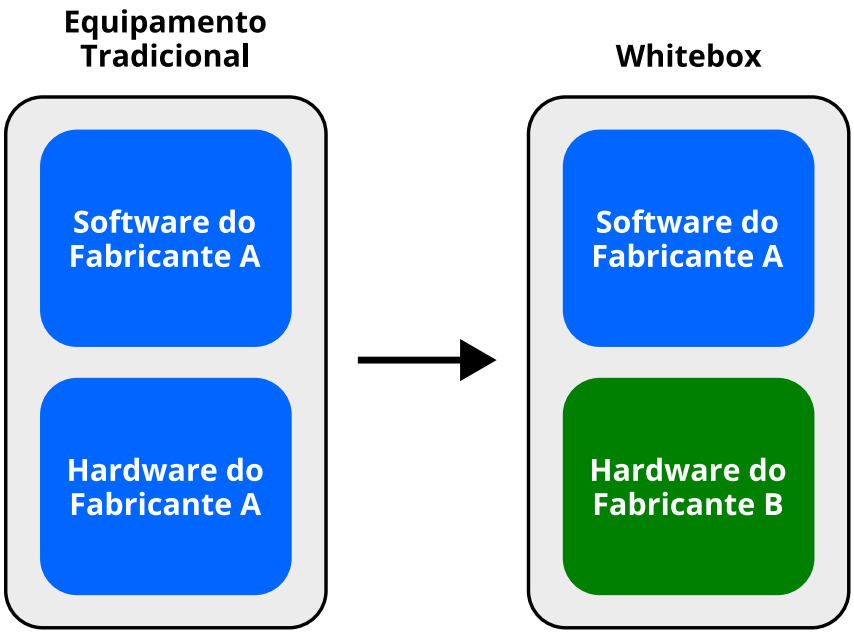
Um *testbed* de simulação comum pode suportar o compartilhamento de software e esforços, bem como estudos aprimorados das interações entre os diferentes subsistemas. Além disso, pode melhorar significativamente a integração dos resultados, pois impõe certos padrões sobre os esforços de desenvolvimento independentes (DUPUY et al., 1990).

Alguns dos *testbeds* mais conhecidos: FIBRE (SALLENT et al., 2012), GENI (BERMAN et al., 2014), GÉANT (FARINA; SZEGEDI; SOBIESKI, 2014), COSMOS (YU et al., 2019) e FABRIC (BALDIN et al., 2019).

2.3 *WHITEBOX*

Também conhecida como *white-box* e *white box*, ou “caixa branca” em tradução livre, o conceito de *whitebox networking* começou a ser adotado na área de redes de computadores como um esforço para remover a dependência do software da plataforma de hardware. O termo “*whitebox*” originalmente era usado para designar um computador pessoal ou servidor sem marca registrada que é montado com “peças de prateleira”, ou seja, produtos de hardware prontos e disponíveis para venda ao público em geral. (KO et al., 2013). As *whiteboxes* possibilitam um passo adiante na abertura da rede (BARGUIL; LOPEZ; GIMENEZ, 2020), permitindo executar qualquer sistema operacional na caixa branca que pudesse ser comprada independentemente. O sistema operacional pode ser desacoplado da plataforma de hardware usando uma *whitebox*, como na Figura 2.

Figura 2 – Equipamento Tradicional vs. *Whithebox*



*Fonte: adaptado de (PalC Networks, 2022)

2.4 RARE/FREERTR

O RARE/freeRtr[1] é um *Router OS Process* (tradução livre, processo do sistema operacional como roteador) gratuito e de código aberto. Projeto este que é usado e mantido pelo RARE[2] (*Router for Academia Research & Education*), que tem como objetivo criar um sistema operacional de roteador aberto e moderno que possa ser adaptado

[1] <http://freertr.org/>
[2] <https://wiki.geant.org/display/RARE/Home>

rapidamente para atender aos inúmeros casos de uso desafiadores da comunidade de Pesquisa e Educação (R&E, *Research and Education*) (LOUI et al., 2022). O RARE é um esforço contínuo sob o "guarda-chuva" da GÉANT[3], que é a colaboração das Redes Nacionais Europeias de Pesquisa e Educação (NRENs, *National Research and Education Networks*). Que possui o objetivo de entregar um ecossistema de informações de infraestrutura e serviços para avançar em pesquisa, educação e inovação em escala global.

A composição do nome RARE/freeRtr possui uma certa semelhança com o nome GNU/Linux, já que o projeto RARE fornece implementações para diferentes destinos de plano de dados programáveis que podem interagir com o plano de controle freeRtr por meio de uma API de mensagem comum bem definida (LOUI et al., 2022). Assim como o projeto GNU abarca o Kernel Linux, o projeto RARE abarcou o projeto freeRtr. O RARE/freeRtr implementa uma API simples de mensagem de texto comum para interagir com as APIs nativas de plano de dados, permitindo assim o acesso programável a partir do plano de controle, ao mesmo tempo em que oferece suporte a planos de dados intercambiáveis (LOUI et al., 2022).

O RARE/freeRtr também é definido como um "canivete suíço" devido às suas capacidades intrínsecas de (re)encapsular pacotes em várias interfaces. Destaca-se por ter implementado um portfólio de protocolos imenso e excepcional (BORGES et al., 2022b), além de ser desenvolvido usando a linguagem de programação Java, sem dependência a bibliotecas externas.

2.4.1 *Namespace*

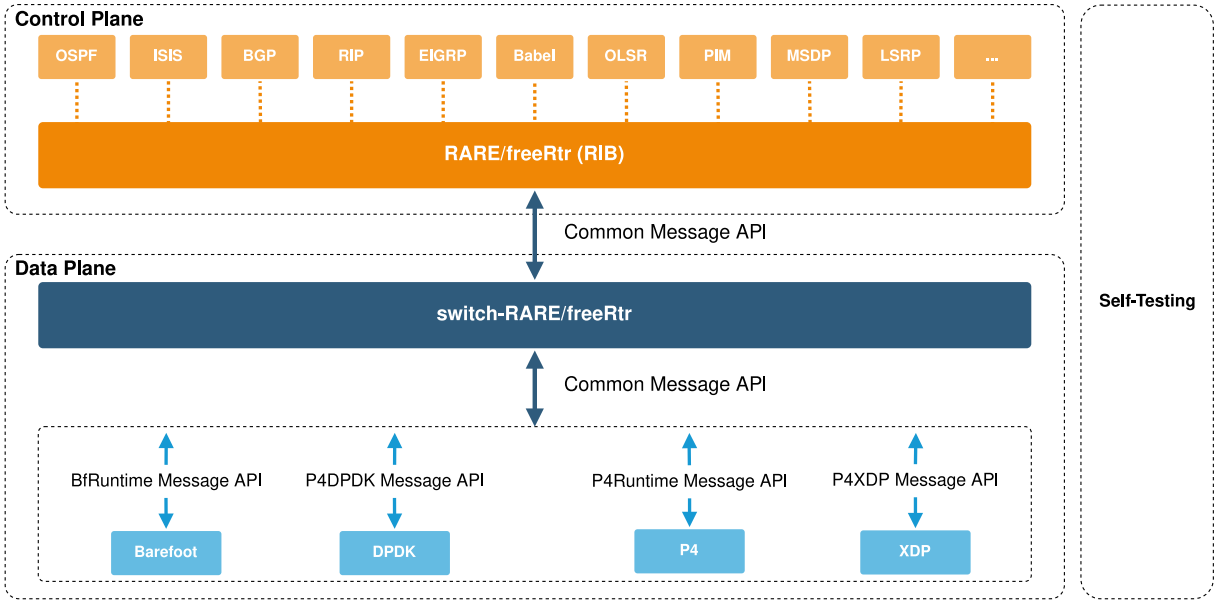
Presente no Kernel do Linux desde a versão 2.4.19, este recurso é utilizado em conjunto com o RARE/freeRtr na comunicação entre o Plano de Dados e Plano de Controle.

2.4.2 *Virtual Routing Forwarding*

Comumente chamado de VRF (ou VRFs), em português Encaminhamento de Roteamento Virtual, é um recurso do Kernel do Linux[4] e Cisco IOS[5], que foi implementado

[3] <https://geant.org/>
[4] <https://docs.kernel.org/networking/vrf.html>
[5] <https://cisco.com/c/en/us/td/docs/voice_ip_comm/cucme/vrf/design/guide/vrfDesignGuide.html>

Figura 3 – Arquitetura do RARE/freeRtr



*Fonte: adaptado de (BORGES et al., 2022b)

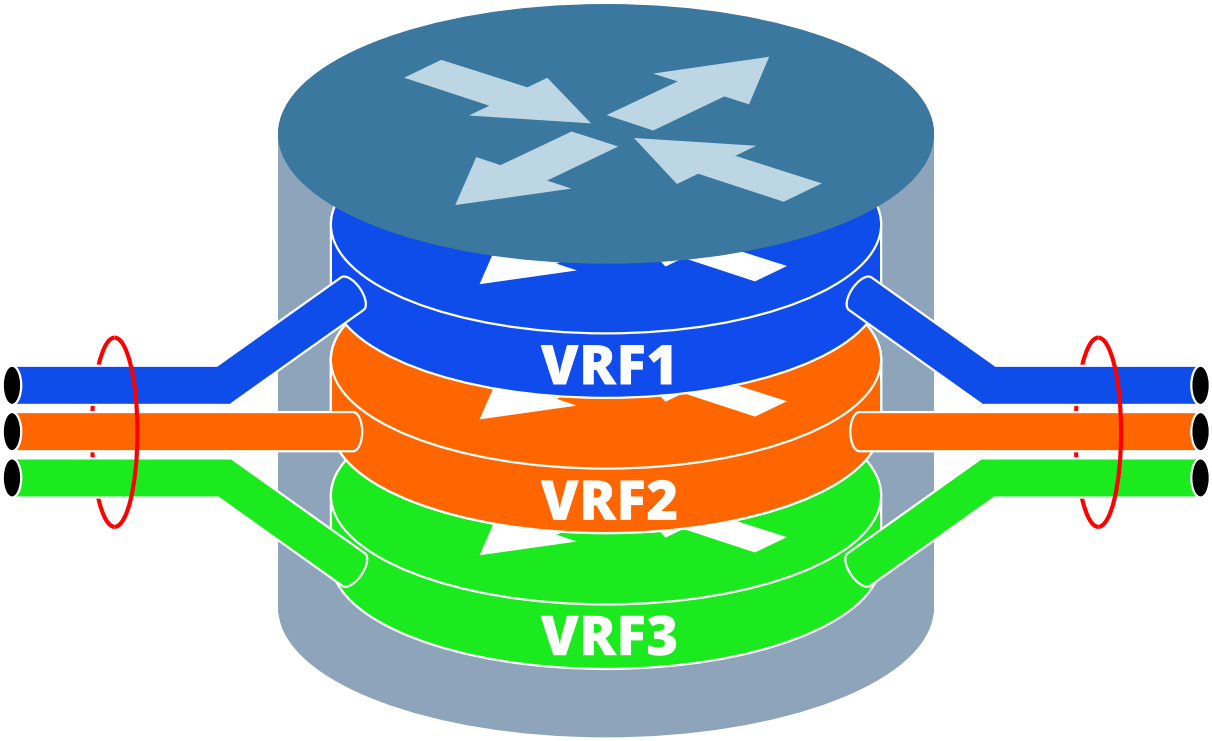
no RARE/freeRtr e é necessário para configurar a maioria de seus recursos.

O VRF está disponível no Kernel do Linux desde a versão 4.3.1[6], combinado com as regras IP este recurso fornece a capacidade de criar roteamento virtual e domínios de encaminhamento na pilha de rede do Linux. Outras características importantes do VRF são: cada inquilino (*tenant*) tem tabelas exclusivas de roteamento e diferentes *gateways* padrão, no mínimo; Os processos podem ser "*VRF aware*" ("conscientes de VRF", em português), ou seja, vinculando um soquete a um VRF, os pacotes usam a tabela de roteamento associada; O VRF permitem que VRFs sejam aninhados em *namespaces*; É possível utilizar regras de IP com prioridade mais alta (*Policy Based Routing*) para ter precedência sobre as regras do VRF que direcionam o tráfego específico conforme desejado; O VRF afeta apenas a camada 3 e camadas posteriores do modelo OSI, ou seja, as aplicações de camada 2 (como o LLDP[7]), não são afetadas.

No RARE/freeRtr, o VRF permite a separação de vários experimentos em uma mesma topologia. Esse recurso promove um "fatiamento" da rede como proposto por (SHERWOOD et al., 2010), com a diferença de que usando o VRF os experimentos são logicamente separados a partir da camada 3 (*layer 3*) do modelo OSI.

6 <https://cdn.kernel.org/pub/linux/kernel/v4.x/ChangeLog-4.3.1>
7 <https://en.wikipedia.org/wiki/Link_Layer_Discovery_Protocol>

Figura 4 – *Virtual Routing Forwarding*



*Fonte: adaptado de (CISCO, 2008).

2.4.3 *Policy-Based Routing*

Policy-Based Routing (PBR), Roteamento com Base em Políticas em português, foi adicionado no Kernel do Linux desde a versão 2.2[8], esse recurso que encaminha e envia pacotes de dados com base em políticas ou filtros. No Cisco IOS, esse recurso foi adicionado na versão 11.0[9]. Esse recurso também está presente no RARE/freeRtr.

[8] <https://en.wikipedia.org/wiki/Policy-based_routing>
[9] <http://ciscopress.com/content/downloads/cisco/bookreg/2237xxd.pdf>

3 MATERIAIS E MÉTODOS

Este capítulo descreve os materiais e métodos utilizados neste trabalho. Entende-se como materiais, todos os equipamentos eletrônicos, o laboratório, os softwares e os periféricos como cabos de rede. Os métodos foram todos os procedimentos e técnicas utilizadas para viabilizar a execução dos experimentos.

3.1 MATERIAIS

O *testbed* foi implantado no Laboratório de Desenvolvimento de Soluções - LEDS, localizado no Instituto Federal do Espírito Santo campus Cachoeiro de Itapemirim. O laboratório disponibiliza de toda a infraestrutura necessária para a realização dos experimentos. Os equipamentos foram configurados no dia 17 de agosto de 2022 e ficaram ativos até o dia 30 de Novembro de 2022.

Para compor o *testbed*, foram utilizados três *Desktop Appliances* Lanner modelo Luna D-125A com a seguinte especificação: processador Intel® Atom® C2316 (Rangeley), memória principal de 2GB DDR3L 1333 MHz Non-ECC, memória secundária de 8GB Onboard M.2 2242 (SATAIII), quatro portas GbE RJ-45, uma porta console RJ-45 e duas portas USB 2.0.

Outro modelo de *Desktop Appliance* também foi utilizado para dois equipamentos, com a seguinte especificação: Desktop Appliances Lanner NCA-1010A, processador Intel® Bay Trail E3815, memória principal de 8GB DDR3L 1067 MHz non-ECC, memória secundária de 16GB Onboard M.2 2242 (SATAIII), uma porta console RJ-45, três portas GbE RJ-45, uma porta USB 2.0, uma porta USB 3.0 e uma porta HDMI.

Para configuração dos *Desktop Appliances* e compor o *testbed* como nós geradores de tráfego, também conhecidos como DTNs (*Data Transfer Nodes*), foram utilizados dois computadores Dell Optiplex 3050 com a seguinte especificação: processador Intel® Kaby Lake Core i5-7500 3.4 Ghz, memória principal de 8GB DDR3, memória secundária de 500GB (SATAIII), uma porta GbE RJ-45, duas portas USB 2.0, duas portas USB 3.0, duas portas HDMI e uma placa *wireless* Realtek.

Para a configuração dos *Desktop Appliances*, foram utilizados: uma placa serial/COM, um cabo conversor serial/RJ-232 e um pendrive USB de 4GB. Para as conexões de rede foram usados 6 cabos CAT5-3 RJ-45.

No total foram utilizados sete equipamentos, sendo eles: três *Desktop Appliances* modelo Luna D-125A, dois *Desktop Appliances* modelo NCA-1010 e dois DTNs modelo Dell Optiplex 3050.

Para configurar os equipamentos que compõem *testbed* e realizar os experimentos, foram utilizados os seguintes softwares:

- UNetbootin para Linux 64-bit (v7.02);
- Minicom (v2.8);
- Sistema Operacional Debian Bullseye/SID (v11.5.0) em todos os equipamentos (Luna D-125A, NCA-1010A e Dell Optiplex 3050);
- Kernel Linux (v6.0.0);
- RARE/freeRtr (v22.10.15) apenas nas *whiteboxes Desktop Appliances Lanner* (Luna D-125A e NCA-1010A);
- GNU Bash (v5.1.16);
- tmux (v3.2a);
- iperf3 (3.11).

Outros materiais como cabos de força, roteador *wireless* e monitores, foram utilizados mas não foram listados, pois não interferem diretamente na execução dos experimentos.

3.2 MÉTODOS

Neste trabalho, o RARE/freeRtr é um habilitador que permite a prototipação e implantação de uma rede experimental, também conhecida como *testbed*, de classe de operadora (*carrier-class*), aberta e portátil utilizando *whiteboxes*. Uma das principais vantagens de utilizá-lo é o acesso a "terra dos protocolos" legados, ou "*protocoland*" (BORGES et al., 2022b), que é o plano de controle que fornece todos os recursos e protocolos esperados de um sistema operacional de roteador de nível empresarial, com suporte adicional para dispositivos de *hardware* com suporte a linguagem P4 e novos protocolos personalizados (LOUI et al., 2022), por exemplo o PolKA (DOMINICINI et al., 2020).

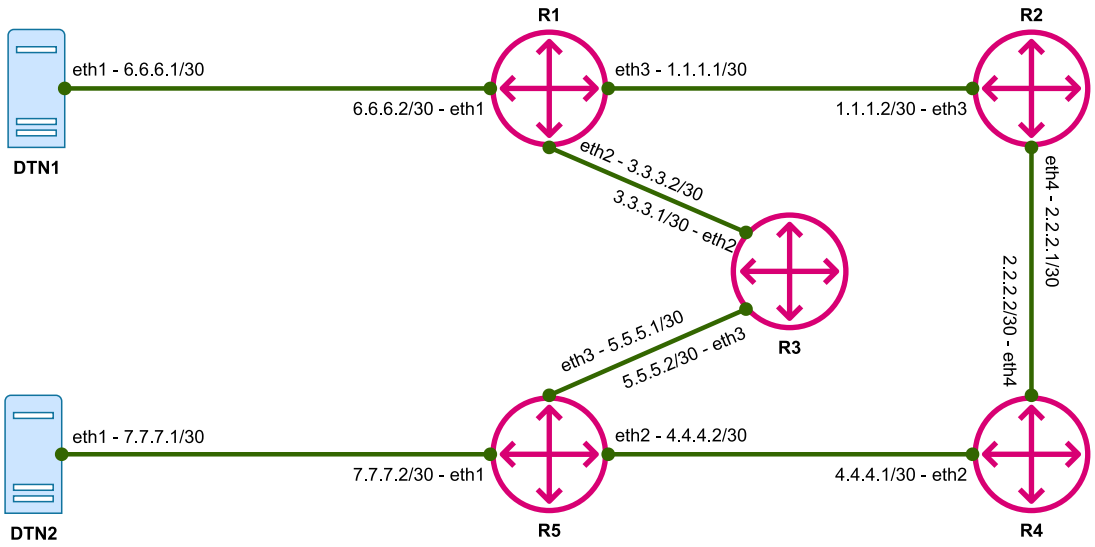
Outro recurso importante do RARE/freeRtr são as implementações para diferentes alvos de planos de dados programáveis (LOUI et al., 2022), que podem interagir com o plano de controle, por meio de uma API bem definida. Essas implementações são as APIs chamadas: `P4Runtime`, `BfRuntime`, `P4DPDK` e `P4XDP`. Cada uma delas interagindo com a APIs de mensagens nativas específica do *hardware* alvo, ou seja, `P4Runtime` interage com a API nativa de hardwares que suportam a linguagem P4; `BfRuntime` com os hardwares Tofino; `P4DPDK` com *hardwares* Intel que tenham suporte a este recurso; e `P4XDP` para *hardwares* com Linux (v4.8+) e acesso a biblioteca eBPF/XDP.

Apesar dos *Desktop Appliances* Lanner D-125A e NCA-1010A disporem de um processador Intel, não foi possível utilizar a API P4DPDK, já que os modelos de processador Rangeley e Bay Trail, respectivamente, não possuem suporte a biblioteca VT-X necessária para a utilização do `P4DPDK`. Desta forma, utilizamos a configuração apenas para o plano de controle do RARE/freeRtr.

Para atingir os objetivos deste trabalho, que são a prototipação e implantação de um *testbed* utilizando *whiteboxes*, foram realizados os seguintes procedimentos: i) a emulação e validação da topologia; ii) a configuração do *testbed*.

Para a execução dos experimentos, emulado e físico, foi escolhida uma topologia próxima a topologia do RARE (DOMINICINI et al., 2021) e (BORGES et al., 2022a), como apresentada na Figura 5.

Figura 5 – Topologia



*Fonte: o próprio autor

3.2.1 Testes em ambiente emulado

Seguindo um dos passos propostos por (BORGES et al., 2022b), que é a validação da configuração da topologia utilizando o RARE/freeRtr antes de iniciar a configuração e implantação nas *whiteboxes*, a topologia para os experimentos, como mostra a Figura 5, foi emulada utilizando as configurações apresentadas no Apêndices A e B. Neste experimento, foi utilizado o protocolo de roteamento dinâmico OSPF[1], como apresentado em (BORGES et al., 2022a).

Para realizar testes com o RARE/freeRtr no ambiente emulado, é necessário utilizar o JAR da aplicação, obtido no site da ferramenta[2]. Outro requisito é o JRE, ou JDK, compatível com Java 11, para a versão do RARE/freeRtr utilizada neste trabalho (v22.6.11). Com estes dois pré-requisitos, é possível executar o comando apresentado abaixo:

Execução do JAR do RARE/freeRtr

```
java -jar rtr.jar
```

O comando exibe as seguintes informações:

[1] <https://datatracker.ietf.org/doc/html/rfc1247>
[2] <http://freertr.org/rtr.jar>

Exemplo do arquivo de hardware

```
java -jar rtr.jar <parameters>
parameters:
  router <cfg>                    - start router background, config url
  routerc <cfg>                   - start router with console, config url
  routerw <cfg>                   - start router with window, config url
  routercw <cfg>                  - start router with console and window,
  ↪  config url
  routers <hwcfg> <swcfg> [hwcfg] - start router from separate configs,
  ↪  config url, config url, config url
  routers <hwcfg> <swcfg>         - start router from separate configs,
  ↪  config url, config url
  routera <swcfg>                 - start router with sw config only,
  ↪  config url
  test <cmd>                      - execute test command, command to
  ↪  execute
  show <cmd>                      - execute show command, command to
  ↪  execute
  exec <cmd>                      - execute exec command, command to
  ↪  execute
  cfgexec <swcfg> <cmd>           - execute exec command, config url,
  ↪  command to execute
```

Os comandos `router`, `routerc`, `routerw`, `routercw`, `routers` e `routera`, são utilizados para iniciar o RARE/freeRtr. O comando `test` é usado para validar uma instrução. O comando `show` exibe as configurações definidas na aplicação. O comando `exec` executa uma instrução na aplicação. Por último, o comando `cfgexec` executa uma instrução na aplicação de acordo com o arquivo de configuração passado como parâmetro.

Cada roteador deverá ter arquivos separados de hardware e software. O arquivo de hardware define informações sobre a plataforma, interfaces, tradução de porta externa para a porta do *namespace* da aplicação do RARE/freeRtr, processo externo iniciado e assistido pela aplicação do RARE/freeRtr, etc. A principal configuração do arquivo de hardware são as interfaces, pois elas definem as ligações físicas entre os roteadores, como se fossem os “cabos de rede”. Segue abaixo a descrição da configuração das interfaces:

Descrição da configuração das interfaces.

```
int <intf_name> <intf_type> <intf_mac> <ip_socket_a> <port_socket_a>
↪  <ip_socket_b> <port_socket_b>
```

Segue abaixo um exemplo de configuração de uma interface, de acordo com os parâmetros apresentados:

Exemplo de configuração das interfaces.

```
int eth1 eth 0000.1111.0001 127.0.0.1 10012 127.0.0.1 10021
```

Nesse exemplo, foi declarada uma interface `int` com o nome `eth1` do tipo `eth` Ethernet[3], com o endereço MAC `0000.1111.0001` e um soquete UDP com os IPs e portas do roteador de origem para o roteador destino, ou seja, o IP do roteador de origem `127.0.0.1` que é o endereço da `loopback` do SO hospedeiro, já que estamos executando um ambiente emulado, com a porta `10012`. Apontando para o IP do roteador de destino `127.0.0.1`, repetindo o endereço do `localhost`, com a porta `10021`.

Neste trabalho, foi utilizada uma padronização dos parâmetros do arquivo de hardware com o objetivo de facilitar a configuração dos demais arquivos dos roteadores que compõem a topologia. No endereço MAC, os quatro primeiros algarismos foram atribuídos o valor zero, `0000`, do 5° algarismo até o 8°, para identificar qual roteador está sendo configurado, foi atribuído o número do roteador, ou seja, para o roteador 1 (R1), foi atribuído o valor 1 para os quatro algarismos, `1111`. Os últimos quatro algarismos, dos doze disponíveis, foram utilizados para identificar o número da interface, ou seja, para a interface `eth1` foi atribuído o valor 1 para o último algarismo, `0001`. Para as portas que compõem o soquete UDP, foi utilizado um valor acima de `10000` para não conflitar com serviços do SO, e abaixo da última porta disponível `65535`. Os dois últimos algarismos do endereço MAC foram usados para identificar o número do roteador que faz parte do soquete, neste exemplo, os roteadores R1 e R2. É necessário estabelecer uma ligação de ida e outra de volta, como `127.0.0.1 10012 127.0.0.1 10021`. Note que os dois últimos números devem ser invertidos, no segundo parâmetro porta, para estabelecer essa ligação.

Outra configuração importante do arquivo de hardware, no ambiente emulado, é o mapeamento da porta externa do SO para a porta interna da aplicação do RARE/freeRtr. Abaixo é possível ver um exemplo desta configuração:

[3] <https://www.ieee802.org/3/>

Exemplo do arquivo de hardware

```
int eth1 eth 0000.1111.0001 127.0.0.1 10012 127.0.0.1 10021
tcp2vrf 2121 v1 23
```

O comando `tcp2vrf 2121 v1 23` estabelece uma ligação entre a porta `2121` do SO com a porta `23` dentro da aplicação do RARE/freeRtr, usando a VRF `v1`. No RARE/freeRtr é obrigatório declarar uma VRF para encaminhamento e recebimento dos pacotes.

O arquivo de software do RARE/freeRtr é similar ao arquivo `startup-config` do sistema operacional Cisco IOS. É neste arquivo que são configurados os parâmetros das interfaces, protocolos de roteamento, serviços de rede, etc. No Apêndice A é apresentado uma amostra do arquivo de configuração usado neste experimento. Foram configurados 5 roteadores e 2 DTNs, com um arquivo de hardware e software cada. Vale a pena ressaltar que os DTNs também são instâncias do RARE/freeRtr, ou seja, ao todo foram utilizados 7 roteadores emulados. Essa decisão foi tomada com o intuito de facilitar a configuração e testes dos nós que compõem a topologia. Abaixo é possível verificar um exemplo da configuração do arquivo de software.

Exemplo do arquivo de software

```
hostname r1
no buggy
!
vrf definition v1
  exit
!
interface ethernet3
  description r1 -> r2
  vrf forwarding v1
  ipv4 address 1.1.1.1 /30
  ipv6 address 1111::1 /64
  template template1
  no shutdown
  exit
!
interface template1
  description template interface r1
  lldp enable
  vrf forwarding v1
  ipv4 address dynamic dynamic
  ipv6 address dynamic dynamic
  router ospf4 1 enable
  router ospf4 1 area 0
  router ospf6 1 enable
  router ospf6 1 area 0
  shutdown
  no log-link-change
  exit
!
server telnet tel
  security protocol telnet
  exec timeout 10000000
  exec logging
  exec colorize prompt
  no exec authorization
  no login authentication
  login logging
  vrf v1
  exit
!
end
```

A primeira declaração no arquivo de software é o nome do roteador, ou `hostname`. Neste experimento, utilizaremos os nomes dos roteadores como `r1`, `r2`, `r3`, `r4` e `r5`, e os DTNs como `dtn1` e `dtn2`. Como no Cisco IOS, são utilizados pontos de exclamação (`!`) para indicar saltos de linha nas instruções. Outra configuração utilizada foi o parâmetro `no buggy`, quando habilitado, este comando inclui funcionalidades experimentais na aplicação. Utilizando o comando `no` é possível desabilitar essa instrução assim como pode ser utilizado para desabilitar outras configurações. Neste trabalho foi utilizado o protocolo de roteamento dinâmico OSPF, é necessário declarar o `router ospf4 1` e `router ospf6 1` para habilitar esse protocolo no roteador e permitir a troca de informações com os outros roteadores que compõem a topologia. Note que é preciso declarar os parâmetros `ospf4` e `ospf6` para habilitar as trocas de IPs com os protocolos IPv4 e IPv6 respectivamente. Ainda no parâmetro `router` é importante notar a declaração do `id` do processo (`1`), que pode ser representado por um número de `1` a `65.535`[4]. O número utilizado como `id` é localmente significativo, o que significa que não precisa ser o mesmo valor nos outros roteadores OSPF para estabelecer adjacências com esses vizinhos. Em seguida é utilizado o comando `interface template1` que cria um conjunto de especificações que podem ser aplicadas as outras interfaces. Evitando dessa forma incompatibilidade de configurações entre as interfaces.

Para acesso externo a aplicação de um roteador, é configurado um servidor *Telnet*, com a instrução `server telnet tel`. Com base na configuração do arquivo de hardware `tcp2vrf 2121 v1 23`, permitindo o acesso ao roteador através do comando `telnet 127.0.0.1 2121`, no SO hospedeiro. Também é possível acessar um roteador através de outro, com o comando `telnet 1.1.1.1 vrf v1`. É obrigatório indicar o IP do roteador e o VRF configurado, respectivamente como `1.1.1.1 vrf v1`. Esta configuração possibilita dinamicidade na execução dos comandos entre os roteadores.

Após configurar os arquivos de hardware e software, é executado o comando `java -jar rtr.jar routersc`, no `shell` do SO hospedeiro. É necessário completar o comando `routersc` com o caminho relativo ou absoluto dos arquivos de hardware e software. Para este trabalho utilizamos a padronização dos nomes dos arquivos para `rx-hw.txt` e `rx-sw.txt`, onde `x` é o número do roteador, ou seja, `r1-hw.txt` e `r1-sw.txt` para o

[4] <https://ccna.network/id-do-roteador-ospf/>

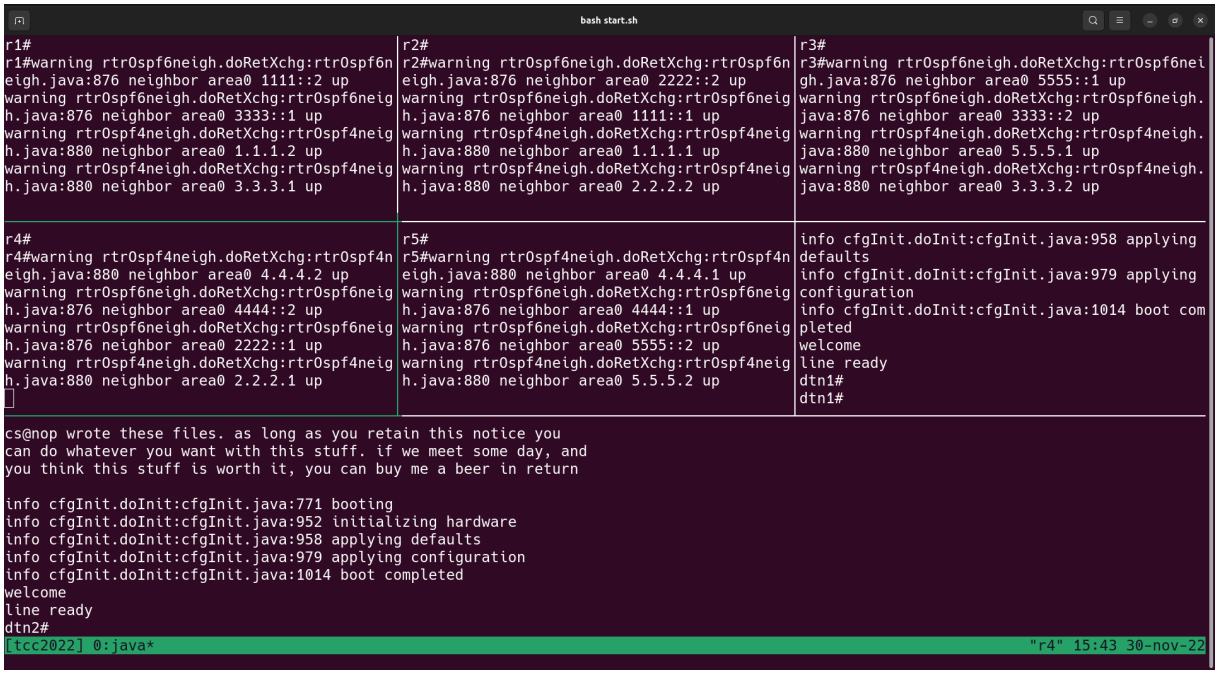
roteador R1. Abaixo é possível verificar o comando completo:

Execução do rtr.jar

```
java -jar /path/rtr.jar routersc /path/r1-hw.txt /path/r1-sw.txt
```

Com o objetivo de agilizar a execução de toda a topologia, foi utilizado um multiplexador de terminal de código aberto para sistemas operacionais do tipo Unix, chamado `tmux`. Desta forma, com apenas um comando foi possível instanciar toda a topologia. O código do script `tmux` está no Apêndice C. Foi desenvolvido o script `start.sh` para iniciar a topologia, que utiliza o comando `new-session` do `tmux`, e o script `stop.sh` para encerrar essa sessão, liberando o acesso ao `shell`. O terminal exibe informações de *log* de todos os roteadores em execução, como mostrado na figura 6.

Figura 6 – Automatização com tmux



*Fonte: o próprio autor

Em outro terminal, é possível executar o Telnet para acessar os roteadores instanciados, com o comando `telnet 127.0.0.1 2121`. Além de permitir um gerenciamento de toda topologia, essa abordagem habilita duas funcionalidades muito importantes para execução das configurações, que são o histórico dos comandos executados, seta para cima no teclado (↑), e o autocompletar dos comandos com a tecla `TAB`. O que não é possível apenas com o `console` habilitado através do comando `java -jar /path/rtr.jar routersc`.

Uma informação importante sobre o funcionamento do RARE/freeRtr pode ser obtida através do comando `pidof java`, pois é possível verificar as instâncias dos roteadores em execução, como exibido no comando abaixo:

Verificando as instâncias do RARE/freeRtr

```
$ pidof java
1413595 1413585 1413525 1413524 1413507 1413261 1412495
```

O comando `pidof` exibe os IDs dos processos de um programa específico em execução, neste caso o `java`. Ou seja, são exibidos os 7 IDs dos processos dos roteadores que foram instanciados no SO hospedeiro.

Com base nas informações apresentadas nesta subseção e modelos das configurações utilizadas nos Apêndices A, B e C, é possível replicar todos os experimentos apresentados neste trabalho. Também é possível acessar as configurações utilizadas neste trabalho no repositório do GitHub[5]

3.2.2 Testes em ambiente físico

Após a validação, da topologia e do experimento, em um ambiente emulado, iniciou-se a etapa da implantação e configuração do *testbed* com as *whiteboxes*. Para configuração dos DTNs e das *whiteboxes*, foi escolhido o SO Debian Bullseye baseando-se na indicação dos desenvolvedores do RARE/freeRtr, mas é possível utilizar qualquer SO baseado em Debian. Além disso, existe a possibilidade de utilizar um repositório no SO Fedora[6]. Como as *whiteboxes* Luna D-125A não possuem uma porta HDMI, foi necessário utilizar uma placa serial em um dos computadores Dell Optiplex 3050 e um cabo conversor serial para RJ-232 para estabelecer conexão e instalar o SO. O mesmo procedimento não é obrigatório para o modelo NCA-1010A, já que ele possui uma porta HDMI.

Um disco de inicialização com o SO Debian Bullseye foi criado utilizando o software UNetbootin e um pendrive. No Dell Optiplex 3050, foi utilizado o software Minicom para estabelecer a conexão com as *whiteboxes*, através do comando `minicom -s on`. É necessário configurar a conexão da porta serial para `115200 8N1 Bps/Par/Bit`.

[5] <https://github.com/edgardcunha/tcc2022>
[6] <https://copr.fedorainfracloud.org/coprs/nucleo/freerouter/>

Estabelecida a conexão, é realizado o *boot* com o pendrive e inicia-se o processo de instalação. É necessário modificar os parâmetros:

Parâmetros de instalação do Debian

```
/install.amd/vmlinuz vga=788 initrd=/install.amd/initrd.gz --- quiet
```

Para:

Modificando os parâmetros de instalação do Debian

```
/install.amd/vmlinuz vga=off initrd=/install.amd/initrd.gz --- quiet
console=ttyS0,115200n8
```

O que possibilita prosseguir o processo de instalação normalmente. Dado o tamanho de 8GB da memória secundária do modelo Luna D-125A, é escolhido o procedimento de uso de todo o disco para evitar erros de espaço insuficiente.

Após a instalação do Debian é necessário a conversão da versão estável (*stable*) para a versão SID, prerrequisito para instalação do RARE/freeRtr e configuração da interface *wireless* (nos DTNs). Antes da instalação do RARE/freeRtr, também é necessário instalar o pacote `wget`. Como apresentado na documentação[7], para iniciar a instalação do é necessário executar o comando: `wget http://freertr.org/install.sh ; sudo bash install.sh`, e digitar `yes`, lembre-se de que este procedimento irá remover a interface gráfica, caso esteja utilizando, além de instalar outros prerrequisitos necessários para configuração e uso do RARE/freeRtr. Ou seja, não é recomendado para computadores que não tenham a função específica de uma equipamento de rede. Outra informação importante, é que o RARE/freeRtr controlará as interfaces de rede e não mais o sistema operacional.

Após a instalação do RARE/freeRtr e reiniciar o equipamento, para acessar o *Desktop Appliance* Luna D-125A, através da porta console com software Minicom é necessário modificar a configuração da conexão da porta serial para `9600 8N1 Bps/Par/Bit`, que foi alterada após a instalação. As informações de inicialização do SO não serão mais exibidas, apenas o *banner* de inicialização do RARE/freeRtr. Vale ressaltar que é possível acessar o `shell` do SO através do comando `test bash`, o que permite a

[7] http://www.freertr.org/

interação com o SO diretamente da conexão com a porta serial. Esta funcionalidade é disponibilizada pelo parâmetro `buggy` no arquivo de software.

Para acessar o RARE/freeRtr, é preciso antes acessar o SO através de uma conexão SSH, em seguida, é necessário utilizar o comando `telnet 10.255.255.254`. Após a instalação, o RARE/freeRtr se tornou um serviço do SO e pode ser conferido com o comando `systemctl status rtr.service`. É importante ressaltar que a instalação do RARE/freeRtr, inicialmente, é configurada para o modo de Plano de Controle. Ou seja, não existe configuração para acessar o Plano de Dados do equipamento.

Os testes do Capítulo 4 foram executados nos dois modos de configuração. Para habilitar o Plano de Dados, após a instalação básica do RARE/freeRtr, é necessário executar os seguintes comandos apresentados a seguir.

Instalação do Plano de Dados emulado (P4emu)

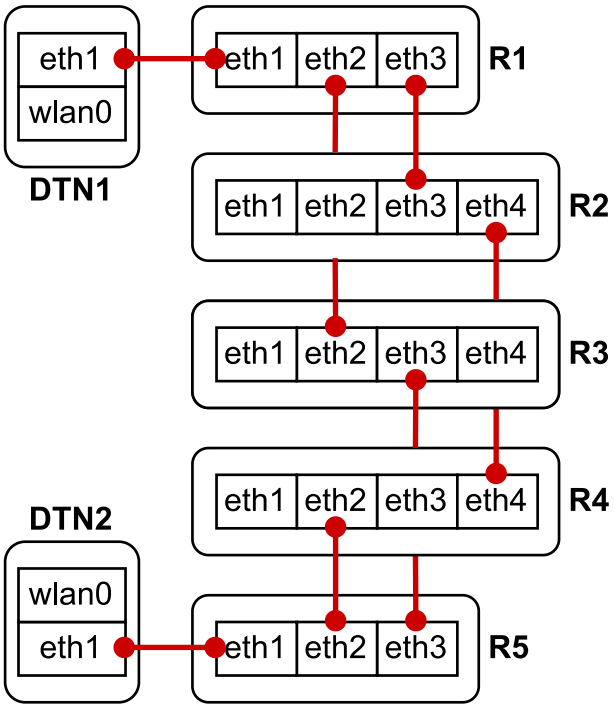
```
freertr#write
freertr#test hwext path /rtr/rtr- dataplane p4emu
freertr#reload cold
```

Salienta-se, que este trabalho utilizou a API `P4Emu` para habilitar o Plano de Dados. Esta API utiliza a biblioteca `libpcap` do Linux para emular o Plano de Dados. Para habilitar outra API, é necessário alterar o nome `p4emu` para o nome da API correspondente ao hardware alvo.

Como apresentado na Figura 7, o *testbed* foi organizado da seguinte forma: as *whiteboxes* modelo NCA-1010A, foram nomeadas como R1 e R5, os equipamentos do modelo Luna D-125A receberam os nomes de R2, R3 e R4. Os computadores modelo Dell Optiplex 3050 foram nomeados como DTN1 e DTN2.

Para recriar uma topologia próxima ao testbed do RARE (DOMINICINI et al., 2021), (BORGES et al., 2022a), as seguintes ligações físicas foram realizadas, de acordo com as figuras 5 e 7: DTN1@eth1 -> R1@eth1, R1@eth2 -> R3@eth2, R1@eth3 -> R2@eth3, R2@eth4 -> R4@eth4, R4@eth2 -> R5@eth2, R5@eth3 -> R3@eth3, R5@eth1 -> DTN2@eth1. Foram utilizados sete cabos de rede CAT5-3 RJ-45.

Figura 7 – Organização do *Testbed*



*Fonte: o próprio autor

4 RESULTADOS

Neste capítulo são apresentados os resultados dos testes executados no *testbed*, bem como a análise das informações obtidas.

4.1 ANÁLISE DOS RESULTADOS

Para analisar os resultados dos testes executados no *testbed*, antes é necessário compreender os possíveis caminhos propostos para a topologia. Na subseção 4.1.1 é demonstrada a diferença entre o caminha mais curto *shortest path* e o caminho mais longo *longest path*. Foram propostos 3 testes para demonstrar a diferença entre modo de configuração, nas *whiteboxes*, do Plano de Controle e Plano de Dados (*Control Plane* e *Data Plane*). Os testes executados em cada modo configuração foram: *Throughput* TCP, *Throughput* UDP e *Throughput* UDP reverso. Nas subseções 4.1.2 e 4.1.3 são apresentados os resultados destes testes.

4.1.1 *Shortest Path vs. Longest Path*

A partir da topologia apresentada na Figura 5, o *testeb* possui no mínimo dois caminhos possíveis, um mais curto (*shortest path*) e outro mais longo (*longest path*). O OSPF sempre escolhe o menor caminho, ou seja, passando pelos roteadores R1 $\leftrightarrow$ R3 $\leftrightarrow$ R5. É possível conferir esse caminho pelo número de saltos (*hops*) resultante do comando `traceroute 7.7.7.1`, executado no DTN1.

Demonstração do caminho mais curto

```
root@dtn1:~ # traceroute 7.7.7.1
traceroute to 7.7.7.1 (7.7.7.1), 30 hops max, 60 byte packets
 1  6.6.6.2 (6.6.6.2)  4.567 ms  4.571 ms  4.534 ms
 2  3.3.3.1 (3.3.3.1)  5.552 ms  5.501 ms  5.450 ms
 3  5.5.5.1 (5.5.5.1)  5.534 ms  6.436 ms  6.388 ms
 4  7.7.7.1 (7.7.7.1)  3.299 ms  3.301 ms  3.305 ms
```

O comando exibe apenas 4 saltos (*hops*), passando pelos roteadores e interfaces `r1@eth1` (`6.6.6.2`), `r3@eth2` (`3.3.3.1`), `r5@eth3` (`5.5.5.1`) até chegar no `dtn2@eth1` (`7.7.7.1`). Como foi mencionado no capítulo 3 subsecção 3.2.2, vale a pena ressaltar que os DTNs possuem uma rota estática para os roteadores que estão diretamente

conectados a eles, ou seja, o DTN1 está conectado ao roteador R1 e o DTN2 ao roteador R5. Como apresentado nas Figuras 5 e 7. Desta forma, são os roteadores R1 e R5, que estão na borda da topologia e possuem o protocolo de roteamento dinâmico (OSPF) ativado, ou seja, são estes nós de borda que resolvem o próximo salto dos DTNs.

O caminho mais longo (*longest path*), passa pelos roteadores R1 $\leftrightarrow$ R2 $\leftrightarrow$ R4 $\leftrightarrow$ R5, e pode ser conferido quando o roteador R3 não está acessível (desligado, por exemplo) ou quando os outros roteadores não possuem conectividade com as interfaces `r3@eth2` e/ou `r3@eth3`. Este caminho pode ser verificado através dos seguintes comandos executados no roteador R3:

Interface r3@sdn2 em modo admin

```
r3#conf
r3(cfg)#int sdn2
r3(cfg-if)#shutdown
```

Após executar estes comandos, a interface `sdn2`, no roteador R3, entra no modo `admin` (administrador, em português), note que esta interface possui o nome `eth2` na configuração das *whiteboxes* no modo de Plano de Controle (*Control Plane*) e no ambiente emulado, portanto é necessário utilizar o comando `r3(cfg)#int eth2` para acessá-la com nestas configurações. Em seguida, é possível executar novamente o comando `traceroute 7.7.7.1` no DTN1 para verificar o caminho percorrido.

Demonstração do caminho mais longo

```
root@dtn1:~ # traceroute 7.7.7.1
traceroute to 7.7.7.1 (7.7.7.1), 30 hops max, 60 byte packets
 1  6.6.6.2 (6.6.6.2)  3.827 ms  3.749 ms  5.844 ms
 2  1.1.1.2 (1.1.1.2)  7.739 ms  9.060 ms  10.002 ms
 3  2.2.2.2 (2.2.2.2)  2.817 ms  2.942 ms  2.890 ms
 4  4.4.4.2 (4.4.4.2)  7.419 ms  7.366 ms  7.348 ms
 5  7.7.7.1 (7.7.7.1)  8.332 ms  8.283 ms  8.233 ms
```

Este comando exibe 5 saltos (*hops*), ao invés de 4 saltos do caminho mais curto, passando pelos roteadores e interfaces `r1@eth1` (`6.6.6.2`), `r2@eth3` (`1.1.1.2`), `r4@eth4` (`2.2.2.2`), `r5@eth2` (`4.4.4.2`) até chegar no `dtn2@eth1` (`7.7.7.1`). Demonstrando desta forma a diferença entre os caminhos: *Shortest Path* e *Longest Path*.

4.1.2 Plano de Controle

No modo de configuração do Plano de Controle (*Control Plane*) das *whiteboxes*, apresentada na seção 3.2 e subseção 3.2.2, foram executados os testes: ***Throughput* TCP**, ***Throughput* UDP** e ***Throughput* UDP reverso**. A seguir são apresentados os resultados destes testes.

Foi utilizada a ferramenta `iperf3`[1] para todos os testes realizados neste trabalho, explorando o caminho mais curto *shortest path*, como explicado na subseção 4.1.1. Esta ferramenta foi escolhida devido a possibilidade de definir vários parâmetros que podem ser usados para testar, otimizar, e/ou ajustar uma rede. Essa ferramenta possui a funcionalidade de configurar clientes e servidor, o que permite medir a taxa de transferência entre as duas extremidades, uni ou bidirecionalmente.

Para executar os testes, é necessário iniciar modo servidor do `iperf3`. Abaixo, é possível verificar o comando responsável por iniciar esse serviço no DTN2. Ou seja, o DTN2 será o servidor e o DTN1 o cliente.

Serviço iperf3 modo servidor

```
root@dtn2:~# iperf3 -s -p 3000
-----------------------------------------------------------
Server listening on 3000 (test #1)
-----------------------------------------------------------
[...]
```

O parâmetro `-s` (ou `--server`) indica que o DTN2 é o servidor `iperf3` que recebe o envio dos dados. O parâmetro `-p 3000` (ou `--port 3000`), indica que o cliente deve enviar os dados para a porta `3000`. As requisições recebidas foram omitidas pelo espaço reservado `[...]`. Abaixo é apresentado o comando executado no DTN1.

Serviço iperf3 modo cliente

```
root@dtn1:~# iperf3 -c 7.7.7.1 -p 3000
Connecting to host 7.7.7.1, port 3000
[  5] local 6.6.6.1 port 32910 connected to 7.7.7.1 port 3000
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[...]
```

[1] <https://iperf.fr/iperf-download.php>

O comando acima utilizou o parâmetro `-c` (ou `--client`) para definir o modo cliente do `iperf3`. Em seguida, é informado o IP do servidor `iperf3`, ou seja, o IP `7.7.7.1` do DTN2. Também foi informado o número da porta configurada no servidor, com o parâmetro `-p 3000` (ou `--port 3000`). Por padrão, o `iperf3` utiliza o protocolo TCP para qualquer comando executado para enviar requisições, caso não especifique outro protocolo. As requisições enviadas foram omitidas pelo espaço reservado `[...]`. Para realizar o **Teste de *Throughput* TCP**, foram adicionados os seguintes parâmetros.

Teste de *Throughput* TCP

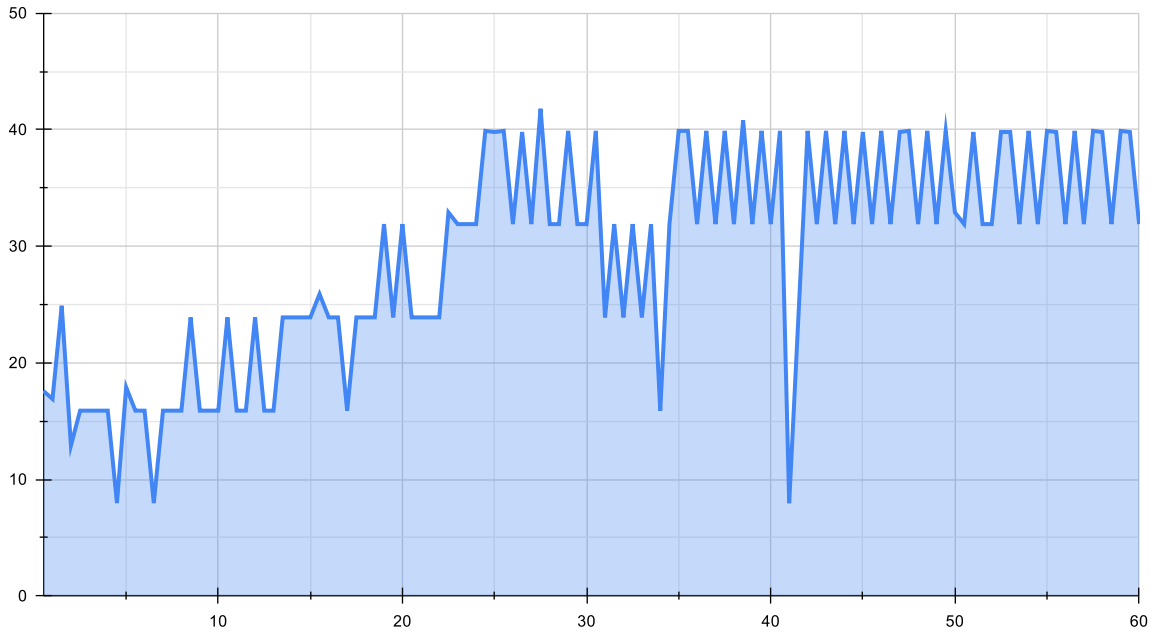
```
root@dtn1:~# iperf3 -c 7.7.7.1 -p 3000 -i 0.5 -t 60
Connecting to host 7.7.7.1, port 3000
[  5] local 6.6.6.1 port 32910 connected to 7.7.7.1 port 3000
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[...]
```

O parâmetro `-i` (ou `--interval`) foi adicionado para especificar um intervalo em segundos para execução de cada requisição. Foi escolhido o valor de `0.5` segundos (ou `500` milissegundos) para este parâmetro, valor este atribuído para melhorar a formatação do gráfico. O parâmetro `-t` (ou `--time`) é usado para definir o tempo em segundos para transmitir os dados, por padrão esse tempo é definido para `10` segundos. Foi utilizado o valor de `60` segundos (ou 1 minuto) para este parâmetro, em todos os comandos executados no DTN1. As requisições enviadas foram omitidas pelo espaço reservado `[...]` e coletadas para gerar o gráfico da Figura 8.

O **Teste de *Throughput* TCP**, representado na figura 8, obteve uma taxa de transferência média de 29.4 Mbps, mínimo de 12.9 Mbps, máximo de 41.8 Mbps, com 101 pacotes retransmitidos e transferindo 211 MB no total. Recebendo uma média de 29.2 Mbps no servidor `iperf3`, com um total de 210 MB.

Vale a pena ressaltar, que o **Teste de *Throughput* TCP** foi executado sem nenhum *tuning* no SO, para alterar o tamanho dos *buffers* padrão para transmissão e recepção proposto por (JACOBSON; BRADEN; BORMAN, 1992). Como mencionado anteriormente, foi utilizado um comando básico do `iperf3` para envio das requisições para o servidor.

Figura 8 – Teste de Throughput TCP (Mbps)



*Fonte: o próprio autor

Para realizar o segundo teste, **Teste de *Throughput* UDP**, foi executado o comando abaixo utilizando os seguintes parâmetros.

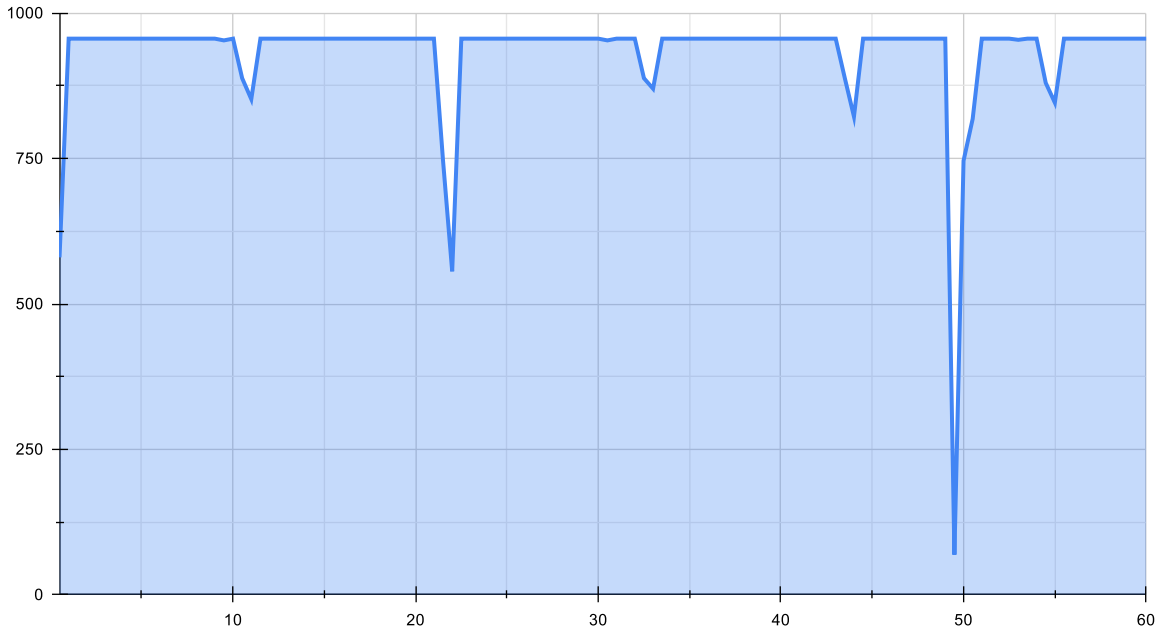
Teste de *Throughput* UDP

```
root@dtn1:~# iperf3 -c 7.7.7.1 -p 3000 -i 0.5 -t 60 -u -b 1G
Connecting to host 7.7.7.1, port 3000
[  5] local 6.6.6.1 port 58924 connected to 7.7.7.1 port 3000
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[...]
```

Diferente do teste com o protocolo TCP, foi utilizado o parâmetro `-u` (ou `--udp`) para especificar a utilização do protocolo UDP. Outro parâmetro utilizado foi o `-b` (ou `--bitrate`) para especificar a largura de banda máxima para 1 Gbps, ou seja, o máximo permitido pelas interfaces da topologia, como apresentado no capítulo 3 seção 3.1. O **Teste de *Throughput* UDP** também fornece informações sobre o *jitter* e a perda de pacotes. O *jitter* é a variação (desvio-padrão) dos tempos de chegada de pacotes, ou seja, essa métrica pode ser considerado como a variação da latência (TANENBAUM, 2003).

O retorno do comando executado no **Teste de *Throughput* UDP**, apresentou no envio: uma taxa de transferência média de 926 Mbps, mínima de 68 Mbps, máxima de 956

Figura 9 – Teste de *Throughput* UDP (Mbps)



*Fonte: o próprio autor

Mbps, com 6.47 GB de dados transmitidos, 0 ms de *jitter*, perda/total de datagramas de 0/4794258, o que corresponde a 0%. No recebimento do servidor: taxa média de transferência de 21 Mbps, e um total de 151 MB entregues, com 1.746 ms de *jitter*, perda/total de datagramas de 4685024/4794258, correspondendo a 98%. Na Figura 9, o gráfico apresenta uma queda brusca próximo a 50 segundos, chegando a 68 Mbps no envio. Outro ponto relevante é que apenas 2% dos dados foram entregues no servidor, demonstrando um bloqueio parcial.

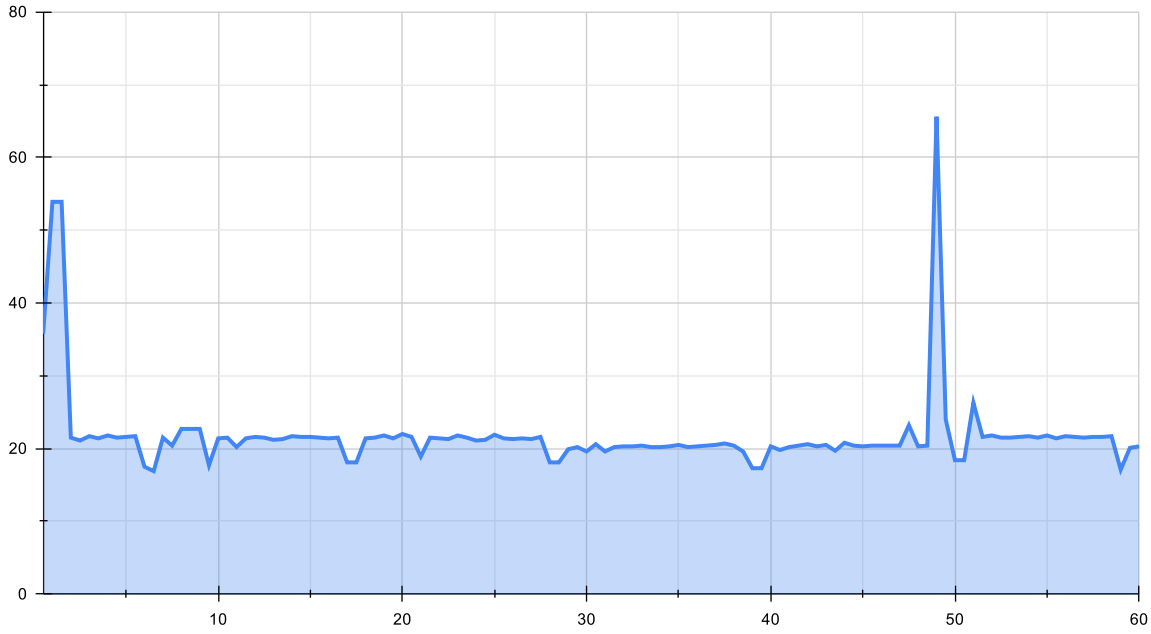
Para realizar o terceiro e último teste no modo do Plano de Controle, que é o **Teste de *Throughput* UDP reverso**, foi executado o mesmo comando do teste anterior, utilizando o parâmetro `-R` (ou `--reverse`), para verificar as informações recebidas pelo servidor `iper3`, ou seja, o servidor envia e o cliente recebe as requisições. Abaixo é possível conferir parte do retorno do deste comando.

Teste de *Throughput* UDP reverso

```
root@dtn1:~# iperf3 -c 7.7.7.1 -p 3000 -i 0.5 -t 60 -u -b 1G -R
Connecting to host 7.7.7.1, port 3000
Reverse mode, remote host 7.7.7.1 is sending
[  5] local 6.6.6.1 port 49678 connected to 7.7.7.1 port 3000
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total
↪  Datagrams
[...]
```

No retorno do comando executado, é possível verificar a inclusão da seguinte linha `Reverse mode, remote host 7.7.7.1 is sending` e a coluna `Jitter`. As requisições enviadas foram omitidas pelo espaço reservado `[...]` e coletadas para gerar o gráfico da figura 10.

Figura 10 – Teste de *Throughput* UDP reverso (Mbps)



*Fonte: o próprio autor

O **Teste de *Throughput* UDP reverso**, apresentou no recebimento do cliente: uma taxa de transferência média de 21.83 Mbps, mínima de 19.6 Mbps, máxima de 65.6 Mbps, com um total de 156 MB de dados transmitidos, 0.742 ms de *jitter*, perda/total de datagramas de 4574320/4683425, o que corresponde a 98% de perda. No envio do servidor: uma taxa de transferência média de 906 Mbps, 6.33 GB transmitidos, com 0 ms de *jitter*, perda/total de datagramas de 0/4692973, correspondendo a 0% de perda. Na Figura 10, o gráfico apresenta uma pico de transmissão próximo a 50

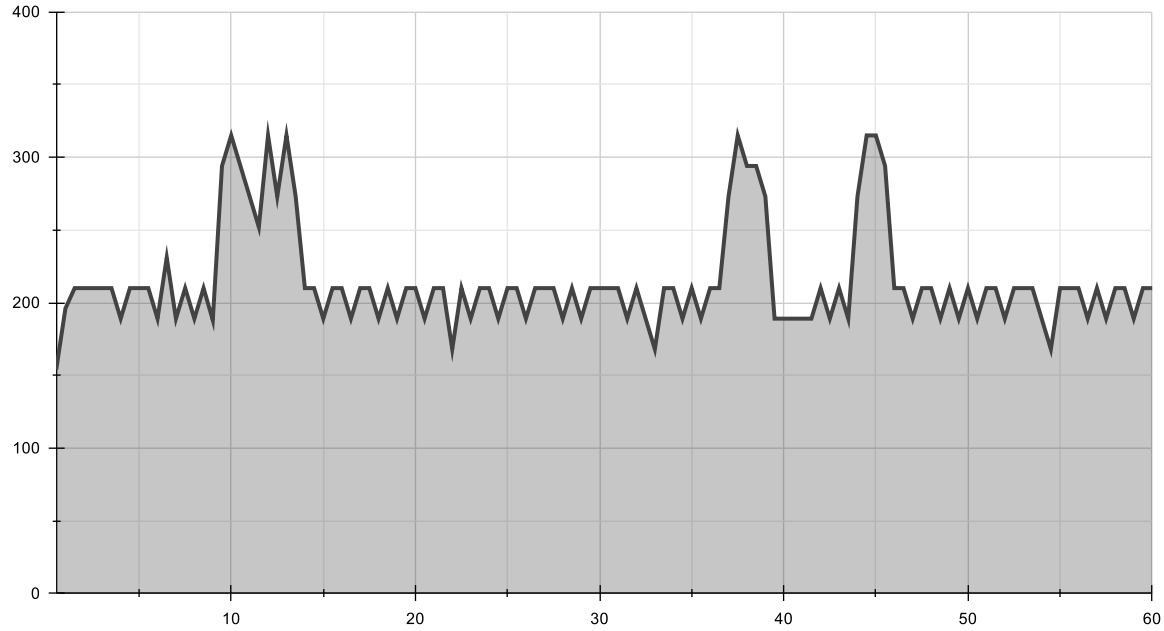
segundos, semelhante a queda brusca do gráfico da Figura 9 chegando a 65.6 Mbps no recebimento. Outro ponto importante é que apenas 2% dos dados foram recebidos no servidor, demonstrando um bloqueio maior do que no teste anterior.

Os três primeiros testes demonstraram que o modo de configuração do Plano Controle nas *whiteboxes* apresenta uma limitação significativa do *throughput* nos testes executados. Pois todas as interfaces dos equipamentos que compõem a topologia possuem a capacidade máxima de 1 Gbps, ou seja, os testes realizados não deviriam apresentar essa limitação. O que indica a necessidade de habilitar a configuração do Plano de Dados das *whiteboxes*.

4.1.3 Plano de Dados

Utilizando as *whiteboxes* configuradas no modo de Plano de Dados (*Data Plane*) emulado com `P4emu`, apresentado no capítulo 3 e subseção 3.2.2, foram executados os seguintes testes: ***Throughput* TCP**, ***Throughput* UDP** e ***Throughput* UDP reverso**. A seguir são apresentados os resultados destes testes.

Figura 11 – Teste de *Throughput* TCP (Mbps)



*Fonte: o próprio autor

Vale a pena ressaltar, que assim como na subseção 4.1.2 deste capítulo, todos os comandos executados utilizaram os mesmos parâmetros citados anteriormente, para

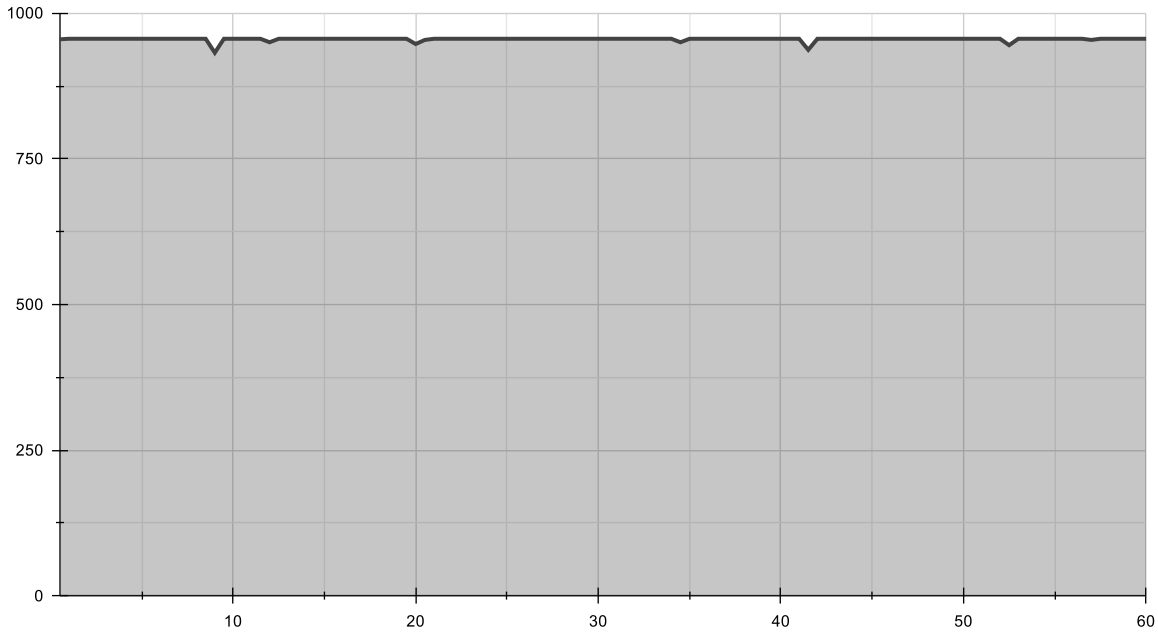
cada teste respectivamente. Com o objetivo de comparar os resultados dos dois modos de configuração das *whiteboxes*.

O **Teste de *Throughput* TCP**, representado no gráfico da Figura 11, obteve uma taxa de transferência média de 214 Mbps, mínima de 154 Mbps, máxima de 315 Mbps, com 81 pacotes sendo retransmitidos e um total de 1.5 GB transferidos. O mesmos resultados foram obtidos tanto para o recebimento quanto para o envio. Comparando com o resultado do mesmo teste da seção 4.1.2, podemos perceber um aumento de mais 7x da taxa de transferência média e máxima, mais de 10x maior do que a mínima, além de um menor número de pacotes retransmitidos e um aumento de mais de 6x do total de dados transferidos.

O **Teste de *Throughput* UDP**, apresentou no envio: uma taxa de transferência média e máxima de 956 Mbps, com mínima de 932 Mbps, transmitindo um total de 6.68 GB de dados, com 0 ms de *jitter*, perda/total de datagramas de 0/4950132, o que corresponde a 0%. No recebimento do servidor: taxa de transferência média foi de 482 Mbps, com 3.37 GB entregues, com 0.04 ms de *jitter*, perda/total de datagramas de 2451006/4950074, correspondendo a 50% de perda. Na **Figura 12**, o gráfico demonstra a taxa de transferência média e máxima de 956 Mbps. Comparando com o resultado do mesmo teste da seção 4.1.2, é possível analisar um aumento da média da taxa de transferência no valor obtido pela máxima do teste anterior, ou seja, 956 Mbps. Além de não apresentar uma queda brusca, este teste manteve uma constância nos dados enviados, como podemos observar pelo valor mínimo da taxa de transferência de 932 Mbps. Outro ponto importante é que 50% dos dados foram entregues no servidor, 25x mais do que o outro teste, entregando uma média de 482 Mbps de taxa de transferência, que corresponde a um aumento de mais de 22x. Por último, o *jitter* teve uma decréscimo de mais 43x, diminuindo de 1.746 ms para 0.04 ms.

O **Teste de *Throughput* UDP reverso**, apresentou no recebimento do cliente: uma média de 349 Mbps de taxa de transferência, mínima de 169 Mbps, máxima de 377 Mbps, com um total de 2.44 GB de dados transmitidos, 0.027 ms de *jitter*, perda/total de datagramas de 3096352/4902672, o que corresponde a 63%. No envio do servidor: uma taxa de transferência de 947 Mbps, 6.62 GB de dados transmitidos, com 0 ms

Figura 12 – Teste de *Throughput* UDP (Mbps)

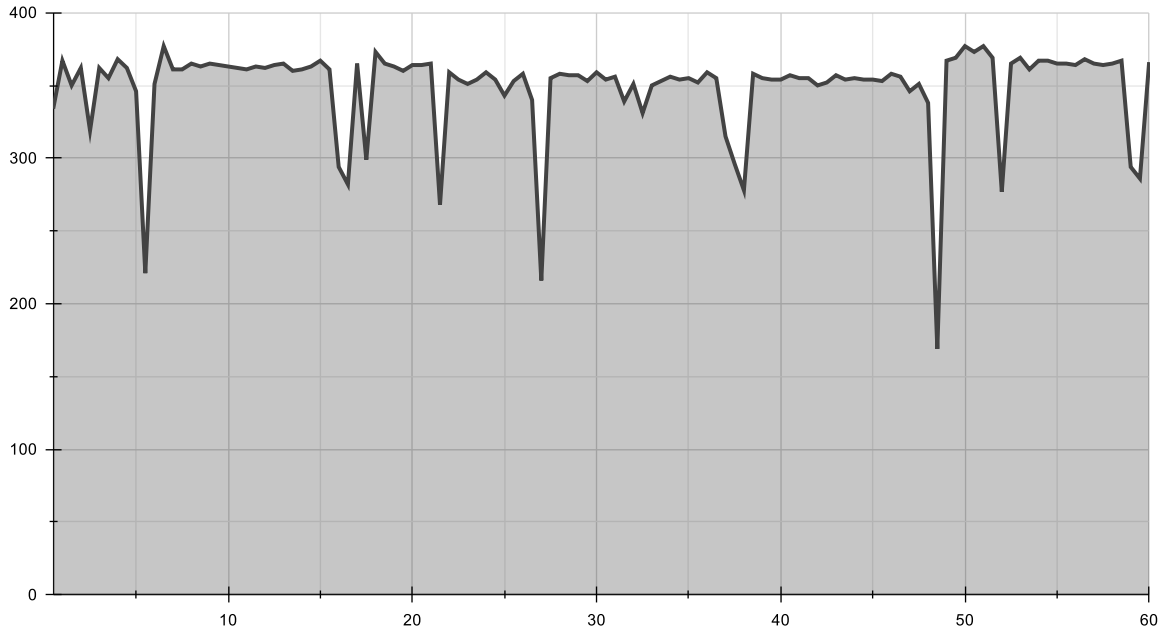


*Fonte: o próprio autor

de *jitter*, perda/total de datagramas de 0/4906124, correspondendo a 0%. Na **Figura 13**, o gráfico apresenta uma queda menos acentuada do que no gráfico da **Figura 9**, próximo a 50 segundos. Comparando com o resultado do mesmo teste da seção 4.1.2, podemos perceber um aumento na taxa de média e total de dados recebidos no cliente de quase 16x, com uma diminuição significativa do *jitter* de 0.742 ms para 0.027 ms. Além de diminuir o percentual de datagramas perdidos de 98% para 63%.

Os testes realizados nesta seção demonstraram resultados significativos da configuração do Plano de Dados emulado com `P4emu` nas *whiteboxes*. Foi possível observar um aumento no *throughput* de todos os testes executados, em comparação com o modo de configuração de Plano de Controle. Resultados como a taxa de transferência média de 214 Mbps, para o **Teste de *Throughput* TCP**, 956 Mbps de transferência média e máxima, para o **Teste de *Throughput* UDP**, e 349 Mbps de taxa de transferência para o **Teste de *Throughput* UDP reverso**. Comprovando a necessidade de habilitar a configuração do Plano de Dados para melhor aproveitamento e desempenho do *testbed*.

Figura 13 – Teste de *Throughput* UDP reverso (Mbps)



*Fonte: o próprio autor

5 CONSIDERAÇÕES FINAIS

Neste capítulo são apresentadas as considerações finais do presente trabalho e as recomendações para futuras pesquisas relacionadas ao tema.

5.1 ANÁLISE GERAL DO TRABALHO

Este trabalho demonstrou, através dos experimentos realizados, como prototipar e implementar uma rede experimental utilizando *whiteboxes* e o RARE/freeRtr. Seguindo a metodologia de prototipar antecipadamente a topologia do *testbed*, por meio da funcionalidade de emulação do RARE/freeRtr, antes de implementá-la fisicamente. Estes dois momentos de experimentação, possibilitaram uma etapa de planejamento e testes dos experimentos, antes da aquisição, despender tempo e esforço na implementação e configuração dos equipamentos, as *whiteboxes*. Com o objetivo de facilitar a reprodutibilidade deste trabalho, foram disponibilizadas as configurações do experimento emulado, em conjunto com as instruções para automatizar a topologia proposta.

Os três testes executados, ***Throughput* TCP**, ***Throughput* UDP** e ***Throughput* UDP reverso**, permitiram uma análise geral sobre o funcionamento das *whiteboxes* nos modos de configuração: Plano de Controle e Plano de Dados. Desta forma, foi possível perceber a limitação no desempenho do *testbed* utilizando estes equipamentos apenas no modo de configuração do Plano de Controle. Sendo necessário executar a configuração do modo de Plano de Dados, que neste trabalho foi utilizado o `P4Emu`, para atingir um desempenho aceitável em uma rede experimental. Vale a pena ressaltar, que este foi um dos maiores benefícios observados, pois a emulação do Plano de Dados utilizando `P4emu` possibilita esse tipo de configuração em qualquer SO que tenha acesso a biblioteca `libpcap` do Linux. O que diminui significativamente as limitações para instalação e utilização do RARE/freeRtr em diferentes tipos de hardware. Neste trabalho, foram utilizados equipamentos específicos de rede, os *Desktop Appliances* Lanner modelos Luna D-125A e NCA-1010, mas poderiam ser utilizados computadores pessoais para o mesmo objetivo. É importante mencionar que caso sejam utilizados

computadores pessoais, a maior limitação para compor um *testbed*, será a quantidade de interfaces de rede disponíveis.

Utilizando o modo de configuração do Plano de Dados emulado com `P4Emu`, o **Teste de *Throughput* TCP** alcançou uma taxa de transferência média de 214 Mbps, o **Teste de *Throughput* UDP** uma média e máxima de 956 Mbps e o **Teste de *Throughput* UDP reverso** uma média de 349 Mbps. É importante ressaltar que não foram usadas técnicas de *tuning* TCP e UDP, pois não fazia parte do escopo deste trabalho. Desta forma, os resultados alcançados comprovam a usabilidade e aplicabilidade do *testbed*, com um desempenho aceitável. Visto que o RARE/freeRtr possibilita a utilização de elementos clássicos de SDN e redes programáveis, além de apresentar uma abordagem modular e desagregada para estes elementos.

O RARE/freeRtr demonstrou ser uma ferramenta poderosa para construção de redes experimentais para fins acadêmicos e/ou de inovação. Recursos como portabilidade, conjunto de testes de interoperabilidade, arquitetura desagregada para o plano de controle e de dados, código aberto (*open source)*, APIs bem definidas para acesso ao plano de dados (DPDK, XDP, Barefoot Tofino e P4), dentre muitas outras funcionalidades. Todos esses recursos possibilitam que uma rede experimental se torne um *testbed* de classe de operadora (*carrier-class)*, aberto e portátil.

5.2 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Neste trabalho não foram explorados experimentos em outros sistemas operacionais, como o Fedora por exemplo, mencionado no capítulo 3 seção 3.2. Desta forma, um estudo comparando os desafios de cada tipo de instalação mostra-se um tópico relevante para obter novos resultados e popularizar o RARE/freeRtr.

Também não foi possível explorar a aceleração do processamento de pacotes no espaço do usuário, utilizando as APIs `P4DPDK`, `BfRuntime` e `P4XDP`, dada a limitação do hardware no caso do `P4DPDK` e `BfRuntime`, e pelo tempo disponível e escopo de pesquisa para o `P4XDP`. Como mencionado neste trabalho, o RARE/freeRtr possui uma *Common Message API* que interage com cada API do hardware alvo. Um trabalho comparando a instalação, configuração e desempenho de cada hardware, também

mostra-se um tópico relevante.

Outro tópico mencionado no trabalho e que merece um estudo detalhado, é a implementação de novos protocolos emergentes como o PolKA e M-PolKA. Estes protocolos podem ser explorados em diferentes topologias e/ou cenários.

REFERÊNCIAS

BALDIN, I. et al. Fabric: A national-scale programmable experimental network infrastructure. *IEEE Internet Computing*, IEEE, v. 23, n. 6, p. 38–47, 2019.

BARGUIL, S.; LOPEZ, V.; GIMENEZ, J. P. F.-P. Towards an open networking architecture. In: IEEE. *2020 International Conference on Optical Network Design and Modeling (ONDM)*. [S.l.], 2020. p. 1–3.

BERMAN, M. et al. Geni: A federated testbed for innovative network experiments. *Computer Networks*, Elsevier, v. 61, p. 5–23, 2014.

BORGES, E. S. et al. A lifecycle experience of polka: From prototyping to deployment at géant lab with rare/freertr. In: SBC. *Anais do XIII Workshop de Pesquisa Experimental da Internet do Futuro*. [S.l.], 2022. p. 35–40.

BORGES, E. S. et al. Freerouter in a nutshell: A “protocoland” routing platform for open and portable carrier-class testbeds. In: SBC. *Anais do I Workshop de Testbeds*. [S.l.], 2022. p. 36–46.

CASADO, M.; FOSTER, N.; GUHA, A. Abstractions for software-defined networks. *Communications of the ACM*, ACM New York, NY, USA, v. 57, n. 10, p. 86–95, 2014.

CISCO. *Virtual Route Forwarding Design Guide*. 2008. <https://www.cisco.com/c/en/us/td/docs/voice_ip_comm/cucme/vrf/design/guide/vrfDesignGuide.html>.

DOMINICINI, C. et al. Deploying polka source routing in p4 switches. In: IEEE. *2021 International Conference on Optical Network Design and Modeling (ONDM)*. [S.l.], 2021. p. 1–3.

DOMINICINI, C. et al. Polka: Polynomial key-based architecture for source routing in network fabrics. In: IEEE. *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. [S.l.], 2020. p. 326–334.

DUPUY, A. et al. Nest: A network simulation and prototyping testbed. *Communications of the ACM*, ACM New York, NY, USA, v. 33, n. 10, p. 63–74, 1990.

FARINA, F.; SZEGEDI, P.; SOBIESKI, J. Géant world testbed facility: federated and distributed testbeds as a service facility of géant. In: IEEE. *2014 26th International Teletraffic Congress (ITC)*. [S.l.], 2014. p. 1–6.

FARREL, A.; VASSEUR, J.-P.; ASH, J. *RFC 4655: A Path Computation Element (PCE)-Based Architecture*. 2006. <https://datatracker.ietf.org/doc/html/rfc3746>.

JACOBSON, V.; BRADEN, R.; BORMAN, D. *RFC1323: TCP extensions for high performance*. [S.l.]: RFC Editor, 1992.

KO, N.-S. et al. Design of a multipurpose whitebox networking platform. *International Journal of Future Computer and Communication*, IACSIT Press, v. 2, n. 4, p. 313, 2013.

KUROSE, J. F.; ROSS, K. W. *Computer Networking: A Top-Down Approach*. 8. ed. [S.l.]: Pearson Education, Inc., 2020. 764 p. ISBN 0-13-359414-9.

LINDVALL, M. et al. Experimenting with software testbeds for evaluating new technologies. *Empirical Software Engineering*, Springer, v. 12, n. 4, p. 417–444, 2007.

LOUI, F. et al. Project overview: Router for academia research & education (rare). Universität Tübingen, 2022.

MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, ACM New York, NY, USA, v. 38, n. 2, p. 69–74, 2008.

NADEAU, T. D.; GRAY, K. *SDN: Software Defined Networks: an authoritative review of network programmability technologies*. [S.l.]: O'Reilly Media, Inc., 2013.

PalC Networks. *Open Networking*. 2022. <https://palcnetworks.com/opennetworking.html>.

RNP. *Serviço de Testbeds da RNP*. 2022. Disponível em: <https://www.rnp.br/servicos/testbeds>. Acesso em: 30 de nov. de 2022.

SALLENT, S. et al. Fibre project: Brazil and europe unite forces and testbeds for the internet of the future. In: SPRINGER. *International Conference on Testbeds and Research Infrastructures*. [S.l.], 2012. p. 372–372.

SHERWOOD, R. et al. Can the production network be the testbed? In: *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*. [S.l.: s.n.], 2010.

TANENBAUM, A. S. Redes de computadores. ed. *Campus-Tradução da Terceira Edição, Rio de Janeiro*, 2003.

WANG, Y.; MATTA, I. Sdn management layer: Design requirements and future direction. In: IEEE. *2014 IEEE 22nd International Conference on Network Protocols*. [S.l.], 2014. p. 555–562.

YANG, L. et al. *rfc3746: Forwarding and control element separation (forces) framework*. [S.l.]: RFC Editor, 2004.

YU, J. et al. Cosmos: Optical architecture and prototyping. In: IEEE. *2019 Optical Fiber Communications Conference and Exhibition (OFC)*. [S.l.], 2019. p. 1–3.

APÊNDICE A – ARQUIVOS DE HARDWARE (EXPERIMENTO EMULADO)

r1-hw.txt

```
hwid r1
int eth1 eth 0000.1111.0001 127.0.0.1 10016 127.0.0.1 10061
int eth2 eth 0000.1111.0002 127.0.0.1 10013 127.0.0.1 10031
int eth3 eth 0000.1111.0003 127.0.0.1 10012 127.0.0.1 10021
tcp2vrf 2121 v1 23
```

r2-hw.txt

```
hwid r2
int eth3 eth 0000.2222.0003 127.0.0.1 10021 127.0.0.1 10012
int eth4 eth 0000.2222.0004 127.0.0.1 10024 127.0.0.1 10042
tcp2vrf 2222 v1 23
```

r3-hw.txt

```
hwid r3
int eth2 eth 0000.3333.0002 127.0.0.1 10031 127.0.0.1 10013
int eth3 eth 0000.3333.0003 127.0.0.1 10035 127.0.0.1 10053
tcp2vrf 2323 v1 23
```

r4-hw.txt

```
hwid r4
int eth2 eth 0000.4444.0002 127.0.0.1 10045 127.0.0.1 10054
int eth4 eth 0000.4444.0004 127.0.0.1 10042 127.0.0.1 10024
tcp2vrf 2424 v1 23
```

r5-hw.txt

```
hwid r5
int eth1 eth 0000.5555.0001 127.0.0.1 10057 127.0.0.1 10075
int eth2 eth 0000.5555.0002 127.0.0.1 10054 127.0.0.1 10045
int eth3 eth 0000.5555.0003 127.0.0.1 10053 127.0.0.1 10035
tcp2vrf 2525 v1 23
```

dtn1-hw.txt

```
hwid dtn1
int eth1 eth 0000.6666.0001 127.0.0.1 10061 127.0.0.1 10016
tcp2vrf 2626 v1 23
```

dtn2-hw.txt

```
hwid dtn2
int eth1 eth 0000.7777.0001 127.0.0.1 10075 127.0.0.1 10057
tcp2vrf 2727 v1 23
```

APÊNDICE B – ARQUIVOS DE SOFTWARE (EXPERIMENTO EMULADO)

r1-sw.txt

```
hostname r1
no buggy
!
vrf definition v1
 exit
!
router ospf4 1
 vrf v1
 router-id 10.1.1.1
 area 0 enable
 redistribute connected
 exit
!
router ospf6 1
 vrf v1
 router-id 10.6.1.1
 area 0 enable
 redistribute connected
 exit
!
interface template1
 description template interface r1
 lldp enable
 vrf forwarding v1
 ipv4 address dynamic dynamic
 ipv6 address dynamic dynamic
 router ospf4 1 enable
 router ospf4 1 area 0
 router ospf6 1 enable
 router ospf6 1 area 0
 shutdown
 no log-link-change
 exit
!
interface loopback0
 description lo r1
 vrf forwarding v1
 ipv4 address 20.20.20.1 /32
 ipv6 address 2020::1 /128
 template template1
 no shutdown
 exit
!
interface ethernet1
 description r1 -> dtn1
 vrf forwarding v1
 ipv4 address 6.6.6.2 /30
 ipv6 address 6666::2 /64
 template template1
 no shutdown
 exit
!
interface ethernet2
 description r1 -> r3
 vrf forwarding v1
 ipv4 address 3.3.3.2 /30
 ipv6 address 3333::2 /64
 template template1
 no shutdown
 exit
!
interface ethernet3
 description r1 -> r2
 vrf forwarding v1
 ipv4 address 1.1.1.1 /30
 ipv6 address 1111::1 /64
 template template1
 no shutdown
 exit
!
server telnet tel
  security protocol telnet
  exec timeout 10000000
  exec logging
  exec colorize prompt
  no exec authorization
  no login authentication
  login logging
  vrf v1
  exit
!
end
```

r2-sw.txt

```
hostname r2
no buggy
!
vrf definition v1
 exit
!
router ospf4 1
 vrf v1
 router-id 10.2.2.2
 area 0 enable
 redistribute connected
 exit
!
router ospf6 1
 vrf v1
 router-id 10.6.2.2
 area 0 enable
 redistribute connected
 exit
!
interface template1
 description template interface r2
 lldp enable
 vrf forwarding v1
 ipv4 address dynamic dynamic
 ipv6 address dynamic dynamic
 router ospf4 1 enable
 router ospf6 1 enable
 shutdown
 no log-link-change
 exit
!
interface loopback0
 description lo r2
 vrf forwarding v1
 ipv4 address 20.20.20.2 /32
 ipv6 address 2020::2 /128
 template template1
 no shutdown
 exit
!
interface ethernet3
 description r2 -> r1
 vrf forwarding v1
 ipv4 address 1.1.1.2 /30
 ipv6 address 1111::2 /64
 template template1
 no shutdown
 exit
!
interface ethernet4
 description r2 -> r4
 vrf forwarding v1
 ipv4 address 2.2.2.1 /30
 ipv6 address 2222::1 /64
 template template1
 no shutdown
 exit
!
server telnet tel
 security protocol telnet
 exec timeout 10000000
 exec logging
 exec colorize prompt
 no exec authorization
 no login authentication
 login logging
 vrf v1
 exit
!
end
```

r3-sw.txt

```
hostname r3
no buggy
!
vrf definition v1
 exit
!
router ospf4 1
 vrf v1
 router-id 10.3.3.3
 area 0 enable
 redistribute connected
 exit
!
router ospf6 1
 vrf v1
 router-id 10.6.3.3
 area 0 enable
 redistribute connected
 exit
!
interface template1
 description template interface r3
 lldp enable
 vrf forwarding v1
 ipv4 address dynamic dynamic
 ipv6 address dynamic dynamic
 polka enable 3 65536 10
 router ospf4 1 enable
 router ospf4 1 area 0
 router ospf6 1 enable
 router ospf6 1 area 0
 shutdown
 no log-link-change
 exit
!
interface loopback0
 description lo r3
 vrf forwarding v1
 ipv4 address 20.20.20.3 /32
 ipv6 address 2020::3 /128
 template template1
 no shutdown
 exit
!
interface ethernet2
 description r3 -> r1
 vrf forwarding v1
 ipv4 address 3.3.3.1 /30
 ipv6 address 3333::1 /64
 template template1
 no shutdown
 exit
!
interface ethernet3
 description r3 -> r5
 vrf forwarding v1
 ipv4 address 5.5.5.2 /30
 ipv6 address 5555::2 /64
 template template1
 no shutdown
 exit
!
server telnet tel
 security protocol telnet
 exec timeout 10000000
 exec logging
 exec colorize prompt
 no exec authorization
 no login authentication
 login logging
 vrf v1
 exit
!
end
```

r4-sw.txt

```
hostname r4
no buggy
!
vrf definition v1
 exit
!
router ospf4 1
 vrf v1
 router-id 10.4.4.4
 area 0 enable
 redistribute connected
 exit
!
router ospf6 1
 vrf v1
 router-id 10.6.4.4
 area 0 enable
 redistribute connected
 exit
!
interface template1
 description template interface r4
 lldp enable
 vrf forwarding v1
 ipv4 address dynamic dynamic
 ipv6 address dynamic dynamic
 router ospf4 1 enable
 router ospf4 1 area 0
 router ospf6 1 enable
 router ospf6 1 area 0
 shutdown
 no log-link-change
 exit
!
interface loopback0
 description lo r4
 vrf forwarding v1
 ipv4 address 20.20.20.4 /32
 ipv6 address 2020::4 /128
 template template1
 no shutdown
 exit
!
interface ethernet2
 description r4 -> r5
 vrf forwarding v1
 ipv4 address 4.4.4.1 /30
 ipv6 address 4444::1 /64
 template template1
 no shutdown
 exit
!
interface ethernet4
 description r4 -> r2
 vrf forwarding v1
 ipv4 address 2.2.2.2 /30
 ipv6 address 2222::2 /64
 template template1
 no shutdown
 exit
!
server telnet tel
 security protocol telnet
 exec timeout 10000000
 exec logging
 exec colorize prompt
 no exec authorization
 no login authentication
 login logging
 vrf v1
 exit
!
end
```

r5-sw.txt

```
hostname r5
no buggy
!
vrf definition v1
 exit
!
router ospf4 1
 vrf v1
 router-id 10.5.5.5
 area 0 enable
 redistribute connected
 exit
!
router ospf6 1
 vrf v1
 router-id 10.6.5.5
 area 0 enable
 redistribute connected
 exit
!
interface template1
 description template interface r5
 lldp enable
 vrf forwarding v1
 ipv4 address dynamic dynamic
 ipv6 address dynamic dynamic
 router ospf4 1 enable
 router ospf4 1 area 0
 router ospf6 1 enable
 router ospf6 1 area 0
 shutdown
 no log-link-change
 exit
!
interface loopback0
 description lo r5
 vrf forwarding v1
 ipv4 address 20.20.20.5 /32
 ipv6 address 2020::5 /128
 template template1
 no shutdown
 exit
!
interface ethernet1
 description r5 -> dtn2
 vrf forwarding v1
 ipv4 address 7.7.7.2 /30
 ipv6 address 7777::2 /64
 template template1
 no shutdown
 exit
!
interface ethernet2
 description r5 -> r4
 vrf forwarding v1
 ipv4 address 4.4.4.2 /30
 ipv6 address 4444::2 /64
 template template1
 no shutdown
 exit
!
interface ethernet3
 description r5 -> r3
 vrf forwarding v1
 ipv4 address 5.5.5.1 /30
 ipv6 address 5555::1 /64
 template template1
 no shutdown
 exit
!
server telnet tel
 security protocol telnet
 exec timeout 10000000
 exec logging
 exec colorize prompt
 no exec authorization
 no login authentication
 login logging
 vrf v1
 exit
!
end
```

dtn1-sw.txt

```
hostname dtn1
no buggy
!
vrf definition v1
 exit
!
interface template1
 description template interface dtn1
 lldp enable
 vrf forwarding v1
 ipv4 address dynamic dynamic
 ipv6 address dynamic dynamic
 shutdown
 no log-link-change
 exit
!
interface loopback0
 description lo dtn1
 vrf forwarding v1
 ipv4 address 20.20.20.6 /32
 ipv6 address 2020::6 /128
 template template1
 no shutdown
 exit
!
interface ethernet1
 description dtn1 -> r1
 vrf forwarding v1
 ipv4 address 6.6.6.1 /30
 ipv6 address 6666::1 /64
 template template1
 no shutdown
 no log-link-change
 exit
!
ipv4 route v1 0.0.0.0 0.0.0.0 6.6.6.2
!
ipv6 route v1 :: :: 6666::2
!
server telnet tel
 security protocol telnet
 exec timeout 10000000
 exec colorize prompt
 exec logging
 no exec authorization
 no login authentication
 login logging
 vrf v1
 exit
!
end
```

dtn1-sw.txt

```
hostname dtn2
no buggy
!
vrf definition v1
 exit
!
interface template1
 description template interface dtn2
 lldp enable
 vrf forwarding v1
 ipv4 address dynamic dynamic
 ipv6 address dynamic dynamic
 shutdown
 no log-link-change
 exit
!
interface loopback0
 no description
 vrf forwarding v1
 ipv4 address 20.20.20.7 /32
 ipv6 address 2020::7 /128
 template template1
 no shutdown
 exit
!
interface ethernet1
 description dtn2 -> r5
 vrf forwarding v1
 ipv4 address 7.7.7.1 /30
 ipv6 address 7777::1 /64
 template template1
 no shutdown
 no log-link-change
 exit
!
ipv4 route v1 0.0.0.0 0.0.0.0 7.7.7.2
!
ipv6 route v1 :: :: 7777::2
!
server telnet tel
 security protocol telnet
 exec timeout 10000000
 exec colorize prompt
 exec logging
 no exec authorization
 no login authentication
 login logging
 vrf v1
 exit
!
end
```

APÊNDICE C – SCRIPTS BASH PARA MULTIPLEXAÇÃO DO TERMINAL

start.sh

```
#!/bin/bash

RTR=/home/$USER/rtr.jar
HWSW=$PWD/

tmux new-session -d -s tcc2022 'java -jar '$RTR' routersc '$HWSW'r1-hw.txt
↪  '$HWSW'r1-sw.txt'
tmux split-window -v -t 0 -p 50
tmux send 'java -jar '$RTR' routersc '$HWSW'dtn1-hw.txt
↪  '$HWSW'dtn1-sw.txt' ENTER;
tmux split-window -h -t 0 -p 50
tmux send 'java -jar '$RTR' routersc '$HWSW'r5-hw.txt '$HWSW'r5-sw.txt'
↪  ENTER;
tmux split-window -h -t 2 -p 50
tmux send 'java -jar '$RTR' routersc '$HWSW'dtn2-hw.txt
↪  '$HWSW'dtn2-sw.txt' ENTER;
tmux split-window -v -t 0 -p 50
tmux send 'java -jar '$RTR' routersc '$HWSW'r3-hw.txt '$HWSW'r3-sw.txt'
↪  ENTER;
tmux split-window -v -t 0 -p 50
tmux send 'java -jar '$RTR' routersc '$HWSW'r2-hw.txt '$HWSW'r2-sw.txt'
↪  ENTER;
tmux split-window -v -t 2 -p 50
tmux send 'java -jar '$RTR' routersc '$HWSW'r4-hw.txt '$HWSW'r4-sw.txt'
↪  ENTER;
tmux select-layout tiled
tmux a;
```

stop.sh

```
#!/bin/bash

tmux kill-window -t tcc2022
```