Pós-Graduação em Ciência da Computação

José Carlos Ferreira de Melo Júnior

Gerenciamento Adaptativo de Redes Baseado em Contexto e SDN

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2020

José Carlos Ferreira de Melo Júnior

Gerenciamento Adaptativo de Redes Baseado em Contexto e SDN

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Redes de Computadores, Redes Definidas por Software
Orientador: Carlos André Guimarães Ferraz

Recife
2020

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

M528g Melo Júnior, José Carlos Ferreira de
Gerenciamento adaptativo de redes baseado em contexto e SDN / José Carlos Ferreira de Melo Júnior. – 2020.
117 f.: il., fig., tab.

Orientador: Carlos André Guimarães Ferraz.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2020.
Inclui referências e apêndice.

1. Redes de computadores. 2. Gerência de redes. I. Ferraz, Carlos André Guimarães (orientador). II. Título.

004.6 CDD (23. ed.) UFPE - CCEN 2020 – 103

José Carlos Ferreira de Melo Júnior

"Gerenciamento Adaptativo de Redes baseado em Contexto e SDN"

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação

Aprovado em: 17/02/2020.

BANCA EXAMINADORA

Prof. Dr. Daniel Carvalho da Cunha
Centro de Informática/ UFPE

Profa. Dra. Zuleika Tenório Cavalcanti do Nascimento
Gerência de Informação Tecnológica /ITEP

Prof. Dr. Carlos André Guimarães Ferraz
Centro de Informática / UFPE
(Orientador)

Para você Laura. Minha amada imortal.

AGRADECIMENTOS

Agradeço a minha família e todos que mantiveram o pensamento positivo e que, de certa forma, ajudaram na concretização deste trabalho. Ao meu orientador Carlos Ferraz e ao companheiro David Cunha pela grande ajuda durante este percurso. Gostaria de agradecer também a todas as pessoas que conheci nessa jornada e em especial a minha adorável esposa, Laura Grace. Por estar sempre ao meu lado, por sempre segurar minha mão quando estive por cair, sempre lutar por mim quando estive por desistir. Eu te amo e vou te amar sempre e para sempre. Obrigado a todos por tudo. Obrigado por acreditarem.

RESUMO

Com redes de computadores em todos os lugares, seu gerenciamento está se tornando cada vez mais complexo, levando a custos mais altos, tempos de resposta mais longos e mais erros humanos na tomada de decisões. Com isso, a busca pelo Smart Management, com menos intervenção humana e mais automação, torna-se bastante importante. Várias tecnologias foram propostas para permitir este "Gerenciamento Inteligente", desde novos protocolos de rede a técnicas de aprendizado de máquina, principalmente para casos mais complexos. O paradigma de Redes Definidas por Software (Software Defined Networking - SDN) é um dos mais promissores. A literatura mostra que ainda existem vários desafios e oportunidades de pesquisa para a automação de muitas tarefas de gerenciamento de rede. Este trabalho propõe adicionar sensibilidade a contexto ao paradigma SDN em uma abordagem leve para o Gerenciamento Inteligente, ou Gerenciamento Adaptativo, para lidar de maneira simplificada com os eventos do dia a dia, como queda de enlace e congestionamento de tráfego. Esta trabalho acadêmico apresenta 3 situações de casos de uso emulados e implementados usando a arquitetura SDN e os resultados mostram que a adaptação ao contexto é rápida alcançando um total de 17,16 ms utilizando o protocolo TCP e 17,744 ms utilizando o protocolo UDP. As ações estabelecidas foram realizadas sem nenhuma intervenção humana.

Palavras-chaves: Gerência de Redes. SDN. Sensibilidade a Contexto.

ABSTRACT

With computer networks everywhere, their management is becoming increasingly complex, leading to higher costs, longer response times and more human error in decision making. With this, the search for Smart Management, with less human intervention and more automation, becomes very important. Several technologies have been proposed to enable this " Intelligent Management ", from new network protocols to machine learning techniques, especially for more complex cases. The Software Defined Networking (SDN) paradigm is one of the most promising. The literature shows that there are still many research challenges and opportunities for automating many network management tasks. This work proposes to add context sensitivity to the SDN paradigm in a light approach to Intelligent Management, or Adaptive Management, to deal in a simplified way with day-to-day events, such as link loss and traffic congestion. This academic paper presents 3 use case situations emulated and implemented using the SDN architecture and the results show that the adaptation to the context is fast reaching a total of 17.16 ms using the TCP protocol and 17.744 ms using the UDP protocol. The established actions were carried out without any human intervention.

Keywords: Network Management. SDN. Context-Awareness.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE SIGLAS

API	APPLICATION PROGRAMMING INTERFACE
CLI	COMMAND LINE INTERFACE
CMIP	COMMON MANAGEMENT INFORMATION PROTOCOL
CPOR	CENTRO DE PREPARAÇÃO DE OFICIAIS DA RESERVA
FUNDAJ/CGF	FUNDAÇÃO JOAQUIM NABUCO - CAMPUS GILBERTO FREYRE
ICONE	INFRAESTRUTURA DE COMUNICAÇÃO ÓTICA PARA ENSINO E PESQUISA
IDS	INTRUSION DETECTION SYSTEM
IFPE	INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE PERNAMBUCO
IOT	INTERNET OF THINGS
MIB	MANAGEMENT INFORMATION BASE
ML	MACHINE LEARNING
NBI	NORTHBOUND INTERFACE
NFV	NETWORK FUNCTION VIRTUALIZATION
NMS	NETWORK MANAGEMENT SYSTEM
NOS	NETWORK OPERATING SYSTEM
OID	OBJECT IDENTIFIERS
ONF	OPEN NETWORK FOUNDATION
OVS	OPEN V SWITCH
POP-PE	PONTO E PRESENÇA EM PERNAMBUCO
RMON	REMOTE MONITORING
RNP	REDE NACIONAL DE ENSINO E PESQUISA
RPC	REMOTE PROCEDURE CALL
RT	RESPONSE TIME

SBI	SOUTHBOUND INTERFACE
SDN	SOFTWARE DEFINED NETWORKING
SNMP	SIMPLE NETWORK MANAGEMENT PROTOCOL
SSL	SECURE SOCKET LAYER
SVG	SCALABLE VECTOR GRAPHICS
XML	EXTENSIBLE MARKUP LANGUAGE
XMPP	EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL

SUMÁRIO

1 INTRODUÇÃO

Nas últimas décadas, as redes de computadores vêm se tornando progressivamente onipresentes, ou ubíquas, sendo utilizadas de forma cada vez mais intensa e diversificada. Serviços avançados de rede são essenciais para viabilizar os mais variados e dominantes tópicos da atualidade, como a computação em nuvem, *internet* móvel (5G), a *internet* das coisas (IoT), o processamento de fluxos (*streams*) complexos, serviços multimídias de alta definição – particularmente, serviços de *video streaming*, entre muitos outros. Neste sentido, os requisitos de gerência de redes são cada vez mais altos e erros humanos se tornam mais frequentes na tomada de decisão.

O gerenciamento de rede, na maioria dos casos, ainda é feito de forma tradicional utilizando o protocolo simples de gerenciamento de rede (Simple Network Management Protocol, SNMP), o que dificulta a implementação de novos paradigmas de infraestruturas e novos dispositivos. A utilização do protocolo SNMP é apenas utilizado no auxílio da gerência de redes. Sua arquitetura simples não possibilita que seja enviado comandos de modificações de configuração, não permitindo que seja feita uma interação maior com os dispositivos de rede. Redes de computadores são estruturas complexas e dinâmicas que estão em constantes mudanças de infraestruturas. Essas estruturas implicam em vários tipos de dispositivos, como *switches*, roteadores, *firewalls* etc. Novas tecnologias estão constantemente se conectando à rede. Impulsionados pela tecnologia de internet das coisas, computação em nuvem e o aquecimento do mercado 5G da tecnologia móvel, o volume de tráfego e o número de dispositivos e *data centers* podem dificultar a administração e gerência dessas tecnologias. De acordo com gerentes de TI, a automação de redes (25%) e a arquitetura de redes definidas por *software* (Software Defined Networking, SDN) (23%) estão entre as tecnologias que terão maior impacto na rede nos próximos cinco anos (CISCO, 2020). De acordo com algumas pesquisas, aproximadamente 70% das atividades de rede de um *data center* são feitas manualmente, o que aumenta o tempo, o custo, a probabilidade de erros e reduz a flexibilidade (FOREST; RICKARD, 2019). Essas e outras razões exigem menos intervenção humana e mais automação. Diversas tecnologias têm surgido para possibilitar o gerenciamento inteligente de redes (*"Smart Management"*), como por exemplo, a arquitetura SDN (KIM; FEAMSTER, 2013).

A arquitetura SDN e tecnologias como Virtualização de Função de Redes (*Network Function Virtualization*, NFV) estão introduzindo o próximo modelo para o gerenciamento de redes. A arquitetura SDN propõe a separação do plano de controle do plano de dados (encaminhamento). Esta separação permite que um único *software* tenha uma visão geral da rede, permitindo também o seu controle. Este tipo de separação permite a programabilidade direta do plano de controle. Com o paradigma SDN, a adaptação dinâmica da rede, de acordo com a demanda atual e os requisitos de serviço, pode ser automatizada

com orquestração.

Algumas tecnologias como Aprendizagem de Máquina (*Machine Learning*, ML) e Sensibilidade a Contexto (*Context-Awareness*) podem ser usadas para automatização da operação e do gerenciamento de redes. Devido à programabilidade oferecida pela arquitetura SDN, ML possibilita o auto-gerenciamento ou o gerenciamento autônomo de redes, porém pode aumentar a complexidade computacional envolvida em treinar modelos de algoritmos. Com a programabilidade disponível na camada de controle, fica possível implementar políticas e regras simples de sensibilidade a contexto que possam ajudar a tornar a rede mais adaptativa (AYOUBI et al., 2018), (BUI et al., 2017).

No entanto, de acordo com (AYOUBI et al., 2018), a automação de muitas tarefas de gerenciamento de redes ainda não foi suficientemente explorada, e, portanto, há diversas oportunidades e desafios de pesquisa em aberto. Para Rojas (2018) o paradigma da arquitetura SDN ainda está no início do seu desenvolvimento e, considerando a automação total e o gerenciamento sem esforço (*effortless*) como os principais objetivos destas redes, diversos desafios precisam ser enfrentados. Portanto, ainda é longo o caminho para realizar uma visão de gerenciamento de rede totalmente autônomo (KHAN et al., 2018).

1.1 MOTIVAÇÃO

Há muitos anos, o uso do protocolo SNMP tem sido usado para gerenciamento de dispositivos. Embora o SNMP tenha tido êxito em atividades de gerenciamento de dispositivos de rede, ele falha em atividades de configuração e alterações da rede. Com o passar do tempo, fica claro que atividades envolvendo SNMP se tornaram insustentáveis para gerenciar de forma adequada a enorme quantidade de dispositivos e tecnologias emergentes. A arquitetura SDN oferece um grande passo para o uso definitivo da automação de rede, permitindo que as equipes de rede gerenciem com mais eficiência e flexibilidade, separando o plano de controle e o plano de encaminhamento. Dessa maneira, o plano de controle se torna diretamente programável tornando a inteligência de rede logicamente centralizada. Hoje, é amplamente reconhecido que a automação de redes é crucial para a eficácia e o sucesso das futuras gerações da operação de redes. A automação pode melhorar a gerência de redes permitindo a adaptabilidade, a disponibilidade, a confiabilidade da operação de redes, poupando os administradores e analistas de rede de atividades diárias e demoradas. Com isso, permite que os engenheiros e administradores de redes concentrem suas energias em outros níveis de serviços de redes.

Alguns conceitos como sensibilidade a contexto ou aprendizagem de máquina (Machine Learning, ML), podem impulsionar a automação de redes, tornando-a mais adaptativa. No entanto, técnicas de ML podem implicar em um peso maior quanto à técnicas de complexidade computacional envolvida. O que não justificaria o uso de uma técnica tão complexa para resolver um problema simples de rede. Por outro lado, a sensibilidade a

contexto em uma rede de computadores pode adicionar a adaptabilidade necessária para o gerenciamento de dispositivos de rede para resolver problemas simples e corriqueiros de uma rede de computadores. A programabilidade oferecida pela arquitetura SDN permite a coleta de alguns dados diretamente do plano de encaminhamento e com isso, fica possível estabelecer regras e políticas simples para caracterizar uma situação contextual (ex. congestionamento de rede). Em geral, tais regras simples devem ser suficientes para lidar com problemas diários ou simples (queda de enlace ou congestionamento), para que então técnicas mais complexas, como ML, possam ser reservadas para um uso mais adequado como classificação de pacotes e melhorar a detecção de ataques de rede.

A Rede Nacional de Ensino e Pesquisa (RNP) possui algumas pesquisas no campo de SDN envolvendo também sua própria infraestrutura (RNP, 2017), (RNP, 2016), (RNP, 2015). A RNP possui vários projetos de infraestruturas de rede para atender a instituições de ensino, pesquisa e saúde. Uma delas, a Redecomep (Redes Comunitárias de Ensino e Pesquisa), atende a instituições nas áreas metropolitanas onde se localizam os pontos de presença da RNP. Em Recife, a Rede ICONE (Infraestrutura de Comunicação Óptica para eNsino e pEsquisa) faz parte do programa Redecomep da RNP atendendo a várias instituições de ensino, pesquisa e saúde em um anel de fibra de aproximadamente 87Km. A Rede ICONE foi escolhida como ponto de partida para esta pesquisa de uma futura implementação da arquitetura SDN.

1.2 OBJETIVOS

O objetivo geral é propor e elaborar uma solução que associe uma abordagem leve utilizando conceitos de sensibilidade a contexto ao paradigma de redes definidas por *software*, visando realizar o gerenciamento adaptativo para lidar com eventos simples de rede. A abordagem leve se refere ao poder computacional envolvido, onde o uso de ML poderia envolver intensas coletas de dados e processo de treinamento de algoritmos.

Para os objetivos específicos temos:

- Propor situações contextuais que se assemelhem às situações cotidianas da Rede ICONE.
- Elaborar um ambiente virtual de experimentação utilizando a arquitetura SDN que reflita as condições da Rede ICONE.
- Implementar uma aplicação sensível a contexto no controlador SDN, de forma a se adaptar às situações contextuais estabelecidas.
- Realizar uma análise estatística das etapas de captura de contexto, inferência de contexto e adaptação a contexto em uma porção emulada da Rede ICONE.

1.3 ESCOPO DA PESQUISA

Este trabalho se concentra na criação de uma abordagem leve (técnicas de sensibilidade a contexto) para realizar o gerenciamento adaptativo de rede. Tal abordagem faz uso do paradigma da arquitetura SDN com regras simples de sensibilidade a contexto. Essas regras de contexto foram determinadas baseadas no estudo de caso da rede metropolitana de fibra óptica em Recife (Rede ICONE), onde essas regras se refletem em problemas corriqueiros envolvendo esta rede. Esta pesquisa está delimitada em emular parte da Rede ICONE para testar e medir o comportamento da solução proposta. Para testar o comportamento do controlador SDN na rede emulada, será feito o uso de um gerador de tráfego, onde a rede será testada tanto com tráfego utilizando o protocolo TCP quanto UDP.

Para o escopo negativo, esta pesquisa foi baseada nos estudos de caso da Rede ICONE, que possui uma topologia em anel e, desta maneira, a abordagem sensível a contexto não foi preparada para reagir a situações que envolvam topologias diferentes desta.

1.4 METODOLOGIA ADOTADA

Por se tratar de uma rede em produção, foi necessário estabelecer um ambiente controlado para testes. Foi elaborado um ambiente virtual experimental para emular a rede. A Rede ICONE contém várias instituições conectadas em uma topologia em anel. Foram escolhidas 3 instituições pertencentes ao um mesmo par de fibra do anel da Rede ICONE. Para representar a saída de *internet*, foi estabelecido mais 1 ponto na rede emulada representando o PoP-PE. Sendo assim, a rede emulada possui 4 pontos em anel com todos os 3 pontos com saída direcionada ao PoP-PE/RNP. O emulador de rede SDN permite a customização da rede através de *scripts*. Com isso, foi elaborado um *script* da topologia com características semelhantes ao da Rede ICONE. A Rede ICONE é considerada uma rede estável, então apenas parâmetros como porta, atraso (*delay*), banda e perda de pacotes (*packet loss*) foram considerados.

Os sistemas de gerenciamentos estão centralizados no *data center* do PoP-PE/RNP e têm acesso a todas as instituições conectadas à Rede ICONE. Do mesmo modo, o ambiente virtual foi fundamentado em máquinas virtuais para reproduzir condições de interação similares ao *data center* do PoP-PE. Com um total de 3 máquinas virtuais, suas funções foram determinadas da seguinte forma: emulação, controlador SDN e monitoramento.

O controlador SDN foi elaborado de forma a reagir a eventos que podem ser gerados aleatoriamente na rede emulada. Os eventos que são capturados são inferidos através de regras de contextos bem definidas. Se a inferência for atendida, a etapa de adaptação é acionada. A máquina virtual que foi utilizada para monitoramento segue o padrão adotado no *data center* do PoP-PE. Nesta máquina, foram criados dois contêineres: um para base de dados de séries temporais para armazenar métricas de consumo de banda e o outro um

sistema de monitoramento para exibir as métricas de consumos de banda. Como prova de conceito, foi necessário observar o comportamento do controlador SDN. Foi necessário criar uma topologia gráfica baseada na Rede ICONE. O tráfego foi gerado utilizando os protocolos TCP e UDP e os tempos de resposta das etapas de captura de contexto, inferência de contexto e adaptação de contexto, foram coletadas e analisadas estatisticamente. Em uma segunda análise, foi feita a comparação de um sistema de gerenciamento baseado em contexto (adaptativo) e um sistema de gerenciamento tradicional baseado em gerenciamento humano com tomada de decisão, considerando que, para parecer instantâneo, o tempo de resposta máximo em uma interface de usuário deve ser na ordem de 10 a 40 milissegundos (tempo limite de percepção humana) (NICOLAOU, 1990).

1.5 ORGANIZAÇÃO DO DOCUMENTO

Os Capítulos desta dissertação estão organizados da seguinte forma: O Capítulo 2 trata da fundamentação teórica onde é explicado um pouco sobre o projeto de Redes Comunitárias de Ensino e Pesquisa da RNP. Trabalha também os conceitos de temas que são relevantes a esta dissertação e, por fim, são apresentados os principais trabalhos que influenciaram no tema envolvido nesta pesquisa. No Capítulo 3, é apresentada uma abordagem leve para o gerenciamento adaptativo. São apresentadas as regras de contexto e o passos que foram seguidos para reproduzir as situações contextuais. O Capítulo 4 é reservado para detalhar toda a experimentação desta pesquisa, onde foi destinado às configurações das ferramentas e codificações desta pesquisa. Também são feitas a apresentação das métricas envolvidas e as análises estatísticas descritivas. Por fim, no Capítulo 5 é apresentada a conclusão com as principais contribuições deste trabalho. Também são apresentadas as dificuldades e limitações encontradas durante o percurso desta pesquisa e as expectativas para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo serão apresentados os temas e conceitos que envolvem esta pesquisa. Na Seção 2.1, é feita uma descrição sobre o projeto de Redes Comunitárias de Educação e Pesquisa ou Redecomep da RNP. A Seção 2.2 faz uma abordagem histórica do Gerenciamento de Rede descrevendo a forma tradicional e as novas tendências e protocolos para esta área. Em seguida, na Seção 2.3 é feita uma descrição sobre o paradigma de redes definidas por software e suas principais características e tecnologias. Na Seção 2.4 são expostos os conceitos da sensibilidade a contexto e por fim, é feito um estudo analítico e comparativo dos trabalhos relacionados com esta pesquisa.

2.1 REDECOMEP - RNP

Instituições de ensino e pesquisa vêm mostrando um aumento no uso de tecnologias de *internet* para colaboração e cooperação. A Redecomep foi um projeto determinado pelo Ministério da Ciência e Tecnologia (MCT), apoiado e executado pela RNP. Essa rede tem o objetivo de agregar instituições de pesquisa e educação superior nas 26 capitais que abrigam os pontos de presença (PoPs) da RNP. Essas redes conectam essas instituições a nível metropolitano por meio de uma infra-estrutura de fibra ótica e inclui materiais e equipamentos para construção da rede lógica (RNP, 2007).

O projeto inicial ou prova de conceito da Redecomep começou a ser operado em 2007 em Belém, conhecida como MetroBel, gerida pela Universidade Federal do Pará (UFPA). Com o investimento inicial perto de R$ 1 milhão, a princípio, foram conectadas 12 instituições de educação e pesquisa públicas e privadas. Após seu início, o projeto conta com várias cidades que foram contempladas com a Redecomep.

A sustentabilidade se mantém com a criação de consórcios entre as instituições, governos locais e federal através de sua concepção compartilhada na sociedade como empreendimento comum e não comercial (SILVA, 2016). Assim como a Redecomep MetroBel, existem no Brasil várias iniciativas do projeto Redecomep e atualmente o projeto não se restringe apenas às capitais. A Figura 1 ilustra a distribuição das Redecomeps nas principais cidades e capitais do país.

Figura 1 – Redes COMEP Brasil.

Fonte: (RNP, 2019)

2.2 GERENCIAMENTO DE REDES

Um dos aspectos mais importantes ao se incorporar uma nova tecnologia de rede é o seu gerenciamento. O gerenciamento de redes é o elemento chave para redes em crescimento, tanto em complexidade quanto em tamanho. A rede cresce à medida que o número de serviços e dispositivos que se conectam a ela também aumenta. Com este crescimento, a atividade e as responsabilidades do gerenciador de redes também se tornam complexas, prejudicando a identificação de problemas e pontos críticos na rede. O gerenciamento de rede tradicional consiste em alguns elementos que formam um modelo de gerenciamento de rede. Este modelo basicamente é composto pelo gerenciador, o sistema de gerencia-

mento, a base de informações e os protocolos de comunicação. Para que um sistema de gerenciamento funcione, é fundamental que os dispositivos troquem informações entre si e para isso, é necessário fazer uso de alguns protocolos de comunicação. Em redes IP, algumas abordagens de protocolos são utilizadas para a comunicação entre dispositivos (ELLIS; PURSELL; RAHMAN, 2003), (SANDUSKY, 2011).

O protocolo SNMP é o protocolo mais popular para o gerenciamento de ativos de rede e telecomunicação. Sua facilidade na implementação e capacidade de operar em redes de médio porte permitiu ser uma escolha rápida para o crescimento rápido da *internet*. Do mesmo modo que o lento progresso de outras tecnologias como o protocolo comum de informações de gerenciamento (*Common Management Information Protocol*, CMIP) e o monitoramento remoto de redes (*Remote Monitoring*, RMON) também ajudou a rápida popularização do SNMP. O SNMP trabalha com a coleta de base de informações de gerenciamento (*Management Information Base*, MIB). A MIB é um grupo de informações de objetos gerenciáveis organizados na forma de árvore hierárquica. A MIB é constituída por Identificadores de Objetos (*Object Identifiers*, OID) que são os objetos identificados em uma MIB. Apesar do gerenciamento de redes ter alcançado uma certa maturidade, ainda é uma área com um grande potencial a ser explorado por técnicas e tecnologias auto-adaptativas, de automação e inteligência artificial (NAKAMURA; KASHIMURA; MOTOMURA, 1995), (NARAYANAN; ILANGOVAN; NARAYANAN, 2013), (SANDUSKY, 2011), (SILVA; SOUZA, 2016).

Com o aumento dos dispositivos e serviços de redes, devido a propagação de tecnologias como Internet das Coisas (*Internet of Things*, IoT), Computação em Nuvem e o aumento no número de *data centers*, o gerenciamento se torna cada vez mais uma tarefa árdua para o administrador de redes. Novos avanços em tecnologias de redes implicam em novos desafios e paradigmas na área de gerenciamento. Questões como qualidade, performance, atrasos, taxa de perda de pacotes, impõem um comprometimento maior e uma sobrecarga de trabalho para administradores de redes. Nas últimas décadas novos termos como *self-adaptive*, *self-management*, *autonomous network* e *smart management* surgiram como novas técnicas para tratar da administração e gerenciamento de uma rede de dispositivos. Algumas pesquisas ratificam o objetivo geral de manter a disponibilidade e melhorar o desempenho da rede, perseguido desde o gerenciamento tradicional, com base em sistemas de gerenciamento de rede ou *Network Management System* (NMS), protocolo SNMP e a base de informações de gerenciamento (MIB). Tradicionalmente, o gerenciamento de rede requer ações fortemente dependentes dos administradores. Para mudar essa realidade, os esforços de pesquisa se concentram em arquiteturas de gerenciamento ágeis e adaptáveis que suportam redes de auto-gerenciamento ou redes inteligentes. Neste contexto o paradigma de Redes Definidas por Software pode ser considerado um candidato ideal para lidar com tais desafios (QUTTOUM et al., 2018), (DERBEL; AGOULMINE; SALAÜN, 2009), (TOY, 2014), (TUNCER et al., 2015).

2.3 REDES DEFINIDAS POR SOFTWARE

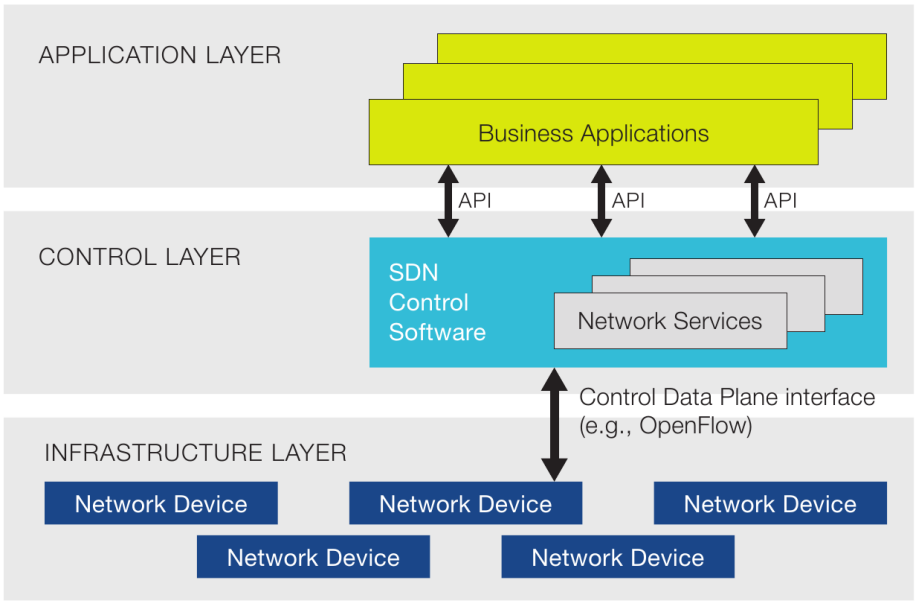
Nas últimas décadas novas tecnologias como IoT, *BlockChain*, Inteligência Artificial, Computação em Nuvem (*Cloud Computing*) e computação móvel vem modelando e transformando a maneira em que se troca informações. Com isso, surgiu a necessidade de inovar em tecnologias de redes. Rede de computadores tradicionais consistem em várias redes interconectadas entre si possuindo vários tipos de dispositivos desde *switches* e roteadores até *middleboxes* como *firewalls*, balanceadores de carga (*load balancers*) e sistemas de detecção de intrusos (*Intrusion Detection System*, IDS). Os roteadores e *switches* executam protocolos complexos e normalmente são fechados e proprietários. Operadores e administradores de rede constantemente implementam políticas cada vez mais sofisticadas e tarefas complexas com um conjunto limitado e altamente restrito de comandos de configuração de dispositivos de baixo nível utilizando linhas de comandos (*Command Line Interface*, CLI). Sendo assim, redes tradicionais são centradas no hardware impedindo a inovação e pesquisas na área de redes e restringindo a inserção de novos protocolos (KIM; FEAMSTER, 2013), (FEAMSTER; REXFORD; ZEGURA, 2014), (MASOUDI; GHAFFARI, 2016).

A SDN, vêm propondo uma mudança em relação às limitações da arquitetura tradicional. Como mudança, a SDN propõe a separação do plano de controle e do plano de encaminhamento ou plano de dados. Este tipo de separação permite a programabilidade direta do plano de controle. Consolidando esta separação, o plano de controle é representado por um único *software* um controlador capaz de administrar múltiplos dispositivos no plano de dados (*switches*, roteadores). O controlador SDN assume o papel de Sistema Operacional de Rede (*Network Operating System*, NOS). Dispositivos localizados no plano de dados são representados por *switches* e consistem de uma ou várias tabelas de encaminhamento que são preenchidas pela a ação do controlador SDN (KIM; FEAMSTER, 2013), (FEAMSTER; REXFORD; ZEGURA, 2014), (MASOUDI; GHAFFARI, 2016), (COX et al., 2017).

A comunicação entre os plano de controle e o plano de dados é estabelecida através de interfaces. Essas interfaces são divididas em: Interface Sul (*SouthBound Interface*, SBI), Interface Norte (*NorthBound Interface*, NBI) e em algumas abordagens para controladores distribuídos, *EastBound Interface* e *WestBound Interface* (LATIF et al., 2019).

A Figura 2 ilustra as 3 camadas que compõe uma arquitetura SDN, sendo a camada de infraestrutura (*Infrastructure Layer*) a responsável por representar o plano de dados e a camada de controle (*Control Layer*), a responsável pelo plano de controle ligadas por uma interface sul (*Control Data Plane Interface*). A camada de aplicação (*Application Layer*) é responsável por representar aplicações SDN ou simples aplicações que se comunicam com o controlador SDN (LATIF et al., 2019).

Figura 2 – Camadas da arquitetura SDN.



Fonte: (CHAGAS; FILENE R., 2018)

A separação do plano de controle do plano de dados levantam questões que podem trazer benefícios. A seguir, seguem as principais características de um modelo SDN (COX et al., 2017), (FEAMSTER; REXFORD; ZEGURA, 2014):

- Permite a separação do plano de controle (*Control Layer*) e o plano de dados (*Infrastructure Layer*).
- Permite o uso de um protocolo padronizado para troca de informações entre o controlador e os dispositivos de rede.
- Fornecer programabilidade de rede através de uma Interface de Programação de Aplicativos (*Application Programming Interface*, API) moderna e extensível.
- Redução de despesas de capital e operacional (CAPEX e OPEX).

2.3.1 Controlador SDN

Redes definidas por software foram um dos tópicos mais discutidos nas ultimas décadas, trazendo novas visões e recursos permitindo resolver muitos problemas herdados da arquitetura de rede tradicional. Enquanto em redes tradicionais os planos de controle e o plano de dados residem em um mesmo *switch*, em uma rede SDN o plano de controle é movido para fora do *switch*. O controlador é um elemento de rede que faz parte da arquitetura SDN que pode fornecer uma interface programável para rede. Ele pode ser usado para implementação de novas atividade de gerenciamento, roteamento, dentre outras funcionalidades (SHALIMOV et al., 2013), (NUNES et al., 2014a).

O controlador é um NOS que visa a oferecer um controle centralizado com interface aberta e API bem definida para proporcionar um ambiente ideal para desenvolvimento de aplicações SDN. Aplicações SDN são construídas utilizando o ambiente fornecido por controladores. Toda inteligência da rede está alocada no controlador que, por conta disto, possui uma visão geral da rede. Deste modo, o controlador pode aplicar políticas de decisão e controle para todos os dispositivos de uma rede SDN. Alguns dos principais controladores podem ser vistos no quadro 1 (GÖRANSSON; BLACK; CULVER, 2017b), (NUNES et al., 2014a).

Quadro 1 – Controladores abertos populares

Controlador	**Linguagem**	**Desenvolvedor**
POX	Python	Nicira
NOX	Python/C++	Nicira
MUL	C	Kulcloud
Trema	Ruby/C	NEC
Beacon	Java	Stanford
Floodlight	Java	BigSwitch
Ryu	Python	NTT, OSGR *group*
NodeFlow	Javascript	Desenvolvedores independentes
ovs-controller	C	Desenvolvedores independentes
RouteFlow	C++	CPqD
ONOS	Java	*Open Networking Lab* (ON.LAB)
Opendaylight	Java	*Linux Foundation*

Fonte: Adaptado de (NUNES et al., 2014b)

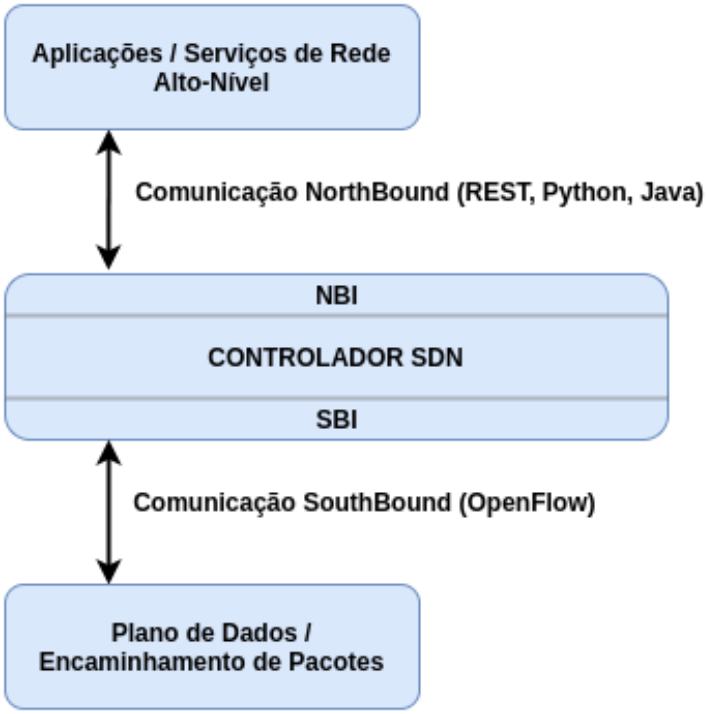
2.3.1.1 Controlador Ryu

O controlador Ryu é um sistema operacional de rede e controlador SDN. Foi escolhido por ser baseado em *python* e ser amplamente utilizado na área de pesquisa, além de fornecer uma API rica para criação de novas aplicações de gerenciamento e controle de rede. O Ryu suporta totalmente o *OpenFlow* v1.0, v1.2, v1.3, v1.4 e as extensões Nicira e seu código fonte estão disponíveis sob licença da *Apache* 2.0. O controlador Ryu é um dos mais usados pela comunidade de pesquisa por seu *design* único de arquitetura *multithread*, na qual novos componentes são executados como uma nova *thread* (COX et al., 2017) (CARGA et al., 2017) (ARBETTU et al., 2016).

O controlador Ryu foi projetado para fornecer uma interface para implementação de aplicações SDN, facilitando a agilidade e o gerenciamento de rede. A Figura 3 ilustra como o controlador Ryu se relaciona com as camadas superiores e inferiores de uma arquitetura SDN. Onde SBI é uma interface que permite que protocolos de comunicação (*OpenFlow*) troquem informações entre o controlador e os dispositivos *OpenFlow*. A NBI

é uma interface (REST, Java, Python) que permite a comunicação entre o controlador e aplicações de alto-nível (RYU, 2014).

Figura 3 – Arquitetura do Controlador Ryu



fonte: Adaptado de (NUNES et al., 2014a).

Os controladores SDN apesar de fornecerem uma forma inovadora para gerenciar dispositivos, seu modelo de implementação centralizado para tratar tais atividades, pode afetar em sua escalabilidade. Dependendo da extensão da rede, esse fator pode afetar seu desempenho. O controlador por si só representa um único ponto vulnerável de falha para toda a rede e por isso, novos modelos de implementação podem ser necessários. Apesar de protocolos como o *OpenFlow* especificarem que apenas *switches* deve ser controlados por controladores, a arquitetura SDN permite um modelo de implantação distribuída de controladores. Embora a comunicação **controlador-controlador** não seja definida pelo *OpenFlow*, ela é necessária para qualquer tipo de distribuição ou redundância. Embora o *OpenFlow* não aceite conexões do tipo "controlador-controlador", ele permite que vários controladores se conectem a um único *switch*, possibilitando que controladores de *backup* assumam o controle em caso de falhas (NUNES et al., 2014a).

A dinâmica de funcionamento do controlador Ryu é caracterizada pelo padrão de programação orientada a eventos. Os eventos podem ser gerados por acontecimentos na rede e capturados por *decorators* em *Python*. Alguns eventos podem ser gerados por classes dentro do código do controlador que por sinal podem gerar outros eventos. O controlador Ryu possui classes responsáveis por enviar mensagens de requisições (*Request*) e classes para tratar as respostas (*Reply*). As mensagens de requisições são enviadas através das classes específicas para este tipo atividade. Em seguida, esses eventos gerados por classes

de requisições são capturados por *decorators* definidos pela classe *@set_ev_cls* e especificando como parâmetro uma classe de resposta (*Reply*). Estas respostas por sinal retornam mensagens de contadores estatísticos, *status* de porta entre outros valores, seguindo assim um desencadeamento de funções e métodos (RYU, 2014).

A programação orientada a eventos é caracterizada por um laço de repetição responsável por gerar eventos que por sinal podem acionar eventos secundários. O controlador Ryu possui várias classes de eventos bem definidas referentes a eventos específicos. Cada classe é responsável por um tipo de evento mensagem que pode ser gerada ou capturada por *decorators*. A classe base responsável pelas mensagens de eventos é *EventOFPMsgBase*. Um evento descreve uma classe de evento do Ryu que foi gerado internamente ou externamente ao ambiente do sistema. A partir desta classe base, outras classes de eventos herdam as mensagens que são geradas (RYU, 2014). Sua descrição pode ser vista no Quadro 2.

Quadro 2 – Classe Base de Eventos do Ryu.

Atributo	Descrição
msg	Instância da classe **ryu.controller.controller.Datapath**
msg.datapath	Instância da classe *ryu.controller.controller.Datapath* na qual descreve um *switch OpenFlow*
timestamp	Registro de *timestamp* quando o *datapath* gera esse evento

Fonte: Adaptado de (RYU, 2014).

Uma aplicação SDN pode utilizar *OpenFlow* para ter uma ação reativa ou proativa. A forma reativa de aplicações SDN recebe pacotes encaminhados pelos dispositivos de rede para o componete *Packet Listener* dentro do controlador geralmente representado pela função *packet_in* em *python*. Em abordagens reativas a aplicação envia instruções de volta ao *switch* que prescreve como lidar com esse pacote. Em aplicações reativas a comunicação entre o *switch OpenFlow* e o controlador normalmente será dimensionada com novos pacotes injetados na rede. Na forma de aplicação proativa se difere por: (a) Definir o comportamento de encaminhamento a priori, em vez de esperar que os pacotes sejam encaminhados; (b) pouco dinamismo é alcançado ao responder eventos externos em relação a aplicações reativas; e (c) permite a escrita de aplicações que residem externo ao controlador podendo ser programada em qualquer linguagem de programação desde que utilizem APIs REST do controlador (GÖRANSSON; BLACK; CULVER, 2017a).

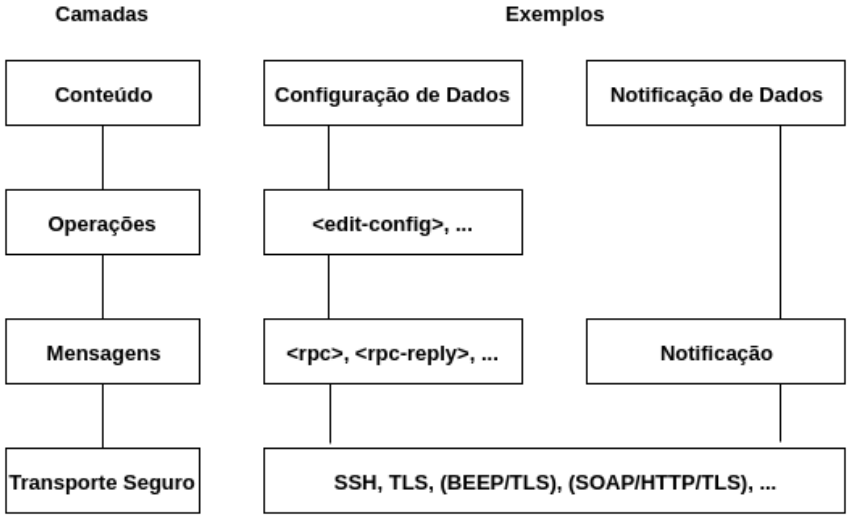
2.3.2 Protocolos

Os protocolos de rede são conjunto de regras utilizadas para envio e recebimento de mensagens para os dispositivos conectados. Como visto em seções anteriores, alguns protocolos de redes foram utilizados para gerenciar dispositivos mas apenas o SNMP ganhou popularidade. Mas com tempo utilizando o protocolo SNMP infelizmente se tornou aparente que, apesar do que foi originalmente planejado, o SNMP não poderia ser usado para configurar equipamentos de rede sendo utilizado apenas para monitoramento. Na mesma época, alguns fabricantes utilizavam uma abordagem de gerenciamento de rede baseada em Linguagem de Marcação (*e**X**tensible **M**arkup **L**anguage* ou XML) para se comunicar com seus dispositivos remotamente. Esse tipo de abordagem foi levada em consideração pela IETF para a comunidade em geral como uma proposta para uma interface de gerenciamento mais uniforme que com isso surgiu o protocolo de gerenciamento de rede NETCONF (GRAY; NADEAU, 2013).

2.3.2.1 Netconf e XMPP

O protocolo **NETCONF** foi elaborado em uma série básica de Chamada de Procedimento Remoto ou *Remote Procedure Call* (RPC) e seus dados transportados em XML ou utilizando o modelo YANG (***Y**et **A**nother **N**ext **G**eneration data modeling language*). Foi basicamente desenvolvido em resposta às limitações do SNMP, projetado para funcionar de forma independente ao *OpenFlow*. O NETCONF pode ser dividido em quatro camadas, como pode ser visto na Figura 4 (GRAY; NADEAU, 2013), (GRAY; NADEAU, 2016), (GÖRANSSON; BLACK; CULVER, 2017a).

Figura 4 – Camadas do protocolo NETCONF.



Fonte: Adaptado de (GRAY; NADEAU, 2016).

Em relação a Figura 4, pode-se dizer que a camada de conteúdo consiste na configuração e notificação de dados. Na camada de operações, é definido um conjunto de operações

para editar ou recuperar dados de configuração. Esses dados são definidos no dispositivo para fazê-lo se comportar de uma maneira específica. Na camada seguinte temos a camada de mensagens onde é fornecido um mecanismo para codificar RPCs e notificações. Nessa camada é possível invocar procedimentos nos dispositivos passando e recebendo parâmetros. E por último, a camada de transporte seguro onde é fornecida uma conexão confiável entre um cliente e um servidor. Por fim, o NETCONF é tido como um protocolo que pode ser usado para monitoramento e configuração em uma rede definida por *software*. Informações como armazenamento, estatísticas, sessões etc, ficam disponíveis para serem usados para atividades importantes facilitando a elaboração de aplicações dinâmicas dentro de controladores SDN (GÖRANSSON; BLACK; CULVER, 2017a), (GRAY; NADEAU, 2016).

Do mesmo modo que o protocolo NETCONF, o Protocolo Extensível de Mensagens e Presença (*e**X**tensible **M**essaging and **P**resence **P**rotocol*, XMPP) pode operar como SBI. O XMPP é um protocolo aberto padronizado pela IETF e assim como o NETCONF, também baseado em XML para troca de informações. Do mesmo modo, o protocolo XMPP fornece um método semelhante a pequenos fluxos para enviar pequenas partes de informações XML de um dispositivo a outro. Além disso alguns estudos sugerem que o protocolo XMPP possa ser usado como protocolo de comunicação de alto-nível em serviços de *software*. Ainda que o XMPP proporcione soluções para *data centers* em larga escala, muitos controladores não fornecem suporte nativo a este protocolo (CHANG; CHEN, 2011), (OSINSKI; DANDOUSH, 2018).

2.3.2.2 OpFlex

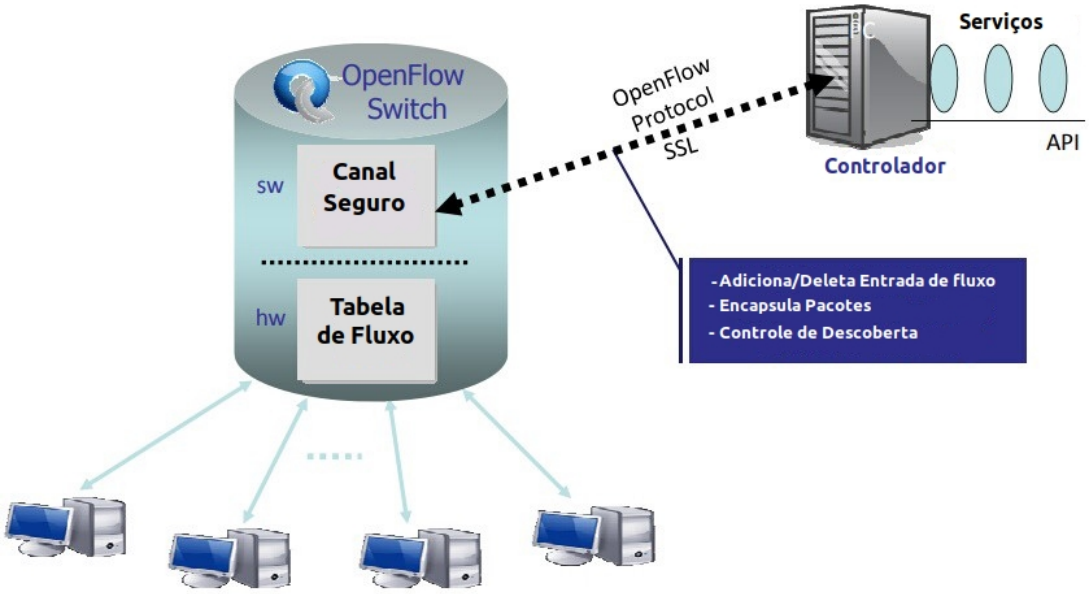
Protocolo OpFlex é um protocolo de comunicação SBI proposto pela Cisco *Systems* para IETF. Este protocolo é capaz de fornecer comunicação entre o plano de controle e o plano de dados, mas com um propósito diferente em comparação ao *OpenFlow*. Assim como o NETCONF, também foi projetado para ter seu funcionamento independente ao *OpenFlow*. O OpFlex é baseado em um modelo declarativo de informações de políticas, o que significa que centraliza apenas as políticas de gerenciamento e implementação mas distribui a inteligência e o controle. Com o objetivo de ser um protocolo escalável, o OpFlex tenta distribuir a complexidade fazendo com que os próprios dispositivos de encaminhamento (plano de dados) sejam responsáveis pelo gerenciamento de toda a rede, exceto políticas. As políticas são definidas no repositório de políticas ou *Policy Repository* (PR) logicamente centralizadas que se comunica com elementos de política ou *Policy Elements* (PE) do protocolo OpFlex. Os dispositivos são conectados ao PE e são registrados pelo módulo *End-Point Registry*, responsável pela adição e remoção de dispositivos. Uma das principais limitações deste protocolo é que ele retira o recurso principal de programação de rede a partir do controlador SDN (LATIF et al., 2019), (SMITH et al., 2016).

2.3.2.3 OpenFlow

Bem como os protocolos já citados, nenhum deles foi de fato padronizado para ser utilizado como SBI em uma arquitetura SDN. A iniciativa ***OpenFlow*** surgiu dos esforços de pesquisadores da universidade de *Stanford*. Originalmente o *OpenFlow* permite a criação e a experimentação de novos protocolos de rede. Sua visão inicial descreve a capacidade de substituir as funcionalidades de protocolos da camada 2 e 3 referindo-se como uma abordagem *clean slate*. Em 2011 empresas como Google, Facebook, Cisco Systems e a Microsoft criaram a *Open Networking Foundation* (ONF) um consórcio sem fins lucrativos para promover o uso do *OpenFlow* como o primeiro padrão de interface de comunicação entre as camadas de controle e de dados (VAUGHAN-NICHOLS, 2011), (GRAY; NADEAU, 2013), (OPEN NETWORKING FOUNDATION, 2012).

Em redes tradicionais, os planos de dados e controle estão alocados dentro de um único dispositivo (*switch* ou roteador), onde o plano de dados (também conhecido como Plano de Encaminhamento ou *Forwarding Plane*) é representado por tabelas de encaminhamento, enquanto o plano de controle é representado pelo NOS. No modelo *OpenFlow* o plano de dados continua sendo representado por *switches*, enquanto o controle pode ser alocado em um servidor padrão onde *switch* e controlador trocam informações (Figura 5).

Figura 5 – Componentes do protocolo Openflow.



Fonte: Adaptado de (ZHU, 2018)

Como visto na Figura 5, o modelo *OpenFlow* possui os seguintes componentes:

- **Protocolo *OpenFlow***: Protocolo aberto usado para troca de mensagens entre o controlador e os equipamentos *OpenFlow*. Essas mensagens trafegam em um canal seguro onde podem ser síncronas (*hello*, *echo vendor)*, assíncronas (*packet in*, *flow removed*, *port status*) ou geradas pelo controlador (*feature*, *configuration*, *modify*

state, *send packet*, *barrier*). Com essas mensagens é possível adicionar, deletar ou modificar entradas de fluxos, encapsular pacotes etc.

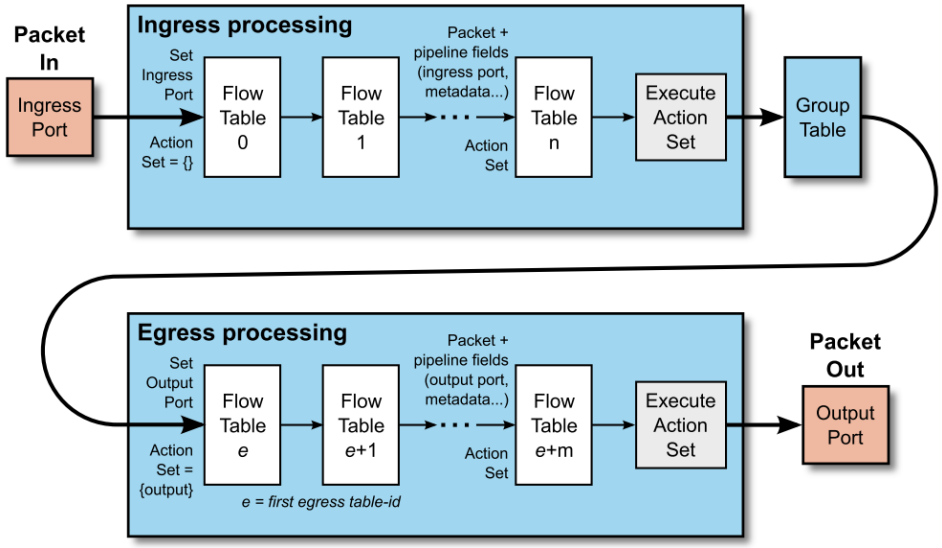
- **Canal Seguro**: Se caracteriza por um meio onde controlador o dispositivo *OpenFlow* podem trocar informações garantindo a confiabilidade das mensagens. Esta interface de acesso seguro se dá por meio do protocolo SSL (*Secure Socket Layer*).
- **Tabelas de Fluxos**: As tabelas de fluxos consistem em entradas de fluxos que possuem ações, instruções, contadores que são essenciais e que definem o modo como os pacotes devem ser processados e encaminhados.

O *OpenFlow* proporciona uma API que dá aos administradores de rede a habilidade de programar e determinar tipo de atributo, prioridade ou até mesmo bloquear pacotes na rede. Permite que a rede se comporte a depender da vontade do gerente de redes. O *OpenFlow* permite o acesso e a manipulação das tabelas de fluxos dos dispositivos conectados. Cada tabela de fluxo pode ser preenchida por entradas de fluxos (*Flow Entries*) onde cada entrada de fluxo representa uma instrução a ser efetuada (ROTHENBERG et al., 2011), (OPEN NETWORKING FOUNDATION, 2012), (VAUGHAN-NICHOLS, 2011), (ONF, 2015).

Quando um pacote chega a um equipamento *OpenFlow* é feita uma comparação com as regras das entradas de fluxos. Caso encontre correspondência os contadores são atualizados e as ações são realizadas. Caso não encontre correspondência nas entradas de fluxos, o novo pacote será enviado ao controlador através de uma mensagem *packet in* onde apenas o cabeçalho é enviado ao controlador mantendo o pacote armazenado no *buffer* do equipamento. O controlador, por fim, envia a modificação das entradas de fluxos para o *switch OpenFlow* em função da aplicação em execução para que então o novos pacotes tenham correspondência às novas regras instaladas no futuro (ROTHENBERG et al., 2011), (CUNHA, 2017).

Após o controlador enviar as novas regras ao *switch*, os pacotes passam por um novo processo de correspondência onde normalmente será feita na primeira tabela de fluxo, a tabela 0, visto na Figura 6. Caso não encontre será enviado a próxima tabela seguindo então um canal de processamento (*Pipeline Processing*). Cada *switch* pode ter múlitplas tabelas de fluxos e cada tabela contendo suas próprias entradas de fluxos.

Figura 6 – Processamento de Pacotes.



Fonte: (ONF, 2015)

As entradas de fluxos possuem campos com um conjunto de características que testam as correspondências dos pacotes. As entradas de fluxos seguem um canal de processamento até que seja encontrada uma correspondência. O pacote será encaminhado para todas as tabelas de fluxos configuradas caso contrário o pacote estará sujeito às configurações da *table miss* onde poderá ser descartado ou encaminhado para o controlador onde será feita uma nova análise. Os pacotes correspondidos serão processados de acordo com um conjunto de ações instalados nas entradas de fluxos (CUNHA, 2017). Entradas de fluxo são construídas a depender da aplicação em execução no controlador. A Figura 7) indica os campos da entrada de fluxo.

Figura 7 – Campos Entrada de Fluxo.

Cookie	Table	Timeouts	Counters	Flags	Priority	Match Fields	Instructions

Fonte: Elaborado pelo autor.

Alguns campos correspondem aos campos criados pela aplicação referente a esta pesquisa (ONF, 2015).

- **Cookie**: É o valor escolhido pelo controlador podendo ser usado para filtrar entradas de fluxos afetadas por requisições de estatísticas, modificação e de exclusão. Este campo não é usado no processamento de pacotes.
- **Table**: Este campo se refere ao número da tabela onde a regra esta instalada.
- **Timeouts**: Refere-se a quantidade máxima de tempo ou tempo ocioso antes que o fluxo seja expirado. Este campo pode ser representado pelos valores *hard_timeout* e *idle_timeout* da API *OpenFlow* do controlador Ryu.

- **Counters**: Contadores são atualizados quando os pacotes são correspondidos. Este campo refere-se aos números de pacotes (*n_packets*) e ao número de *bytes* (*n_bytes)*) da API *OpenFlow* do controlador Ryu.
- **Flags**: Alteram a forma como as entradas de fluxos são gerenciadas
- **Priority**: Refere-se a precedência de correspondência do fluxo, onde o maior número representa maior prioridade da entrada de fluxo.
- **Match Fields**: Refere-se aos campos para correspondência com os pacotes. Este campo pode gerar correspondência com os valores de ip de origem, ip de destino e *EtherType* que é usado para indicar que tipo de protocolo esta encapsulado no *payload*.
- **Instructions**: Este campo refere-se ao conjunto de ações que fazem parte do canal de processamento do pacote.

Nesta pesquisa o protocolo *OpenFlow* foi escolhido por ser o mais usado entre a comunidade como padrão de uma arquitetura SDN. Sua verão 1.4 foi implementada por fornecer suporte a propriedades de portas óticas. Essa adição permite o uso de tecnologias de canais de fibra ou *Fibre Channel over Ethernet* (FCoE). FCoE é uma tecnologia de rede de computador que encapsula quadros de canais de fibra em redes *Ethernet*, que permite que o canal de fibra use redes *Ethernet* de 10 Gbps (ou velocidades mais altas), preservando o protocolo FCoE (REN; XU, 2014).

2.3.3 Software Switch

Uma das proeminentes tendências na área da computação em redes são os crescentes atuações de *software* no plano de dados. Atualmente existe um grande número de abordagens de *software switch*. Algumas implementações atendem casos que vão desde simples Funções de Redes Virtualizadas (*Network Function Virtualization*, NFV) até virtualização de redes para experimentação de projetos de novos protocolos. *Software switches* são muito mais ricos e extensíveis do que os padrões L2. *Software switch* pode ser implementado para realizar diversos procedimentos sendo possível implantar políticas de controle de acesso entre contêiners. Da mesma maneira que podem ser usados instanciar redes de sobreposição separadas em malha em um *data center* ou ser utilizado para um simples roteamento em NFV. Todos os *switches* possuem um certo número de portas que pode ser físicas (NIC) ou virtuais (conectadas a um contêiner ou máquina virtual) (FANG et al., 2018).

2.3.3.1 Open vSwitch

É o *software switch* mais popular da comunidade com integrabilidade ao *Mininet* (que será discutido na seção 2.3.4.3). Projeto *open source* pela Linux *Foundation*, multicamada, licenciado pela Apache 2 . Foi implementado para que suportasse interfaces de gerenciamento padrão e funções abertas para encaminhamento e controles programáticos. O *Open vSwitch* (OvS) é bastante adequado em ambientes virtuais. Possui uma boa performance por oferecer um modo *kernel* sendo propício para ambientes de produção. Além de expor as interfaces de controle e visibilidade padrão para a camada de rede virtual, foi projetado para oferecer suporte a vários servidores físicos e várias tecnologias baseadas em Linux, em suas últimas versões pode-se incluir *Xen/XenServer* 6.0, *Xen Cloud Plataform*, KVM, Proxmox VE e *VirtualBox*. Também foi integrado em vários sistemas de gerenciamento virtual como *OpenStack*, *openQRM*, *OpenNebula* e *oVirt*. O OvS possui suporte a encaminhamento de mensagens *OpenFlow* versões mais recentes dão suporte ao *OpenFlow* 1.5. A maior parte do OvS foi desenvolvida em C independente de plataforma para ser facilmente transportado para outros ambientes operacionais. Dentre as tecnologias suportadas pelo OvS algumas são (PIOLINK, 2013) (FERNANDES; ROTHENBERG, 2014):

- Modelo VLAN 802.1Q portas *Trunk* e *Access*
- NIC com ou sem LACP (*Link Aggregation Control Protocol*)
- Monitoramento de tráfego com NetFlow, sFlow(R), IPFIX, SPAN, RSPAN.
- Configuração de QoS.
- Multi-tunelamento GRE, VXLAN, STT e Geneve com suporte a IPsec.
- Suporte a prevenção de loop com STP e RSTP.
- LLDP (*Link Layer Discovery Protocol*) e SPBM (*Shortest Path Bridging MAC*)
- Suporte ao algorítmo de agendamento de rede ou *Hierarchical Fair Service Curve* HFSC qfisc.
- NIC com balanceamento de carga de MAC de origem.
- IPv6
- Suporte a novas versões do protocolo *OpenFlow* 1.5
- Banco de dados de configuração transacional com ligações em C e *Python*.
- Encaminhamento de alto desempenho usando módulo *kernel* do Linux.

Além do OvS, exitem outras soluções mais recentes disponíveis para *software switch* com suporte ao protocolo *OpenFlow* (RAHIMI et al., 2016), (FERNANDES; ROTHENBERG, 2014), (HAN; SCIENCES, 2019), (MOLNÁR et al., 2016):

- **LINC**: *Software switch* desenvolvido em *Erlang*. Suporta versões do *OpenFlow* 1.2 e 1.3 possui integração com o *Mininet*.
- **Lagopus**: É um *software switch OpenFlow* projetado para prestadores de serviços de área ampla. O Lagopus foi projetado para ser executado em ambientes multi-processamento. Algumas de suas características são: classificadores de pacotes e mecanismos de QoS.
- **BESS ou *Berkeley Extensible Software Switch***: É uma plataforma programável para implementação de plano de dados em redes. Um de seus aspectos são o de introduzir atributos dinâmicos de metadados por pacotes, que substituem a noção estática, ineficiente e propensa a erros de "anotações"e incorporar um agendador que reconhece recursos, na qual permite políticas flexíveis de agendamento. O agendador fornece recursos compartilhados, como tempo do processador e largura de banda do *link* entre VMs, aplicativos ou fluxos.
- **Trema Edge Switch**: Faz parte do código do controlador SDN Trema. Uma de suas características é a possibilidade de criar suas topologias virtuais utilizando ferramenta própria, diferente de outras abordagens que necessitam de ferramentas externas como *Mininet*.
- **ESwitch**: O ESwitch é uma nova arquitetura de *switch OpenFlow* que foi criada com o objetivo de descobrir e fornecer recursos mais rápido do que *switches* de uso geral. É considerado um *switch* cuidadosamente adaptado para o canal de processamento *OpenFlow*. Utiliza um código gerado baseado em modelo para derivar o caminho de dados personalizados.

2.3.4 Emulação de Rede

Com todas as vantagens que a arquitetura SDN possa oferecer e como toda nova tecnologia ela precisa ser validada e testada. A maioria dos campos da engenharia inclui padrões ou referências para validar as melhorias propostas. Novas arquiteturas, redes sem fio, modelos de tolerância a falhas e muitos outros utilizam emuladores para comparar, testar novas arquiteturas, protocolos e métodos com os recursos existentes. Para a arquitetura SDN não é diferente. No entanto mesas de teste ou *testbeds* ainda são necessárias para validação de performance, escalabilidade, tolerância a falhas e estabelecer casos de uso para fins de gerenciamento. Infelizmente o custo de *hardware* específicos são altos e falhas de rede não são aceitáveis, o que trás problemas na realização de testes em uma rede ativa. Por

esta razão, são necessárias plataformas de simulação e/ou emulação para fornecer aos pesquisadores e engenheiros um meio para validar suas propostas antes de aplicá-las em uma rede em produção. Existem algumas ferramentas que estão disponíveis no âmbito de redes definidas por *software*. Essas ferramentas são importantes para visualizar e avaliar alterações na infraestrutura ou nos protocolos de rede (COX et al., 2017), (VEENA et al., 2014).

2.3.4.1 vSDNEmul

Estudos recentes apresentam o *vSDNEmul* como uma proposta de um emulador de redes capaz de fornercer um ambiente SDN. Este ambiente pode ser composto de *hosts* ou dispositivos baseados em contêiners de *software* conectados entre si por tecnologia de encaminhamentos. O *vSDNEmul* tenta entregar um plano de dados real com fidelidade onde o conceito de contêiners teve um papel fundamental para isso. Diferentemente do *Mininet*, o *vSDNEmul* oferece nós totalmente isolados contendo seus próprios recursos de computação, tais como: memória, cpu, redes. Esta característica permite aumenta sua precisão em oferecer uma avalização de desempenho, dado que, cada nó trabalha de forma semelhante a um ambiente real. Por último, o *vSDNEmul* permite a construção de novos dispositivos ou aplicações através de *templates* de imagens disponibilizados pelo *Docker* facilitando a escolha do sistema operacional, bibliotecas e protocolos que são utilizadas na execução do nó (FARIAS; SALVADOR; ABELÉM, 2018).

2.3.4.2 MaxiNet

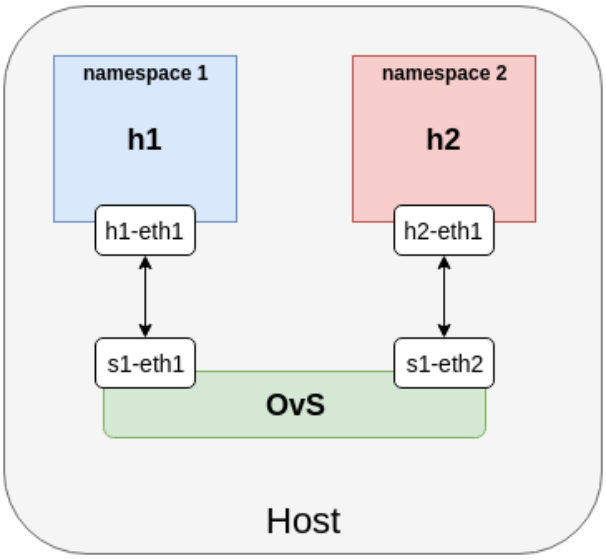
O *MaxiNet* pode ser considerado uma camada de abstração que conecta várias instâncias do *Mininet* não modificadas em execução em diferentes *hosts*. É fornecida uma API central para acessar o *cluster* de instâncias do *Mininet*. Túneis GRE são usados para interconectar os nós emulados em diferentes *hosts*. O *MaxiNet* funciona como um *front-end* para o *Mininet* que configura todas as instâncias do *Mininet*, executa comandos nos nós e configura os túneis necessários para a conectividade adequada. Dessa forma, com o *MaxiNet* é possível emular topologias de *data centers* em escala razoável de tempo usando um *cluster* de máquinas físicas. Isso, aliado a um gerador de tráfego, abre as portas para uma avaliação realista de novos protocolos de roteamento de *data centers* através de emulação que pode ser usados para avaliações rápidas de prototipagem (WETTE et al., 2014) (ROJAS et al., 2018).

2.3.4.3 Mininet

O *Mininet* esta disponível como uma ferramenta *open source* que emula dispositivos *OpenFlow* e também controladores SDN para experimentos. O *Mininet* acaba sendo a ferramenta predominante para criação e testes de novos conceitos e topologias baseados em

redes definidas por *software*. É considerado uma ferramenta leve, o *Mininet* foi projetado com base em processos simples do *shell*. O *Mininet* faz uso de processos Linux em *namespaces* de rede em uma área de trabalho simples para criação de redes dimensionáveis, virtuais e definidas por *software*. O conceito de *namespace* de rede permite a criação de domínios virtuais de rede com seus próprios conjuntos de interfaces, endereços ips, tabelas de roteamento etc (Figura 8). O uso da virtualização baseada em processo permite que o *Mininet* emule uma rede com dezenas de *hosts* e *switches* em uma única máquina. Nesse ambiente, um pesquisador ou operador de rede pode desenvolver uma lógica de controle e testá-la em uma emulação em larga escala no plano de dados. Com isso, o *Mininet* é capaz de oferecer um ambiente leve para pesquisa, desenvolvimento, prototipagem, teste, depuração e outras tarefas que possam beneficiar uma rede experimental em apenas um laptop ou desktop (KAUR; SINGH; GHUMMAN, 2014), (FEAMSTER; REXFORD; ZEGURA, 2014), (COX et al., 2017).

Figura 8 – *Namespace* do *Mininet*.



fonte: Adaptado de (HONGWEI, 2017).

Os processos do *shell* envolvendo o *Mininet* podem atuar como *hosts*, *switches* e controladores e exigem significativamente menos recursos que máquinas virtuais, porém, permite apenas a criação de redes compostas por sistemas homogêneos. A construção da rede emulada pode ser feita e administrada por uma API *python*, que permite a criação e customização da topologia de rede e de seus componentes. Além disso, através da biblioteca padrão do *python*, tarefas específicas para monitoramento de rede ou geração de tráfego podem ser agendadas para execução nos processos gerados. O *Mininet* também disponibiliza uma interface gráfica (GUI) *MiniEdit* que permite a criação e visualização dessas topologias de rede de uma forma mais amigável (COX et al., 2017), (KAUR; SINGH; GHUMMAN, 2014).

O *Mininet* foi utilizado nesta pesquisa por permitir que as aplicações instaladas sejam utilizadas pelos *hosts* criados através da técnica de *namespace* o que dá a liberdade do pesquisador realizar testes utilizando outros tipos de *software* baixados através de listas de repositórios.

2.4 SENSIBILIDADE A CONTEXTO

Sensibilidade a contexto está diretamente ligada a computação pervasiva (ubíqua) onde foi afirmado que as tecnologias mais profundas são aquelas que desaparecem. Da mesma forma, é afirmado que tal desaparecimento é consequência do fator psicológico humano onde uma pessoa que aprende a realizar uma tarefa suficientemente bem e repetidas vezes pode fazê-la sem ter consciência dela. E por realizar tal tarefa repetidas vezes esta atividade iria desaparecer (inconscientemente), consequentemente deixando o indivíduo livre para realizar outras tarefas. Tendo isso em vista, aplicando esta temática à computação ubíqua, é correto afirmar que tecnologias envolvendo computação ubíqua irão ajudar a resolver o problema da sobrecarga de trabalho (WEISER, 1991).

Uma das áreas da computação pervasiva (ubíqua) é a computação sensível a contexto (*context-awareness*). Contexto significa capturar informações com o uso de sensores (físicos ou lógicos), inferir essas informações para então adaptá-las automaticamente ao usuário. Há algum tempo atrás, a habilidade de perceber mudanças de contexto era considerada uma característica humana. Mudança de ambiente, temperatura, intensidade luminosa ou até a interação humano-computador geram informações que podem caracterizar um contexto. Um sistema sensível a contexto está apto a adaptar seus serviços baseado na captura do contexto atual. Para o uso eficiente do contexto, foi necessário entender o que seria contexto e como usá-lo em um ambiente computacional (ABOWD et al., 1999),(GANDODHAR; CHAWARE, 2019).

Segundo o conceito introduzido por (DEY, 2000), contexto é "qualquer informação que possa ser usada para caracterizar a situação de uma entidade (pessoa, lugar ou objeto) e que é considerada relevante para a interação entre usuário e aplicação, incluindo ambos". Sensibilidade a contexto é a capacidade do sistema computacional sentir, observar características e identificar mudanças no ambiente ao seu redor e, além disso, estar apto para reagir a tais mudanças. Alguns estudos, descrevem sistemas sensíveis a contexto como aqueles que gerenciam elementos contextuais relacionados a uma aplicação em um domínio e usam esses elementos para apoiar um agente na execução de alguma tarefa. Esse apoio pode ser alcançado pelo aumento da percepção do agente em relação à tarefa sendo executada ou pelo provimento de adaptações que facilitem a execução da tarefa. Do mesmo modo, outra abordagem importante é apresentada, na qual definem e distinguem dois conceitos: O elemento contextual é qualquer dado, informação ou conhecimento que permite caracterizar uma entidade em um domínio. O contexto da interação entre um agente e uma aplicação, para executar alguma tarefa, é o conjunto de elementos con-

textuais instanciados que são necessários para apoiar a tarefa atual (VIEIRA; TEDESCO; SALGADO, 2009).

Para facilitar o processo de identificação de contexto, alguns modelos de classificação foram propostos. Um elemento de contexto pode ser classificado seguindo o modelo 5W+1H: Quem (*Who*), O que (*What*), Onde (*Where*), Quando (*When*), Porque (*Why*) e Como (*How*) que serão descritos a seguir (VIEIRA, 2011), (MORSE; ARMSTRONG; DEY, 2000):

- **Quem** (*Who*): Indica informação contextual sobre a entidade.
- **Onde** (*Where*): Indica informação contextual de localização. Um dos mais usados na computação móvel.
- **O que** (*What*): Indica o que foi ou o que está sendo realizado pela entidade. É considerado uma das informações contextuais mais difíceis de identificar porque depende da variedade de atividade que um sistema pode realizar.
- **Quando** (*When*): Indica informação contextual de tempo.
- **Porque** (*Why*): Indica o motivo que levou a entidade a realizar a atividade.
- **Como** (*How*): Indica como os elementos contextuais foram recolhidos ou executados.

Do mesmo modo, a classificação de elementos contextuais pode ser básica ou complexa. Este tipo de classificação está relacionada a granularidade da informação. Informações contextuais básicas se referem a informações mais brutas e baixas, quando não é necessária a aplicação de cálculos ou processamento de informações, enquanto no contexto complexo é necessário um método para processar a informação.

Os sistemas sensíveis ao contexto atuais, normalmente, envolvem três principais módulos ou fases: Captura de Dados, Inferência de Contexto e Adaptação ao Contexto (VIEIRA; TEDESCO; SALGADO, 2009).

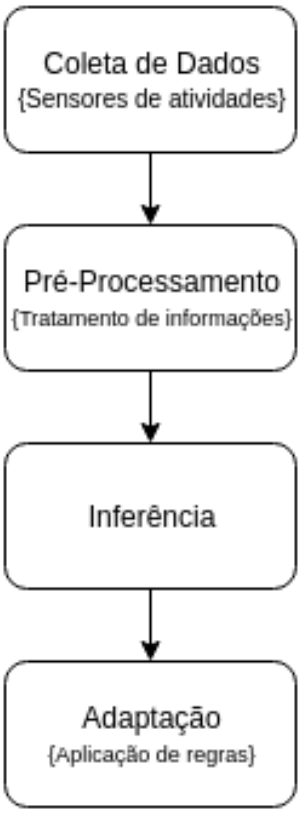
Captura de dados. A primeira etapa é de coleta, o que envolve toda a parte de sensoreamento, inserção de dados via dispositivos de entrada, ou mesmo com origem em informações históricas ou em outras aplicações.

Inferência de contexto. É o processo no qual são realizadas as análises dos dados coletados, para que seja feita a inferência de situações. Nessa etapa, está a inteligência do sistema para interpretar as informações contextuais. É muitas vezes considerada como sinônimo ou complemento da fase de raciocínio. Em geral, são utilizados dois tipos de abordagem: uma mais simples, estática, baseada na definição prévia de regras pelos administradores e desenvolvedores; a segunda abordagem é essencialmente baseada em decisões automáticas, aprendidas pelo sistema de forma dinâmica, utilizando algoritmos de ML.

Adaptação ao contexto. Nessa fase, uma vez que o sistema inferiu a ocorrência de uma determinada situação contextual, ele poderá tomar certas ações mais adequadas. Para o gerenciamento de redes de computadores, essa é uma fase fundamental, pois permite que o sistema não só perceba o ambiente, mas reaja em tempo mínimo a uma situação de falha ou degradação da rede, o que de fato dará a característica de autonomia à gestão da rede.

Para esta pesquisa, algumas adaptações foram necessárias para a arquitetura de processamento de informações de contexto proposta em (GANDODHAR; CHAWARE, 2019). Na Figura 9, é possível observar o processamento de informações de contexto adaptado: inicialmente é feita a coleta de dados, capturando eventos que ocorrem na rede; em seguida é feito o pre-processamento desses dados para extrair as informações necessárias. Logo após, com essas informações, é feita a inferência de contexto; e, por fim, o sistema se adapta com a aplicação da regras de contexto.

Figura 9 – Arquitetura do processamento da informação de contexto.



Fonte: Adaptado de (GANDODHAR; CHAWARE, 2019)

Existem vários desafios a serem superados envolvendo sensibilidade a contexto e a captura de elementos contextuais. Um exemplo disto é tido na computação móvel distribuída onde existe a dificuldade de explorar mudanças de ambientes, sendo este, considerado um evento de troca de contexto. Para uma rede estática, onde não há mudança de ambiente, outros elementos como mudança de tráfego, *status* de porta, ocupação da banda, serviços envolvendo protocolos de rede devem ser considerados. O paradigma da arquitetura

SDN permite programar o comportamento do controlador SDN para que possa reagir a elementos contextuais tornando esta rede mais adaptativa. Deste modo, uma rede de computadores reagiria a eventos simples, deixando o administrador de rede livre para realizar outras atividades de gerência ou projetando novas arquiteturas de rede.

2.4.1 Qualidade de Contexto

Estar ciente do contexto permite sistemas computacionais a realizarem atividades automaticamente, permite inferir sobre determinados eventos utilizando informações contextuais. Com isso, sistemas podem adaptar seus serviços para fornecer serviços personalizados e por conseguinte com uma melhor qualidade. Sistemas sensíveis a contexto geralmente trabalham com uma quantidade grandiosa de informações contextuais. Informações que são coletadas de alguma forma por sensores. Ultimamente, uma grande quantidade de aplicações sensíveis a contexto vem surgindo e qualidade se tornou uma questão crítica. Um dos grandes temas discutidos quando se oferece um serviço personalizado é oferecê-lo com qualidade. A qualidade de contexto ou *Quality of Context* (QoC) está diretamente ligada à qualidade da informação não ao processo. Alguns estudos afirmam que a qualidade de contexto como um elemento referente a uma informação contextual que pode determinar o valor desta informação para uma aplicação (KRAUSE; HOCHSTATTER, 2005), (SCHMIDT, 2011), (BU et al., 2006).

A qualidade de contexto pode ser considerada a causa para uma boa modelagem de contexto. Com informações imprecisas, o sistema sensível a contexto pode oferecer resultados sem sentidos ou inferências incorretas. Sendo assim, é necessário distinguir e analisar as diferentes categorias de qualidades: Qualidade de Contexto (QoC), Qualidade de Serviço (QoS) e Qualidade de Dispositivo (QoD). Em sistemas envolvendo sensibilidade a contexto, as informações pode ser incorretas, inconsistentes, redundantes ou incompletas. Pesquisas sugerem 4 tipos de imperfeições contextuais, sendo elas (HENRICKSEN; INDULSKA, 2004), (SCHMIDT, 2011), (PRADEEP; KRISHNAMOORTHY, 2019):

- **Informação Desconhecida**: Quando nenhuma informação está disponível.
- **Informação Ambígua**: Quando informações diferentes ou com mais de um sentido estão disponíveis.
- **Informação Imprecisa**: Quando as informações que não condizem totalmente com a natureza da situação estão disponíveis.
- **Informação Incorreta**: Quando a informação apresentada está totalmente incorreta.

Tais imperfeições podem ocorrer devido a fatores externos como simples falhas nos sensores, resultado por problemas mecânicos de imprecisão ou devido à própria natureza

dinâmica do sistema decorrente de incompatibilidade semântica. Os dados coletados através de sensores ou simplesmente gerado por um sistema externo podem fazer uso de etapas de processamento de informação. Esse processamento pode ser necessário para alcançar um estado de qualidade favorável para uma boa inferência de contexto (BOBEK; NALEPA, 2017).

2.4.2 Aprendizagem de Máquina SDN

A ML é uma técnica poderosa para extrair conhecimento dos dados, mas seu potencial ainda é raramente usado para soluções práticas em redes adaptativas, registrando que existem limitações impostas pela visão e pelos recursos das redes tradicionais. No entanto, com a capacidade de programação do SDN, as grandes quantidades de fontes de dados atuais e a alta disponibilidade do poder de computação fornecido pela computação em nuvem, torna-se possível o surgimento de soluções baseadas em ML. Fora isso, existem pesquisas que descrevem o uso de técnicas de ML combinadas com o paradigma de rede definida por software como *Supervised* e *Unsupervised Learning* no auxílio de detecção de ataque de negação de serviço distribuído. A separação dos planos de dados e controle permite ao administrador gerenciar e controlar a rede por meio de técnicas de programação, o que permite incorporar a ML. As técnicas de ML discutidas têm algumas tarefas, como coleta e treinamento (algoritmos) para classificar problemas de dados e segurança, o que faz com que haja um aumento no nível de complexidade computacional envolvida (LIU; XU, 2019), (AYOUBI et al., 2018).

2.5 TRABALHOS RELACIONADOS

De acordo com Lee e Shin (LEE; SHIN, 2018), a *zero-touch network* (KOLEY, 2016) é uma rede que minimiza o tempo de inatividade do serviço e os custos operacionais, graças à remoção da intervenção humana. A rede é construída usando tecnologias de nuvem e SDN (ROSSEM et al., 2017). Para Samba (SAMBA, 2018), é enfatizado como o gerenciamento de rede é importante ao introduzir uma nova estrutura para gerenciar arquiteturas SDN, onde foi proposto usar módulos específicos definidos nas funções alocadas no controlador. Tais funções podem fazer parte da própria API do controlador para facilitar o gerenciamento do mesmo. No entanto, foi informado que mesmo que o controlador possa desempenhar o papel de gerente, essa ação poderá sobrecarregá-lo, pois esse mesmo controlador já está encarregado de administrar todos os serviços de rede. Da mesma forma, é mencionado que existem áreas ativas de pesquisas envolvendo a separação do gerenciamento SDN do controlador.

Para Megyesi (MEGYESI et al., 2017), a arquitetura SDN já permite a extração de métricas diretamente da rede por meio de parâmetros de largura de banda, perda de pacotes e atraso. No entanto, destaca-se o desafio de calcular a largura de banda disponível

em ambientes dinâmicos. O artigo utiliza o *Mininet* como emulador para propor o uso de técnicas passivas para estimar a medição do ABW com base nas mensagens da API *northbound* do controlador, para então descobrir a topologia e monitorar os links pela extensão do protocolo *OpenFlow*. Eles consideraram apenas expandir a proposta usando a funcionalidade do *OpenFlows* 1.3, além de restringir os testes aos controladores OpenDayLight, ONOS e FloodLight. Nesta pesquisa, não é feita uma abordagem adaptativa para fazer com que a rede reaja a determinada sobrecarga de tráfego.

Martini et al. (2015) apresentaram um Controlador SDN baseado em serviço, explorando os recursos SDN e NFV para executar a distribuição de dados com reconhecimento de contexto (entrega diferenciada), criando redes de cadeias de serviço. Eles realizaram a composição dinâmica dos serviços por meio das funções virtuais e a abordagem trouxe vantagens em termos de escalabilidade. Porém, esta pesquisa se limita à versão 1.0 do *OpenFlow*.

Do mesmo modo, como em (LANGE et al., 2018), testou-se um sistema de gerenciamento de rede (NMS), no qual faziam solicitações constantes ao controlador ONOS para teste de desempenho. Para isso, eles testaram e ampliaram três técnicas de interação com o controlador usando a interface REST. Foi testada a capacidade de receber e usar medições com base no gerenciamento de rede e na largura de banda dos links e no consumo da largura de banda em fluxos individuais. A topologia do experimento usada no *Mininet* demonstra semelhança com a desta pesquisa, no entanto, o experimento se resume à utilização e avaliação de desempenho de 3 versões do controlador ONOS (Java).

Também baseado em SDN, Ghannami e Shao (2016) analisam um ambiente de rede de caminhos múltiplos, concentrando-se no processo de recuperação rápida de falhas usando o *OpenFlow*. A proposta visa monitorar caminhos de rede, identificar rapidamente falhas e redirecionar o tráfego para o melhor caminho alternativo utilizando o controlador SDN FloodLight (Java). Na proposta, eles usaram as tabelas de grupo do *OpenFlow* para enviar os fluxos, utilizando outra rota em caso de falha nos links. Para o caso de falhas, foi utilizado o protocolo para detecção de encaminhamento bidirecional para utilizar tanto o link principal quanto o de backup. O artigo mencionou um interesse futuro em investigar situações de cenários de congestionamento de rede.

Em (SILVA, 2016), é proposto um serviço de monitoramento de tráfego utilizando arquitetura SDN com o objetivo de fornecer uma visão do tráfego em três níveis de granularidade. Foi proposto também um serviço de balanceamento de carga utilizando os algoritmos *Round-Robin* e o *Bandwidth-Based* utilizando o controlador *OpenDayLight* (Java). A topologia em anel em um dos cenários propostos se assemelha ao desta pesquisa. Porém, o serviço de monitoramento apenas apresenta os resultados. O balanceamento não é feito de forma automática, deste modo, não foi feita qualquer gerência adaptativa baseada nas informações contextuais de banda.

Em (ZHOU et al., 2019), foi proposto um modelo adaptativo de coleta de dados baseado

em vários módulos em uma rede definida por *software*. Esta proposta visa o modelo de classificação 5W + 1H com o objetivo de responder as perguntas: quais dados, onde, como e quando coletá-los por meio de avaliação do *status* da rede. O sistema seleciona adaptativamente os nós de coleta adequados com o objetivo de reduzir a redundância dos dados, ao mesmo tempo que garante a percepção completa do tráfego. Durante a coleta, emprega a geração dinâmica de probabilidade e amostragem para determinar as regras de amostragens para reduzir ainda mais a redundância dos dados. Porém, este modelo foi implementado para garantir a precisão da detecção de ataques de negação distribuída de serviço (*Distributed Deny of Service* – DDoS).

2.6 CONSIDERAÇÕES FINAIS

Novas tecnologias estão modelando o processo de como o gerenciamento de redes pode ser realizado. Limitações do gerenciamento tradicional estão impondo novas mudanças nesta área. Cada vez mais, fica clara a necessidade por modificações em métodos que antes pareciam suficientes mas que, com o decorrer do tempo, exigiam interações maiores do que apenas monitorar desempenho de redes. Com novas tecnologias se conectando à rede, novas abordagens são necessárias para seu gerenciamento. A arquitetura SDN utiliza a separação do plano de controle com o plano de dados para proporcionar a programabilidade e a liberdade para testar novos protocolos e abordagens em tecnologias de rede. Com a programabilidade da rede ao alcance do gerente/administrador de redes, fica possível designar funções específicas para o comportamento de uma rede, de forma a automatizar algumas atividades. Do mesmo modo, vários tipos de conceitos e técnicas computacionais podem ser incluídas para facilitar estas atividades. O paradigma de redes definidas por *software*, unido a conceitos de regras simples para sensibilidade a contexto, podem proporcionar um gerenciamento mais adaptativo, que minimiza o tempo de inatividade do serviço e os custos operacionais, graças à remoção ou diminuição da intervenção humana, o que pode ser fundamental para uma rede em constante crescimento.

3 UMA ABORDAGEM PESO LEVE PARA GERENCIAMENTO DE REDES

Este Capítulo irá descrever os processos e métodos seguidos para alcançar a gerência adaptativa de redes. A proposta desta pesquisa consiste em utilizar o paradigma SDN com sensibilidade a contexto baseada em regras ou políticas simples na gerência de redes utilizando uma pequena porção da Rede ICONE (RNP). Tais regras simples, intuitivas (baseadas no conhecimento/experiência do gerente de rede (ROJAS, 2018) - ex. se o tráfego ou vazão na rede estiver maior que 80% da banda passante, então a rede está sobrecarregada/congestionada), são empregadas em situações corriqueiras do funcionamento das redes atuais óticas (ex. queda de enlace, congestionamento), especificamente redes com comportamentos mais estáveis (ex. Rede ICONE), que, em geral, não justificam o uso de técnicas mais complexas de ML. A metodologia definida para a execução do projeto está fundamentada no uso de emulação e experimentação com o auxílio de ferramentas para armazenar e apresentar o tráfego de uma arquitetura SDN.

3.1 REDE ICONE

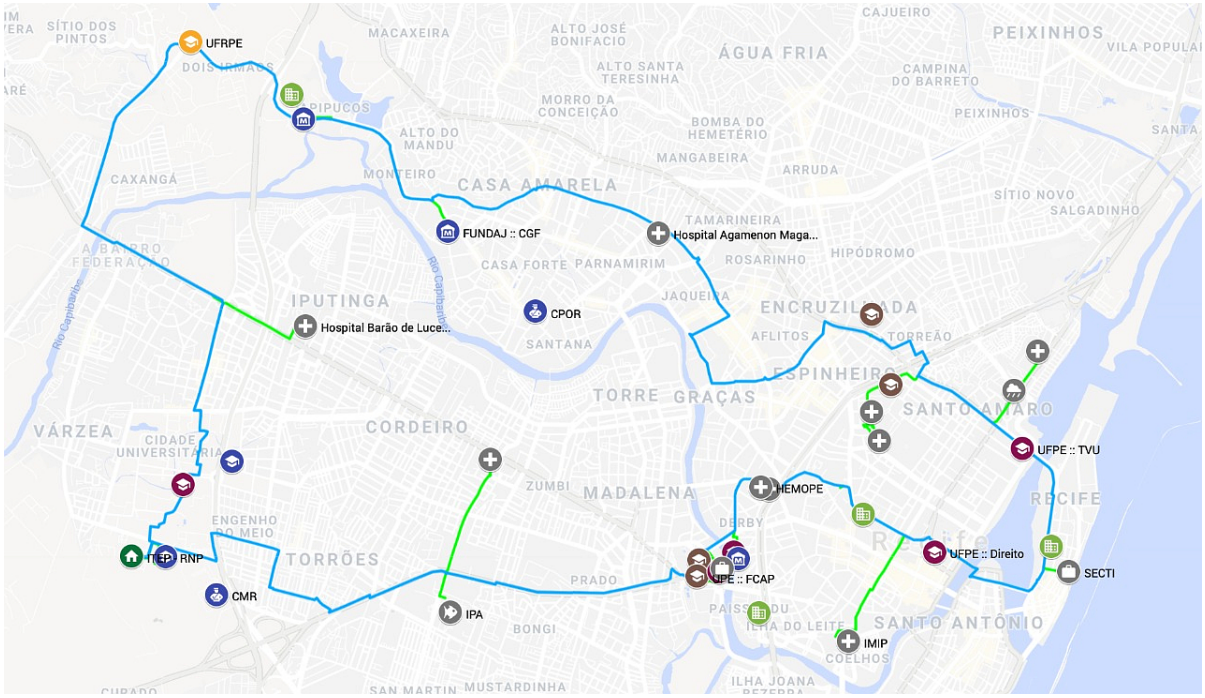
A rede ICONE é a **I**nfraestrutura de **C**omunicação **Ó**tica para e**N**sino e p**E**squisa instalada em Recife como parte do Programa Redecomep - Redes Comunitárias de Educação e Pesquisa da RNP, cujo objetivo é melhorar a qualidade de interconexão das instituições de pesquisa e educação através de redes metropolitanas de alta velocidade. Em Recife, a rede é gerenciada pelo Instituto de Tecnologia de Pernambuco (ITEP), que abriga o Ponto de Presença (PoP) da RNP no estado. Desta forma, a ICONE se conecta nacional e internacionalmente através do PoP-PE/RNP. Atualmente, 34 pontos de 22 instituições estão conectados a um anel de aproximadamente 87Km de fibra com enlaces de 1 Gbps, onde alguns anéis já foram atualizados para 10 Gbps. O mapa da rede pode ser visto na Figura 10.

Como em muitas das Redecomeps espalhadas pelo Brasil, a rede de fibras óticas da ICONE é aérea, instalada por meio de parceria com a Companhia Energética de Pernambuco (Celpe). É comum a quebra de fibras em função de acidentes ou mesmo em função de erros no trabalho de prestadores de serviço nos postes, hoje tão carregados de cabos elétricos e de comunicação. A **queda de link** (*link down*), consequentemente, é uma das ocorrências mais frequentes na rede.

Ainda sobre problemas de desempenho, são comuns os **congestionamentos** em trechos da rede. Como exemplo, imaginaremos o cenário em que a Fundação Joaquim Nabuco (FUNDAJ), uma das instituições conectadas à rede ICONE, realiza uma transmissão ao vivo de um vídeo em 4K (4 vezes a resolução de TV de alta definição - HDTV) para o Museu do Amanhã no Rio de Janeiro. Considerando que um fluxo de vídeo 4K (a 30

quadros/s) descomprimido - geralmente, a melhor opção para transmissão ao vivo - requer uma banda de 6Gbps (KIMIYAMA et al., 2015). Levando em consideração os enlaces da rede ICONE de 10 Gbps, todo o trecho entre a FUNDAJ e o PoP-PE/RNP (ITEP) na rede ICONE estaria ocupando pelo menos 60% da banda, ou bem mais, dado que há outras instituições gerando tráfego (*inbound*/*outbound*) no caminho da transmissão do vídeo 4K. Este cenário permite concluir que teríamos um congestionamento que chamaremos de **médio** (por ex., tráfego acima de 60% da banda disponível) ou até mesmo um congestionamento **pesado** (ex. tráfego acima de 80% da banda).

Figura 10 – Mapa da Rede ICONE



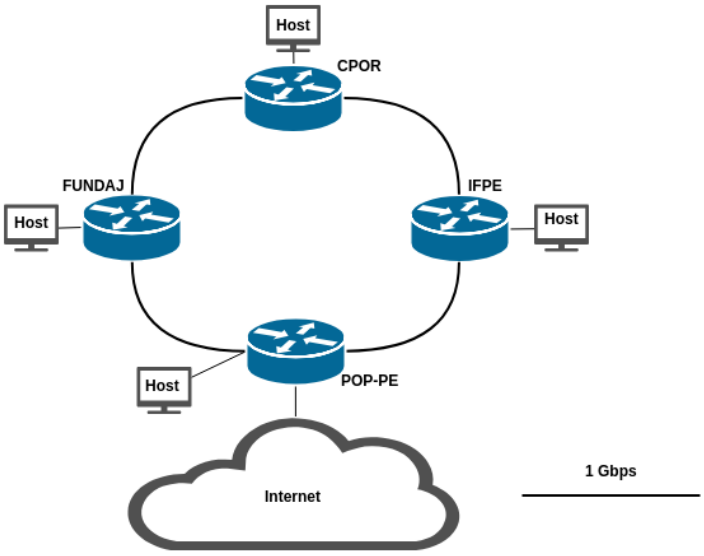
Fonte: PoP-PE/RNP.

3.2 EMULAÇÃO

Emulação, em um contexto de *software*, refere-se ao uso de uma aplicação ou ferramenta para imitar o comportamento de outro software ou dispositivo (*hardware*). Neste trabalho, a ferramenta *Mininet* foi usada para emular o comportamento de parte da rede ICONE para fins de gerência associada à tecnologia SDN, usando o controlador Ryu. Em um próximo passo, **experimentos controlados** serão realizados na rede ICONE propriamente dita, a fim de verificar a adequação do uso combinado de *SDN* com contexto na gerência da rede (verificação de relação causa-efeito).

Como se observa na Figura 11, apenas uma pequena parte da rede ICONE foi escolhida para ser virtualizada. Os critérios de escolha para os três nós da rede virtualizada, além do PoP-PE (Ponto de Presença da RNP em Pernambuco, também servindo de ponto de conexão da Rede ICONE com a RNP/Internet), foram:

Figura 11 – Rede ICONE Emulada.



Fonte: Elaborado pelo Autor.

1. CPOR (Centro de Preparação de Oficiais da Reserva - Exército Brasileiro): gerador de tráfego pequeno, localizado mais ao centro do anel com distâncias relativamente grandes do PoP-PE, tanto na direção leste quanto a oeste, como visto anteriormente da Figura 10 - Mapa da Rede ICONE;

2. FUNDAJ (Fundação Joaquim Nabuco/CGF - Campus Gilberto Freyre): eventual grande gerador de tráfego, localizado na direção oeste do PoP-PE;

3. IFPE (Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco): frequente grande gerador de tráfego, localizado na direção leste do PoP-PE.

3.3 ARQUITETURA

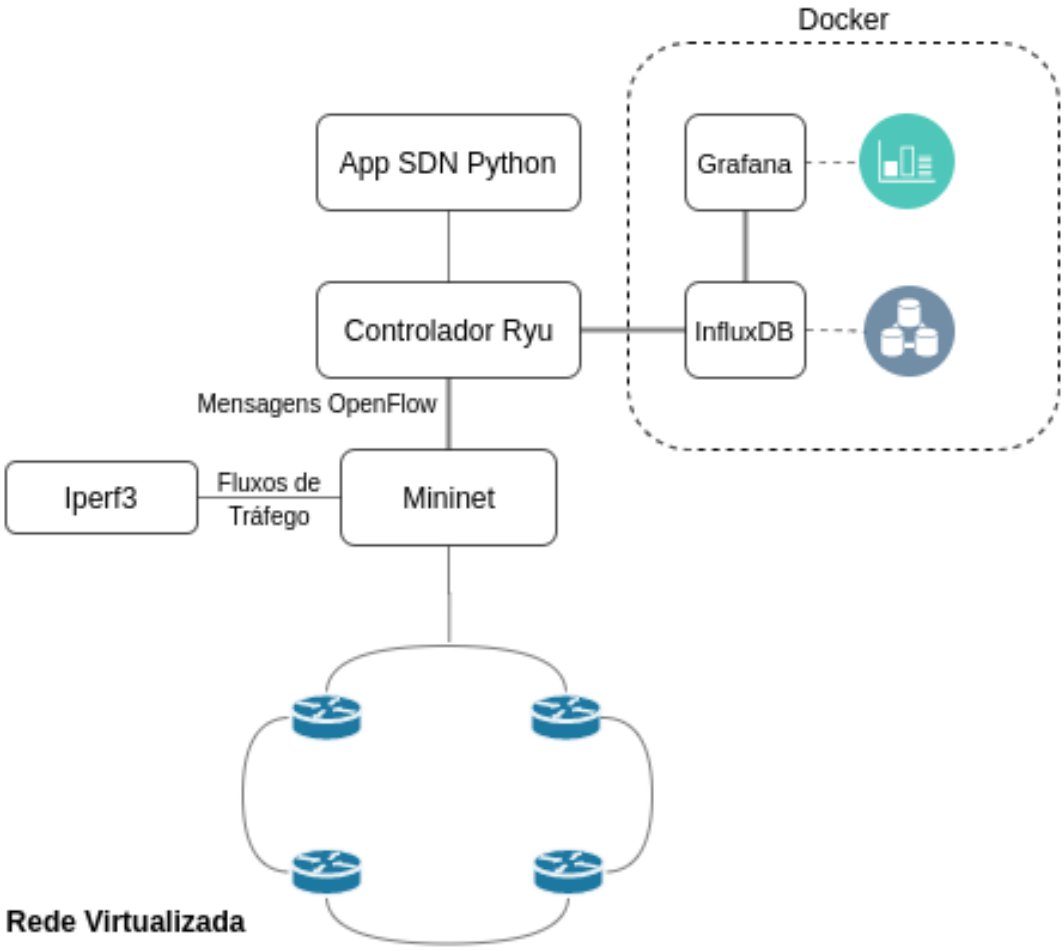
A arquitetura de comunicação da rede é mostrada da Figura 12. Inicialmente, foi criado um ambiente de emulação e análise que permite emular uma rede virtual utilizando a arquitetura SDN e o protocolo *OpenFlow* para gerenciar, programaticamente, o comportamento da rede de forma dinâmica (sensível a contexto). O *Mininet* é um dos principais emuladores de SDN que cria uma rede virtual de *hosts*, *switches*, *links* etc.

O Ryu é um *framework* SDN que facilita a criação de aplicações de gerenciamento e controle de rede - já destacado na subseção 2.3.1.1. Neste trabalho o Ryu se comunica com o *Mininet* usando o protocolo *OpenFlow* através da SBI. Uma Aplicação de Gerência de Rede Sensível a Contexto, implementada em *python*, recebe, através da NBI do controlador Ryu, dados relativos ao tráfego da rede virtualizada. Após receber os dados da rede virtualizada, infere-se o contexto (ex. tráfego fortemente congestionado no *link* IFPE $\rightarrow$ POP-PE) e se adapta a tal contexto (ex. direciona o tráfego para um *link* al-

ternativo), permitindo a adaptação da gerência de desempenho da rede. Desta forma, contribuindo para reduzir a intervenção humana nas atividades de gerência de redes.

O teste de desempenho da rede requer a geração de tráfego a várias taxas e a medição de taxa de transferência (*throughput* ou vazão), perda, latência, *jitter* e assim por diante. O *IPerf3*[1] foi a ferramenta escolhida para gerar tráfego e realizar medições de *throughput* de rede nos experimentos do projeto. Para fins de armazenamento e apresentação das medições, foi feito o uso do *InfluxDB*[2] e do *Grafana*[3] como contêiners dentro do *Docker*[4].

Figura 12 – Arquitetura do Sistema Emulado



Fonte: Elaborado pelo Autor.

3.4 SITUAÇÕES DE CONTEXTO

Para testar o uso combinado de SDN com sensibilidade a contexto na gerência da rede ICONE, foram definidas quatro situações contextuais baseadas em ocorrências comuns de rede, como os descritos adiante.

[1] iPerf - The TCP, UDP and SCTP network bandwidth measurement tool. http://iperf.fr/.
[2] InfluxDB - Purpose-Built Open Source Time Series Database. https://www.influxdata.com
[3] Grafana - The open observability platform | Grafana Labs. https://grafana.com
[4] Docker - Enterprise Container Platform. https://www.docker.com

1. Situação de **tráfego normal**;
2. Situação de **queda de enlace**;
3. Situação de **congestionamento pesado**;
4. Situação de **congestionamento médio**.

3.4.1 Tráfego Normal

Neste caso, temos uma situação, chamada **estado de observação**, onde o controlador apenas observa (percebe) as condições de tráfego, capturando dados de desempenho da rede. Aqui, a regra que permite inferir um contexto de **tráfego normal** é especificada como

$$Se\ vazão \leq 60\%\ da\ banda\ passante, então\ contexto\ do\ tráfego = NORMAL \quad (3.1)$$

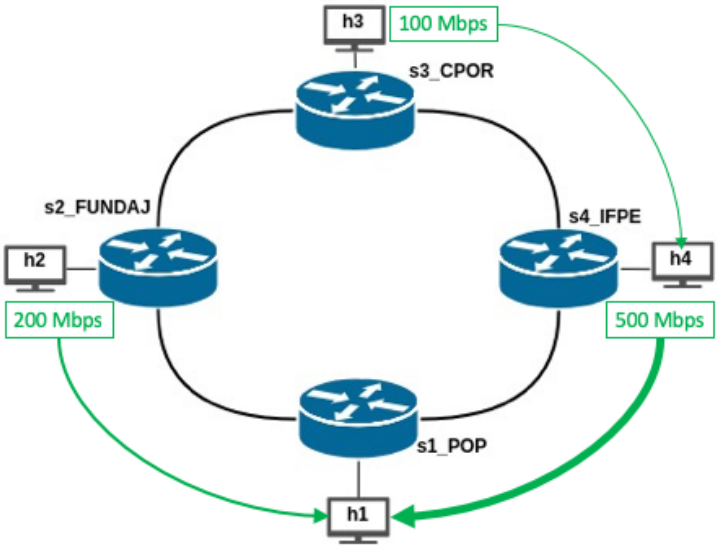
A regra (3.1) acima relaciona dois elementos contextuais (*vazão* e *banda*) e ao *inferir* que o tráfego é "normal" (neste caso, quando $vazão \leq 0,6 \times banda$, por exemplo), o sistema de gerência sensível a contexto não precisa se *adaptar*, permanecendo em **estado de observação** (Figura 13).

No cenário emulado que se verifica na Figura 13, o fluxo do tráfego na rede foi configurado para teste da seguinte forma:

- um *host* (h2) na FUNDAJ foi preparado para enviar dados no sentido FUNDAJ → POP-PE (no exemplo, a vazão de dados foi de 200 Mbps);
- um *host* (h3) no CPOR foi preparado para enviar dados no sentido CPOR → IFPE → POP-PE (no exemplo, a vazão de saída foi de 100 Mbps);
- um *host* (h4) no IFPE foi preparado para enviar dados no sentido IFPE → POP-PE (no exemplo, a vazão de dados saindo de h4 foi de 500 Mbps).

Na rede ICONE, atualmente, cada *link* tem uma banda passante de 1 Gbps (1000 Mbps). Na Figura 13 todos os *links* onde há tráfego atendem à regra de **tráfego normal** ($vazão \leq 60\% da banda, ou seja, 600\ Mbps$) – no máximo, o *link* IFPE → POP-PE chega ao limite de 600 Mbps ($vazão \leq 0,6 \times 1000\ Mbps$), agregando-se 100 Mbps que chegam do CPOR (h3) aos 500 Mbps que saem do h4-IFPE Portanto, em respeito à *regra de inferência de contexto* estabelecida (3.1), conclui-se que neste caso o tráfego é **normal**.

Figura 13 – Exemplo de cenário de tráfego normal.

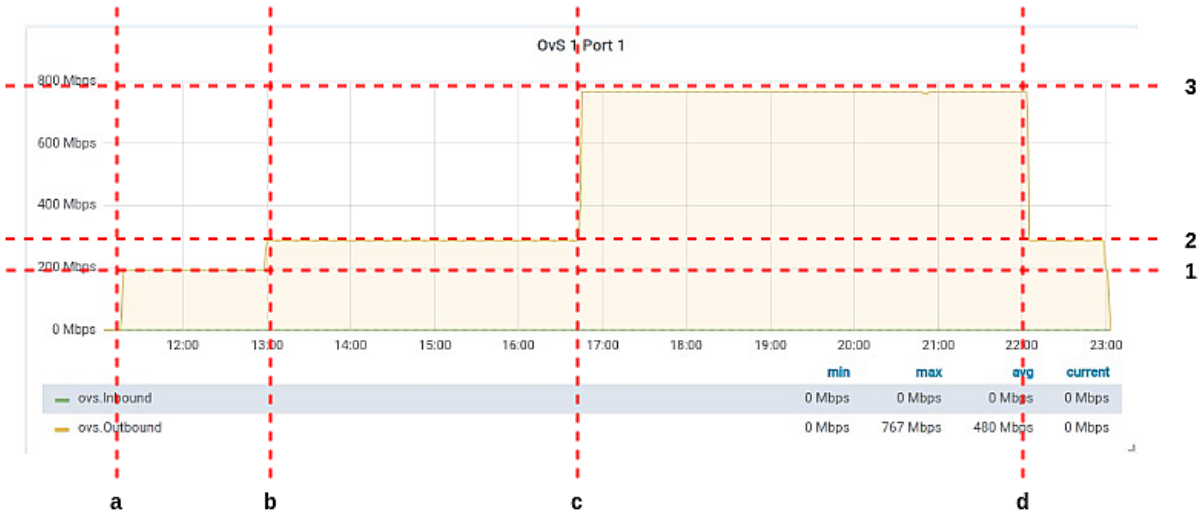


Fonte: Elaborado Pelo Autor.

No cenário de observação de tráfego emulado, o gráfico da Figura 14 (onde letras são instantes de tempo e números são patamares de tráfego) mostra o seguinte:

- do instante (a) até (b): tráfego de 200 Mbps (patamar 1) representando o fluxo entre a FUNDAJ e o POP-PE, conforme Figura 13 anterior, e que sai deste último (*Outbound*);
- do instante (b) até (c): tráfego agregado no patamar de 300 Mbps (2) saindo do POP-PE (*Outbound*), explicado pela agregação de 100 Mbps que passaram a fluir a partir do instante (b) entre o CPOR e o POP-PE (rota CPOR → IFPE → POP-PE) com a continuidade do tráfego de 200 Mbps entre a FUNDAJ e o POP-PE – até o instante (c), o IFPE ainda não havia iniciado a geração de tráfego próprio (*host* h4);
- do instante (c) até (d): tráfego agregado no patamar de 800 Mbps (3) saindo do POP-PE (*Outbound*), verificando-se a continuidade de 200 Mbps passando pela rota FUNDAJ → POP-PE e de 100 Mbps passando pela rota CPOR → IFPE → POP-PE, mais 500 Mbps incrementando o tráfego na rota IFPE → POP-PE a partir do instante (c).

Figura 14 – Tráfego de saída para a Internet agregado no POP-PE ao longo do período de observação.



Fonte: Elaborado pelo Autor.

Ao final, a observação com duração de cerca de **12 horas** do tráfego emulado na rede virtualizada de parte da ICONE mostra um aumento de tráfego agregado de 200 Mbps para 800 Mbps na interface de saída do POP-PE (*Outbound*) no período de (a) até (d).

3.4.2 Queda de Enlace

Neste contexto, **queda de enlace** (*link down*) refere-se à situação onde é capturado o evento no momento em que o enlace entre dois *switches* fica indisponível (*down*). A regra que permite a *inferência* do contexto como *link down*, e consequente adaptação, é estabelecida como:

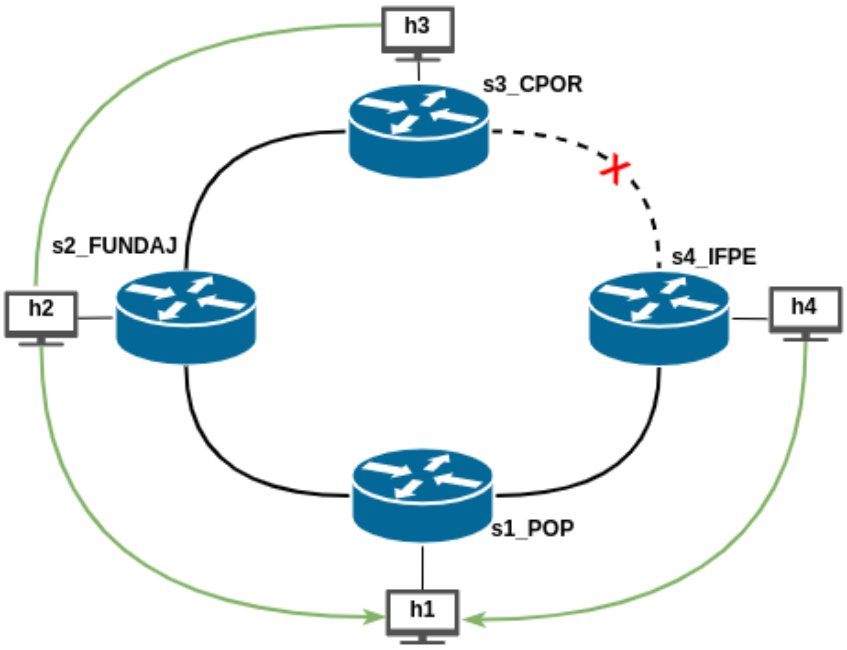
$$Se\ Estado\ do\ Link = Link\_Down, então\ muda\ o\ fluxo \tag{3.2}$$

Neste caso, a regra que permite inferir a **queda de enlace** (3.2) considera o estado do link como elemento contextual. Inicialmente, foram seguidos os seguintes passos:

- um host (h3) no CPOR foi preparado para enviar dados no sentido CPOR → IFPE → POP-PE (vazão de dados igual a 200 Mbps);
- um host (h4) no IFPE foi preparado para enviar dados no sentido IFPE → POP-PE (vazão de saída foi de 400 Mbps).

Nesta condição, há necessidade de adaptação ao contexto estabelecido. O emulador *Mininet* permite que qualquer enlace da rede emulada seja derrubado a critério do pesquisador. Esta ação foi elaborada como mostra a Figura 15.

Figura 15 – Exemplo de Queda de Enlace.

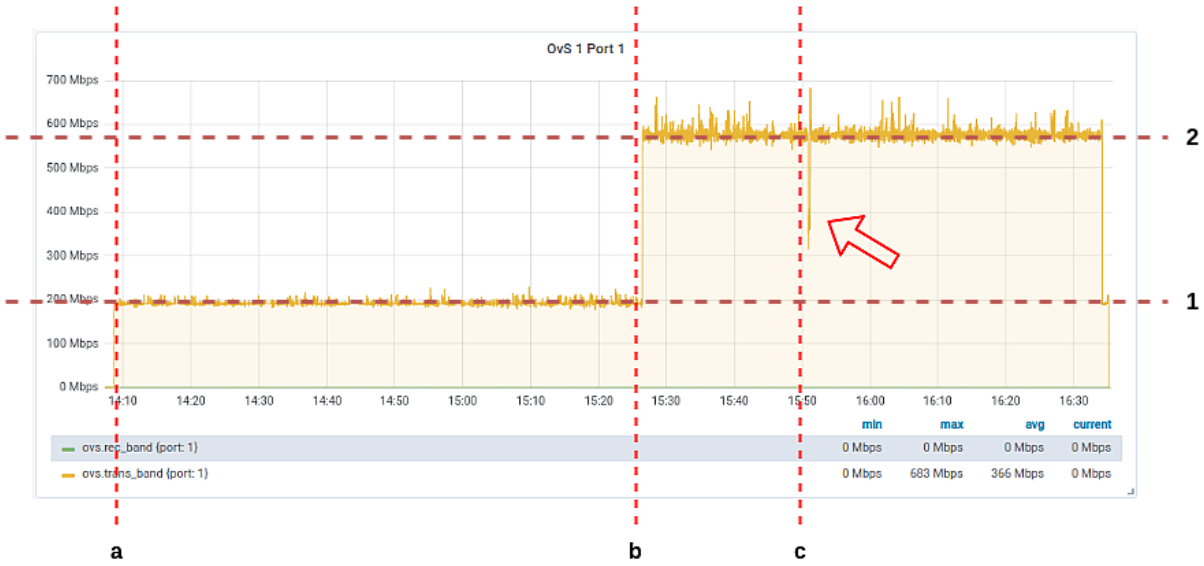


Fonte: Elaborado pelo Autor.

O gráfico da Figura 16 mostra o fluxo agregado na porta de saída do PoP-PE obedecendo os seguintes passos:

- do instante (a) até (b): foi gerado um tráfego de 200 Mbps (patamar 1) fluindo entre o CPOR → IFPE → POP-PE;
- do instante (b) até (c): tráfego agregado (patamar 2) de vazão ≈ 600 Mbps sendo transmitido pelo IFPE, explicado pela agregação de 400 Mbps que passaram a fluir a partir do instante (b) entre o IFPE e o POP-PE (rota IFPE → POP-PE) com a continuidade do tráfego de 200 Mbps entre o CPOR e o POP-PE;
- a partir do instante (c): momento da queda do enlace CPOR → IFPE, que pode ser observada pela pequena baixa de tráfego no gráfico da Figura 16 (seta vermelha), mostrando a continuidade de 600 Mbps passando pela porta de saída do POP-PE (Figura 16 instante (c)); observa-se que a nova rota do CPOR passa a ser CPOR → FUNDAJ → POP-PE (`OvS_1 port 2` na Figura 17) com a transmissão contínua do CPOR a 200 Mbps, enquanto que os 400 Mbps provenientes do IFPE continuam chegando (*Inbound*) pela rota IFPE → POP-PE (`OvS_1 port 3`).

Figura 16 – Mudanças de tráfego *Outbound* em função de queda de enlace.



Fonte: Elaborado pelo Autor.

Figura 17 – Mudanças de tráfego *Inbound* em função de queda de enlace.

Fonte: Elaborado pelo Autor.

Vale enfatizar que no gráfico (Fig. 16) que mostra a interface `OvS 1 Port 1` (*Outbound*) é possível observar (destaque com seta vermelha) uma rápida e pequena baixa na saída de dados no instante (`c`) e logo em seguida o tráfego volta ao normal, após a mudança de rota do CPOR via FUNDAJ devido a mudança de rota das entradas de fluxos.

3.4.3 Congetionamento Pesado

Nesta condição, também chamado de **estado de bloqueio**, o tráfego praticamente não flui por estar muito congestionado, levando à necessidade de mudança de rota. A regra definida para inferir este contexto é:

$$Se\ vazão \geq 80\%\ da\ banda\ passante, então\ contexto\ do\ tráfego = PESADO \quad (3.3)$$

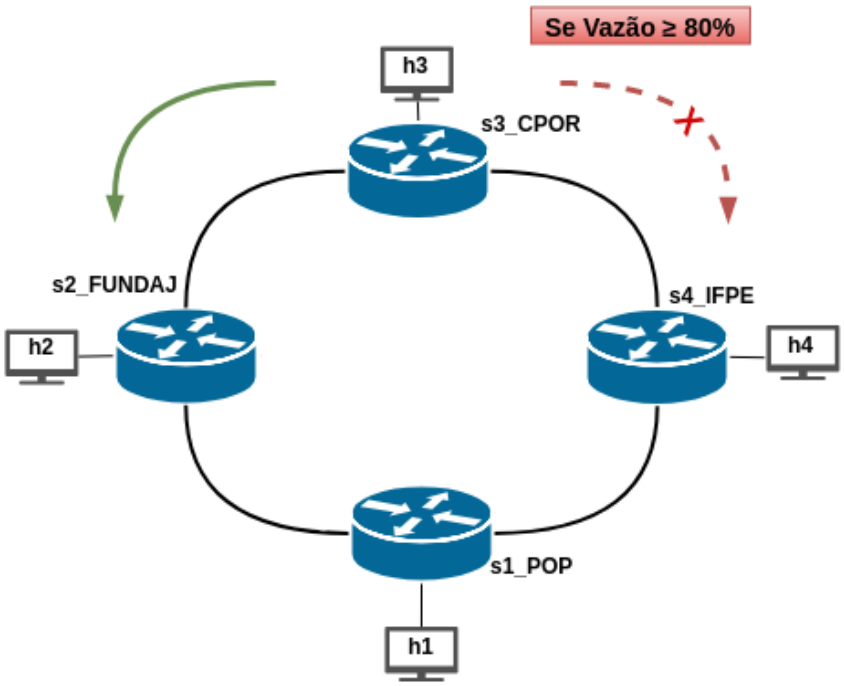
Aqui a regra de inferência tem sua análise contextual baseada nas informações estatísticas da vazão e banda. Nesta situação, infere-se que o tráfego está muito congestionado quando a vazão $\geq$ 0,8 $\times$ banda (ex. se a banda disponível em determinado enlace for igual a 1 Gbps, o congestionamento é considerado **pesado** se a vazão neste enlace for maior ou igual a 800 Mbps).

Assim, para criar a situação de congestionamento da Figura 18, algumas etapas foram cumpridas da seguinte forma:

- um *host* (h2) na FUNDAJ foi preparado para enviar dados no sentido FUNDAJ $\rightarrow$ POP-PE (vazão de dados igual a 50 Mbps);
- um *host* (h3) no CPOR foi preparado para enviar dados no sentido CPOR $\rightarrow$ IFPE $\rightarrow$ POP-PE (vazão de saída $\approx$ 350 Mbps);
- em um segundo momento, o *host* h3 aumenta a vazão de envio de dados para 800 Mbps no sentido CPOR $\rightarrow$ IFPE $\rightarrow$ POP-PE fazendo com que rapidamente seja detectado o congestionamento (vazão de 800 Mbps), mudando o sentido da vazão dos dados para CPOR $\rightarrow$ FUNDAJ $\rightarrow$ POP-PE;

Cabe esclarecer que, neste exemplo, em caso de detecção de **congestionamento pesado** em ambos os lados, ou seja, tráfego(CPOR $\rightarrow$ IFPE $\rightarrow$ POP-PE) $\geq$ 80% e tráfego(CPOR $\rightarrow$ FUNDAJ $\rightarrow$ POP-PE) $\geq$ 80%, terá que ser realizado um balanceamento de carga nos dois enlaces, como em 3.4.4.

Figura 18 – Exemplo de Congestionamento Pesado.

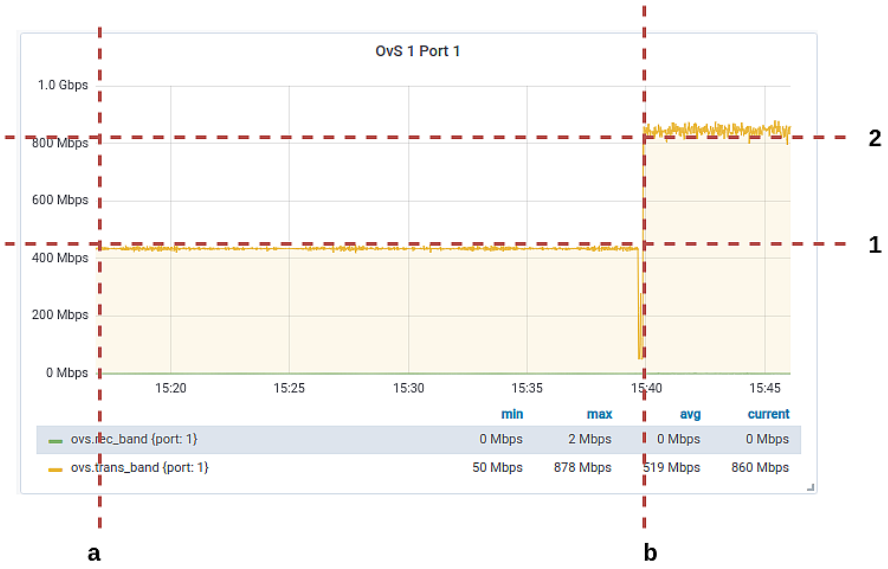


Fonte: Elaborado pelo Autor.

As Figuras 19 e 20 mostram o fluxo na porta de saída do PoP-PE e o fluxo referente aos dois lados do PoP-PE (trechos FUNDAJ ↔ POP-PE e IFPE ↔ POP-PE). Para gerar os gráficos capturados do congestionamento pesado (Figuras 19 e 20), a emulação seguiu em dois momentos

- no primeiro momento (instante (a) até (b)): tráfego de 400 Mbps atingindo o patamar 1 fluindo a partir de dois pontos FUNDAJ → POP-PE (50 Mbps) e CPOR → IFPE → POP-PE (350 Mbps);
- e no segundo momento (instante (b) em diante): o tráfego agregado (patamar 2) é aumentado em 800 Mbps transmitidos pelo CPOR, agregado ao valor de 50 Mbps da FUNDAJ, que justifica o valor final agregado em ≈ 850 Mbps do patamar 2 mostrado na Figura 19.

Figura 19 – Mudanças de tráfego *Outbound* em função do Congestionamento Pesado.



Fonte: Elaborado pelo Autor.

Figura 20 – Mudanças de tráfego *Inbound* em função do Congestionamento Pesado.

Fonte: Elaborado pelo Autor.

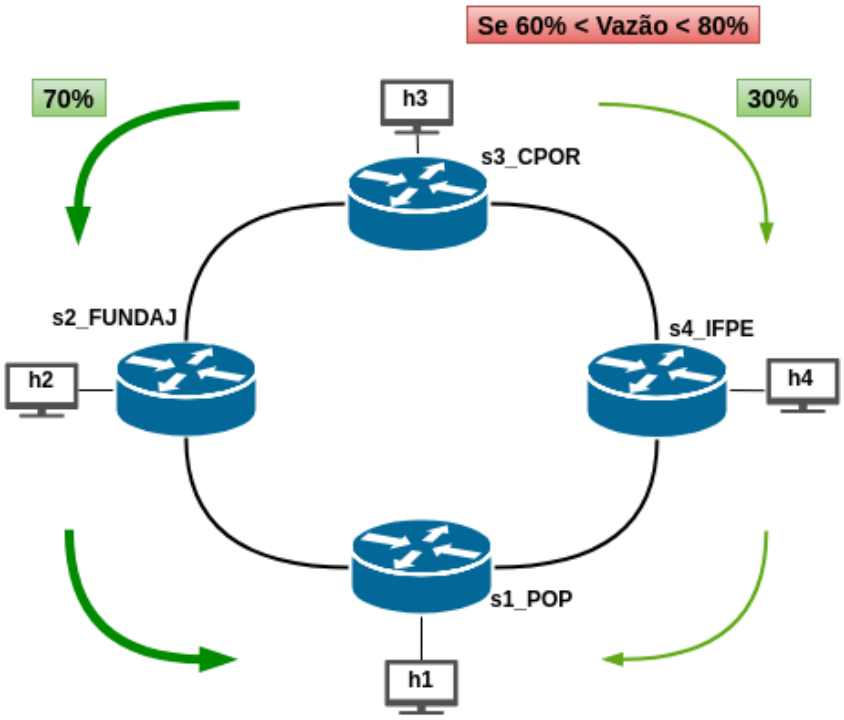
Com duração de cerca de 1h, como pode ser visto nas figuras anteriores (Figura 19 e 20) houve uma mudança de rota em função da ocupação da banda. Na Figura 20, do instante (a) ao instante (b), o tráfego segue normal sem violar a regra de **congestionamento pesado**. Do instante (b) em diante, o *host* 3 atinge o limiar de 800 Mbps e então é impedido de continuar por esta rota, explicando a mudança de tráfego *Inbound* do *OvS_1 port 3* para o *OvS_1 port 2*.

3.4.4 Congestionamento Médio ou Moderado

Este situação é chamada de **contexto de balanceamento**, ilustrada por uma situação em que o tráfego está congestionado, mas pode ser balanceado, dividindo-se (ex. 50%-50% ou 70%-30%) por enlaces distintos. A regra estabelecida para inferir esse contexto é:

$$Se\ 60\% < vazão < 80\%\ da\ banda\ passante, então\ contexto\ do\ tráfego = MODERADO \tag{3.4}$$

Figura 21 – Exemplo de Congestionamento Moderado.



Fonte: Elaborado pelo Autor.

Neste caso, é inferido que a rede está com **congestionamento moderado** quando a vazão atinge valores acima de 60% e abaixo de 80% da banda passante. Como se observa na Figura 21, a regra de balanceamento faz com que o fluxo seja dividido (neste exemplo, 70% pelo link CPOR → FUNDAJ e 30% pelo link CPOR → IFPE) de forma automática (adaptação) assim que a regra de contexto (3.4) é confirmada.

3.5 CONSIDERAÇÕES FINAIS

Neste Capítulo foram apresentadas quatro regras de contexto das quais três (Tráfego Normal, Queda de Enlace e Congestionamento Pesado) foram implementadas. A topologia foi baseada na rede ICONE parcialmente emulada. Em seguida, foram estabelecidas as regras baseada em problemas simples e corriqueiros de redes, onde a primeira regra foi definida para um contexto de **tráfego normal**, a segunda para **queda de enlace**, a terceira regra foi criada para tratar **congestionamento pesado** e a quarta para inferir um contexto de **congestionamento médio** ou **moderado**. Como foi visto, o ambiente foi elaborado baseado na arquitetura SDN, onde foram definidos três *switches* mais o do PoP-PE como saída para a Internet com a utilização de um controlador SDN. Também foram discutidos os passos para testar as regras definidas com o uso combinado de SDN e sensibilidade a contexto.

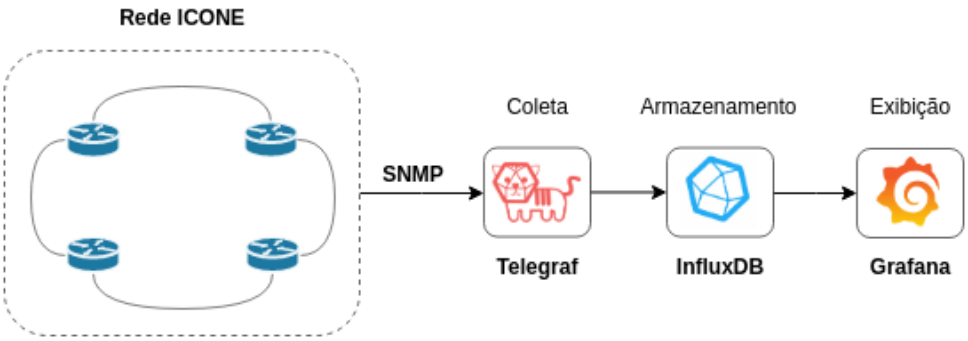
4 EXPERIMENTAÇÃO

Este Capítulo será dedicado à descrição das etapas envolvidas na experimentação deste trabalho. Os detalhes a serem abordados incluem toda a implementação do ambiente virtual, configuração das ferramentas utilizadas para complementar esta pesquisa (Grafana e o *InfluxDB*), codificação e a avaliação estatística dos resultados. Na Seção 4.1 é detalhado todo o ambiente virtual utilizado neste experimento que envolve toda a preparação das ferramentas envolvidas e codificação. A Seção 4.2 explica como foi elaborado o plano de experimentação e como foi estabelecida uma prova de conceito, além de apresentar quais métricas foram adotadas para a realização das análises estatísticas. Na Seção 4.3 são apresentados os resultados juntamente com a análise estatística e, por fim, na Seção 4.4 são mencionadas as considerações finais.

4.1 AMBIENTE VIRTUAL

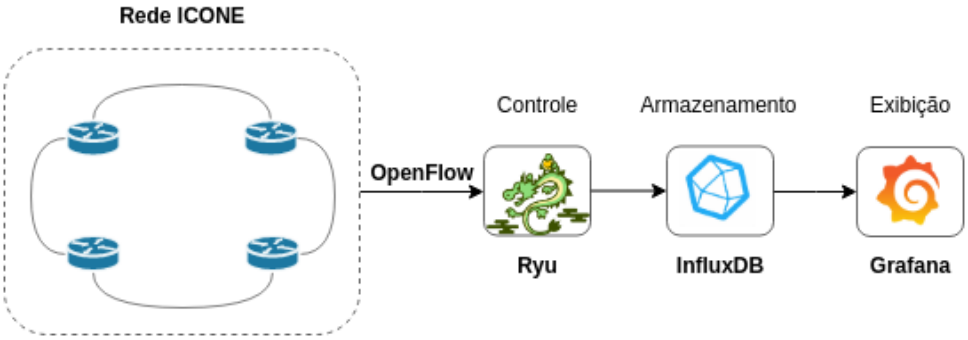
Para esta pesquisa de gerenciamento adaptativo baseado em contexto e SDN, foi decidido basear o ambiente virtual no mesmo modelo de gerenciamento do PoP-PE/RNP. A Rede ICONE realiza o gerenciamento utilizando o protocolo SNMP de forma tradicional. Para realizar esta tarefa, é feito o uso de uma ferramenta (*Telegraf*) para a coleta de dados da Rede ICONE. Com o objetivo de remodelar a arquitetura tradicional de gerenciamento da Rede ICONE para uma arquitetura SDN, o padrão do gerenciamento foi mantido, substituindo apenas a coleta dos dados do *Telegraf* (SNMP) pelo controlador SDN Ryu (*OpenFlow*) como mostrado nas Figuras 22 e 23.

Figura 22 – Gerenciamento Tradicional do PoP-PE/RNP.



Fonte: Elaborado pelo Autor.

Figura 23 – Gerenciamento Adaptativo.



Fonte: Elaborado pelo Autor.

Deste modo, o ambiente virtual foi criado baseado em máquinas virtuais com funções separadas, com o objetivo de emular a situação real de interação do controlador SDN com a camada de monitoramento (*Grafana* e o *InfluxDB*) dentro do *data center* do PoP-PE. Essas ferramentas foram escolhidas por se tratarem das mesmas ferramentas utilizadas pelo PoP-PE para o monitoramento da rede ICONE.

4.1.1 Oracle Virtualbox

Para a realização deste experimento, a infraestrutura foi criada em um ambiente virtual com a utilização de máquinas virtuais. A base deste experimento se reflete no estudo da rede metropolitana Rede ICONE, mantida e gerenciada pelo ponto de presença da RNP em Pernambuco (PoP-PE/RNP), localizado no Instituto de Tecnologia de Pernambuco (ITEP). Os ativos que gerenciam e coletam informações da base de informações de gerenciamento (MIB), através do protocolo SNMP, estão localizados no *data center* do PoP-PE. Deste modo, a organização do ambiente virtual segue o princípio de virtualizações de funções de redes (*Network Functions Virtualization* – NFV) representando máquinas e serviços que podem ser instalados no *data center* do PoP-PE. A literatura nos mostra que há vários ambientes e *testbeds* disponíveis para experimentação de redes definidas por software (SDN) (RAZA; CHOWDHURY; ROBERTSON, 2016), (ALCORN; MELTON; CHOW, 2017), (CZIVA et al., 2016), (IBM, 2013), (SALSANO et al., 2013). Ainda que os ambientes virtuais encontrados na literatura possam ser mais restritos ou pouco documentados, eles normalmente são caracterizados por *Hypervisors* (*VMWare*, *XenServer*, *Hyper-V*) para construção desses ambientes virtuais. Soluções menores como *hypervisors* do tipo-2, como *VMware Workstation Player* e *Oracle VirtualBox*, são de fácil acesso e atendem aos requisitos desta pesquisa na criação de redes virtuais isoladas para estudos e experimentos que simulam as camadas básicas de rede. Por motivos de facilidade de uso e portabilidade, optou-se pelo *Oracle VirtualBox* para a realização deste experimento.

Com o intuito de que o ambiente virtual pudesse oferecer condições de interações próximas da realidade, foi utilizada a organização mostrada na Figura 24, ressaltando

que neste ambiente apenas o controlador SDN e o *Docker* poderiam ser replicados de fato em um *data center*, visto que o *Mininet* tem apenas a função de emular um ambiente de rede definidas por software, neste caso emulando uma pequena porção da Rede ICONE.

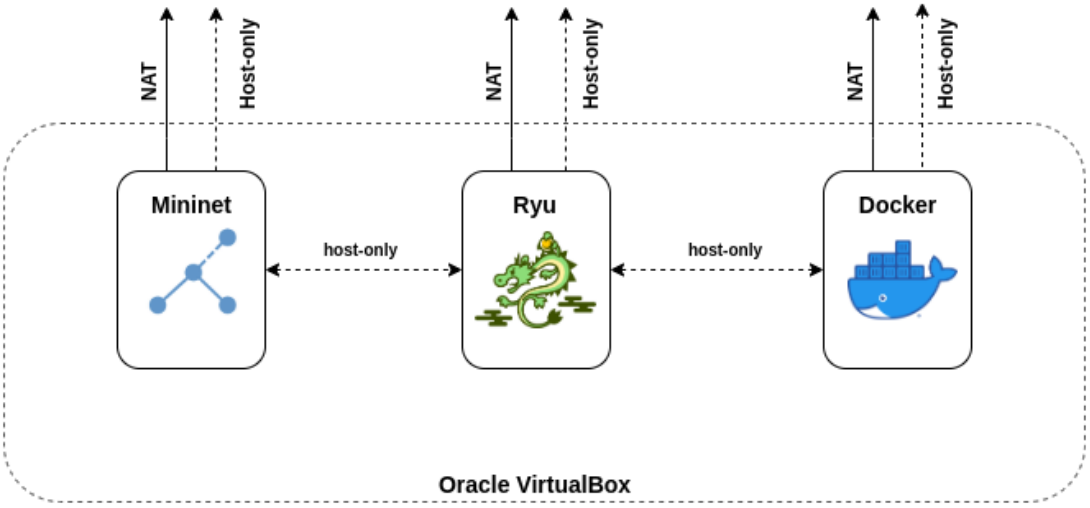
A seguir, foram criadas 3 máquinas virtuais com funções separadas, sendo elas:

- 1º Máquina Virtual - Controlador SDN Ryu
- 2º Máquina Virtual - Emulador de redes SDN *Mininet*
- 3º Máquina Virtual - *Docker*

Cada máquina virtual foi configurada com duas placas de rede como pode ser visto na Figura 24. As placas da rede virtual do *Oracle VirtualBox* foram criadas da seguinte maneira:

- Placa de rede NAT - Esta placa tem o único objetivo de acessar repositórios na internet para atualizar e instalar os pacotes e programas necessários para a execução deste experimento.
- Placa de rede *host-only* - Com o propósito de estabelecer a comunicação entre as máquinas virtuais. Este tipo de configuração também permite, através de uma interface lógica (placa de rede exclusiva), que as máquinas virtuais se comuniquem com a máquina física (hospedeiro).

Figura 24 – Rede Virtual

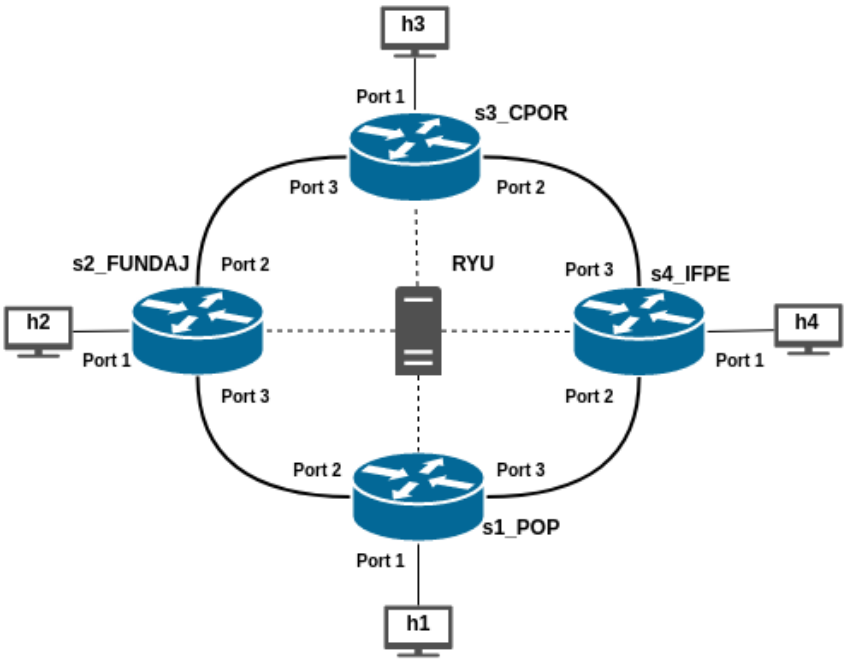


Fonte: Elaborado pelo Autor

4.1.2 Mininet

Esta Subseção explica como foi elaborado o *script* para replicar a topologia da rede ICONE. Para este trabalho foi necessário a elaboração de um *script* que emulasse condições e métricas semelhantes ao da rede ICONE. O *Mininet* permite a elaboração de topologias através da customização de *scripts* escritos em *Python*. Alguns parâmetros foram adotados para a elaboração do *script* da topologia que serão descritos adiante. A topologia detalhada da porção da Rede ICONE que foi emulada pode ser vista na Figura 25.

Figura 25 – Topologia Detalhada.



Fonte: Elaborado pelo Autor

Os *hosts* foram implementados utilizando a função ***addHost()*** dentro do *script* da topologia. Para os *hosts*, os parâmetros de `ip`, `mac`, `defaultRoute` e `cpu` foram implementados. Cada *host* conectado a um *switch* é um objeto criado recebendo os parâmetros que serão detalhados abaixo:

- **IP:** Para cada *host* foi atribuído um IP, representando o ip ou bloco ip da instituição conectada à Rede ICONE.
- **MAC:** Este parâmetro foi modificado atribuindo um endereço MAC manualmente para que o endereço MAC pudesse ser identificado com mais facilidade nas mensagens *OpenFlow*. Caso não fosse feito isso o *Mininet* atribuiriam endereços MAC aleatórios para os *hosts*.
- **DefaultRoute:** O parâmetro *defaultRoute* é utilizado para adicionar uma rota padrão, fazendo com que os *hosts* sejam alcançáveis. Este item se mostrou necessário

pois é preciso estabelecer uma rota, além de adicionar regras de fluxos nos *switches* (OvS) emulados utilizando *OpenFlow*.

- **CPU:** O parâmetro de *cpu* indica limitação no uso de processamento para cada *host* criado. Neste caso, foi configurado para o menor valor de processamento por questões de desempenho da máquina virtual do *mininet*.

Para a criação dos *switches* foi utilizada a função ***addSwitch()*** dentro do *script* da topologia. A função de criação de *switches* suporta poucos parâmetros e apenas dois (`name` e`cls`) foram suficientes para este experimento.

- **Name:** Atribui um nome de identificação ao *switch* OvS. Aqui valores dos *switches* são nomes representando a instituição pertencente.

- **CLS:** Este perâmetro especifica o constructor ou classe do *switch OpenFlow* que será usado. Neste campo foi escolhido o valor ***OVSKernelSwitch*** por ser o *switch* com desempenho melhor ao utilizar o *kernel space*.

Os *links* criados conectam todos os dispositivos virtuais da topologia. Para conectar *hosts* e *switches* foi utilizada a função ***addLink()*** dentro do `script` da topologia. Os parâmetros `port`, `bw`, `delay` e `loss` foram escolhidos para caracterizar as conexões (*script* da topologia 4.1.1).

- **Port:** O parâmetro `port1` indica o número da porta de origem e `port2` a porta de destino dos dispositivos conectados da rede emulada. Foram escolhidas portas alternadas entre 2 e 3 de modo que as portas não fossem repetidas entre enlaces. Este tipo de configuração foi escolhida por se assemelhar ao padrão adotado na rede ICONE (detalhes das distribuições das portas podem ser vistas na Figura 25). Nas conexões entre *hosts* e *switches*, a primeira porta do *host* como porta 0 e a primeira porta do *switch* foi a porta 1.

- **Bw:** O parâmetro `bw` especifica o limite de banda em Mbps. Este parâmetro fica disponível após importar o módulo *TCIntf* (*Traffic Control Interface* ou Interface de Controle de Tráfego). Neste parâmetro, o valor designado foi de 1000 representando 1Gbps de enlace da rede ICONE.

- **Delay:** Aqui o valor de `delay` indica o tempo que um bit leva de um ponto a outro no enlace. Os valores foram escolhidos após realizar uma média de 1000 pacotes ICMP entre cada dispositivo da topologia (valores detalhados podem ser vistos no código 4.1.1).

- **Loss:** Indica a porcentagem (%) de perda de pacotes que podem ocorrer entre enlaces. Após a coleta de 1000 pacotes ICMP de todos os enlaces, os valores foram zero (0) para todos os casos.

Código Da topologia do Mininet.

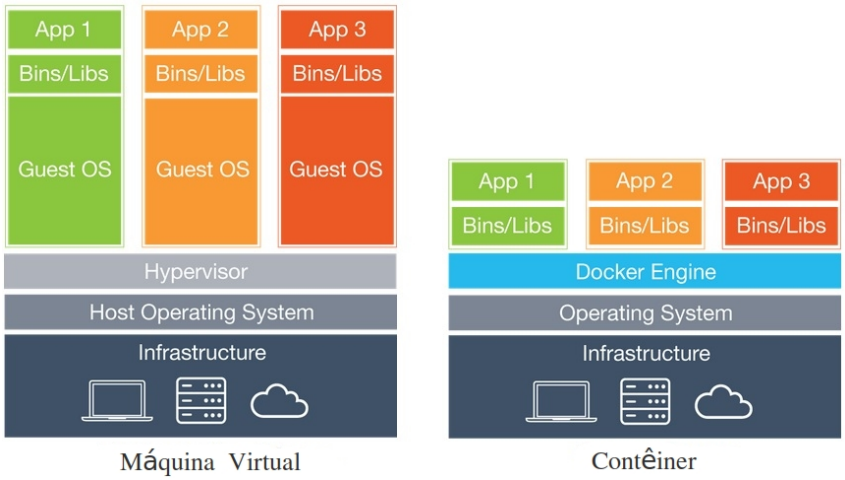
```
#HOSTS
h1=self.addHost('h1',ip='192.168.1.1/24',mac='00:00:00:00:00:01',defaultRoute='via 192.168.1.1',cpu=.1)
h2=self.addHost('h2',ip='192.168.2.2/24',mac='00:00:00:00:00:02',defaultRoute='via 192.168.2.2',cpu=.1)
h3=self.addHost('h3',ip='192.168.3.3/24',mac='00:00:00:00:00:03',defaultRoute='via 192.168.3.3',cpu=.1)
h4=self.addHost('h4',ip='192.168.4.4/24',mac='00:00:00:00:00:04',defaultRoute='via 192.168.4.4',cpu=.1)

#SWITCHES
s1 = self.addSwitch('s1_POP', cls=OVSKernelSwitch)
s2 = self.addSwitch('s2_FUNDAJ', cls=OVSKernelSwitch)
s3 = self.addSwitch('s3_CPOR', cls=OVSKernelSwitch)
s4 = self.addSwitch('s4_IFPE', cls=OVSKernelSwitch)

#LINK BETWEEN SWITCHES
self.addLink(s1, s2, port1=2, port2=3, bw=1000, delay='6.754ms', loss=0)
self.addLink(s2, s3, port1=2, port2=3, bw=1000, delay='6.214ms', loss=0)
self.addLink(s1, s4, port1=3, port2=2, bw=1000, delay='6.322ms', loss=0)
self.addLink(s3, s4, port1=2, port2=3, bw=1000, delay='6.177ms', loss=0)

#LINK BETWEEN HOSTS AND SWITCHES
self.addLink(h1, s1, port1=0, port2=1, bw=1000, delay='1ms', loss=0)
self.addLink(h2, s2, port1=0, port2=1, bw=1000, delay='0.875ms', loss=0)
self.addLink(h3, s3, port1=0, port2=1, bw=1000, delay='3.426ms', loss=0)
self.addLink(h4, s4, port1=0, port2=1, bw=1000, delay='1.168ms', loss=0)
```

Código 4.1.1 – Elaborado pelo Autor

4.1.3 Docker

Docker é um projeto *open source* que fornece uma plataforma para desenvolvimento e execução de várias aplicações. O *Docker* é um conjunto de vários conceitos e tecnologias, sendo contêineres (LXC) a principal delas, combinando uma plataforma leve de virtualização de contêineres com fluxos de trabalho e ferramentas que ajudam a gerenciar e implantar aplicativos. O *Docker* acaba sendo uma ótima solução para virtualização em nível de sistema operacional, podendo até economizar recursos que geralmente teriam alguns gigabytes a mais em uma máquina virtual tradicional como pode ser visto na Figura 26 (PREETH et al., 2016), (BOETTIGER, 2015).

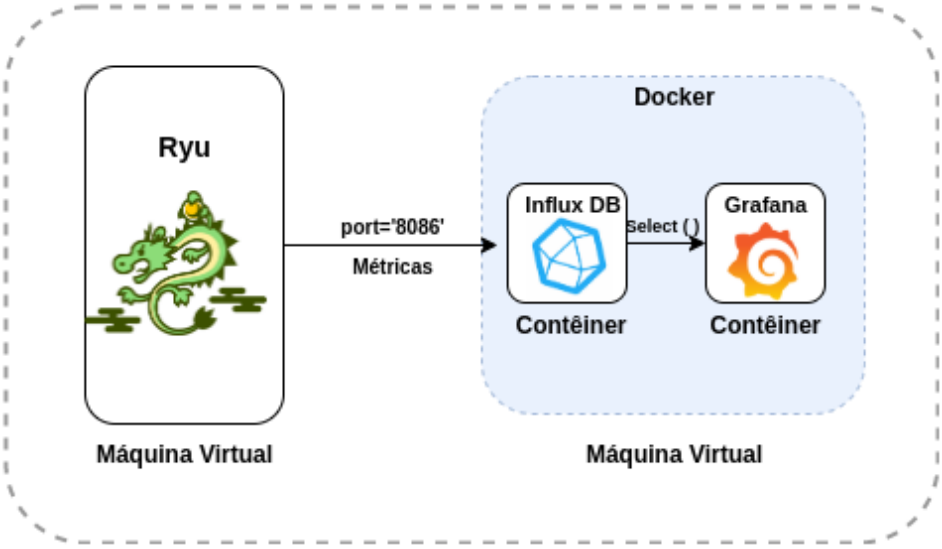
Figura 26 – Comparação de Máquinas Virtuais e Contêiner.



Fonte: Adaptado de (CONTAINER-VS-VMS, 2017)

Neste trabalho, o *Docker* foi utilizado para que o ambiente virtual se assemelhasse ao ambiente do *data center* de PoP-PE ao mesmo tempo em que economizamos recursos utilizados por máquinas virtuais. O *Docker* foi instalado em uma terceira máquina virtual separada, evidenciando a interação entre os sistemas como exibido na Figura27. O *Docker* foi instalado apenas com o pacote ***docker-ce*** (*Docker Community Edition*) que é uma versão gratuita em comparação com o ***docker-ee*** (*Docker Enterprise Edition*). Como mencionado nas seções anteriores, em um ambiente tradicional de redes, o gerenciamento é feito com a coleta da MIB dos ativos de rede. A coleta dessas informações é feita utilizando o protocolo SNMP e exibida em software de monitoramento de rede, como Cacti, Nagios, Zabbix etc, (IQBAL; PATTINSON; KOR, 2015), (LI et al., 2016), (RENITA; ELIZABETH, 2018), (SILVA; SOUZA, 2016). Para esta tarefa foram utilizados o *InfluxDB* para armazenar informações e o *Grafana* para exibição. Sendo assim, foram implementados dois contêineres *InfluxDB* e o *Grafana*.

Figura 27 – Interação: Controlador-Docker.

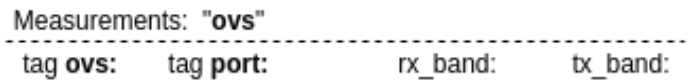


Fonte: Elaborado pelo Autor

4.1.3.1 InfluxDB

InfluxDB é um banco de dados distribuído projetado para armazenar dados de séries temporais com requisitos mínimos de hardware (1 GB, 1 núcleo). Seus recursos contínuos de consulta e retenção de dados podem executar operações nos dados sem intervenção manual. Devido ao seu design leve, foi projetado para utilização mínima de recursos sendo altamente flexível e fácil de estender a novos sensores. Considerado uma base de dados de alta performance, é capaz de lidar com elevados fluxos de dados (JOSHI; CHODISETTY; RAVEENDRAN, 2019). Neste trabalho o *InfluxDB* foi utilizado para reunir e enviar as métricas e resultados para exibição no *Grafana*. Foi necessário criar um esquema de base de dados no *InfluxDB* para que pudesse atender ao experimento. Um banco de dados de séries temporais como *InfluxDB* apresenta um conceito diferente na criação da base de dados. Diferente de um banco de dados relacional (MySQL, PostgreSQL), o *InfluxDB* apresenta o conceito de *measurement* (medidas), que é similar ao nome da tabela, *Tags*, que são semelhantes às colunas indexadas no SQL e *Fields*, que são como as colunas não indexadas no SQL. Há também o campo *timestamp*, porém seria necessário informar um valor em UTC a cada entrada e devido a isto não foi necessário utilizar esse campo. Com isso, a criação do esquema se deu por linha de comando (CLI) e seguiu o seguinte padrão: <*measurement*>, <*tag_key*>=<*tag_value*> <*tag_key*>=<*tag_value*>, <*field_key*>=<*field_value*>, <*field_key*>=<*field_value*>. Para atender às necessidades do experimento, foi necessária a criação de dois campos *tags* sendo o primeiro deles identificando o *switch OpenFlow* (ovs) e o segundo a porta (port) separados e sem vígula. Em seguida, os campos *field* rx_band (representando a banda recebida) e tx_band (representando a banda transmitida) foram criados separados com vírgula. O resultado desta criação é mostrado na Figura 28.

Figura 28 – Influx *Schema*.



Fonte: Elaborado pelo Autor

Apenas uma tabela *measurement* foi criada neste experimento. Para alimentar a tabela foi necessário criar a conexão entre o controlador SDN e o contêiner *InfluxDB* através do método ***InfluxDBClient()*** em *Python*. Este método recebe como parâmetros para conexão: nome da base de dados, endereço ip e porta de comunicação (padrão 8086). O controlador *SDN* foi programado para receber informações das portas dos *switches OpenFlow*. Para enviar essas informações coletadas do controlador *SDN* para o *InfluxDB* foi utilizado a estrutura em ***JSON*** (como mostra o código 4.1.2) para então alimentar a base de dados do *InfluxDB*.

Código 4.1.2 – JSON InfluxDB

```
events = [
          {
              "measurement": "ovs",
              "tags": {
                  "ovs": dpid,
                  "port": port_no
              },
              "fields": {
                  "rx_band": <banda_recebida>,
                  "tx_band": <banda_transmitida>
              }
          }
          ]
```

Fonte: Elaborado pelo Autor.

4.1.3.2 Grafana

Como uma ferramenta de visualização *Web*, o *Grafana* se concentra na análise e monitoramento de dados. É conhecido por não apresentar uma base de dados padrão durante a instalação, diferentemente de outras abordagens, como *Cacti* ou *Nagios*, que utilizam por padrão o *MySQL* para armazenar os dados (CHAN, 2019). O *Grafana* não possui suporte nativo ao protocolo SNMP para coletar informações MIB de dispositivos de rede, sendo necessário usar outras soluções para coletar essas informações. Sua flexibilidade de interação e configuração, não apenas com o *InfluxDB* mas com outras bases de dados, e

pelo fato de ser utilizado pelo PoP-PE/RNP, facilitaram a escolha desta ferramenta para exibição das métricas.

Neste experimento, o *Grafana* foi utilizado para estabelecer uma camada de monitoramento para exibição das métricas e observar o comportamento do controlador Ryu com a aplicação sensível a contexto. Após a instalação e integração da base de dados do *InfluxDB*, foi necessário a elaboração de *query* SQL para consulta no *InfluxDB* (como visto no código 4.1). A *query* segue o seguinte padrão: *SELECT <field_key>=<field_value> FROM <measurement_name> WHERE <conditional_expression>AND|OR <conditional_expression>GROUP BY<tag_key>*.

Para os *switches* CPOR, FUNDAJ e IFPE foram necessárias duas *queries* de consulta, representando os dois lados da topologia em anel, com exceção do PoP-PE, em que foram elaboradas três *queries*. A terceira *query* do PoP-PE, se refere à porta do tráfego de saída para exibir o fluxo agregado dos outros OvS's. O ***SELECT*** seleciona o *field key* banda recebida (rx_band) e a banda transmitida (tx_band). O *Grafana* dispõe de funções de agregação (*Aggregation*) para exibição dos valores. A função utilizada foi a *last()* que retorna o valor do *field key* associado ao *timestamp* mais recente. A função de agregação *last()* é também utilizada pelo PoP-PE/RNP em sua *query* de consulta. A cláusula ***FROM*** utiliza o valor de *measurement* para a pesquisa. A cláusula ***WHERE*** retorna os valores onde os números representam o *switch* OvS e a porta do *switch* agrupados (***GROUP BY***) pelo valor da porta. Com isso foi possível criar gráficos de utilização de banda para cada porta dos *switches* OvS.

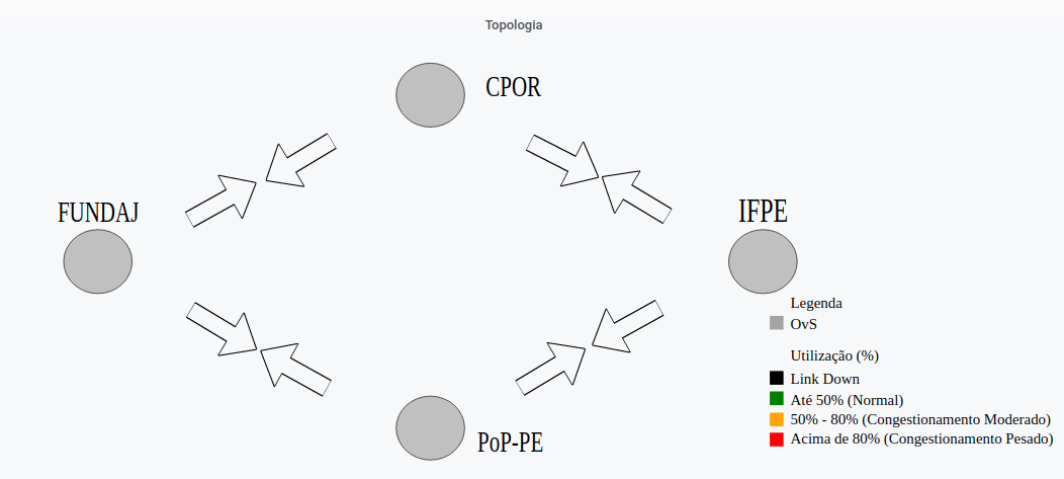
Código 4.1 – Query do InfluxDB no Grafana

```
1        SELECT
2            last("rx_band") AS "Inbound",
3            last("tx_band") AS "Outbound"
4        FROM "ovs" WHERE
5            ("ovs" = '<value>' AND "port"='<value>')
6        GROUP BY "port"
```

O *Grafana* permite a elaboração gráfica de *dashboards* e topologias através de *plugins*. Para a exibição das mudanças de comportamento de tráfego da topologia foi feito o uso do *plugin* SVG. O SVG (*Scalable Vector Graphics* ou Gráficos Vetoriais Escalonáveis) é uma linguagem que descreve gráficos 2D vetoriais em **XML**. Este tipo de gráfico permite que outras tecnologias, como *JavaScript*, manipulem o comportamento de imagens criando várias possibilidades de utilização em diversos tipos de aplicações (MORENO; OLIVEIRA, 2009). O PoP-PE/RNP utiliza o *plugin* SVG para exibição em tempo real do tráfego e *status* da Rede ICONE, sendo assim foi feito o uso deste *plugin*. Após a instalação foi criada uma imagem que representasse a rede emulada com a utilização de um editor que suportasse imagem SVG (Figura 29). O *plugin* SVG dispõe de 3 áreas para realizar a

tarefa de manipulação da imagem SVG . O *plugin* do SVG disponibiliza um editor próprio (*SVG Builder*) com algumas opções de imagens e formas geométricas próprias. É possível também importar o código de imagens criadas em outros editores de imagens que tenham suporte ao SVG. A importação se trata em copiar o código (XML) da imagem e colar na área *SVG Data* representado pela primeira área do *plugin*.

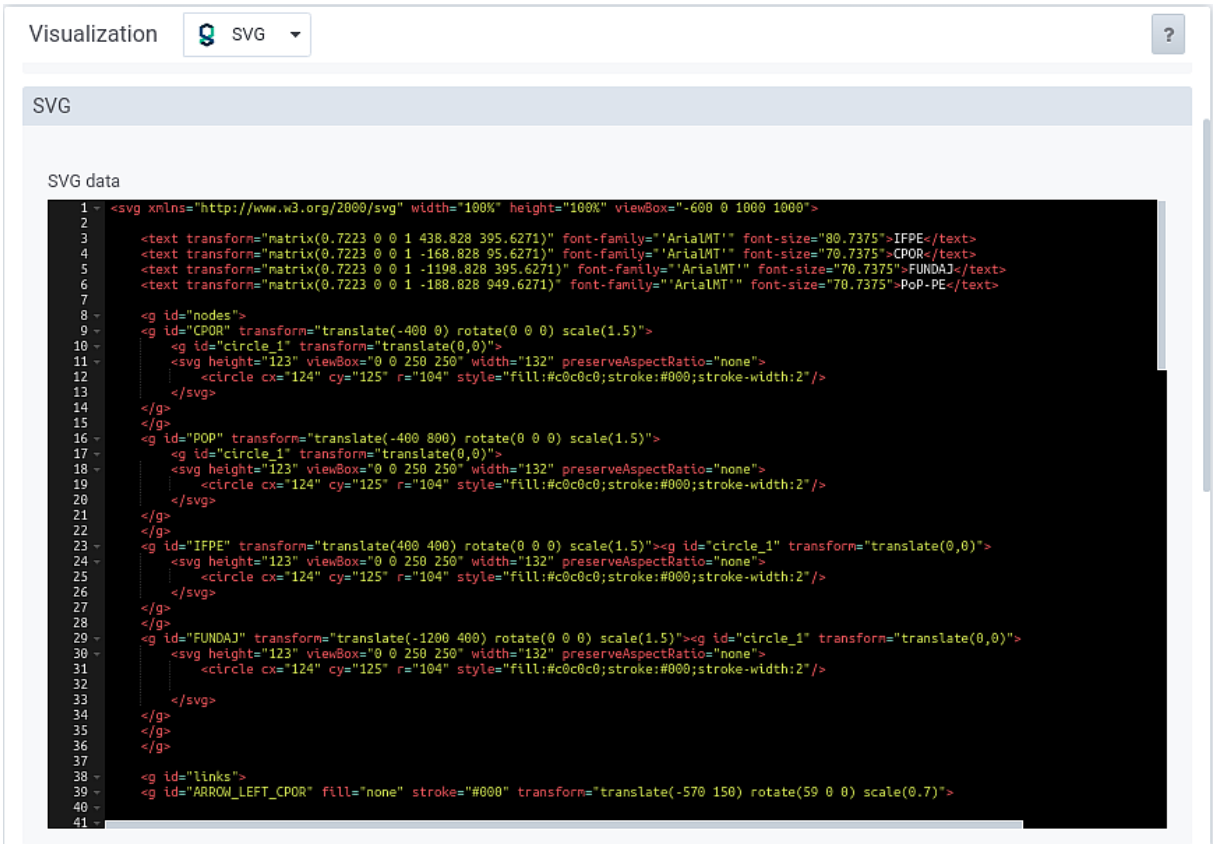
Figura 29 – Imagem SVG representando a topologia emulada.



Fonte: Elaborado pelo Autor.

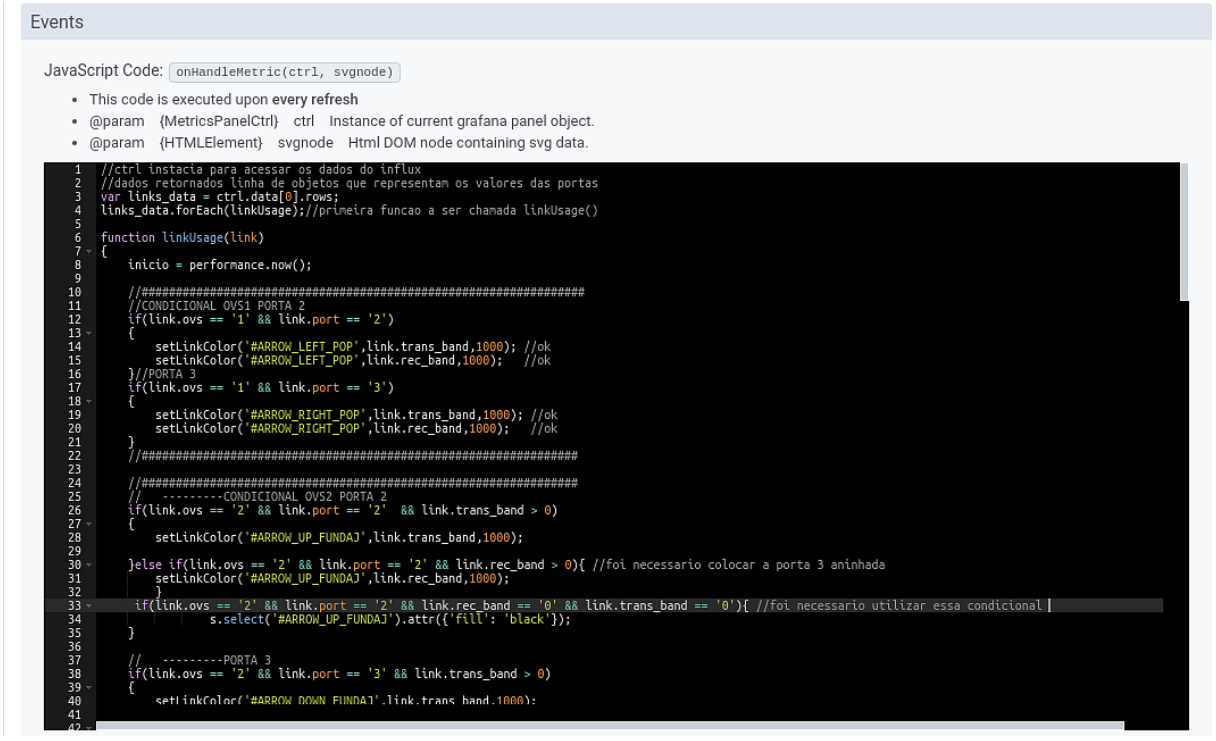
A primeira área (*SVG Data*) (Figura 30) é onde se localiza o código (XML) da topologia. Neste campo é possível manipular o posicionamento e o tamanho de cada figura geométrica da topologia. Apesar de ter sido usado o *SVG Builder* neste trabalho, o código da imagem continua disponível nesta área. As duas áreas seguintes (Figuras 31 e 32) são áreas para edição do comportamento da topologia. Nestas áreas é utilizada a linguagem *JavaScript*. Como o PoP-PE/RNP utiliza a mesma lógica para manipular comportamentos das figuras geométricas. O código SVG comentado por ser visto em Apêndice A.

Figura 30 – *Plugin* SVG primeira área.



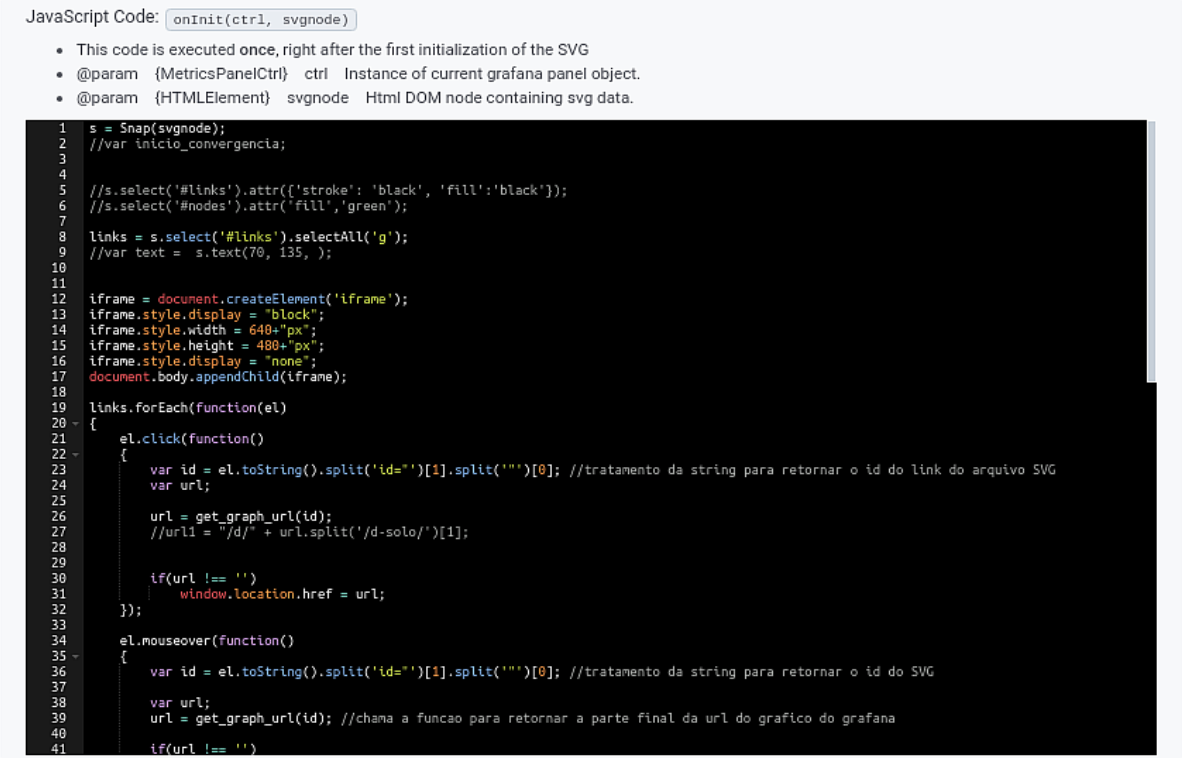
Fonte: Elaborado pelo Autor

Figura 31 – *Plugin* SVG segunda área.



Fonte: Elaborado pelo Autor

Figura 32 – *Plugin* SVG terceira área



Fonte: Elaborado pelo Autor

Para criar animações na imagem da topologia (Figura 29) foi necessária a criação de uma única *query* para que os valores retornados fossem acessados através da linguagem *JavaScript*. Seguindo a mesma lógica da consulta de cada porta do OvS (código 4.1), nesta situação é necessário que a consulta percorra todos os *switches* OvS e todas as portas. Sendo assim, a *query* para a consulta no *InfluxDB* foi elaborada como é mostrado no código 4.2. Esta consulta se assemelha à usada para as portas (código 4.1), com exceção dos valores dos *switches* e as portas na cláusula ***WHERE***. Para esta pesquisa foi importante ter estruturado a *query* com dois valores *tags* e agrupa-los (***GROUP BY***) para que posteriormente fosse possível identificar as portas por cada *switch* OvS atráves do *JavaScript*.

Código 4.2 – Query da Topologia no Grafana

```
1        SELECT
2            last("rx_band") as "rx_band",
3            last("tx_band") as "tx_band"
4        FROM "ovs" WHERE
5            ("ovs" = '1' or "ovs" = '2' or "ovs" = '3' or
6            "ovs" = '4')
7            AND ("port" = '1' or "port" = '2' or "port"='3'))
8        GROUP BY "ovs", "port"
```

Com a *query* estruturada com duas *tags*, a primeira representando o *switch* OvS e a segunda representando a porta, fica possível aplicar regras de mudanças de comportamento utilizando o *JavaScript*. As duas ultimas áreas do *plugin* SVG (como visto na Figura 31 e 32) são destinadas à programabilidade do comportamento da imagem da topologia. Com o *plugin* SVG duas ações devem ser consideradas:

1. Que todos os elementos da estrutura XML da imagem SVG sejam acessados. Para acessar os atributos de cada forma geométrica da topologia, é necessário utilizar a biblioteca Snap do *JavaScript*. O **svgnode** representa o *HTMLElement* do painel SVG. Ao utilizar o método **Snap(svgnode)** é retornado um objeto SVG com o *HTMLElement* da imagem da topologia, permitindo o acesso e a utilização do ID das figuras geométricas (GRAFANA, 2014).

2. Que todos os dados armazenados no *InfluxDB* sejam acessados. O acesso aos dados coletados pelo *InfluxDB* é feito através da instância ***ctrl*** contendo todas as métricas coletadas em forma de um *Array* de objetos (GRAFANA, 2014). Este *Array* pode ser acessado com o método **ctrl.data[0].rows** em *JavaScript*.

4.1.4 Controlador Ryu

Esta Subseção descreve como o controlador SDN foi trabalhado para o gerenciamento de rede sensível a contexto. O controlador foi instalado em uma distribuição Ubuntu seguindo a documentação oficial do Ryu. Neste experimento, o controlador SDN foi preparado para reagir a eventos de redes considerados eventos simples, como foram descritos em capítulos anteriores. Para tornar o controlador sensível a contexto, isso implica em responder a eventos de redes utilizando classes e funções definidas pela API do controlador Ryu. A documentação do controlador Ryu disponibiliza alguns exemplos de classes e funções básicas que podem ser usadas dependendo da necessidade do usuário. Essas funções retornam mensagens que são fundamentais para enviar requisições de configuração e mudança nos *switches*. Tendo em vista isso, o código do controlador foi elaborado da seguinte forma:

1. Utilizar um algoritmo para tornar todos os **nós** conectados aos *switches* alcançáveis.

2. Tratar eventos relacionados ao *status* das portas dos *switches*.

3. Tratar mensagens relacionadas a informações de estatísticas de entradas de fluxos dos *switches*.

4. Monitorar e tratar mensagens relacionadas a informações de estatísticas das portas dos *switches*.

A princípio foi utilizado um algoritmo genérico de Busca em Profundidade (*Depth-First Search* – DFS) em *Python* com a finalidade de tornar todos os **nós** conectados

aos *switches* alcançáveis, evitando *loop* de tráfego por conta da topologia em anel. Este algoritmo foi adquirido e modificado para que atendesse aos procedimentos simples desta pesquisa (SYAHIDILLAH, 2018). O controlador foi elaborado de forma **reativa**. Com isso, as entradas de fluxos (*Flow Entries*) são instaladas à medida que os pacotes vão sendo enviados para o controlador, distribuídos e instalados nos *switches*. A escolha de outros algoritmos não influencia neste experimento contanto que não resulte em conflitos com as regras a serem estabelecidas posteriormente.

Para tratar dos eventos referentes a **queda de enlace**, a classe que foi utilizada para capturar mensagens de mudança de porta foi **ryu.controller.dpset.EventPortModify**. Esta classe é definida pela API do controlador Ryu e ao ser utilizada como *decorator*, automaticamente ativa a função responsável para tratar da mudança de *status* de porta. A função *port_mod* foi criada para tratar as mensagens deste evento (**ev.msg**) da qual contém duas informações (vide também o Quadro 3).

- A primeira informação é um objeto da classe **ryu.controller.controller.Datapath** contendo informações sobre o *switch* (*Datapath).*
- A segunda informação é o número da porta afetada pelo evento.

Quadro 3 – Classe EventPortModify(Datapath, Porta).

Atributo	Descrição
Datapath	Instância da classe **ryu.controller.controller.Datapath**
Porta	Número da porta

Fonte: Adaptado de (RYU, 2014).

As informações de atributo do *Datapath* (exibidas no quadro 3) são necessárias para enviar as requisições aos *switches*. Adquirindo essas informações, elas são passadas adiante utilizando a função **`send_flow_stats_request`**. Esta função utiliza a classe **`ofp_event.OFPFlowStatsRequest`** e foi criada para realizar *requests* das entradas de fluxos dos *switches* utilizando o método **`datapath.send_msg()`**. Esse *request* gera um evento de *reply* (resposta) que é capturado pela classe **`ofp_event.EventOFPFlowStatsReply`**.

Finalmente são retornadas as informações de todas as linhas de fluxos dos *switches* afetados pelo evento. Foi criado uma função para remover e criar as linhas de fluxos nos *switches* de forma dinâmica e automática para evitar que esta tarefa fosse feita manualmente. A estrutura da mensagem da estatística de linhas de fluxos, retornada pela classe

`ofp_event.EventOFPFlowStatsReply`, pode ser vista no quadro 4.

Quadro 4 – Campos de Estatística de Entradas de Fluxos.

Atributo	Descrição
table_id	Indica o ID da tabela na próxima linha de processamento de pacotes.
duration_sec	O tempo em que o fluxo está ativo em segundos.
duration_nsec	O tempo em que o fluxo está ativo em nanosegundos.
priority	prioridade da linha de fluxo.
idle_timeout	Número de segundos inativos antes da expiração.
hard_timeout	Número de segundos antes da expiração.
flags	*Bitmap* que descrevem uma característica do OFPFF_*
importance	Precedência de despejo de fluxo (*OpenFlow*1.4).
cookie	Identificador emitido pelo controlador opaco.
packet_count	Contador de pacotes na linha de fluxo.
byte_count	Contador de *bytes* na linha de fluxo.
match	Campos a comparar.
instructions	Lista de instruções do OFPInstruction.

Fonte: Adaptado de (RYU, 2014).

Os campos que foram utilizados para alterar as linhas de fluxos foram o campo *match* e o campo *instructions*. Na Figura 33 são ilustrados os processos que envolvem o contexto de **queda de enlace**, onde é possível visualizar as etapas responsáveis por tratar a situação de queda de enlace. A situação de queda de enlace, exibida na Figura 33, é ativada através de um evento (círculo azul), em seguida, várias funções para o tratamento deste incidente são chamadas.

A função **`EventPortModify`** captura o evento de modificação de *status* de porta onde é retirada a informação de porta e OvS afetado pelo evento. A próxima função **`send_flow_stats_request`**, realiza uma requisição das entradas de fluxo afetada pelo evento. A função **`send_flow_stats_request`** executa o método **`datapath.send_msg(OFPFlowStatsRequest)`** enviando a requisição para o OvS. Esta ação, gera um evento de resposta que é capturado

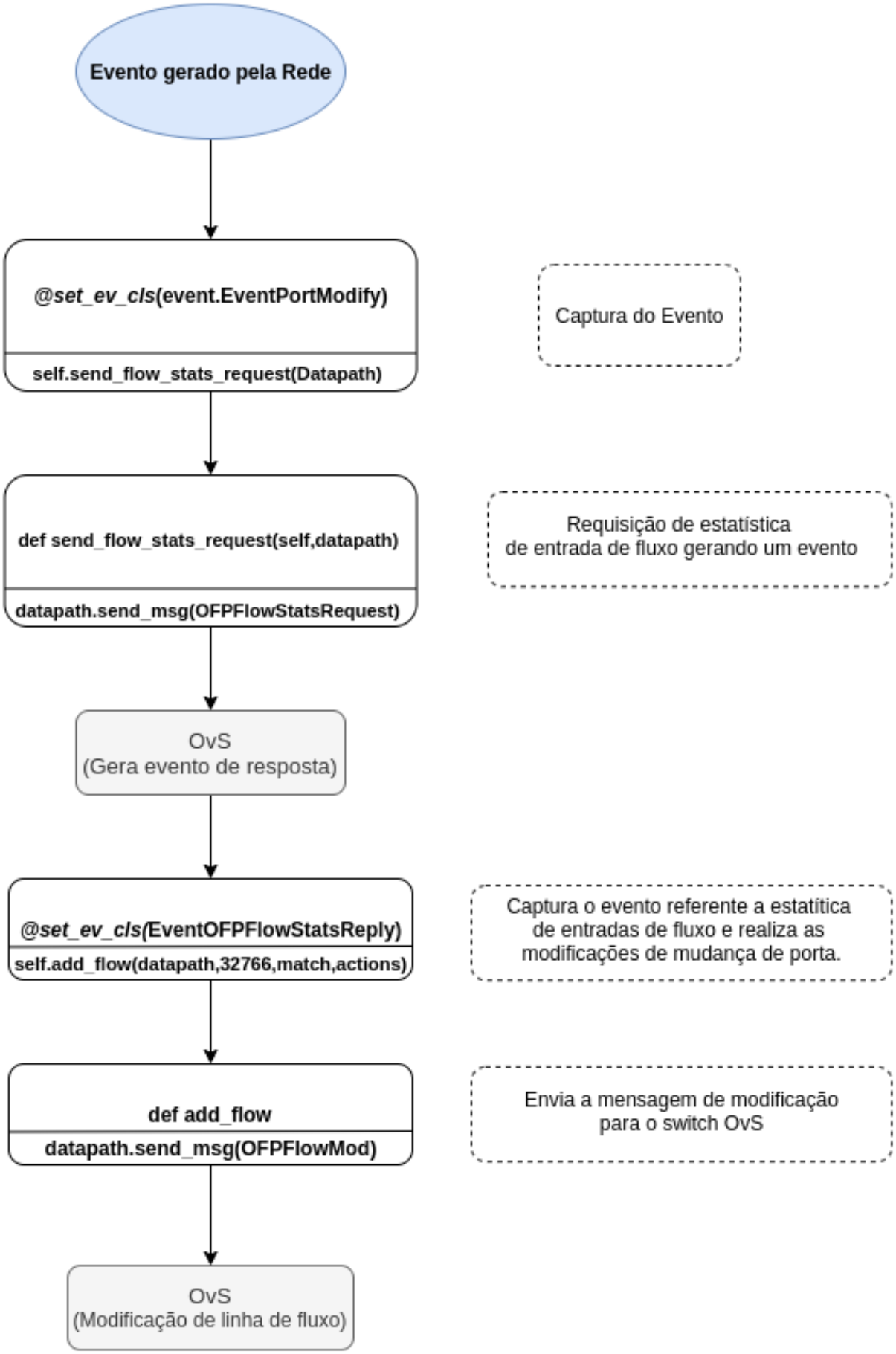
pelo *decorator* **@set_ev_cls(EventOFPFlowStatsReply**) onde são realizadas as modificações de troca de portas. A modificação se trata apenas de trocar os valores das portas nas entradas de fluxos nos OvS's.

Por fim a função **add_flow** é chamada para enviar as modificações ao OvS. Por conveniência, também foi feito o uso do módulo *Topology Discovery* que facilitam detecção de mudanças de topologia na rede. Este módulo importa métodos e funções que podem detectar novos *switches* ou *hosts* que se conectam à rede. As funções importadas pelo módulo *Topology Discovery* simplificam o mapeamento automático dos ips dos *hosts* conectados. Este módulo pôde ser importado no código do controlador através da linha *ryu.topology.api* (RYU, 2014).

Já para os processos que envolvem o contexto de **congestionamento pesado** (Figura 34), foi utilizado um procedimento de monitoramento das estatísticas de consumo das portas dos *switches*. Para monitorar as estatísticas de portas, foi feito o uso de uma função para monitorar as interfaces dos *switches* (`_monitor`), que está exemplificada na documentação do controlador Ryu ((RYU, 2014)) como *Traffic Monitor*. Esta função cria um *loop* que envia requisições periodicamente aos *switches* utilizando a função **_request_stat()**. Seguindo o modelo da documentação, foi feita a implementação desta função que gera um *loop* infinito, realizando requisições que utilizam uma lista dos *switches* detectados pelo módulo *Topology Discovery* (visto no parágrafo anterior) (TEAM, 2014).

As requisições enviadas pela função **_request_stat()** ao OvS gera um evento de resposta. Esse evento de resposta é capturado pelo *decorator* **@set_ev_cls(EventOFPPort StatsReply)**. Este *decorator* executa algumas funções e métodos para realizar os cálculos de banda. Se a condicional de congestionamento pesado for violada, será feita a mudança da entrada de fluxo para mudar a rota utilizando a função **send_flow_mod()**. Em seguida, a função **send_flow_mod()** chama chama uma última função **add_flow** para enviar as mensagens de mudança para o OvS.

Figura 33 – Diagrama de Tratamento do Contexto Queda de Enlace.



Fonte: Elaborado pelo Autor

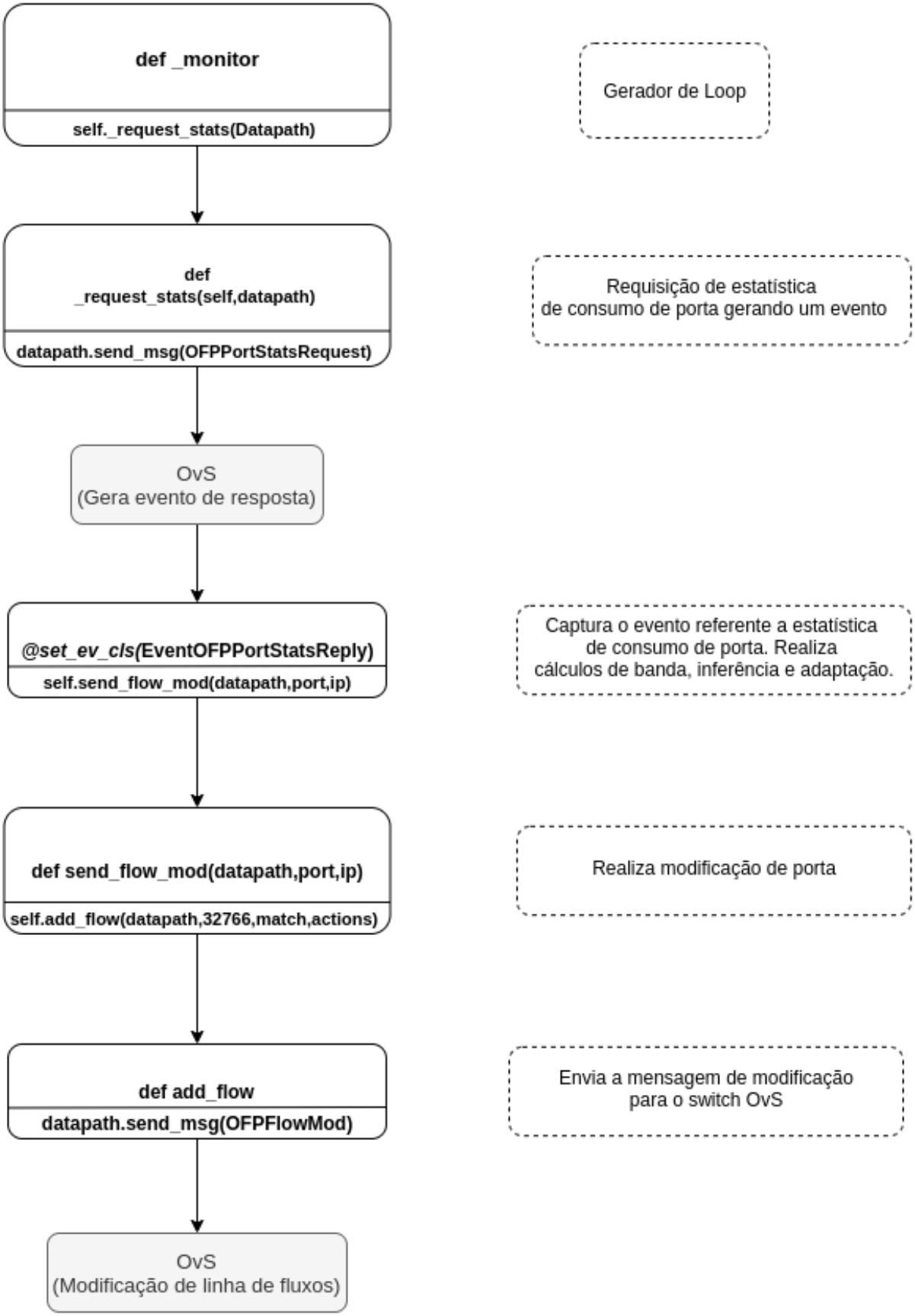
Neste experimento as requisições feitas para monitorar as estatísticas de portas foram ajustadas para a frequência de 1 segundo. A cada ciclo é feita uma requisição de estatística de consumo de porta usando a função *_request_stats*. Feita essa requisição, é gerado um evento de resposta que é capturado pela classe *EventOFPPortStatsReply* e tratado pela função *send_flow_mod*. Na Figura 34 é exibido no lado esquerdo a descrição do processo que engloba o contexto de **congestionamento pesado** e à direita as funções ou métodos que estão relacionados a cada processo do lado esquerdo. Logo abaixo, no quadro 5, é possível observar as informações da estatística de utilização de porta que são retornadas com a classe *EventOFPPortStatsReply*.

Quadro 5 – Campos de Estatística de utilização de Porta.

Atributo	Descrição
length	Comprimento da mensagem.
port_no	Número da porta.
duration_sec	Tempo em que a porta esteve ativa em segundos.
duration_nsec	Tempo em que a porta esteve ativa em nanosegundos.
rx_packets	Número de pacotes recebidos.
tx_packets	Número de pacotes transmitidos.
rx_bytes	Número de *bytes* recebidos
tx_bytes	Número de *bytes* transmitidos.
rx_dropped	Número de pacotes recebidos descartados.
tx_dropped	Número de pacotes transmitidos descartados.
rx_errors	Taxa de erros recebidos.
tx_errors	Taxa de erros transmitidos.
properties	Descrição de propriedades de portas óticas.

Fonte: Adaptado de (OPEN NETWORKING FOUNDATION, 2015).

Figura 34 – Diagrama de Tratamento do Contexto Congestionamento Pesado.



Fonte: Elaborado pelo Autor.

4.2 PLANO DE EXPERIMENTAÇÃO

Esta Seção trata do planejamento feito para emular parte de Rede ICONE utilizando o *Mininet* e o controlador Ryu executando uma aplicação sensível a contexto. O experimento deste trabalho consistiu em gerar tráfego com a presença de um controlador SDN preparado para reagir de forma adaptativa a eventos simples de rede. Conforme o tráfego fosse gerado, alguns passos foram seguidos para a realização da sensibilidade a contexto (captura de dados, inferência de contexto e adaptação) nos casos onde há necessidade de adaptação.

Na situação de **queda de enlace**, significando tráfego **impedido**, onde a adaptação se faz por um desvio de rota:

- Foram gerados tráfegos utilizando os protocolos TCP e UDP.
- Os eventos de *Link Down* foram acionados manualmente através de linhas de comandos no emulador *Mininet*.
- Na ocorrência do evento de queda de enlace (*Link Down*), o sistema foi programado para direcionar o tráfego por outra porta do *switch* virtual.

Na situação de **congestionamento pesado**, tráfego considerado em estado de **bloqueio**, exigindo desvio de rota:

- Foram gerados tráfegos utilizando os protocolos TCP e UDP.
- O tráfego foi gerado de forma dinâmica para reproduzir o evento de congestionamento considerado pesado (tráfego superior a 80% da banda).
- Na ocorrência de congestionamento pesado, o sistema foi igualmente programado para direcionar o tráfego por outra porta do *switch* virtual.

O tráfego de rede foi gerado com o uso da ferramenta *Iperf* versão 3.1.3 instalado na mesma máquina virtual do emulador *Mininet*. Foram utilizadas múltiplas instâncias do programa *iperf* de forma simultânea para gerar tráfego. Esta ação se fez necessária para reproduzir o consumo dinâmico da Rede ICONE onde fosse possível visualizar o consumo gradativo em tempo real. Como discutido anteriormente, a mudança de rota foi a forma tratada para o controlador SDN se adaptar a eventos nas duas situações contextuais. A mudança de rota implica em manipular as portas de saída nas linhas de fluxos dos *switches*.

4.2.1 Prova de Conceito: Queda de Enlace

Para testar o comportamento do controlador SDN, para esta pesquisa, foi considerado estabelecer, inicialmente, uma prova de conceito. Foi escolhida a situação de **queda de**

enlace para validar as regras instaladas no controlador e observar o comportamento do fluxo pela topologia montada no *Grafana*. Com a topologia possuindo quatro enlaces, foram estabelecidos 4 casos de queda dos enlaces que interconectam as instituições. Nos testes descritos a seguir, foi apenas utilizado o protocolo UDP no *Mininet*.

Para estabelecer uma nova rota de saída de fluxo, é preciso alterar os valores correspondentes à porta de saída das entradas de fluxos (*flow entries*) de forma automática. O método adotado para alterar as entradas de fluxos foi deleção da antiga entrada e adição de uma nova. Para que a mudança seja feita, é necessário adquirir informações das linhas de fluxos que irão ser modificadas. Para adquirir essas informações, foi utilizada a classe **OFPFlowStatsRequest** explicada na Subseção 4.1.4. Em seguida, a requisição é capturada com as informações das entradas de fluxos. O campo *instructions* (seção 2.3.2.3), especificado pela classe **OFPInstructionActions**, foi utilizado para realizar as modificações. Esse campo possui valores estruturados em dicionário em *Python*. A estrutura das mensagens *OpenFlow* 1.4 do Ryu possui vários tipos de **ações** que podem ser adotadas a critério do pesquisador. Para indicar mudança de porta foi feito o uso da classe **OFPActionOutput** alterando seu valor para a próxima porta disponível.

Neste trabalho, para realizar mudanças nas tabelas de fluxo, algumas informações fundamentais foram necessárias para que as mudanças de rota fossem possíveis. Essas informações são essenciais para montar a requisição para serem eviadas aos *switches*. São elas:

- *Datapath*: O controlador foi preparado para detectar quando um *switch* (*datapath*) se conecta à rede emulada. Quando um novo *switch* se conecta à rede, um evento é gerado. Essa mensagem de evento contém informações necessárias para o envio das instruções de modificações nos *switches*. Este tipo de evento é capturado implementando um *decorator* com a seguinte classe: **event.EventSwitchEnter**.

- Ip: O endereço ip é necessário para especificar na regra de fluxo a origem e o destino dos pacotes. O ip foi obtido ao implementar o método *get_host* na qual detecta todos os ips dos *hosts* que se conectam à rede emulada. O método *get_host* é importado pelo módulo do *Topology Discovery* (Subseção 4.1.4)

- Porta: A informação de porta é essencial para realizar a mudança de rota nos *switches*. Ela pode ser adquirida de algumas formas e nesta pesquisa foram utilizadas duas delas. Com um *decorator* apontando para a classe **event.EventPortModify** (Quadro 3) e com o método *get_link* também importado pelo módulo *Topology Discovery*

4.2.1.1 Teste de Queda de Enlace CPOR ↔ IFPE

Neste primeiro caso, descreve-se os passos seguidos para testar o enlace CPOR ↔ IFPE:

1. um *host* (h3) foi preparado para enviar dados no sentido CPOR → IFPE → PoP-PE, (vazão de dados foi de 300 Mbps).

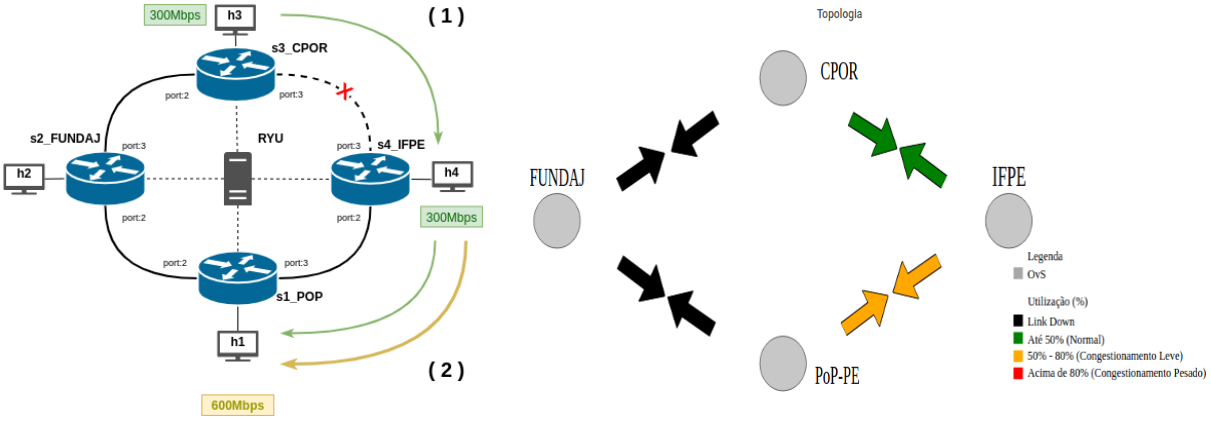
2. um *host* (h4) foi preparado para enviar dados no sentido IFPE → PoP-PE vazão de dados de 300 Mbps exibido na Figura 35a e seu comportamento comprovado na Figura 35b.

3. em um segundo momento o trecho CPOR ↔ IFPE foi derrubado. (Figura 36a).

4. Em seguida o fluxo do CPOR é redirecionado obedecendo o trajeto CPOR → FUNDAJ → PoP-PE como pode ser visto na Figura 36b.

Figura 35 – Teste do enlace CPOR ↔ IFPE.

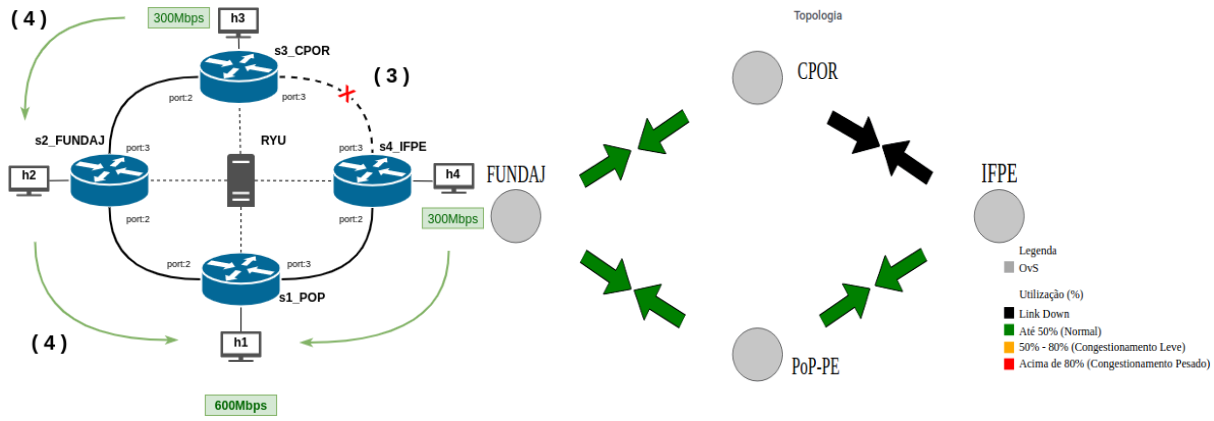
(a) Cenário Emulado. (b) Resultado apresentado no Grafana.



Fonte: Elaborado pelo Autor.

Figura 36 – Queda do enlace CPOR ↔ IFPE.

(a) Cenário Emulado. (b) Resultado apresentado no Grafana.



Fonte: Elaborado pelo Autor.

4.2.1.2 Teste de Queda de Enlace IFPE ↔ PoP-PE.

Neste segundo caso, serão descritos os passos para testar o enlace IFPE → PoP-PE:

1. um *host* (h3) foi preparado para enviar dados no sentido CPOR → IFPE → PoP-PE, vazão de dados foi de 300 Mbps (Figura 35a).

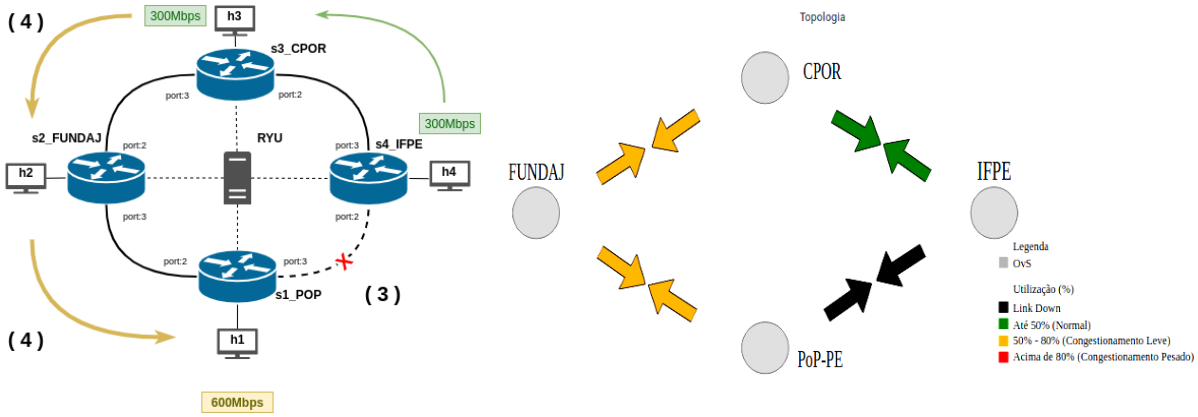
2. um *host* (h4) foi preparado para enviar dados no sentido IFPE → PoP-PE, vazão de dados de 300 Mbps, com isso, explicando o fluxo agregado no trecho IFPE → PoP-PE da topologia (Figura 35b).

3. em um segundo momento o trecho IFPE ↔ PoP-PE foi derrubado (Figura 37a).

4. Em seguida a rota de saída do IFPE é alterada para IFPE → CPOR → FUNDAJ → PoP-PE. Como pode ser comprovado na imagem da topologia na Figura 37b, o fluxo agregado passa a ser nos trechos do CPOR ↔ FUNDAJ e FUNDAJ ↔ PoP-PE.

Figura 37 – Queda do enlace POP ↔ IFPE.

(a) Cenário Emulado. (b) Resultado apresentado no Grafana.



Fonte: Elaborado pelo Autor.

O próximo enlace a ser testado será o FUNDAJ ↔ PoP-PE que segue os seguintes passos:

1. um *host* (h2) foi preparado para enviar dados no sentido FUNDAJ → PoP-PE, vazão de dados foi de 300 Mbps .

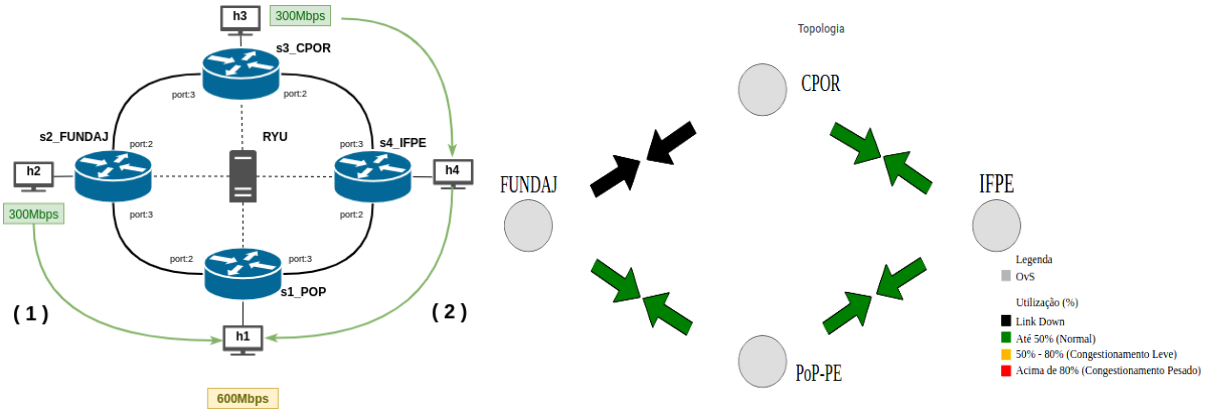
2. um *host* (h3) foi preparado para enviar dados no sentido CPOR → IFPE → PoP-PE com vazão de 300 Mbps. Sendo assim, o tráfego flui por caminhos separados (Figura 38a e 38b).

3. em um segundo momento o trecho FUNDAJ ↔ PoP-PE é derrubado (Figura 39a).

4. Em seguida a rota de saída é alterada fazendo o tráfego fluir no sentido FUNDAJ → CPOR → IFPE → PoP-PE. Com isso, explicando o fluxo agregado nos trechos CPOR → IFPE e IFPE → PoP-PE (Figura 39b).

Figura 38 – Teste do enlace FUNDAJ ↔ POP.

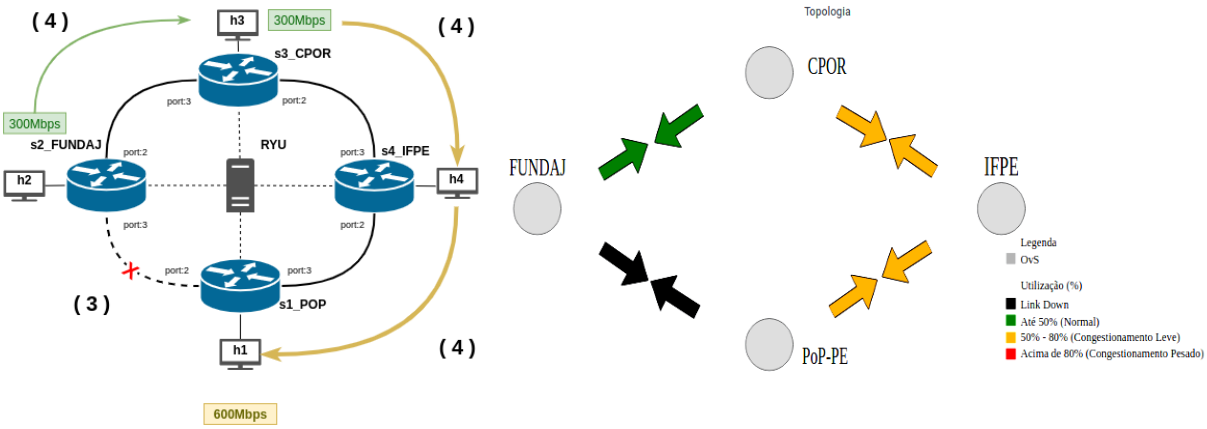
(a) Cenário Emulado. (b) Resultado apresentado no Grafana.



Fonte: Elaborado pelo Autor.

Figura 39 – Queda do enlace FUNDAJ ↔ POP.

(a) Cenário Emulado. (b) Resultado apresentado no Grafana.



Fonte: Elaborado pelo Autor.

4.2.1.3 Teste de Queda de Enlace FUNDAJ ↔ CPOR.

O fluxo inicial sempre obedece o trajeto CPOR → IFPE → PoP-PE deixando o trecho CPOR → FUNDAJ livre. Para este caso foram necessárias duas quedas: a primeira para que o fluxo do CPOR obedecesse o trajeto CPOR → FUNDAJ → PoP-PE e a segunda entre o trecho CPOR → FUNDAJ.

O teste seguiu os seguintes passos:

1. um *host* (h2) foi preparado para enviar dados no sentido FUNDAJ → PoP-PE com vazão de dados de 50Mbps.

2. um *host* (h3) foi preparado para enviar dados no sentido CPOR → IFPE → PoP-PE com vazão de dados de 400 Mbps.

3. um *host* (h4) foi preparado para enviar dados no sentido IFPE → PoP-PE com vazão de dados de 200 Mbps explicando o fluxo agregado no trecho IFPE → PoP-PE (Figura 40a e 40b).

4. em um segundo momento o trecho CPOR → IFPE foi derrubado.

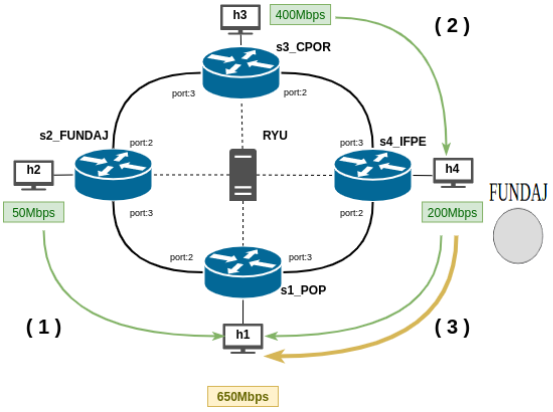
5. com o trecho CPOR ↔ IFPE *down*, o tráfego a seguir pelo caminho CPOR → FUNDAJ → PoP-PE como exibido na Figura 41a e 41b.

6. em um terceiro momento (após recuperar o trecho CPOR → IFPE) o enlace FUNDAJ → CPOR foi derrubado (Figura 42a).

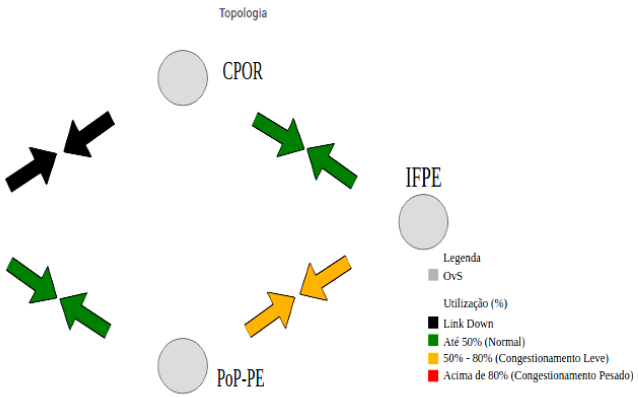
7. em seguida o tráfego segue novamente pelo trajeto CPOR → IFPE → PoP-PE como exibido na topologia (Figura 42b)

Figura 40 – Teste do enlace FUNDAJ ↔ CPOR.

(a) Cenário Emulado.



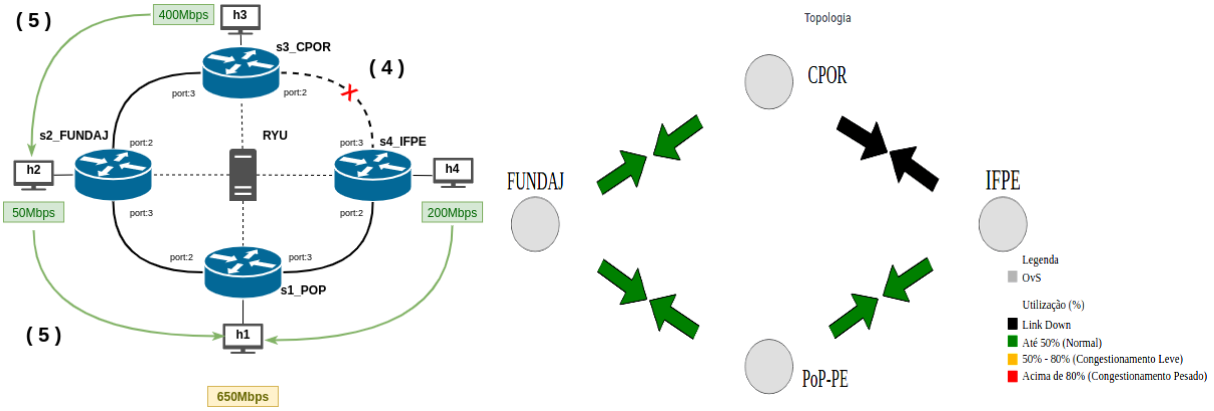
(b) Resultado apresentado no Grafana.



Fonte: Elaborado pelo Autor.

Figura 41 – Primeira parte do teste do enlace FUNDAJ ↔ CPOR.

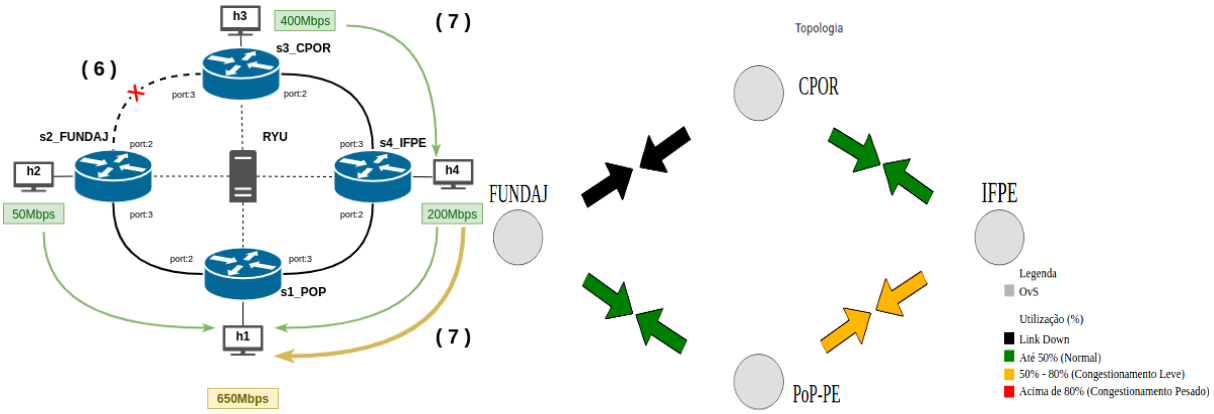
(a) Cenário Emulado. (b) Resultado apresentado no Grafana.



Fonte: Elaborado pelo Autor.

Figura 42 – Queda do Enlace FUNDAJ ↔ CPOR.

(a) Cenário Emulado. (b) Resultado apresentado no Grafana.



Fonte: Elaborado pelo Autor.

4.2.2 Análise Estatística: Congestionamento Pesado

Nesta Subseção, foi feita a avaliação da situação Congestionamento Pesado. A avalização consiste em analisar o tempo de resposta das etapas de captura, inferência e adaptação de contexto. Para avaliar esta situação de congestionamento pesado, foi gerado um tráfego em apenas um trecho. O trecho selecionado para esta avaliação foi CPOR ↔ IFPE. O total de repetições executadas foi igual a 30 repetições para o protocolo TCP e 30 para UDP somando um total de 60 repetições. Nesta avaliação, o experimento consiste em gerar tráfego maior ou igual a 80% (regra de contexto estabelecida para Congestionamento Pesado definida na Subseção 3.4.3) suficiente para que esta regra seja violada e ocorra a mudança de rota.

4.2.2.1 Métricas

As métricas aqui analisadas foram usadas com dois propósitos. O primeiro objetivo é coletar informações contextuais para alterar o comportamento do controlador, enquanto o segundo objetivo é apresentar os resultados da análise estatística descritiva. As métricas *Throughput* e tempo de resposta (*Response Time* — RT) foram escolhidas para esta pesquisa. A métrica *Throughput* foi estabelecida ao realizar um cálculo de banda feito em todas as saídas e entradas das portas dos *switches*, e o RT, para medir o desempenho das etapas de contexto.

4.2.2.2 Resultados

Esta Seção será destinada a apresentar os métodos utilizados para realizar os processos de captura, inferência e adaptação, bem como seus resultados. Para este trabalho a etapa de pré-processamento não se aplica aos casos que serão apresentados. Os resultados a serem exibidos fazem parte da análise estatística descritiva que foi realizada utilizando a linguagem R com o programa *RStudio*[1]. Estes resultados foram obtidos levando em consideração os testes com os protocolos TCP e UDP.

4.2.2.3 Captura de Dados para Contexto

A captura de dados envolve o sensoriamento e a aquisição de informações específicas que possam ser usadas para definir uma determinada situação. Em uma rede onde há um grande fluxo de dados, é necessário que o monitoramento de estatísticas de rede seja feito de forma contínua e estável. Com essa observação, foi feita a implementação da função *Traffic Monitor* (explicado na Seção 4.1.4) ajustado para o tempo de coleta de 1 segundo. Sendo assim, O *Traffic Monitor* implementa um *loop* executando coleta de estatísticas de porta a cada segundo. Com o tempo de 1 segundo, a análise dos elementos contextuais (vazão e banda) é feita em megabit por segundo (*Mbps*) seguindo uma equação simples de cálculo de banda que é realizado em cada porta dos *switches* da rede emulada como visto no código 4.3. Os *bytes* são capturados através de uma estrutura de dados (dicionário em *Python)* especificada como `stat` e o valor dos *bytes* acessado em `tx_bytes`, para *bytes* transmitidos (linha 2) e `rx_bytes` para *bytes* recebidos (linha 10).

O `stat.tx_bytes` faz parte da mensagem do evento **EvenOFPPortStatsReply** capturado, descrito na seção 2.3.1.1 (quadro 5). A partir do momento em que o experimento é iniciado os *bytes* transmitidos `tx_ini` e os *bytes* recebidos `rx_bytes` são armazenados na variável (linhas 1, 2, 9 e 10) em seguida, todos os *bytes* seguintes serão considerados como *bytes* finais e armazenados em *tx_fin*. Com isso, é feito o cálculo da banda subtraindo os *Bytes* finais dos *Bytes* iniciais do tráfego capturados (linhas 5 e 13) e multiplicado por 8 *bits*, em seguida transformando em Mbps dividindo a *bandwidth* por 1 Mbit (1,048,576

[1] RStudio - Open Source Professional Software for data Science Teams. https://rstudio.com/

bits) (linhas 6 e 14). E por fim, os valores de *bytes* finais são enviados para variável de *bytes* iniciais (linhas 7 e 15) (MEGYESI et al., 2017), (SHU et al., 2016).

Código 4.3 – Captura para Congestionamento Pesado

```
1                 if tx_ini == 0:
2                     tx_ini = stat.tx_bytes
3
4                 tx_fin = stat.tx_bytes
5                 bandwidth = (tx_fin-tx_ini)*8
6                 result_tx = int(bandwidth/1048576)
7                 tx_ini = tx_fin
8
9                 if rx_ini == 0:
10                    rx_ini = stat.rx_bytes
11
12                rx_fin = stat.rx_bytes
13                bandwidth = (rx_fin-rx_ini)*8
14                result_rx = int(bandwidth/1048576)
15                rx_ini = rx_fin
```

4.2.2.3.1 TCP

Aqui foram apresentados os resultados do RT referentes à etapa de captura dos dados exibidos em milissegundos em TCP. Nesta análise descritiva do tempo de resposta, foram obtidos alguns resultados como exibido no Quadro 6.

Quadro 6 – Resultado Captura de Dados para Contexto - tráfego TCP.

	TCP (ms)
Min.	5,094
1st Qu.:	6,495
Median:	**8,481**
Mean:	8,481
3rd Qu.:	9,558
Max.:	15,724
Var.	4,754
Sd.	2,180

Fonte: Elaborado pelo Autor.

4.2.2.3.2 UDP

No Quadro 7, os resultados exibidos são referentes à captura de contexto utilizando tráfego com o protocolo UDP. Como é possível visualizar, a mediana analisada se aproxima do valor exibido no quadro TCP anterior (Quadro 6).

Quadro 7 – Resultado Captura de Dados Contexto - tráfego UDP.

	UDP (ms)
Min.	4,454
1st Qu.:	7,683
Median:	**8,574**
Mean:	8,822
3rd Qu.:	9,347
Max.:	25,066
Var.	11,395
Sd.	3,375

Fonte: Elaborado pelo Autor.

4.2.2.4 Inferência de Contexto

A inferência trata as informações capturadas das situações contextuais desta pesquisa. A regra para a inferência de **congestionamento pesado** (vazão $\geq 80\%$) pode ser detalhada seguindo o simples cálculo do Código 4.4 onde a vazão (*Throughput*) é o resultado da soma de *Bytes* transmitidos **`result_tx`** com *Bytes* recebidos **`result_rx`** (cálculo de banda visto no Código 4.3).

Código 4.4 – Inferência para congestionamento Pesado

```
Throughput = (result_tx + result_rx)
if (Throughput >= MAX_BAND*0.8):
```

Com isso, é estabelecido uma constante MAX_BAND onde foi atribuído um valor 1000 representando 1000 Mbps (1Gbps) para aplicação da regra de inferência. Por fim a condicional if testa a regra onde Se a vazão for maior ou igual a 80% da banda máxima será inferido que o enlace está com congestionamento pesado.

4.2.2.4.1 TCP

Aqui são exibidos os resultados do RT nesta etapa de inferência de contexto (Quadro 8). Nesta etapa os valores de mediana apresentados estão abaixo dos valores da etapa de captura vistos anteriormente (Quadros 6 e 7).

Quadro 8 – Resultado Inferência de Contexto TCP.

	TCP (ms)
Min.	3,200
1st Qu.:	4,746
Median:	**6,9**
Mean:	6,484
3rd Qu.:	7,657
Max.:	10,528
Var.	3,701
Sd.	1,923

Fonte: Elaborado pelo Autor.

4.2.2.4.2 UDP

Neste resultado de inferência de contexto, foi utilizado o protocolo UDP (Quadro 9). Neste caso, a mediana se mostrou acima do resultado TCP para esta mesma etapa (mediana **6,9** no Quadro 8), mas abaixo dos resultados de TCP e UDP para etapa de captura de contexto (Quadros 6 e 7).

Quadro 9 – Resultado Inferência de Contexto UDP.

	UDP (ms)
Min.	3,079
1st Qu.:	4,432
Median:	**7,360**
Mean:	7,127
3rd Qu.:	8,096
Max.:	18,667
Var.	10,270
Sd.	3,204

Fonte: Elaborado pelo Autor.

4.2.2.5 Adaptação a Contexto

A abordagem adotada para o processo de adaptação referente ao contexto de congestionamento pesado envolve apenas chamar a função responsável por modificar entradas de fluxos. Ao inferir vazão $\geq 80\%$ descrito pelo Código 4.4, a função para modificação de fluxo será chamada. Esta função atua utilizando o método de remoção e adição de linhas de fluxo através de mensagens **OFPFC_ADD** e **OFPFC_DELETE**. Essas mensagens fazem parte de uma estrutura de mensagens de modificação da tabela de fluxo *OpenFlow*

((OPEN NETWORKING FOUNDATION, 2015)) que, por meio da classe **OFPFlowMod** do controlador Ryu, podem ser utilizadas para manipulação de entradas da tabela de fluxo.

4.2.2.5.1 TCP

No Quadro 10, logo abaixo, são apresentados os resultados do RT da etapa de adaptação de contexto utilizando o protocolo TCP.

Quadro 10 – Resultado Adaptação de Contexto TCP.

	TCP (ms)
Min.	1,419
1st Qu.:	1,592
Median:	**1,781**
Mean:	1,976
3rd Qu.:	2,153
Max.:	3,631
Var.	0,314
Sd.	0,560

Fonte: Elaborado pelo Autor.

4.2.2.5.2 UDP

A seguir no Quadro 11, são apresentados os resultados do RT da etapa de adaptação envolvendo o protocolo UDP. O resultado da mediana mostra que esta etapa foi levemente superior do que a utilização do protocolo TCP (Quadro 10), mas inferior às etapas de captura e inferência nos quadros anteriores (Quadros 6, 7, 8 e 9).

Quadro 11 – Resultado Inferência de Contexto UDP.

	UDP (ms)
Min.	1,444
1st Qu.:	1,663
Median:	**1,814**
Mean:	1,939
3rd Qu.:	2,038
Max.:	3,157
Var.	0,154
Sd.	0,393

Fonte: Elaborado pelo Autor.

4.3 AVALIAÇÃO

Para testar a eficácia desta proposta, foram feitas duas análises estatísticas do experimento referente ao cenário de **congestionamento pesado**.

4.3.1 Análise 1

Para esta primeira análise, foram coletados os RTs das 3 etapas de contexto (captura contexto, inferência de contexto e adaptação de contexto) onde foi feita uma comparação dos RTs dessas etapas. Foram feitas coletas de amostras com tamanho de 30 repetições com base no Teorema Central do Limite descrito por (LARSON; FARBER, 2010, p. 221), que afirma que quando o tamanho da amostra aumenta, a distribuição das médias amostrais se aproxima de uma distribuição normal. Nos resultados dos quadros exibidos na Subseção anterior (Quadros 6, 7, 8, 9, 10 e 11) os valores das medianas (destaque em negrito) foram considerados por não sofrer muita influência de *outliers* sinalizados no *BoxPlot*, o que pode acarretar em alterações nos resultados envolvendo a média. Foi levado em consideração o impacto do RT de cada etapa, que são: (a) captura de dados, (b) inferência de contexto, (c) adaptação ao contexto, que aqui foram chamados de RTa, RTb e RTc, tanto para o protocolo TCP quanto UDP. Nesta análise descritiva dos RTs, os valores medianos foram **8,48**, **6,9**, **1,78** para o protocolo TCP e **8,574**, **7,36** e **1,81** para o protocolo UDP. Neste cenário, levou-se mais tempo para (a) capturar os dados para a inferência de contexto (b), que levou um tempo menor, enquanto que a adaptação ao contexto (c) foi ainda mais rápida, dada a simplicidade da ação necessária (uma mudança de rota). Tecnicamente, esta diferença de tempo entre as etapas é justificada porque a etapa (a) tem um atraso de tempo por conta do cálculo da taxa de transferência em tempo real com base nos fluxos *OpenFlow*, enquanto (b) leva um pouco menos tempo para conhecer o contexto e inferir uma situação de congestionamento, e a adaptação (c) é um processo mais simples onde é executada uma função para realizar a inversão de portas de saída (Tabela 1).

Tabela 1 – Análise de impacto dos RTs de captura de contexto, inferência de contexto e adaptação de contexto (tempos em ms)

Etapas / Protocolo	Captura	Inferência	Adaptação
TCP	8,4	6,9	1,7
UDP	8,5	7,3	1,8

Fonte: Elaborado pelo Autor.

4.3.2 Análise 2

Na segunda análise, foram comparadas duas abordagens, chamadas **SDNC** - experimentos conduzidos em um ambiente SDN com gerenciamento de rede baseado em contexto, e **SDNH** - experimentos conduzidos em um ambiente SDN baseado em gerenciamento tradicional com intervenção humana.

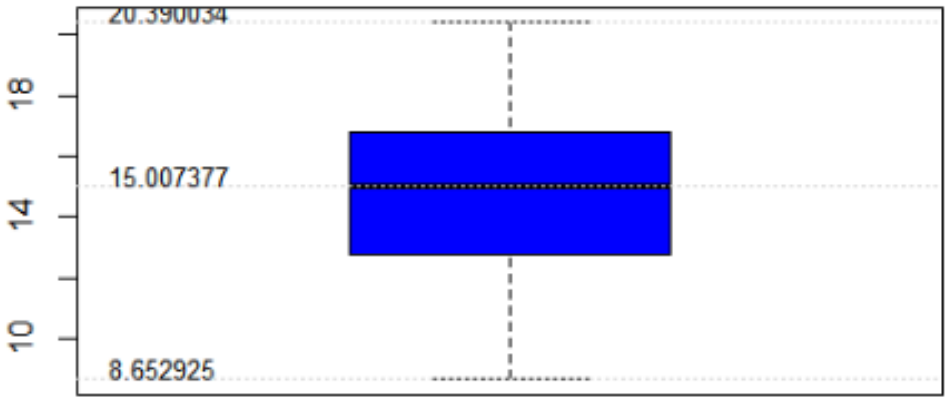
A abordagem **SDNC** representa um sistema de gerenciamento de redes adaptativo baseado em contexto. Em outra palavra, um sistema de gerenciamento que se adapta sem interferência de um gerente/administrador de redes. Para isso, são considerados apenas as etapas de (a) Captura e (b) Inferência. Apenas essas duas etapas foram consideradas, pois a etapa (c) Adaptação, envolveria testes com a presença de administradores de redes representando um sistema de gerenciamento de rede tradicional.

A abordagem **SDNH** onde representa um sistema de gerenciamento de redes tradicional (com a presença de um gerente/administrador de redes). Neste caso, foram acrescentados os passos (d) exibição de alarme e (e) - percepção humana, representando a inferência em um sistema de gerenciamento tradicional. Observe que a etapa (c) adaptação ao contexto não foi considerado nas duas abordagens porque envolveria a presença de administradores de redes para marcar o tempo de atuação em uma situação real, o que poderia tornar esta pesquisa mais complexa de se realizar.

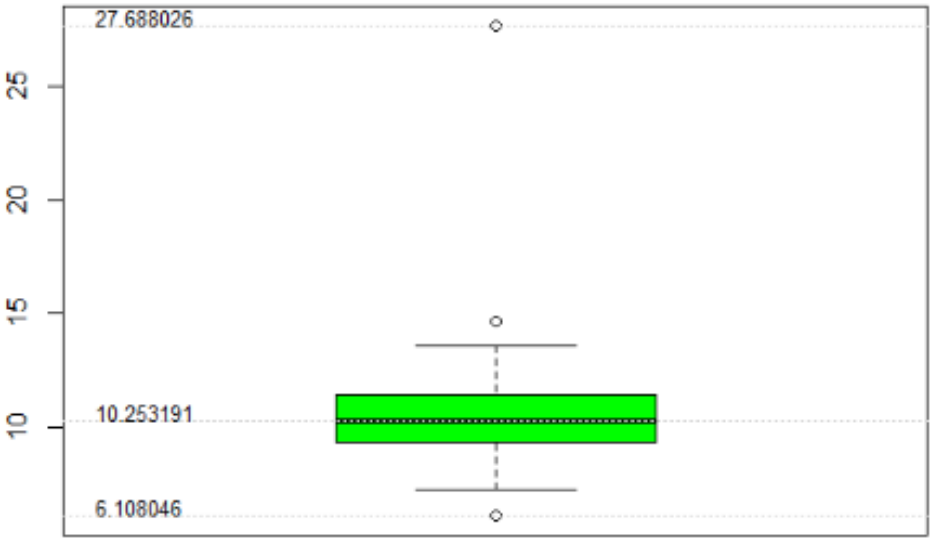
Comparando as duas abordagens em relação à análise estatística descritiva das duas amostras (SDNC e SDNH), foi observado que na abordagem SDNC (BoxPlot, Figura 43a e 43b), que envolve apenas os passos de Captura (a) e Inferência (b), a mediana dos valores amostrados é igual a **15,007 ms** para TCP (Figura 43a) e **10,253 ms** para UDP (Figura 43b). Para a abordagem SDNH, foi obtido o BoxPlot, Figura 44a com a mediana **15,082 ms**, para o tráfego TCP (Figura 44a), e **10,323 ms**, para o tráfego UDP (Figura 44b).

Figura 43 – Boxplot do Gerenciamento Baseado em Contexto

(a) SDN+Contexto TCP.



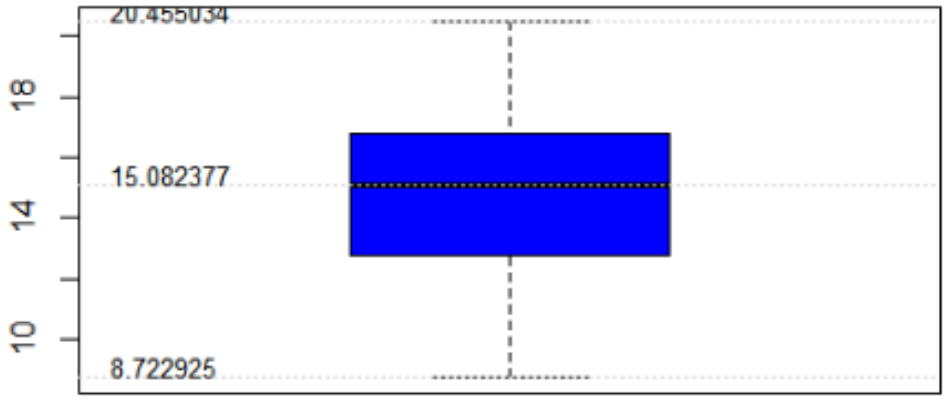
(b) SDN+Contexto UDP.



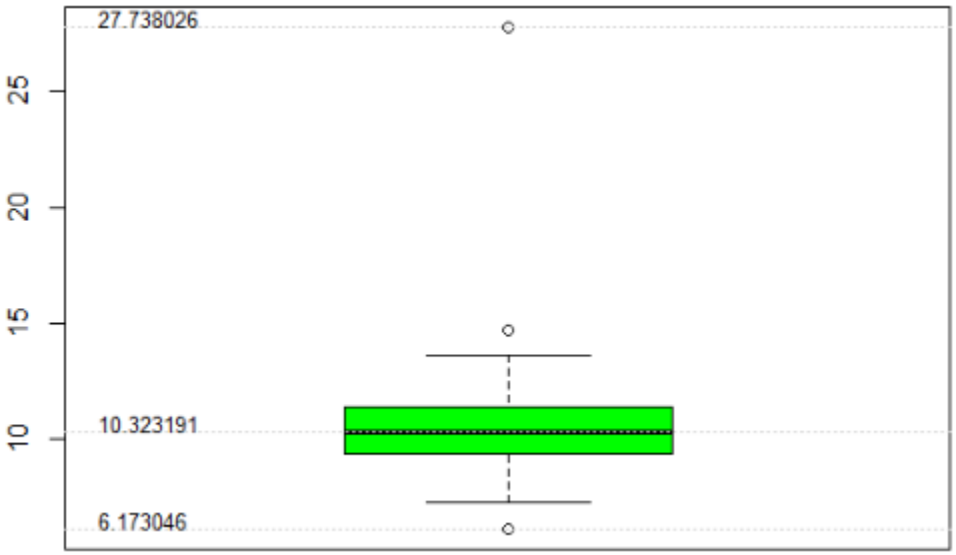
Fonte: Elaborado pelo Autor.

Figura 44 – Boxplot do Gerenciamento Tradicional.

(a) SDN+Humano TCP.



(b) SDN+Humano UDP.



Fonte: Elaborado pelo Autor.

Considerando o trabalho de Nicolaou (NICOLAOU, 1990), que observou que o limiar de percepção humana nas interfaces do usuário (UI) está no intervalo de 10 a 40 ms (vamos chamá-lo de etapa (e) – percepção humana), na abordagem SDNH o tempo de resposta estaria entre **25,082 ms** e **55,082 ms**, para o tráfego TCP, e **20,323 ms** e **50,323 ms**, para o tráfego UDP, uma vez que $SDNC = RTa + RTb$ e $SDNH = RTa + RTb + RTd + RTe$ (Tabela 2). Portanto, nossa abordagem leve (sensível ao contexto) provou ser viável para o gerenciamento adaptável de rede. Comparamos SDNC e SDNH sem RTc, pois o processo de adaptação seria significativamente diferente nas duas abordagens, dada a intervenção humana em SDNH, sujeita a erros e atrasos imprevisíveis. No entanto, a análise anterior mostrou que o RTc é bastante curto (1,78 ms) no SDNC. É naturalmente esperado que a adaptação humana a uma situação de rede seja da ordem de alguns segundos.

Tabela 2 – Comparação SDN-Contexto *vs* SDN-Humano – percepção situacional (tempos em ms)

Abordagem / Protocolo	SDNC	SDNH
TCP	15,007	25,082 – 55,082
UDP	10,253	20,323 – 50,323

Fonte: Elaborado pelo Autor.

4.4 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados os detalhes abrangendo a implementação, configuração e codificação da abordagem peso leve para o Gerenciamento Adaptativo de Redes Baseado em Contexto e SDN, dos experimentos realizados, bem como os resultados estatísticos das etapas de captura de contexto, inferência de contexto e adaptação de contexto. O ambiente virtual foi elaborado a partir do modelo de gerenciamento do PoP-PE/RNP. Foram utilizadas máquinas virtuais para a organização e a instalação das ferramentas. Foram instalados o controlador Ryu (*Python*), o emulador *Mininet* e o *Docker* com os contêiners *InfluxDB* e o *Grafana*. O *Mininet* foi configurado para ter condições semelhantes ao da Rede ICONE, onde foram preparados 3 *switches* representando as instituições CPOR, IFPE e FUNDAJ, e um quarto *switch* representando o PoP-PE/RNP para saída de *internet*, formando uma topologia em anel. Uma aplicação de gerência sensível a contexto foi implementada utilizando a API *OpenFlow* do Ryu de forma reativa. Foram testadas duas situações contextuais: queda de enlace e congestionamento pesado. A situação de queda de enlace foi utilizada como **prova de conceito**, enquanto que para a situação de congestionamento pesado foram feitas duas **análises estatísticas**. Na primeira análise, os resultados mostraram que a captura teve maior tempo por conta do cálculo da taxa de transferência em comparação com as etapas de inferência e de adaptação. Na segunda análise foi apresentada uma comparação de duas abordagens, representando um sistema de gerenciamento baseado em contexto (SDNC) e um sistema de gerenciamento tradicional realizado por um ser humano (SDNH), onde foi mostrado que a abordagem SDNH é mais fortemente impactada no sentido de ficar mais lenta por introduzir o tempo máximo (limite) de percepção humana (10ms - 40ms) (NICOLAOU, 1990).

5 CONCLUSÕES

Este trabalho apresentou uma abordagem SDN para Gerenciamento Adaptativo de Rede com o auxílio de conceitos de sensibilidade a contexto e da arquitetura SDN. Foi estabelecido um ambiente virtual de experimentação controlado, utilizando o emulador de redes SDN *Mininet*. A rede emulada foi baseada na rede ICONE (rede metropolitana de ensino e pesquisa) gerenciada pelo Instituto de Tecnologia de Pernambuco (ITEP). Em seguida foi feito o uso do controlador Ryu (*python*) para a elaboração de uma aplicação sensível a contexto. Quatro contextos foram apresentados, dos quais três foram reproduzidos: **tráfego normal**, **queda de enlace** e **congestionamento pesado**.

Em questão ao contexto de tráfego normal, o controlador apenas **observa** a rede, o que indica que não houve adaptação já que a rede está em condição normal de operação. No contexto de **queda de enlace**, o controlador foi ajustado para responder a eventos de rede de forma reativa, de modo que quando for detectada uma queda de enlace o tráfego é redirecionado. Já na análise do contexto **congestionamento pesado**, é feito um cálculo de vazão utilizando informações estatísticas recolhidas nos *switches*, onde se o trafego for acima de 80% da banda, então o tráfego está congestionado, o que implica em mudança de rota. Após o estabelecimento das regras de contexto, foi feita a implementação no controlador Ryu para que se comportasse de uma forma reativa às regras de contexto. Foi construída uma rede em anel para emular os comportamentos da Rede ICONE. Foi instalado o *Docker* e com ele foram levantados dois contêineres, *InfluxDB* e *Grafana*, para monitoramento do tráfego e validação do comportamento do controlador.

A validação do comportamento do controlador se deu pela organização de uma prova de conceito com a situação de queda de enlace, onde os resultados se mostraram satisfatórios em relação à mudança de rota. Por fim, algumas análises estatísticas foram feitas em relação às etapas de captura, inferência e adaptação de contexto, onde foi verificado que os tempos da captura foram superiores aos de inferência e adaptação por envolver cálculos de taxa de transferência (vazão). Na segunda análise, foi feita uma comparação entre duas abordagens. A primeira, foi chamada de SDNC, que representa o gerenciamento adaptativo baseado em contexto, e a segunda, SDNH, representando um sistema de gerência tradicional, envolvendo intervenção humana na gerência de rede. A diferença se deu no tempo de percepção de mudança de situação (contexto), que com o ser humano é naturalmente mais lenta.

5.1 PRINCIPAIS CONTRIBUIÇÕES

Este trabalho ofereceu uma abordagem leve, sensível a contexto, capaz de reagir a eventos simples de rede de forma automática e adaptativa. Foi possível entender a dinâmica de uma

programação orientada a eventos do controlador Ryu utilizando o protocolo *OpenFlow*. O controlador *Ryu* fornece algumas classes e funções bem definidas que foram exploradas para a coleta de informações contextuais, como vazão e banda. Outras informações contextuais podem ser exploradas, uma vez que o acesso a elas seja alcançado. Uma vez alcançada, pode-se aplicar regras simples para manipulação do tráfego a depender de cada situação, ficando a critério do pesquisador/administrador da rede. Existem algumas soluções privadas para tratar de mudança de rotas de forma dinâmica e automática, como o protocolo SPF (*Shortest Path First*) baseado no algoritmo *Dijkstra* em redes tradicionais (NAKAHODO; NAITO; OKI, 2014), (WALEED et al., 2017).

Apesar disso, este pesquisa apresentou uma solução código aberto, sensível a contexto utilizando o paradigma SDN onde foi implementado o algoritmo DFS *Depth-First Search*. Além disso, foi verificado, através da análise do tempo de resposta, que a etapa de captura possui o tempo mais longo, uma vez que envolve o processo de cálculo de banda. Os resultados, exibidos nesta pesquisa, mostram que o sistema se adapta automaticamente às mudanças do ambiente, sem a necessidade de intervenção humana. Conclui-se que sensibilidade a contexto pode realizar a autonomia de gerência de redes definidas por *software*, pelo menos em casos mais simples e corriqueiros, e pode ser um complemento ao aprendizado de máquina para situações mais complexas.

5.2 LIMITAÇÕES E DIFICULDADES

Aqui são descritas algumas limitações e dificuldades encontradas durante a execução desta pesquisa:

5.2.1 LIMITAÇÕES

Aqui são listadas as limitações encontradas no percurso desta pesquisa:

- Devido a uma limitação do OvS por conta de versionamento, não foi possível explorar a *meter_table*, responsável por possuir alguns medidores de banda, o que permite também o policiamento de QoS. Por conta disto, foram feitos cálculos de banda por porta, o que pode causar um *overhead* no controlador.
- Limitações do emulador *Mininet* impediram que outras informações contextuais como *packet_loss* ou taxa de erros fossem levadas em consideração.
- Apesar do controlador Ryu ser um dos controladores mais populares, sua documentação apenas fornece exemplos de uso de funções de uma maneira muito básica. As explicações de alguns parâmetros ou campos dessas funções deixam a desejar, o que prejudicou um pouco a criação de abordagens para manipulação de entradas de fluxos de uma forma mais dinâmica.

5.2.2 DIFICULDADES

Nesta pesquisa acadêmica houve algumas dificuldades que prejudicaram o progresso das atividades.

- Quanto ao ambiente virtual, no início das atividades referentes a esta pesquisa, o PoP-PE estava passando por um processo de expansão e reestruturação do NOC, fazendo com que alguns serviços ficassem indisponíveis ou instáveis. Por conta disto, apesar do PoP-PE fornecer um ambiente de servidores com serviços de virtualizações adequados, ficou decidido que o processo de experimentação fosse executado em máquina local.

- A programação utilizando a API *SouthBound* para arquitetura SDN é algo muito específico. A falta de experiência e conhecimento técnico para entender a dinâmica e a lógica da programação envolvendo o protocolo *OpenFlow* prejudicou o progresso do experimento.

- Tendo em vista o problema acima, foi feito o uso de listas de discussões do projeto do controlador Ryu. Porém, a maioria dos temas debatidos não condiziam com o tema pesquisado. As questões levantadas demoravam a ser respondidas e muitas vezes as dúvidas não eram respondidas como esperado.

- O método de captura de eventos e o processo para tratar as mensagens para retirar informações relevantes para depois enviar para outras funções, e em seguida encapsular essas mensagens, pode ser um procedimento muito trabalhoso. A falta de um padrão para desenvolvimento utilizando o protocolo *OpenFlow* no controlador Ryu pode gerar muitas linhas de código, dificultando o entendimento caso o objetivo seja realizar algo mais complexo.

- A programação orientada a eventos partindo do controlador Ryu pode ser bastante confusa em certas situações. Na programação reativa, geralmente caracterizada por *loops*, um evento pode ser detectado mais de uma vez em uma rede em anel. Isto pode dificultar o entendimento, caso o código fique mais complexo.

- O Grafana possui o plugin SVG, bastante poderoso para projetar e customizar a topologia monitorada, permitindo várias possibilidades de ideias de visualizações. Porém, em um ambiente dinâmico, adicionar ou remover dispositivos um a um pode ser um trabalho extremamente custoso.

5.3 TRABALHOS FUTUROS

Para trabalhos futuros e continuação desta pesquisa, algumas propostas podem ser consideradas:

- Testar o desempenho do controlador com todas as 34 instituições da Rede ICONE em um ambiente controlado.
- Será indicado a elevação da experimentação para simulação eliminando o uso do emulador *Mininet* e utilizando alguns simuladores do mercado. E em seguida a prototipagem com *switches OpenFlow* reais.
- Informações contextuais de *Packet_loss*, taxa de erros, pacotes descartados podem ser considerados. Feito isso, será discutida a possibilidade de captura, inferência e adaptação de outras informações contextuais levando em conta o tipo de serviço fornecido pelo protocolo, utilizando o campo *EtherType* (ELECTRICAL; IEEE, 2019). Além disso, propriedades e valores referentes a fibras ópticas fornecida pelo *OpenFlow* versão 1.4 (usado nesta pesquisa).
- É esperada a implementação, juntamente com a análise estatística, do contexto de congestionamento moderado para realização de balanceamento dividindo o fluxo de tráfego (50%-50% ou 70%-30%). Isto fará com que o fluxo seja mais distribuído, tornando a engenharia de tráfego mais dinâmica.
- Melhoria contínua do código do controlador Ryu[1] de modo a reconhecer outras topologias.

[1] https://github.com/josecarlosjr/SDN-CONTEXT/blob/master/ryu_multipath_1_7.py

REFERÊNCIAS

ABOWD, G. D.; DEY, A. K.; BROWN, P. J.; DAVIES, N.; SMITH, M.; STEGGLES, P. Towards a better understanding of context and context-awareness. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 1707, p. 304–307, 1999. ISSN 16113349.

ALCORN, J.; MELTON, S.; CHOW, C. E. Portable SDN Testbed Prototype. *Proceedings - 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN-W 2017*, p. 109–110, 2017.

ARBETTU, R. K.; KHONDOKER, R.; BAYAROU, K.; WEBER, F. Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers. *2016 17th International Telecommunications Network Strategy and Planning Symposium, Networks 2016 - Conference Proceedings*, p. 37–44, 2016.

AYOUBI, S.; LIMAM, N.; SALAHUDDIN, M. A.; SHAHRIAR, N.; BOUTABA, R.; ESTRADA-SOLANO, F.; CAICEDO, O. M. Machine learning for cognitive network management. *IEEE Communications Magazine*, v. 56, n. 1, p. 158–165, Jan 2018. ISSN 0163-6804.

BOBEK, S.; NALEPA, G. J. Uncertainty handling in rule-based mobile context-aware systems. *Pervasive and Mobile Computing*, Elsevier, v. 39, p. 159–179, 2017.

BOETTIGER, C. An Introduction to Docker for Reproducible Research. p. 71–79, 2015.

BU, Y.; GU, T.; TAO, X.; LI, J.; CHEN, S.; LU, J. Managing quality of context in pervasive computing. *Proceedings - International Conference on Quality Software*, p. 193–200, 2006. ISSN 15506002.

BUI, N.; CESANA, M.; HOSSEINI, S. A.; LIAO, Q.; MALANCHINI, I.; WIDMER, J. A survey of anticipatory mobile networking: Context-based classification, prediction methodologies, and optimization techniques. *IEEE Communications Surveys Tutorials*, v. 19, n. 3, p. 1790–1821, thirdquarter 2017. ISSN 1553-877X.

CARGA, B. D. E.; REDES, E. M.; FIO, S. E. M.; SOFTWARE, P. O. R. Eric Ribas Moraes Machado. 2017.

CHAGAS, L.; FILENE R., E. *SDN - Redes Definidas por Software*. UFRJ, 2018. Disponível em: <https://www.gta.ufrj.br/ensino/eel879/trabalhos_v1_2018_2/sdn/>. Acesso em: 2 jan. 2020.

CHAN, N. A resource utilization analytics platform using grafana and telegraf for the savio supercluster. In: ACM. *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning)*. [S.l.], 2019. p. 31.

CHANG, F. C.; CHEN, D. K. The design of an XMPP-based service integration scheme. *Proceedings - 7th International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIHMSP 2011*, p. 33–36, 2011.

CISCO. 2025 Global Trends Report. *Intelligence*, 2020.

CONTAINER-VS-VMS. Xebia, 2017. Disponível em: <https://xebia.com/blog/deep-dive-into-windows-server-containers-and-docker-part-1-why-should-we-care/container-vs-vms/>. Acesso em: 5 jan. 2020.

COX, J. H.; CHUNG, J.; DONOVAN, S.; IVEY, J.; CLARK, R. J.; RILEY, G.; OWEN, H. L. Advancing software-defined networks: A survey. *IEEE Access*, v. 5, p. 25487–25526, 2017. ISSN 21693536.

CUNHA, D. C. P. da. PROPOSTA DE SUPORTE À QUALIDADE DE SERVIÇO BASEADA EM SDN: UM ESTUDO DE CASO PARA INSTITUIÇÕES DE ENSINO FEDERAL (UFPE). 2017.

CZIVA, R.; JOUËT, S.; STAPLETON, D.; TSO, F. P.; PEZAROS, D. P. IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT SDN-based Virtual Machine Management for Cloud Data Centers. p. 1–14, 2016.

DERBEL, H.; AGOULMINE, N.; SALAÜN, M. ANEMA: Autonomic network management architecture to support self-configuration and self-optimization in IP networks. *Computer Networks*, Elsevier B.V., v. 53, n. 3, p. 418–430, 2009. ISSN 13891286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2008.10.022>.

DEY, A. K. *Providing Architectural Support for Building Context-aware Applications*. Tese (Doutorado) — Georgia Institute of Technology, Atlanta, GA, USA, 2000. AAI9994400.

ELECTRICAL, I. of; IEEE, E. E. *IEEE 802 Numbers*. IEEE, 2019. Disponível em: <https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml>. Acesso em: 07 jan. 2020.

ELLIS, J.; PURSELL, C.; RAHMAN, J. *Voice, video, and data network convergence: architecture and design, from VoIP to wireless*. Elsevier, 2003. Pages 221–243 p. Disponível em: <https://doi.org/10.1016/B978-012236542-3/50011-1>.

FANG, V.; LÉVAI, T.; HAN, S.; RATNASAMY, S.; RAGHAVAN, B.; SHERRY, J.; BME, T. L. Evaluating Software Switches: Hard or Hopeless? 2018. Disponível em: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-136.html>.

FARIAS, F. N. N.; SALVADOR, P. M.; ABELÉM, A. J. G. vSDNEmul: Emulador de Redes Definidas Por Software Usando Contêineres. v. 9, 2018. Disponível em: <https://portaldeconteudo.sbc.org.br/index.php/wpeif/article/view/2320>.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The Road to SDN: An Intellectual History of Programmable Networks. *ACM Sigcomm Computer Communication*, v. 44, p. 87–98, 2014. ISSN 01464833. Disponível em: <http://dl.acm.org/citation.cfm?id=2602204.2602219{&}coll=DL{&}dl=ACM{&}CFID=429855848{&}CFTO>.

FERNANDES, E. L.; ROTHENBERG, C. E. OpenFlow 1.3 Software Switch. *Anais do 32º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos – SBRC*, p. 1021–1028, 2014.

FOREST, J.; RICKARD, N. 2019 Strategic Roadmap for Networking. *Intelligence*, Gartner, 2019.

GANDODHAR, P. S.; CHAWARE, S. M. Context aware computing systems: A survey. *Proceedings of the International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud), I-SMAC 2018*, IEEE, p. 605–608, 2019.

GHANNAMI, A.; SHAO, C. Efficient fast recovery mechanism in software-defined networks: multipath routing approach. In: IEEE. *Internet Technology and Secured Transactions (ICITST), 2016 11th International Conference for*. [S.l.], 2016. p. 432–435.

GÖRANSSON, P.; BLACK, C.; CULVER, T. Emerging Protocol, Controller, and Application Models. In: *Software Defined Networks (Second Edition)*. [S.l.: s.n.], 2017. cap. Chapter 7, p. 167–189. ISBN 9780128045558.

GÖRANSSON, P.; BLACK, C.; CULVER, T. How SDN Works. *Software Defined Networks*, p. 61–88, 2017.

GRAFANA. *Grafana Monitoring and Analytics*. 2014. Disponível em: <https://grafana.com.> Acesso em: 04 dez. 2019.

GRAY, K.; NADEAU, T. D. *SDN Software Defined Networks*. [s.n.], 2013. 384 p. ISSN 1098-6596. ISBN 0163-6804 VO - 51. Disponível em: <www.vm.ibm.com/vm40hist.pdf{\%}0Ahttps://www.rfc-editor.org/info/rfc7>.

GRAY, K.; NADEAU, T. D. IETF RELATED STANDARDS: NETMOD, NETCONF, SFC AND SPRING. In: *Network Function Virtualization*. [S.l.: s.n.], 2016. cap. Chapter 4, p. 77–102.

HAN, S.; SCIENCES, C. System Design for Software Packet Processing. 2019.

HENRICKSEN, K.; INDULSKA, J. Modelling and using imperfect context information. In: CITESEER. *PerCom Workshops*. [S.l.], 2004. p. 33–37.

HONGWEI, Q. *Crie uma rede parecida com Mininet - Mininet*. 2017. Disponível em: <https://www.hwchiu.com/setup-mininet-like-environment.html>. Acesso em: 5 jan. 2020.

IBM. Implementing IBM Software Defined Network for Virtual Environments. *White Paper*, n. June, 2013.

IQBAL, A.; PATTINSON, C.; KOR, A.-L. Performance monitoring of virtual machines (vms) of type i and ii hypervisors with snmpv3. In: IEEE. *2015 World Congress on Sustainable Technologies (WCST)*. [S.l.], 2015. p. 98–99.

JOSHI, J.; CHODISETTY, L. S.; RAVEENDRAN, V. A quality attribute-based evaluation of time-series databases for edge-centric architectures. *ACM International Conference Proceeding Series*, Part F1481, p. 98–103, 2019.

KAUR, K.; SINGH, J.; GHUMMAN, N. S. *Mininet as Software Defined Networking Testing Platform*. 2014. 3–6 p.

KHAN, M. A.; PETERS, S.; SAHINEL, D.; POZO-PARDO, F. D.; DANG, X.-T. Understanding autonomic network management: A look into the past, a solution for the future. *Computer Communications*, v. 122, p. 93 – 117, 2018. ISSN 0140-3664. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0140366417305327>.

KIM, H.; FEAMSTER, N. Improving network management with software defined networking. *IEEE Communications Magazine*, v. 51, n. 2, p. 114–119, 2013. ISSN 01636804.

KIMIYAMA, H.; MARUYAMA, M.; KOBAYASHI, M.; SAKAI, M. Uncompressed 8k-video system using high-speed video server system over ip network. In: *2015 Asia Pacific Conference on Multimedia and Broadcasting*. [S.l.: s.n.], 2015. p. 1–7.

KOLEY, B. The zero touch network. *IEEE Conference on Network and Service Management - CNSM*, 2016.

KRAUSE, M.; HOCHSTATTER, I. Challenges in modelling and using quality of context (qoc). In: SPRINGER. *International Workshop on Mobile Agents for Telecommunication Applications*. [S.l.], 2005. p. 324–333.

LANGE, S.; REINHART, L.; ZINNER, T.; HOCK, D.; GRAY, N.; TRAN-GIA, P. Integrating network management information into the SDN control plane. *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018*, IEEE, p. 1–9, 2018.

LARSON, R.; FARBER, B. *Estatística Aplicada*. 4. ed. São Paulo, Brasil: Pearson Prentice Hall, 2010.

LATIF, Z.; SHARIF, K.; LI, F.; KARIM, M. M.; WANG, Y. A Comprehensive Survey of Interface Protocols for Software Defined Networks. p. 1–30, 2019. Disponível em: <http://arxiv.org/abs/1902.07913>.

LEE, C.; SHIN, S. Fault tolerance for software-defined networking in smart grid. In: IEEE. *Big Data and Smart Computing (BigComp), 2018 IEEE International Conference on*. [S.l.], 2018. p. 705–708.

LI, C.; JI, Z.; WANG, F.; WANG, P.; WANG, Y.; ZHANG, Z. The network monitoring system based on cacti for east. In: IEEE. *2016 IEEE-NPSS Real Time Conference (RT)*. [S.l.], 2016. p. 1–5.

LIU, J.; XU, Q. Machine Learning in Software Defined Network. *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, IEEE, n. Itnec, p. 1114–1120, 2019.

MARTINI, B.; PAGANELLI, F.; MOHAMMED, A. A.; GHARBAOUI, M.; SGAMBELLURI, A.; CASTOLDI, P. Sdn controller for context-aware data delivery in dynamic service chaining. In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. [S.l.: s.n.], 2015. p. 1–5.

MASOUDI, R.; GHAFFARI, A. Software defined networks: A survey. *Journal of Network and Computer Applications*, v. 67, p. 1–25, 2016. ISSN 10958592.

MEGYESI, P.; BOTTA, A.; ACETO, G.; PESCAPÉ, A.; MOLNÁR, S. Challenges and solution for measuring available bandwidth in software defined networks. *Computer Communications*, Elsevier, v. 99, p. 48–61, 2017.

MOLNÁR, L.; PONGRÁCZ, G.; ENYEDI, G.; KIS, Z. L.; CSIKOR, L.; JUHÁSZ, F.; KORÖSI, A.; RÉTVÁRI, G. Dataplane specialization for high-performance OpenFlow software switching. *SIGCOMM 2016 - Proceedings of the 2016 ACM Conference on Special Interest Group on Data Communication*, p. 539–552, 2016.

MORENO, E. D.; OLIVEIRA, J. I. F. de. Architectural impact of the SVG-based graphical components in web applications. *Computer Standards and Interfaces*, Elsevier B.V., v. 31, n. 6, p. 1150–1157, 2009. Disponível em: <http://dx.doi.org/10.1016/j.csi.2008.12.007>.

MORSE, D. R.; ARMSTRONG, S.; DEY, A. K. The what, who, where, when, why and how of context-awareness. In: ACM. *CHI'00 Extended Abstracts on Human Factors in Computing Systems*. [S.l.], 2000. p. 371–371.

NAKAHODO, Y.; NAITO, T.; OKI, E. Implementation of smart-ospf in hybrid software-defined network. In: IEEE. *2014 4th IEEE International Conference on Network Infrastructure and Digital Content*. [S.l.], 2014. p. 374–378.

NAKAMURA, N.; KASHIMURA, N.; MOTOMURA, K. CMIP to SNMP translation technique based on rule description. *Proceedings of the International Conference on Computer Communications and Networks, ICCCN*, p. 266–271, 1995.

NARAYANAN, H. T.; ILANGOVAN, G.; NARAYANAN, S. Feasibility of SNMP OID compression. *Journal of King Saud University - Computer and Information Sciences*, King Saud University, v. 25, n. 1, p. 35–42, 2013. ISSN 13191578. Disponível em: <http://dx.doi.org/10.1016/j.jksuci.2012.05.006>.

NICOLAOU, C. An architecture for real-time multimedia communication systems. *IEEE Journal on Selected Areas in Communications*, v. 8, n. 3, p. 391–400, April 1990. ISSN 0733-8716.

NUNES, B. A. A.; MENDONCA, M.; NGUYEN, X.-N.; OBRACZKA, K.; TURLETTI, T. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, IEEE, v. 16, n. 3, p. 1617–1634, 2014.

NUNES, B. A. A.; MENDONCA, M.; NGUYEN, X. N.; OBRACZKA, K.; TURLETTI, T. A survey of software-defined networking. past, present, and future of programmable networks. *IEEE Communications Surveys Tutorials*, v. 16, n. 3, p. 1617–1634, 2014.

ONF. ONF TS-025 OpenFlow Switch Specification. V 1.5.1, p. 1–283, 2015.

OPEN NETWORKING FOUNDATION. Software-Defined Networking: The New Norm for Networks [white paper]. p. 1–12, 2012.

OPEN NETWORKING FOUNDATION. *OpenFlow Switch Specification 1.5.1*. Open Networking Foundation, 2015. 277 p. Disponível em: <https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>.

OSINSKI, T.; DANDOUSH, A. XMPP as a scalable multi-tenants isolation solution for ONOS-based Software-Defined Cloud Networks. *14th International Conference on Network and Service Management, CNSM 2018 and Workshops, 1st International*

Workshop on High-Precision Networks Operations and Control, HiPNet 2018 and 1st Workshop on Segment Routing and Service Function Chaining, SR+SFC 2, p. 300–302, 2018.

PIOLINK. Open vSwitch . n. November, p. 54–55, 2013.

PRADEEP, P.; KRISHNAMOORTHY, S. The MOM of context-aware systems: A survey. *Computer Communications*, Elsevier B.V., v. 137, 2019. ISSN 1873703X. Disponível em: <https://doi.org/10.1016/j.comcom.2019.02.002>.

PREETH, E. N.; MULERICKAL, J. P.; PAUL, B.; SASTRI, Y. Evaluation of Docker containers based on hardware utilization. *2015 International Conference on Control, Communication and Computing India, ICCC 2015*, n. November, p. 697–700, 2016.

QUTTOUM, A. N.; ALSARAIREH, H.; KHADJIEV, O.; ALQUDAIMAT, M. SMS: Smart Management Scheme via Software Defined Networks. *Procedia Computer Science*, Elsevier B.V., v. 134, p. 303–308, 2018. ISSN 18770509. Disponível em: <https://doi.org/10.1016/j.procs.2018.07.175>.

RAHIMI, R.; VEERARAGHAVAN, M.; NAKAJIMA, Y.; TAKAHASHI, H.; NAKAJIMA, Y.; OKAMOTO, S.; YAMANAKA, N. A high-performance OpenFlow software switch. *IEEE International Conference on High Performance Switching and Routing, HPSR*, v. 2016-July, p. 93–99, 2016. ISSN 23255609.

RAZA, M.; CHOWDHURY, S.; ROBERTSON, W. SDN based emulation of an academic networking testbed. *Canadian Conference on Electrical and Computer Engineering*, v. 2016-Octob, p. 1–6, 2016. ISSN 08407789.

REN, T.; XU, Y. Analysis of the New Features of OpenFlow 1.4. n. Icieac, p. 73–77, 2014.

RENITA, J.; ELIZABETH, N. E. Network's server monitoring and analysis using Nagios. *Proceedings of the 2017 International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2017*, v. 2018-Janua, p. 1904–1909, 2018.

RNP, R. N. de Ensino e P. *RedeCOMEP*. 2007. Disponível em: <https://www.rnp.br/sistema-rnp/redecomep>. Acesso em: 21 ago. 2019.

RNP, R. N. de Ensino e P. *RNP realiza demonstração em 4K configurada por redes definidas por software nos EUA*. IEEE, 2015. Disponível em: <https://www.rnp.br/noticias/rnp-realiza-demonstracao-em-4k-configurada-por-redes-definidas-por-software-nos-eua>. Acesso em: 18 jan. 2020.

RNP, R. N. de Ensino e P. *Demonstração testa tráfego internacional por redes definidas por software*. IEEE, 2016. Disponível em: <https://www.rnp.br/noticias/demonstracao-testa-trafego-internacional-por-redes-definidas-por-software>. Acesso em: 18 jan. 2020.

RNP, R. N. de Ensino e P. *Conheça as vantagens em migrar para a tecnologia SDN*. IEEE, 2017. Disponível em: <https://www.rnp.br/noticias/conheca-vantagens-em-migrar-para-tecnologia-sdn>. Acesso em: 18 jan. 2020.

RNP, R. N. de Ensino e P. *Redecompep*. RNP, 2019. Disponível em: <https://www.rnp.br/noticias/rnp-realiza-demonstracao-em-4k-configurada-por-redes-definidas-por-software-nos-eua>. Acesso em: 3 jan. 2020.

ROJAS, E. From software-defined to human-defined networking: Challenges and opportunities. *IEEE Network*, v. 32, n. 1, p. 179–185, Jan 2018. ISSN 0890-8044.

ROJAS, E.; DORIGUZZI-CORIN, R.; TAMUREJO, S.; BEATO, A.; SCHWABE, A.; PHEMIUS, K.; GUERRERO, C. Are we ready to drive software-defined networks? a comprehensive survey on management tools and techniques. *ACM Computing Surveys (CSUR)*, ACM, v. 51, n. 2, p. 27, 2018.

ROSSEM, S. V.; CAI, X.; CERRATO, I.; DANIELSSON, P.; NÉMETH, F.; PECHENOT, B.; PELLE, I.; RISSO, F.; SHARMA, S.; SKÖLDSTRÖM, P.; JOHN, W. NFV service dynamicity with a DevOps approach: Insights from a use-case realization. *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network and Service Management*, p. 674–679, 2017.

ROTHENBERG, C. E.; NASCIMENTO, M. R.; SALVADOR, M. R.; MAGALHÃES, M. F. OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. *Cadernos CPqD Tecnologia*, v. 7, n. 1, p. 65–75, 2011.

RYU. *Ryu application API*. 2014. Disponível em: <https://ryu.readthedocs.io/en/latest/ryu_app_api.html?highlight=eventofpmsgbase#ryu.controller.ofp_event.EventOFPMsgBase>. Acesso em: 02 jan. 2020.

SALSANO, S.; BLEFARI-MELAZZI, N.; DETTI, A.; MORABITO, G.; VELTRI, L. Information centric networking over SDN and OpenFlow: Architectural aspects and experiments on the OFELIA testbed. *Computer Networks*, Elsevier B.V., v. 57, n. 16, p. 3207–3221, 2013. ISSN 13891286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2013.07.031>.

SAMBA, A. A Network Management Framework for SDN. *Modeling and Simulation Tools for Emerging Telecommunication Networks*, p. 179–200, 2018.

SANDUSKY, R. J. Network management. *Understanding Information Retrieval Systems: Management, Types, and Standards*, p. 149–160, 2011. ISSN 01419331.

SCHMIDT, C. Context-aware computing. *Berlin Institute of Technology*, 2011.

SHALIMOV, A.; ZUIKOV, D.; ZIMARINA, D.; PASHKOV, V.; SMELIANSKY, R. Advanced study of SDN/OpenFlow controllers. *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia on - CEE-SECR '13*, n. October, p. 1–6, 2013. Disponível em: <http://dl.acm.org/citation.cfm?id=2556610.2556621>.

SHU, Z.; WAN, J.; LIN, J.; WANG, S.; LI, D.; RHO, S.; YANG, C. Traffic engineering in software-defined networking: Measurement and management. *IEEE Access*, IEEE, v. 4, p. 3246–3256, 2016.

SILVA, E. F. da. SDNMonitor: um serviço de monitoramento de tráfego em redes definidas por software. *Ufs*, 2016.

SILVA, E. H. A. da; SOUZA, A. C. d. S. Gerenciador Simplificado de Redes Baseado no Protocolo SNMP. 2016.

SILVA, N. S. da. Redes Comunitárias: Uma Construção Sociotécnica de Politicas de Comunicação. p. 1–238, 2016. Disponível em: <http://ebooks.cambridge.org/ref/id/CBO9781107415324A009>.

SMITH, M.; DVORKIN, M.; LARIBI, V.; PANDEY, V.; GARG, P.; WEIDENBACHER, N. *OpFlex Control Protocol*. Internet Engineering Task Force - IETF, 2016. Disponível em: <https://tools.ietf.org/pdf/draft-smith-opflex-03.pdf>. Acesso em: 01 jan. 2020.

SYAHIDILLAH, W. M. *Multipath Routing with Load Balancing using RYU OpenFlow Controller*. Wordpress, 2018. Disponível em: <https://wildanmsyah.wordpress.com/2018/01/13/multipath-routing-with-load-balancing-using-ryu-openflow-controller/>. Acesso em: 22 mar. 2018.

TEAM, R. project. SDN RYU-Controller Framework. 2014. Disponível em: <https://www.ietf.org/proceedings/82/sdn.html>.

TOY, M. Self-managed networks with fault management hierarchy. Elsevier Masson SAS, v. 36, n. C, p. 373–380, 2014. Disponível em: <http://dx.doi.org/10.1016/j.procs.2014.09.008>.

TUNCER, D.; CHARALAMBIDES, M.; CLAYMAN, S.; PAVLOU, G. Adaptive resource management and control in software defined networks. *IEEE Transactions on Network and Service Management*, v. 12, n. 1, p. 18–33, 2015. ISSN 19324537.

VAUGHAN-NICHOLS, S. J. OpenFlow: The Next Generation of the Network? *Computer*, v. 44, n. 8, p. 13–15, 2011. ISSN 0018-9162.

VEENA, S.; PAL, C.; RUSTAGI, R. P.; MURTHY, K. N. B. A Framework for Implementing Realistic Custom Network Topology in {Mininet}. *International Journal of Science and Research (IJSR)*, v. 3, n. 7, p. 1316–1323, 2014.

VIEIRA, V. Computação Sensível ao Contexto. 2011.

VIEIRA, V.; TEDESCO, P.; SALGADO, A. C. Modelos e processos para o desenvolvimento de sistemas sensíveis ao contexto. *André Ponce de Leon F. de Carvalho, Tomasz Kowaltowski.(Org.). Jornadas de Atualização em Informática*, p. 381–431, 2009.

WALEED, S.; FAIZAN, M.; IQBAL, M.; ANIS, M. I. Demonstration of single link failure recovery using bellman ford and dijikstra algorithm in sdn. In: IEEE. *2017 International Conference on Innovations in Electrical Engineering and Computational Technologies (ICIEECT)*. [S.l.], 2017. p. 1–4.

WEISER, M. The computer for the 21st century. *Scientific American (International Edition)*, v. 265, n. 3, p. 66–75, 1991. ISSN 0036-8733. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=993141>.

WETTE, P.; DRÄXLER, M.; SCHWABE, A.; WALLASCHEK, F.; ZAHRAEE, M. H.; KARL, H. Maxinet: Distributed emulation of software-defined networks. In: IEEE. *2014 IFIP Networking Conference*. [S.l.], 2014. p. 1–9.

ZHOU, D.; YAN, Z.; LIU, G.; ATIQUZZAMAN, M. An Adaptive Network Data Collection System in SDN. *IEEE Transactions on Cognitive Communications and Networking*, IEEE, PP, n. c, p. 1, 2019. ISSN 23327731.

ZHU, A. *OpenFlow Switch: What Is It and How Does it Work?* Medium, 2018. Disponível em: <https://medium.com/@AriaZhu/openflow-switch-what-is-it-and-how-does-it-work-7589ea7ea29c>. Acesso em: 2 jan. 2020.

APÊNDICE A – CÓDIGO DO PLUGIN SVG

Código A.1 – Código XML da Topologia da Rede Emulada

```
<svg xmlns="http://www.w3.org/2000/svg" width="100%" height="100%" viewBox="-600 0 1000 1000">

    <text transform="matrix(0.7223 0 0 1 438.828 395.6271)" font-family="'ArialMT'" font-size="80.7375">IFPE</text>
    <text transform="matrix(0.7223 0 0 1 -168.828 95.6271)" font-family="'ArialMT'" font-size="70.7375">CPOR</text>
    <text transform="matrix(0.7223 0 0 1 -1198.828 395.6271)" font-family="'ArialMT'" font-size="70.7375">FUNDAJ</text>
    <text transform="matrix(0.7223 0 0 1 -188.828 949.6271)" font-family="'ArialMT'" font-size="70.7375">PoP-PE</text>

    <!--- criacao dos switch OvS CPOR--->
    <g id="nodes">
    <g id="CPOR" transform="translate(-400 0) rotate(0 0 0) scale(1.5)">
        <g id="circle_1" transform="translate(0,0)">
    <svg height="123" viewBox="0 0 250 250" width="132" preserveAspectRatio="none">
      <circle cx="124" cy="125" r="104" style="fill:#c0c0c0;stroke:#000;stroke-width:2"/>
    </svg>
  </g>
  </g>
   <!--- criacao dos switch OvS POP--->
  <g id="POP" transform="translate(-400 800) rotate(0 0 0) scale(1.5)">
      <g id="circle_1" transform="translate(0,0)">
    <svg height="123" viewBox="0 0 250 250" width="132" preserveAspectRatio="none">
      <circle cx="124" cy="125" r="104" style="fill:#c0c0c0;stroke:#000;stroke-width:2"/>
    </svg>
  </g>
  </g>
     <!--- criacao dos switch OvS IFPE--->
  <g id="IFPE" transform="translate(400 400) rotate(0 0 0) scale(1.5)"><g id="circle_1"
      transform="translate(0,0)">
    <svg height="123" viewBox="0 0 250 250" width="132" preserveAspectRatio="none">
      <circle cx="124" cy="125" r="104" style="fill:#c0c0c0;stroke:#000;stroke-width:2"/>
    </svg>
  </g>
  </g>
   <!--- criacao dos switch OvS FUNDAJ--->
  <g id="FUNDAJ" transform="translate(-1200 400) rotate(0 0 0) scale(1.5)"><g id="circle_1"
      transform="translate(0,0)">
    <svg height="123" viewBox="0 0 250 250" width="132" preserveAspectRatio="none">
      <circle cx="124" cy="125" r="104" style="fill:#c0c0c0;stroke:#000;stroke-width:2"/>

    </svg>
  </g>
  </g>
  </g>

  <!--- Setas representando os links --->
  <g id="links">
  <g id="ARROW_LEFT_CPOR" fill="none" stroke="#000" transform="translate(-570 150) rotate(59 0
```

```
      0) scale(0.7)">
45      <g>
46        <svg height="265" viewBox="0 0 400 994" width="173" preserveAspectRatio="none">
47            <g>
48                <path stroke-width="10" d="M201,994 l199,-387 -132,0 0,-607 -135,0 0,607
                      -133,0 201,387z"/>
49            </g>
50
51        </svg>
52      </g>
53  </g>
54  <g id="ARROW_UP_FUNDAJ" fill="none" stroke="#000" stroke-width="10" transform="translate
      (-750 250) rotate(61 0 0) scale(0.7)">
55      <g>
56        <svg height="263" viewBox="0 0 400 994" width="171" preserveAspectRatio="none">
57            <g>
58              <path  d="M201,0 l199,387 -132,0 0,607 -135,0 0,-607 -133,0 201,-387z"/>
59          </g>
60        </svg>
61      </g>
62  </g>
63  <g id="ARROW_DOWN_FUNDAJ" fill="none" transform="translate(-910 660) rotate(-59 0 0) scale
      (0.7)">
64      <g>
65        <svg height="265" viewBox="0 0 400 994" width="173" preserveAspectRatio="none">
66           <g>
67              <path stroke="#000" stroke-width="10" d="M201,994 l199,-387 -132,0 0,-607 -135,0
                    0,607 -133,0 201,387z"/>
68         </g>
69        </svg>
70      </g>
71  </g>
72  <g id="ARROW_LEFT_POP" fill="none" stroke="#000" stroke-width="10" transform="translate(-740
       759) rotate(-61 0 0) scale(0.7)">
73      <g>
74        <svg height="263" viewBox="0 0 400 994" width="171" preserveAspectRatio="none">
75            <g>
76              <path  d="M201,0 l199,387 -132,0 0,607 -143,0 0,-607 -123,0 201,-387z"/>
77          </g>
78        </svg>
79      </g>
80  </g>
81  <g id="ARROW_RIGHT_CPOR" fill="none" stroke="#000" stroke-width="10" transform="translate
      (-90 260) rotate(-63 0 0) scale(0.7)">
82      <g>
83        <svg height="265" viewBox="0 0 400 994" width="173" preserveAspectRatio="none">
84            <g>
85              <path  d="M201,994 l199,-387 -132,0 0,-607 -135,0 0,607 -133,0 201,387z"/>
86          </g>
87        </svg>
88      </g>
89  </g>
90      <g id="ARROW_UP_IFPE" fill="none" stroke="#000" stroke-width="10" transform="translate
          (80 340) rotate(-59 0 0) scale(0.7)">
91          <g>
92            <svg height="263" viewBox="0 0 400 994" width="171" preserveAspectRatio="none">
93                <g>
```

```
94                  <path  d="M201,0 l199,387 -132,0 0,607 -135,0 0,-607 -133,0 201,-387z"/>
95              </g>
96            </svg>
97          </g>
98      </g>
99      <g id="ARROW_DOWN_IFPE" fill="none" stroke="#000" stroke-width="10" transform="translate
            (220 550) rotate(61 0 0) scale(0.7)">
100         <g>
101           <svg height="265" viewBox="0 0 400 994" width="173" preserveAspectRatio="none">
102               <g>
103                 <path  d="M201,994 l199,-387 -132,0 0,-607 -135,0 0,607 -133,0 201,387z"/>
104             </g>
105
106           </svg>
107         </g>
108     </g>
109     <g id="ARROW_RIGHT_POP" fill="none" stroke="#000" stroke-width="10" transform="translate
            (40 650) rotate(59 0 0) scale(0.7)">
110         <g>
111           <svg height="263" viewBox="0 0 400 994" width="171" preserveAspectRatio="none">
112               <g>
113                 <path  d="M201,0 l199,387 -132,0 0,607 -135,1 0,-607 -133,0 201,-387z"/>
114             </g>
115           </svg>
116         </g>
117     </g>
118 </g>
119    <!--- caixa de legendas --->
120 <g id="legenda">
121     <path fill="none" d="M572.316,503.164"/>
122
123     <rect x="514.625" y="622.692" fill="#A4A4A4" width="32.961" height="32.961"/> <!--- grey
            --->
124     <rect x="514.625" y="755.136"  width="32.961" height="32.961"/>  <!--- black --->
125     <rect x="514.625" y="804.196" fill="green" width="32.961" height="32.961"/> <!--- blue
            --->
126     <rect x="514.625" y="855.225" fill="orange" width="32.961" height="32.961"/> <!--- green
            --->
127     <rect x="514.625" y="903.286" fill="red" width="32.961" height="32.961"/> <!---
            Congestionamento Moderado --->
128     <!---<rect x="514.625" y="954.314" fill="#F7931E" width="32.961" height="32.961"/> "#3
            FA9F5"--->
129     <!---<rect x="554.625" y="965.343" fill="#FF2C50" width="25.737" height="25.961"/>--->
130     <text transform="matrix(0.9639 0 0 1 564.1167 603.1608)" font-family="'ArialMT'" font-
            size="37.6441">Legenda</text>
131     <text transform="matrix(0.9639 0 0 1 564.363 653.6788)" font-family="'ArialMT'" font-
            size="37.6441">OvS</text>
132     <text transform="matrix(0.9639 0 0 1 564.4767 729.1608)" font-family="'ArialMT'" font-
            size="37.6441">Utilizacao (\%)</text>
133
134     <text transform="matrix(0.9639 0 0 1 564.363 785.9786)" font-family="'ArialMT'" font-
            size="37.6441">Link Down</text>
135     <text transform="matrix(0.9639 0 0 1 564.363 837.0195)" font-family="'ArialMT'" font-
            size="37.6441">Ate 50% (Normal)</text>
136     <text transform="matrix(0.9639 0 0 1 564.363 883.0604)" font-family="'ArialMT'" font-
            size="37.6441">50% - 80% (Congestionamento Moderado)</text>
137     <text transform="matrix(0.9639 0 0 1 564.363 929.1027)" font-family="'ArialMT'" font-
```

```
size="37.6441">Acima de 80% (Congestionamento Pesado)</text>
```

Código A.2 – Código JavaScript do PLugin SVG executado a cada *Refresh*

```
1
2  //ctrl instacia para acessar os dados do influx
3  //dados retornados linha de objetos que representam os valores das portas
4  var links_data = ctrl.data[0].rows;
5  links_data.forEach(linkUsage);//primeira funcao a ser chamada linkUsage()
6
7  function linkUsage(link)
8  {
9      inicio = performance.now();
10
11     //#####################################################################
12     //CONDICIONAL OVS1 PORTA 2
13     if(link.ovs == '1' && link.port == '2')
14     {
15       setLinkColor('#ARROW_LEFT_POP',link.trans_band,1000); //ok
16       setLinkColor('#ARROW_LEFT_POP',link.rec_band,1000);   //ok
17     }//PORTA 3
18     if(link.ovs == '1' && link.port == '3')
19     {
20       setLinkColor('#ARROW_RIGHT_POP',link.trans_band,1000); //ok
21       setLinkColor('#ARROW_RIGHT_POP',link.rec_band,1000);   //ok
22     }
23     //#####################################################################
24
25     //#####################################################################
26     //   ---------CONDICIONAL OVS2 PORTA 2
27     if(link.ovs == '2' && link.port == '2'  && link.trans_band > 0)
28     {
29       setLinkColor('#ARROW_UP_FUNDAJ',link.trans_band,1000);
30
31     }else if(link.ovs == '2' && link.port == '2' && link.rec_band > 0){ //foi necessario
           colocar a porta 3 aninhada
32         setLinkColor('#ARROW_UP_FUNDAJ',link.rec_band,1000);             //pq nao estava
               funcionado dentro de uma mesma condicional
33         }
34      if(link.ovs == '2' && link.port == '2' && link.rec_band == '0' && link.trans_band == '0')
            { //foi necessario utilizar essa condicional
35                 s.select('#ARROW_UP_FUNDAJ').attr({'fill': 'black'});                      //
                       pq ele nao estava pintando de preto na ultima condicional deste codigo
36     }                                                                                      //
           essa mesma ideia pode ser vista nos ovs's abaixo
37
38     //   ---------PORTA 3
39     if(link.ovs == '2' && link.port == '3' && link.trans_band > 0)
40     {
41       setLinkColor('#ARROW_DOWN_FUNDAJ',link.trans_band,1000);
42
43     } else if(link.ovs == '2' && link.port == '3' && link.rec_band > 0){ //foi necessario
           colocar a porta 3 aninhada
44             setLinkColor('#ARROW_DOWN_FUNDAJ',link.rec_band,1000);         //pq nao estava
                   funcionado dentro de uma mesma condicional
45         }
```

```
46      if(link.ovs == '2' && link.port == '3' && link.rec_band == '0' && link.trans_band == '0'){
            // nao estava pintando de preto na ultima condicional deste codigo
47                  s.select('#ARROW_DOWN_FUNDAJ').attr({'fill': 'black'});
48      }
49      //#####################################################################
50
51      //#####################################################################
52      //   ----------CONDICIONAL OVS3 PORTA 2
53      if(link.ovs == '3' && link.port == '2' && link.trans_band > 0)
54      {
55          setLinkColor('#ARROW_RIGHT_CPOR',link.trans_band,1000);
56
57      }else if(link.ovs == '3' && link.port == '2' && link.rec_band > 0){//foi necessario
            colocar a porta 3 aninhada
58              setLinkColor('#ARROW_RIGHT_CPOR',link.rec_band,1000); //pq nao estava funcionado
                    dentro da mesma condicional
59          }
60
61      if(link.ovs == '3' && link.port == '2' && link.rec_band == '0' && link.trans_band == '0'){
            // nao estava pintando de preto na ultima condicional deste codigo
62                  s.select('#ARROW_RIGHT_CPOR').attr({'fill': 'black'});
63                  //situacao 1 queda de enlace
64                  //var end_time =  performance.now() - inicio;
65                  //var text = s.text(60, 80, end_time + ' ms');
66                  //var time_interval1 = end_time + 10;
67                  //var text1 = s.text(70, 110, time_interval1 + ' ms');
68                  //var time_interval2 = end_time + 40;
69                  //var text2 = s.text(80, 135, time_interval2 + ' ms');
70      }
71      // --------------PORTA 3
72      if(link.ovs == '3' && link.port == '3' && link.trans_band > 0)
73      {
74        setLinkColor('#ARROW_LEFT_CPOR',link.trans_band,1000);
75
76      }else if(link.ovs == '3' && link.port =='3' && link.rec_band > 0){//foi necessario colocar
            a porta 3 aninhada
77          setLinkColor('#ARROW_LEFT_CPOR',link.rec_band,1000);  //pq nao estava funcionado
                dentro da mesma condicional
78
79          }
80      if(link.ovs == '3' && link.port == '3' && link.rec_band == '0' && link.trans_band == '0'){
            // nao estava pintando de preto na ultima condicional deste codigo
81          s.select('#ARROW_LEFT_CPOR').attr({'fill': 'black'});
82
83      }
84      //#####################################################################
85
86
87      //#####################################################################
88      //    -------------CONDICIONAL OVS4 PORTA 2
89      if(link.ovs == '4' && link.port == '2' && link.trans_band > 0)
90      {
91        setLinkColor('#ARROW_DOWN_IFPE',link.trans_band,1000);
92
93      } else if (link.ovs == '4' && link.port == '2' && link.rec_band > 0){//foi necessario
            colocar a porta 2 aninhada
94          setLinkColor('#ARROW_DOWN_IFPE',link.rec_band,1000); //pq nao estava funcionado dentro
```

```
            da mesma condicional
95      }
96      if(link.ovs == '4' && link.port == '2' && link.rec_band == '0' && link.trans_band == '0'){
            // nao estava pintando de preto na ultima condicional deste codigo
97          s.select('#ARROW_DOWN_IFPE').attr({'fill': 'black'});
98      }
99      //    ---------------PORTA 3
100     if(link.ovs == '4' && link.port == '3' && link.trans_band > 0)
101     {
102       setLinkColor('#ARROW_UP_IFPE',link.trans_band,1000);
103
104     } else if(link.ovs == '4' && link.port == '3' && link.rec_band > 0){//foi necessario
            colocar a porta 3 aninhada
105         setLinkColor('#ARROW_UP_IFPE',link.rec_band,1000);   //pq nao estava funcionado dentro
                da mesma condicional
106         }
107     if(link.ovs == '4' && link.port == '3' && link.rec_band == '0' && link.trans_band == '0'){
            // nao estava pintando de preto na ultima condicional deste codigo
108         s.select('#ARROW_UP_IFPE').attr({'fill': 'black'});
109     }
110 }
111
112 //Funcao para pintar as setas de acordo com a porcetagem de utilizacao;
113 function setLinkColor(linkID,traffic,link_cap)
114 {
115     var inicio = performance.now();
116     var l = s.select(linkID);
117
118     var util = parseInt((traffic/link_cap)*100); // porcentagem_de_uso
119     //var util = ((traffic/1000000)/link_cap)*100; //%usage para link que nao foi convertido
            em megabit 1000000
120
121     if(util === 0){
122         l.attr({'fill':'black'});
123         }
124     if(util > 0 && util <= 50){
125         l.attr({'fill': 'green'});
126         }
127     if(util > 50 && util < 60){
128         l.attr({'fill': 'orange'});
129         }
130     if(util >= 80){
131         //var text2 = s.text(70, 155, util);
132         l.attr({'fill': 'red'});
133         if(linkID == '#ARROW_RIGHT_CPOR' ){
134
135                 //situacao 2 congestionamento pesado exibe o tempo na tela
136                 //var end_time =  performance.now() - inicio;
137                 //var text = s.text(60, 80, end_time + ' ms');
138                 //var time_interval1 = end_time + 10;
139                 //var text1 = s.text(70, 110, time_interval1 + ' ms');
140                 //var time_interval2 = end_time + 40;
141                 //var text2 = s.text(80, 135, time_interval2 + ' ms');
142         }
143
144     }
145
```

```
146 }
```

Código A.3 – Código JavaScript do plugin SVG executado apenas um vez.

```
1  s = Snap(svgnode);
2
3
4  //s.select('#links').attr({'stroke': 'black', 'fill':'black'});
5  //s.select('#nodes').attr('fill','green');
6
7  links = s.select('#links').selectAll('g');
8  //var text =  s.text(70, 135, );
9
10
11 iframe = document.createElement('iframe');
12 iframe.style.display = "block";
13 iframe.style.width = 640+"px";
14 iframe.style.height = 480+"px";
15 iframe.style.display = "none";
16 document.body.appendChild(iframe);
17
18 links.forEach(function(el)
19 {
20     el.click(function()
21     {
22         var id = el.toString().split('id="')[1].split('"')[0]; //tratamento da string para
               retornar o id do link do arquivo SVG
23         var url;
24
25         url = get_graph_url(id);
26         //url1 = "/d/" + url.split('/d-solo/')[1];
27
28
29         if(url !== '')
30             window.location.href = url;
31     });
32
33     el.mouseover(function()
34     {
35         var id = el.toString().split('id="')[1].split('"')[0]; //tratamento da string para
               retornar o id do SVG
36
37         var url;
38         url = get_graph_url(id); //chama a funcao para retornar a parte final da url do
               grafico do grafana
39
40         if(url !== '')
41         {
42             iframe.style.position = "absolute";
43             iframe.style.top= (event.clientY +  5) + "px";
44             iframe.style.left= (event.clientX + 5) + "px";
45             iframe.style.zIndex= "101";
46             iframe.src = window.location.protocol + '//' + window.location.hostname + ':' +
                   window.location.port + url;
47             //iframe.src = window.location.href + url;
48             iframe.style.display = "block";
49             //var text = s.text(70, 35, iframe.src);
```

```
50              //text.attr({'font-size':20});
51          }
52      });
53
54       el.mouseout(function()
55      {
56          iframe.style.display = "none";
57      });
58  });
59
60  //funcao para retornar o endereco dos graficos do grafana
61  function get_graph_url(id)
62  {
63      // links
64      if(id == 'ARROW_RIGHT_CPOR') //id do SVG
65         return '/d/FZ3_lwtWk/ovs_3-port-2?orgId=1&fullscreen&panelId=2'; //parte final do
               endereco do grafico no grafana
66
67      if(id == 'ARROW_UP_IFPE')
68         return '/d/Mx3SilpZk/ovs_4-port-3?orgId=1&fullscreen&panelId=2';
69
70      if(id == 'ARROW_DOWN_FUNDAJ')
71         return '/d/mavB_wpWz/ovs_2-port-3?orgId=1&refresh=5s';
72
73      if(id == 'ARROW_LEFT_POP')
74         return '/d/_9GqQwpZk/ovs_1-port-3?orgId=1&refresh=5s&fullscreen&panelId=2';
75
76      if(id == 'ARROW_RIGHT_POP')
77         return '/d/kbO_wwpZz/ovs_1-port-2?orgId=1&fullscreen&panelId=2';
78
79      if(id == 'ARROW_DOWN_IFPE')
80         return '/d/efQ7i_pZk/ovs_4-port-2?orgId=1&fullscreen&panelId=2';
81
82      if(id == 'ARROW_LEFT_CPOR')
83         return '/d/VEn6z_pWz/ovs_3-port-3?orgId=1&fullscreen&panelId=2';
84
85      if(id == 'ARROW_UP_FUNDAJ')
86         return '/d/O_W1_wpWk/ovs_2-port-2?orgId=1&fullscreen&panelId=2';
87
88      return '';
89
90  }
```