LST: Testbed Emulado Leve para Redes SDN Aplicado ao Contexto de Segurança

Alexandre M. Kaihara[1], Lucas Bondan[2], João J. C. Gondim[1], Gabriel S. Rodrigues[1], Marcelo A. Marotta[1], Genaína N. Rodrigues[1]

[1]Departamento de Ciência da Computação – Universidade de Brasília (UnB)

[2]Rede Nacional de Ensino e Pesquisa (RNP)

{alexandre.kaihara,siqueira.rodrigues}@aluno.unb.br,lucas.bondan@rnp.br

{gondim, marcelo.marotta, genaina}@unb.br

Abstract. *There are many Emulated Testbeds proposed for Software-Defined Networking (SDN). Nonetheless, there are still few studies that focus on security and some of them are restricted to the context in which it was designed and/or do not always meet requirements such as easy installation and configuration, topology configurability, and low cost, harming the reusability of the tool. Aiming to fulfill the identified gap, we propose the Lightweight SDN Testbed (LST), an easy-to-use tool for local executions for SDN and security studies. To reduce the computational cost as well as provide configurability to the proposed testbed, LST was developed using Docker, Docker Compose, Python, and Open vSwitch.*

Resumo. *Existem muitos testbeds emulados propostos para experimentação de Redes Definidas por Software, do inglês Software-Defined Networking (SDN). Entretanto, poucas propostas focam em segurança, as quais geralmente são restritas ao contexto de aplicação para o qual foram desenvolvidas e/ou nem sempre atendem a requisitos como facilidade de instalação e configuração, configurabilidade da topologia e baixo custo, prejudicando a reusabilidade da ferramenta. Visando preencher a lacuna de pesquisa identificada, o presente trabalho apresenta o Lightweight SDN Testbed (LST), uma ferramenta leve para execuções locais de aplicações para SDN no contexto de segurança. Para diminuir o custo computacional bem como propiciar configurabilidade ao testbed proposto, o LST foi desenvolvido utilizando Docker, Docker Compose, Python e Open vSwitch.*

1. Introdução

Redes Definidas por Software é um paradigma de rede em crescimento que proporciona a programabilidade do plano de controle de redes, aprimorando a flexibilidade do monitoramento e gestão de redes. Muitos estudos em SDN têm voltado esforços para a melhoria da segurança, como por exemplo na detecção e mitigação de ataques e visando tornar a própria arquitetura SDN mais segura [D. Kreutz 2015]. Com o surgimento de novas propostas aos desafios de segurança, surge a necessidade de validar tais soluções. Para tal validação, tornam-se importantes meios que permitam a reprodução dos cenários de ataques e dos sistemas.

Um dos meios de destaque para realização e reprodução de cenários de redes e ataques são os *testbeds*, que auxiliam na criação de experimentos sobre infraestruturas físicas

utilizando virtualização das mesmas. Esses *testbeds* permitem a criação de cenários experimentais de rede seguindo rigorosos critérios que permitem identificar vulnerabilidades, analisar o impacto de ataques e testar mecanismos de defesa através de um ambiente emulado adequado [M. Khan 2020]. No entanto, em vários estudos de segurança são criados *testbeds* próprios [K. Wrona 2017], os quais geralmente são restritos ao contexto de aplicação para o qual foram desenvolvidos, [S. Srisawai 2018] [M. Ring 2017], limitando o reuso da ferramenta.

Em tais *testbeds*, são importantes requisitos como fácil instalação e configuração, configurabilidade da topologia, aplicabilidade em diferentes domínios e baixo consumo de recursos computacionais. Existem propostas que atendem a tais requisitos (*e.g.* [E. Petersen 2020, G. Bonofiglio 2018]), mas que não foram desenvolvidas com foco em aplicações de segurança. Por outro lado, ferramentas desenvolvidas exclusivamente para experimentos em segurança [M. Khan 2020, M. Šimon 2016] geralmente estão limitadas quanto ao contexto de uso e quanto à aplicabilidade em redes de última geração que utilizam, por exemplo, SDN. Portanto, no melhor dos conhecimentos dos autores, faltam ferramentas capazes de uma maior configurabilidade da topologia, facilidade de instalação e configuração, menor consumo de recursos computacionais e baixo custo, que possam ser utilizadas para experimentar aplicações de segurança, mas sem estarem limitadas a apenas esse contexto.

Visando tais requisitos, neste trabalho é proposta uma ferramenta para emulação de *testbeds* SDN com foco em segurança, utilizando a tecnologia de contêineres, chamada *Lightweight SDN Testbed* (LST). LST permite testar tanto aplicações em SDN (*e.g.* novas políticas e comportamentos do controlador) quanto aplicações focadas em segurança, como por exemplo, no teste de vulnerabilidades do controlador a DoS e/ou de sistemas de detecção de intrusão em redes SDN. Para a montagem dos cenários de estudo, o LST disponibiliza imagens pré-construídas que simulam o ambiente de uma pequena organização, onde é possível gerar fluxos benignos e maliciosos para testar os mecanismos de segurança desenvolvidos. Utilizando virtualização baseada em contêineres através do Docker, é possível alcançar maior configurabilidade nos experimentos, bem como garantir o isolamento dos contêineres e das aplicações desenvolvidas.

O presente trabalho está organizado da seguinte forma: na Seção 2, são abordados trabalhos relacionados ao tema deste artigo. Em seguida, a Seção 3 apresenta as principais funcionalidades e a arquitetura do sistema proposto. Na Seção 4, é apresentado um estudo de caso utilizando o LST. Por fim, a Seção 5 apresenta as conclusões e perspectivas de trabalhos futuros.

2. Trabalhos Relacionados

Mininet é um emulador de rede de código aberto amplamente conhecido, capaz de criar com facilidade topologias de redes SDN com poucos recursos computacionais. É possível utilizar o Mininet para gerar experimentos locais com ataques, por exemplo, DDoS [R. Kandoi 2015]. Todavia, os nós de rede criados no Mininet têm acesso aos processos do hospedeiro principal, compartilham o plano de dados, de usuários, além de lançamento de processos compartilhados [O. Flauzac 2019]. Por esses motivos que o Mininet não apresenta o isolamento necessário para execução de algumas aplicações, principalmente as de segurança, podendo levar a resultados incorretos ou mascarar potenciais ataques

sendo experimentados [O. Flauzac 2019].

Muitos *testbeds* emulados têm utilizado tecnologias de contêineres como o Docker para a virtualização de recursos [E. Petersen 2020, M. Khan 2020, S. Srisawai 2018, K. Wrona 2017], como identificado na Tabela 1. A adoção dessa tecnologia traz outros benefícios, como uso mais eficiente da CPU e memória em relação às máquinas virtuais, aprimora a portabilidade da aplicação e facilita a criação de topologias e funcionalidades novas. O estudo de [K. Wrona 2017] propõe um *testbed* emulado focado na experimentação dos serviços de segurança fornecidos pelo SDN que estão voltados tanto para a rede quanto para a camada de aplicação, utilizando Docker para emular máquinas físicas à rede. Entretanto, a ferramenta apresenta restrições quanto a prover uma interface que facilite a execução e configuração da ferramenta. Por outro lado, [E. Petersen 2020, M. Khan 2020, S. Srisawai 2018] utilizam uma interface para facilitar e agilizar a criação de experimentos para redes SDN. Em particular, [S. Srisawai 2018] utiliza Ansible para execução remota em diferentes computadores e o Docker para execução local. No entanto, a proposta da ferramenta não contempla a possibilidade de se criar topologias personalizadas e não apresenta foco nos estudos de segurança.

A configurabilidade da topologia também é um requisito proposto em *testbeds* emulados [E. Petersen 2020, G. Bonofiglio 2018, M. Khan 2020]. Em [G. Bonofiglio 2018], a definição de topologias é feita utilizando uma interface de linha de comando e vários *scripts* de configuração. A partir desses arquivos, é possível criar contêineres e definir como estes se conectam à rede. Entretanto, apesar de fornecer imagens pré-construídas do Docker, não provê imagens que permitem injetar fluxos maliciosos em rede para estudos de segurança.

No que tange a um *testbed* emulado de SDN de fácil instalação e uso, a ferramenta DockSDN [E. Petersen 2020] segue uma proposta semelhante. É possível definir o número de máquinas, de *switches* e controladores, conectados via interfaces virtuais de rede Linux. No entanto, a reprodutibilidade de eventos é permitida apenas com as ferramentas já instaladas nas imagens Docker fornecidas (*e.g.* Iperf e Ping), o que limita a capacidade de reprodução do comportamento de ambientes reais nos experimentos.

Assim, no melhor dos conhecimentos dos autores sobre *testbeds* emulados, apesar de serem explorados os requisitos de propor uma ferramenta de fácil instalação e configuração, baixo custo, topologia configurável, de virtualização leve e execução local, não foram identificados estudos que tenham propostos *testbeds* emulados que atendam a esses requisitos no contexto de SDN e segurança simultaneamente.

	Segurança	SDN	Virtualização Leve	Fácil Instalação e Configuração	Configurabilidade da Topologia	Exec. local	Isolamento
Mininet		✓	✓	✓	✓	✓	
[R. Kandoi 2015]	✓	✓	✓	✓	✓	✓	
SDN Owl		✓	✓	✓		✓	✓
Káthara		✓	✓	✓	✓	✓	✓
[K. Wrona 2017]	✓	✓	✓				✓
DockSDN		✓	✓	✓	✓	✓	✓
[M. Ring 2017]	✓	✓	✓			✓	✓
[M. Khan 2020]	✓		✓	✓	✓	✓	✓
LST	✓	✓	✓	✓	✓	✓	✓

Tabela 1. Características dos *Testbeds* Emulados em Relação às Características do LST

3. Sobre o LST

O objetivo do LST é prover uma interface de fácil instalação e configuração de modo que usuários com pouco domínio em redes e acesso a poucos recursos computacionais consigam executar experimentos localmente. Para tal fim, a maioria das configurações dos experimentos é centralizada em um único arquivo JSON. Nas seções seguintes serão descritas as principais funcionalidades da ferramenta, o seu funcionamento e a composição do arquivo JSON.

3.1. Principais Funcionalidades

Em suma, as principais funcionalidades da ferramenta são:

- **Configurabilidade da Topologia:** É possível definir a quantidade de máquinas conectadas diretamente a um *switch* virtual com conexão a um controlador local ou remoto;
- **Virtualização leve:** Todos os serviços de rede do experimento são contêineres Docker. Podem ser usadas as imagens pré-construídas que criam servidores (*web*, e-mail, arquivos e *backup*), máquinas do tipo Cliente (utilizam os serviços dos servidores e podem realizar ataques), impressoras ou outras imagens Docker;
- **Nós de rede:** Todos os nós de rede são construídos a partir do uso de *switches* virtuais com Open vSwitch. O usuário pode conectá-los a controladores para que os *switches* interconectem os serviços finais de rede;
- **Emulação de fluxos:** Os contêineres do tipo Clientes permitem configurar os horários dos expedientes em que a máquina será usada, definir seu comportamento na rede, como acessar servidor de e-mail, acessar a *web*, entre outros. Cada máquina pode receber um perfil de comportamento específico na rede ou ser completamente configurada do zero, conforme a necessidade do usuário;
- **Ataques:** É possível realizar ataques na rede através dos contêineres do tipo Cliente, representando máquinas infectadas. Por padrão, há ataques do tipo DoS, *Port Scan* e *Brute Force*. Novos ataques e comportamentos podem ser implementados e adicionados aos Clientes conforme as necessidades do usuário;
- **Controlador:** O controlador pode ser instanciado em uma máquina virtual, contêiner, na rede local ou no próprio computador hospedeiro. Através desse controlador é possível alterar as funções desempenhadas pelos *switches* virtuais para a realização de análises na rede, identificação e mitigação de ataques, alteração de políticas de roteamento de redes, entre outras. Por padrão, o controlador realiza apenas as funções de roteamento de camada de rede e enlace comuns;
- **Logs:** Cada cliente gera um arquivo de *log*, permitindo a identificação do momento em que foi realizado cada operação na rede, seja de ataque, de acesso à internet ou acesso aos servidores.

As funções desempenhadas pelo LST se diferem das desempenhadas por outros emuladores por possibilitar trazer vários perfis de serviços de rede, clientes e de ataques maliciosos, com capacidade para instanciar novos *switches* virtuais acopláveis a quaisquer controladores disponibilizados na rede local, contêineres ou instalados na própria máquina hospedeira. Além disso, LST pode ser utilizado para gerar diferentes arquiteturas de rede, com diversas topologias, para atender vários interesses.

3.2. Fluxograma de execução

A arquitetura do LST é composta por um único módulo que executa uma sequência de comandos a partir do arquivo JSON com as configurações do experimento. O arquivo JSON é interpretado para gerar todos os outros arquivos de configuração, como os arquivos do Docker Compose, de configuração de rede dos hospedeiros e de configuração dos contêineres do tipo Cliente.

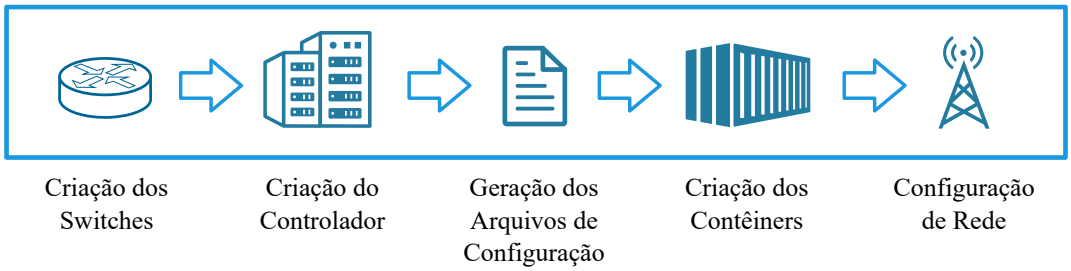

Figura 1. Fluxograma de execução do LST

A Figura 1 mostra as etapas da execução da ferramenta. Inicialmente, são criados e configurados os Open vSwitch *switches*, o controlador Ryu e arquivos de configuração a partir do arquivo JSON de entrada. Por fim, são realizadas a criação dos contêineres através do Docker Compose e as respectivas configurações de rede.

Todo o processo de criação e destruição de contêineres é gerenciado pelo Docker Compose. Por sua vez, as configurações de rede são feitas pelo LST através do uso de interfaces de rede virtual Linux. No caso dos contêineres, visando maior flexibilidade na configuração de rede, é feito o acesso direto aos *namespaces* dos contêineres. Quando o experimento for interrompido, a ferramenta automaticamente irá destruir os contêineres e desfazer todas as configurações de rede realizadas.

3.3. Configuração de Experimentos

Para se criar um experimento no LST, é necessário entender a estrutura do arquivo de configurações em formato JSON que será utilizado para executar a ferramenta. O arquivo é um dicionário, onde cada elemento representa uma máquina a ser criada na topologia. Assim, cada máquina é definida através de um dicionário que contêm vários parâmetros de configuração. Estes são:

- **Nome do dicionário:** Todo dicionário que define uma máquina deve ter um nome entre aspas duplas seguido de dois pontos, por exemplo, **“Servidor1”**;
- **Nome da imagem Docker:** O campo **“image”** representa o nome da imagem Docker a ser criada. Esse parâmetro pode tanto ser o nome de uma imagem local ou pertencente a um repositório do Docker Hub. Essa configuração permite prover novos serviços para adequar o experimento às necessidades do usuário;
- **IP da máquina e sub-rede:** O campo **“IP”** atribui um IP específico para cada máquina, porém devem ser únicos para evitar conflitos de IP. Além disso, sub-redes podem ser geradas a partir da atribuição de diferentes máscaras configuradas pelo usuário;
- **Switch:** O campo **“bridge”** define a qual *switch* o contêiner vai se conectar;

- **Depende de:** O campo **"depends_on"** é um parâmetro para o Docker Compose definido como uma lista de nomes dos dicionários das máquinas que devem ser inicializadas antes do contêiner;
- **Servidor DNS:** O campo **"dns"** representa o endereço de IP do servidor DNS que será utilizado pela máquina.

4. Demonstração planejada do LST

O LST foi desenvolvido para ambientes Linux utilizando a imagem do Ubuntu Server versão 20.04.2 LTS através do VirtualBox versão 6.1.26, configurada com 15 GB de RAM, 4 núcleos de CPU e 32 GB de espaço em disco. A seção seguinte descreverá em detalhes a execução da ferramenta.

4.1. Execução da ferramenta

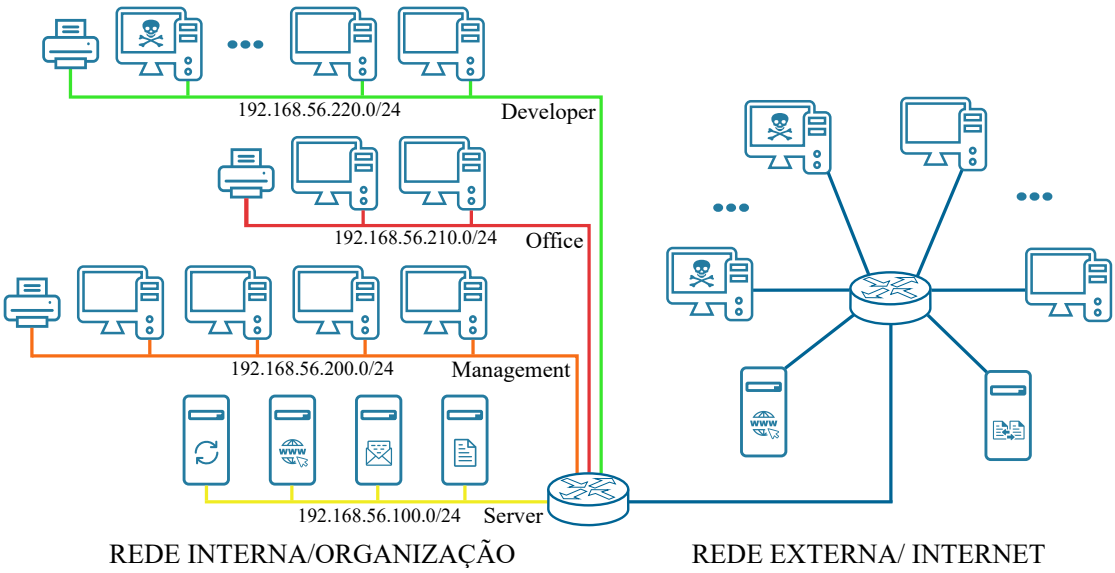

Figura 2. Topologia do Experimento Padrão

Para a demonstração da execução da ferramenta será utilizado o exemplo do experimento padrão baseado no estudo de [M. Ring 2017]. Nesse experimento são criados duas topologias representadas na Figura 2. Uma é a topologia interna que simula uma pequena organização e a outra é a topologia externa que contém vários clientes e servidores. Nessas topologias, existem clientes infectados e clientes saudáveis. Dessa forma, os clientes geram tanto fluxos benignos quanto maliciosos.

A topologia externa representa máquinas conectadas diretamente na Internet. Composta por um servidor Seafile (serviço para armazenamento e partilha de arquivos), um servidor *web*, máquinas do tipo Cliente, em que uma parte são máquinas infectadas. A topologia interna é composta por quatro sub-redes. A sub-rede dos servidores é composta dos servidores de e-mail, arquivos, *web* e *backup*. E as demais sub-redes são gestão, escritório e desenvolvimento, cada uma composta por uma impressora e máquinas do tipo Cliente. Cada Cliente tem um comportamento específico para cada sub-rede a qual ele pertence. Mais detalhes do experimento, consultar o estudo de [M. Ring 2017].

Para executar o LST, é necessário criar o arquivo JSON do experimento passando-o como argumento na linha de comando ao executar o arquivo "setup.sh". É disponibilizado o arquivo JSON do experimento padrão no repositório do projeto [1] e para mais instruções está disponível o vídeo de demonstração da ferramenta no Youtube [2].

```
LST    Copyright (C) 2022 Alexandre Mitsuru Kaihara

       This program comes with ABSOLUTELY NO WARRANTY;
       This is free software, and you are welcome to redistribute it
       under certain conditions;

[LST] Setting up all environment variables
[LST] Remove all remaining network configuration from previous experiments
[LST] Creating bridges br-int and br-ex
[LST] Setting up controller on 127.0.0.1:9001
[LST] Creating Seafile server
213398704cj19837910498712940879870928374190k7891032740d87987491
[LST] Seafolder ID is ef2c0c74-5d13-4263-a6f3-c4de6ce351ba
[LST] Creating all configuration files of linuxclient from partial_experiment.json
[LST] Setting up all containers
```

Figura 3. Início da Execução

Ao executar a ferramenta, o LST seguirá as etapas do fluxo de execução da ferramenta exibindo na tela o seu andamento como ilustrado na Figura 3. E, então, serão instanciadas as máquinas e realizadas as respectivas configurações de rede. Em seguida, as máquinas começarão a gerar pacotes na rede sendo exibidos na tela como fluxos instanciados no controlador. A Figura 4 ilustra os dados dos fluxos instanciados de pacotes de camada três e *packet-in* (pacotes que chegam ao controlador quando não existe uma regra de encaminhamento nos *switches*). Para encerrar o experimento basta pressionar "CTRL + C" e aguardar o programa desfazer todas as configurações de rede e destruir os contêineres.

```
Datefirstseen = 2022-04-07 20:44:00:628585 SrcIPAddr = 192.168.100.2 SrcPt = 445
DstIPAddr = 192.168.220.6 DstPt = 34370 Proto = TCP Tos = 0 Flags = .....F Datapath =
85958796789835 Outport = 18
packet in 85958796789835 4e:2d:d7:8c:48:4b e6:cd:3d:57:26:15 4294967294
```

Figura 4. Fluxos de Rede

Durante a execução do experimento, o controlador gera um *log* dos fluxos instanciados e dos *packet-in*. E todas as máquinas do tipo Cliente geram *logs* das atividades na rede, especificando o tipo da ação, o momento que ocorreu e uma *flag* indicando se foi possível realizar ou não a ação. Para o caso das máquinas infectadas que geram ataques na rede, também é armazenado nos *logs* os ataques realizados.

5. Conclusão e Trabalhos futuros

O presente trabalho objetivou apresentar uma ferramenta de testes destinada tanto para a área de redes SDN quanto para segurança chamada Lighweight SDN Testbed (LST). A solução proposta baseou-se na tecnologia de contêineres do Docker para diminuição

[1] https://github.com/alexandrekaihara/lst
[2] https://www.youtube.com/watch?v=ln0Np3dH6kk

do custo de uso, facilitar a portabilidade da ferramenta e flexibilidade para agregar novas funcionalidades à ferramenta. Além disso, priorizou-se a configurabilidade da topologia dos experimentos assim como a viabilidade de execução dos experimentos.

Como trabalho futuro, pretende-se aprimorar a configurabilidade do LST permitindo a criação de topologias mais complexas, como definir qualquer número de *switches* (atualmente limitada a dois), controladores, sub-redes e maior flexibilidade na definição de como os dispositivos se conectam na rede. Adicionalmente, é interessante estender algumas capacidades de configuração do arquivo JSON como a definição do *Gateway*, aceitar argumentos de configuração do Docker e definir a execução do contêineres em modo privilegiado.

Agradecimentos

Este trabalho foi realizado com apoio financeiro do projeto PROFISSA - Programmable Future Internet for Secure Software Architectures sob a bolsa 2020/05152-7 e da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

Referências

D. Kreutz, e. a. (2015). Software-defined networking: A comprehensive survey. In *Proceedings of the IEEE*, volume 103. IEEE.

E. Petersen, M. A. (2020). Docksdn: A hybrid container-based sdn emulation tool. In *2020 IEEE Latin-American Conference on Communications (LATINCOM)*. IEEE.

G. Bonofiglio, e. a. (2018). Kathará: A container-based framework for implementing network function virtualization and software defined networks. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE.

K. Wrona, S. S. (2017). Sdn testbed for validation of cross-layer data-centric security policies. In *2017 International Conference on Military Communications and Information Systems (ICMCIS)*. IEEE.

M. Khan, O. Rehman, I. R. (2020). Lightweight testbed for cybersecurity experiments in scada-based systems. In *020 International Conference on Computing and Information Technology 10 Sep. 2020*, volume 1. IEEE, Tabuk, Saudi Arabia.

M. Ring, e. a. (2017). Flowbased benchmark datasets for intrusion detection. In *Proceedings of the 16 th European Conference on Cyber Warfare and Security (ECCWS)*, page 361369. ACPI.

M. Šimon, L. H. (2016). Ddos testbed based on peer-to-peer grid. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*. IEEE.

O. Flauzac, E. R. (2019). Is mininet the right solution for an sdn testbed? In *IEEE Global Communications Conference (GLOBECOM)*. IEEE.

R. Kandoi, M. A. (2015). Denial-of-service attacks in openflow sdn networks. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE.

S. Srisawai, P. U. (2018). Rapid building of software-based sdn testbed using sdn owl. In *2018 22nd International Computer Science and Engineering Conference (ICSEC)*. IEEE.