Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

LST: Testbed Emulado Baseado em Contêineres para Redes SDN Seguras

Alexandre M. Kaihara

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Marcelo Antônio Marotta

Brasília
2022

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

LST: Testbed Emulado Baseado em Contêineres para Redes SDN Seguras

Alexandre M. Kaihara

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Marcelo Antônio Marotta (Orientador)
CIC/UnB

Prof. Dr.
Universidade de Brasília

Dr.
Universidade de Brasília

Prof.a Dr.a
Coordenadora do Bacharelado em Ciência da Computação

Brasília, 17 de Novembro de 2022

Dedicatória

Dedico esse trabalho à todos os meus familiares que me apoiaram todos esses anos e contribuíram para o meu crescimento para finalizar esse curso. Também aos meus amigos que acompanharam essa jornada e compartilharam momentos importantes e fizeram da faculdade um lugar especial para mim.

Agradecimentos

Agradeço à todos os meus mentores pela excelência no ensino, em especial ao meu orientador Dr. Marcelo Marotta pela excelente orientação como mentor e como amigo para concretizar todos os estudos e pesquisas realizados. Também agradeço ao projeto PROFISSA pelo apoio acadêmico e à FAPESP pelo apoio financeiro dos projetos desenvolvidos.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Nem todo *testbed* emulado é adequado para experimentações em segurança. Frequentemente os *testbeds* em segurança estão restritos ao contexto de aplicação para o qual foram desenvolvidos ou têm por base outras tecnologias que apresentam limitações de configurabilidade e de aplicações de segurança (*e.g.*, Mininet), enquanto outras propostas de *testbeds* (*e.g.*, DockSDN) permitem maior configurabilidade, mas não possuem foco em aplicações de segurança ou não estão inseridas no contexto de redes SDN. Visando atender à lacuna de pesquisa identificada, é proposta o Lighweight SDN Testbed (LST) - ferramenta leve capaz de atender a diversos contextos de aplicação tanto de segurança quanto de redes SDN programadas e em tempo real via Python. É possível monitorar a rede e coletar métricas fazendo uso de Netflow, sFlow, IPFIX ou CICFlowMeter. Ainda, imagens Docker pré-construídas são disponibilizadas para a emulação de fluxos de rede tanto benignos quanto maliciosos.

Palavras-chave: Testbed; SDN; Emulação Leve; Segurança Computacional; Tempo Real

Abstract

Not all emulated testbeds are suitable for security experimentation. Often security testbeds are restricted to their application context or are based on other technologies which have configurability and security application limitations (*e.g.*, Mininet). While other proposals allow greater configurability, they do not focus on security applications or are not inserted in the context of SDN networks. To address the identified research gap, the Lightweight SDN Testbed (LST) is proposed. LST is a lightweight tool capable of supporting different application contexts both for security and SDN networks programmatically and in real-time through Python. It is possible to monitor the network and collect metrics using Netflow, sFlow, IPFIX, or CICFlowMeter. In addition, pre-built Docker images are available for emulating both benign and malicious network flows.

Keywords: Testbed; SDN; Lightweight Emulation; Security; Real Time

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CPU Central Processing Unit.

DDoS Distributed Denial Of Service.

DoS Denial Of Service.

JSON JavaScript Object Notation.

LST Lightweight SDN Testbed.

SDN Software-Defined Networking.

VM Virtual Machine.

Capítulo 1

Introdução

1.1 Redes Definidas por Software

Em redes de computadores há o conceito dos planos de controle e do plano de dados. O plano de controle refere-se às lógicas e funções que determinam a rota para o pacote ou quadro chegar ao seu destino. Enquanto que o plano de dados se refere às funções e processos que efetivamente encaminham os pacotes ou quadros entre uma interface e outra seguindo a lógica do plano de controle. Nas redes legadas, são constituídas por vários dispositivos de rede com *hardware* dedicados (*e.g.*, roteadores e *switches*) que lidam com ambos os planos de controle e de dados (Figura 1.1), ou seja, é feita tanto a decisão da rota quanto o efetivo encaminhamento dos pacotes ou quadros entre interfaces dentro de cada um desses dispositivos de rede.

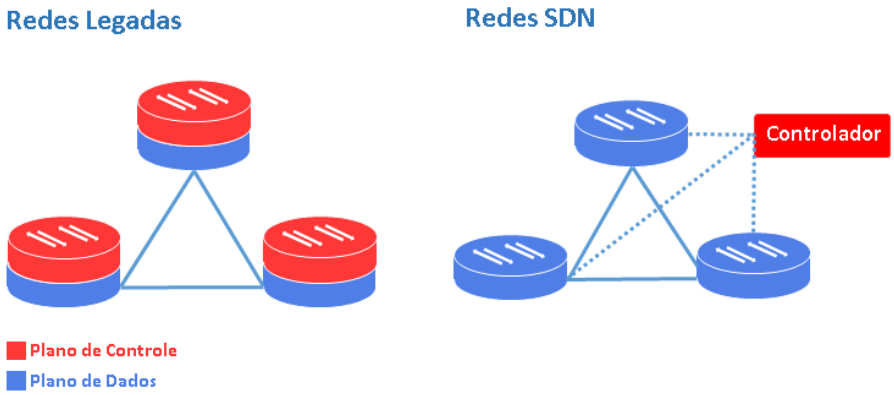

Figura 1.1: Plano de Controle e Dados em Redes Legadas e Redes SDN

Nesse contexto, as Redes Definidas por software ou do inglês *Software-Defined Networking* (SDN), se diferem das redes legadas por separarem do plano de controle do plano de dados. Ou seja, todos os processos e as lógicas de roteamento são feitas separadamente

no chamado controlador. Dentro do controlador é possível modificar como as redes são configuradas, controladas e gerenciadas, permitindo redução de custos, dado uma menor necessidade por *hardwares* sob medida, permite maior controle sobre o tráfego e introduzir mais mecanismos inteligentes para o monitoramento e proteção da rede, entre outros.

Devido a separação do plano de dados do plano de controle e a introdução de controladores na rede, as Redes Definidas por Software permitem a criação de novas aplicações, por exemplo, na criação de novas políticas de redes, otimização e eficiência dos processos de roteamento, facilidade no monitoramento e coleta de métricas da rede, desenvolvimento novos mecanismos de detecção intrusão em rede, métodos mais inteligentes para lidar com erros e ataques na rede. Entretanto, a centralização do plano de controle trouxeram desafios de escalabilidade e de segurança, tanto aquelas incorporadas no controlador quanto ponto único de falha, ficando vulnerável a ataques volumétricos como *Denial of Service* (DoS).

1.2 Testbeds

Em face à diversidade de tópicos de estudo em redes SDN como também em segurança, potencialmente, as soluções propostas necessitam de serem avaliados e validados. Nesse contexto, os *testbeds* são muito úteis para a criação de cenários experimentais adequados que permitirão a reprodução dos sistemas e dos cenários de ataque e, por consequência, conseguir analisar o impacto, as vulnerabilidades e limitações da solução.

Dentre as possibilidades de construção de *testbeds* é a utilização de componentes físicos, emulados e simulados. A utilização de componentes físicos para a construção de tipologias é altamente custosa, com baixa flexibilidade de rede, porém com altíssima fidelidade do experimento. A simulação é uma alternativa aos altos custos e inflexibilidade do uso de componentes físicos que, através de modelos matemáticos que visam prever algum comportamento ou funcionamento de um sistema. Porém simulações são apenas aproximações da realidade, não conseguindo atingir alto grau de realismo nos experimentos. Já a emulação, é um método que permite a reprodução mais semelhante a um ambiente real, mas sem a necessidade de ter toda a infraestrutura física para a criação dos experimentos, sendo, portanto, o método mais adequado para obter maior grau de realismo nos experimentos com baixo custo.

Nos capítulos subsequentes serão apresentados os artigos desenvolvidos e publicados em dois Simpósios Nacionais de redes e segurança com objetivo de propor uma ferramenta emulada de baixo custo, fácil utilização e altamente configurável para experimentações em redes SDN e segurança através do uso da tecnologia de contêineres.

Capítulo 2

XL SBRC 2022

O Lightweight SDN Testbed (LST)[1] é uma ferramenta capaz de gerar topologias de Redes Definidas por Software (do inglês, *Software-Defined Networking* - SDN) para aplicações em redes, segurança ou em ambas. A proposta da ferramenta é ser leve, emulada e principalmente trazer facilidade na configuração e instalação. Este estudo foi aceito e publicado em 15 de Maio de 2022 no 40º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos [1] - maior evento científico nacional sobre redes de computadores e sistemas distribuídos devido à qualidade e o número de submissões, além da atualidade, diversidade e abrangência dos temas dos trabalhos submetidos.

A criação de cenários experimentais são fundamentais validar soluções propostas, sendo uma tarefa extensa e complexa, exigindo conhecimentos práticos e teóricos para o desenvolvimento de experimentos válidos. Dessa forma, é importante que o *testbed* seja de fácil uso, instalação e que permita ser utilizada em diferentes domínios de aplicação. Nesse contexto, vários *testbeds* têm sido propostos, entretanto, observa-se que os estudos de segurança, em geral, são contexto-específicos, limitando-se o reuso e a configurabilidade da ferramenta [2, 3]. Por outro lado, outros *testbeds* em rede, apesar de proverem mais recursos e maior configurabilidade, não possuem foco em aplicações de segurança — como facilitações de geração de ataques em rede [4, 5]. Assim, visando atender à lacuna de pesquisa identificada, que é proposta o LST - uma ferramenta de emulação leve baseada em contêineres de fácil uso para redes SDN e segurança.

As principais contribuições do LST estão em prover uma interface de fácil uso por meio de um único arquivo JSON — que permite definir todas as configurações da topologia. O uso da tecnologia de contêineres diminui drasticamente o número de dependências, facilitando a instalação da ferramenta, além de tornar a ferramenta leve, quando comparado ao uso de máquinas virtuais. Por fim, o presente estudo disponibiliza imagens Docker pré-construídas que simulam o ambiente de uma pequena organização, gerando fluxos tanto benignos quando maliciosos na rede, auxiliando estudos de segurança, por

exemplo, no desenvolvimento de sistemas de detecção de intrusão em rede.

2.1 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 [6]. 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 [2]. No entanto, em vários estudos de segurança são criados *testbeds* próprios [7], os quais geralmente são restritos ao contexto de aplicação para o qual foram desenvolvidos, [8] [9], 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.* [4, 5]), mas que não foram desenvolvidas com foco em aplicações de segurança. Por outro lado, ferramentas desenvolvidas exclusivamente para experimentos em segurança [2, 3] 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 dis-

ponibiliza 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.

2.2 Trabalhos Relacionados

Mininet[1] é um emulador de rede de código aberto amplamente conhecido, capaz de criar com facilidade topologias de redes SDN com poucos recursos computacionais. Com poucos comandos ou utilizando-se scripts em Python, a ferramenta é capaz de emular diversos nós de rede e permitir a conexão dos switches com controladores, habilitando SDN na rede. É possível utilizar o Mininet para gerar experimentos locais com ataques, por exemplo, DDoS [**?**]. 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 [10]. 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 [10].

Muitos *testbeds* emulados têm utilizado tecnologias de contêineres como o Docker para a virtualização de recursos [4, 2, 8, 7], 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 [7] 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, [4, 2, 8] utilizam uma interface para facilitar e agilizar a criação de experimentos para redes SDN. Em particular, [8] 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.

[1] http://mininet.org/

A configurabilidade da topologia também é um requisito proposto em *testbeds* emulados [4, 5, 2]. Em [5], 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 [4] 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		✓	✓	✓	✓	✓	
[?]	✓	✓	✓	✓	✓	✓	
SDN Owl		✓	✓	✓		✓	✓
Káthara		✓	✓	✓	✓	✓	✓
[7]	✓	✓	✓				✓
DockSDN		✓	✓	✓	✓	✓	✓
[9]	✓	✓	✓			✓	✓
[2]	✓		✓	✓	✓	✓	✓
LST	✓	✓	✓	✓	✓	✓	✓

Tabela 2.1: Características dos *Testbeds* Emulados em Relação às Características do LST

2.3 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.

2.3.1 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, mais detalhes serão descritos na Seção 4.2.2;
- **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.

2.3.2 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.

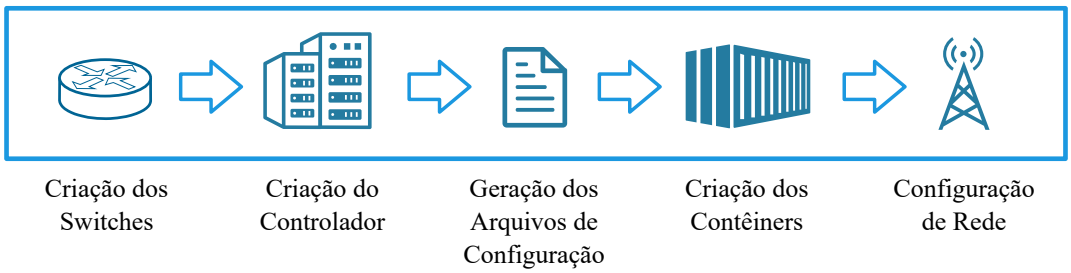

Figura 2.1: Fluxograma de execução do LST

A Figura 3.1 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.

2.3.3 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.

2.4 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.

2.4.1 Execução da ferramenta

Para a demonstração da execução da ferramenta será utilizado o exemplo do experimento padrão baseado no estudo de [9]. Nesse experimento são criados duas topologias representadas na Figura 3.2. 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

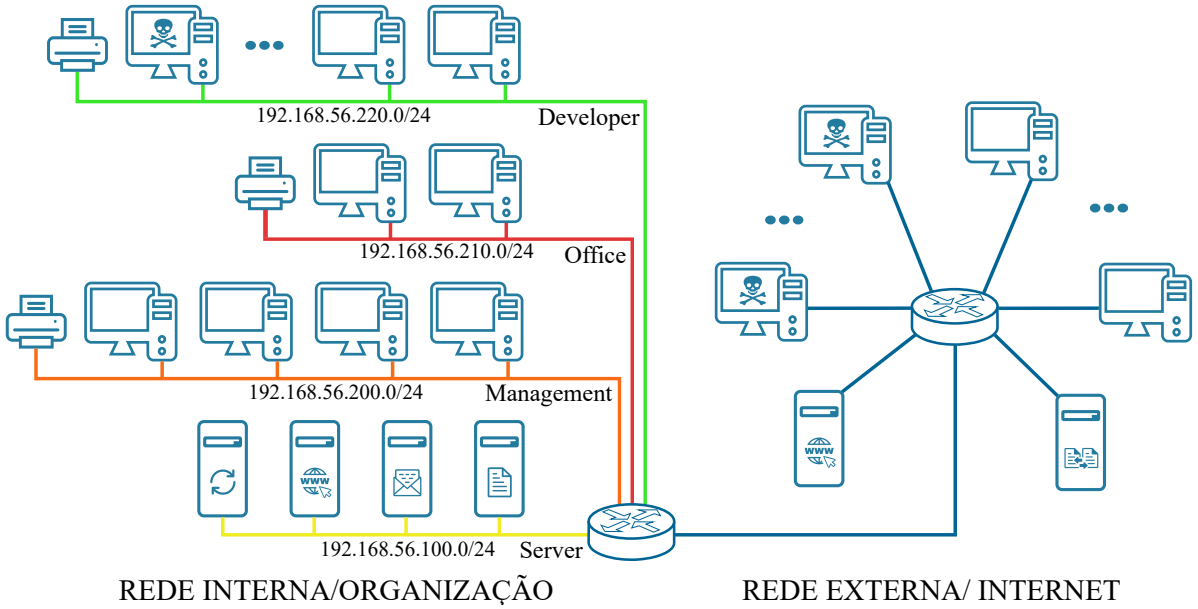

Figura 2.2: Topologia do Experimento Padrão

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 [9].

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 [2] e para mais instruções está disponível o vídeo de demonstração da ferramenta no Youtube [3].

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 2.3. 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 2.4 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.

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

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

```
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 2.3: Início da Execução

```
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 2.4: Fluxos de Rede

realizar ou não a ação. Para o caso das máquinas infectadas que geram ataques na rede, também é armazenado nos *logs* os ataques realizados.

Capítulo 3

XXII SBSEG 2022

O Lightweight SDN Testbed 2.0[11] é um *testbed* emulado leve baseado em contêineres visando alta configurabilidade para atender uma gama maior de aplicações voltados para redes SDN e segurança. Dentre as principais características do LST 2.0[11], destaca-se a capacidade de se gerar topologias programaticamente e em tempo real através de métodos de linguagem de programação, permitindo beneficiar-se da automação e dos recursos providos pela linguagem de programação. O presente estudo foi submetido e aprovado no dia 24 de Agosto de 2022 no XXII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais [11] - principal fórum do país de divulgação de trabalhos na área de segurança da informação e sistemas computacionais.

A arquitetura de redes SDN ao mesmo tempo que permitiu aprimoramento do controle e segurança na rede, trouxe à tona outras questões de segurança (*e.g.*, vulnerabilidades do controlador), além das já existentes, como por exemplo, ataques TCP/IP SYN, ataques volumétricos, entre outros. Diante os vários cenários estudos em redes SDN, *testbeds* devem ser trazer configurabilidade suficiente para atender às especificidades de cada experimento. Entretanto, nem todo *testbed* é adequado para experimentações em segurança por, por exemplo, a falta de isolamento entre os nós criados no Mininet, podendo gerar resultados incorretos [10]. Dessa forma é proposto a segunda versão do LST[11], que traz alto nível de configurabilidade para a geração de topologias para aplicações em redes, segurança ou em ambos os contexto simultaneamente.

O LST 2.0[11] objetiva a alta configurabilidade dos experimentos, sendo possível definir quaisquer números de *hosts*, *switches* e controladores conectados entre si via interfaces Virtuais Ethernet Linux. A interface de uso consiste em um conjunto reduzido de métodos de linguagem de programação simples, porém permitem executar mais configurações de rede e de segurança (*e.g., firewall*), gerar topologias programaticamente e em tempo real. Além de disponibilizar as imagens que simulam os fluxos de uma pequena organização, a ferramenta traz possibilidade de monitorar a rede através do Netflow, IPFIX e/ou sFlow e

gerar várias estatísticas através do CICFlowMeter, contribuindo ainda mais para estudos tanto em rede e aplicações em segurança.

3.1 Introdução

As Redes Definidas por Software (do inglês *Software-Defined Networking* — SDN), tornaram-se importantes arquiteturas de rede, permitindo maior flexibilidade na gestão e no monitoramento via separação do plano de controle do plano de dados. Tal característica permitiu o aprimoramento de mecanismos de segurança para a rede, como, por exemplo, no reforço de políticas de rede, na detecção e mitigação de ataques e na proteção do próprio controlador de rede [6]. Em contrapartida, a arquitetura SDN trouxe outras questões de segurança, tais como possíveis vulnerabilidades incorporadas ao controlador, ataques TCP/IP SYN, UDP *flooding*, *Man in the Middle*, entre outras [12]. Assim, em face da diversidade de cenários de estudo, é possível beneficiar-se do uso de *testbeds* para a criação de ambientes experimentais adequados e na condução de experimentos em segurança que permitam validar as soluções propostas.

Em geral, a aquisição de equipamentos habilitados com SDN é custosa e sujeita às restrições de orçamento. Uma alternativa é recorrer às soluções de baixo custo, como, por exemplo, a emulação dos componentes de rede. Entretanto, nem todo *testbed* emulado é adequado para a condução de experimentos de segurança ou possui o nível de configurabilidade necessária para o estudo, como o Mininet, que compartilha o mesmo espaço de dados e não provê isolamento de recursos por nó [10]. Ao mesmo tempo, *testbeds* virtualizados com máquinas virtuais consomem muitos recursos onerando o sistema hospedeiro, tornando difícil a experimentação com ataques de alto consumo de recurso, como ataques volumétricos (e.g., *Denial of Service* — DOS). Assim, os estudos em segurança exigem requisitos específicos da ferramenta para o correto uso e criação dos experimentos.

Em vários estudos de segurança têm-se o desenvolvimento de *testbeds* próprios. No entanto, muitas vezes, tais *testbeds* estão restritos ao contexto de aplicação ao qual foram desenvolvidos [13, 14, 15], limitando o reuso da ferramenta. Embora outras propostas de *testbeds* permitam maior configurabilidade da ferramenta [4, 16, 17], estas não possuem foco em aplicações de segurança ou não estão inseridas no contexto de redes SDN, evidenciando a necessidade por *testbeds* que permitam explorar os múltiplos cenários de redes SDN, mas que também atendam às necessidades específicas em segurança.

Diante do exposto, é proposta uma nova versão do LST focado em segurança com base em virtualização leve com contêineres no contexto de redes SDN, denominado *Lightweight SDN Testbed* (LST 2.0) — disponível no repositório do GitHub[1] sob a licença GNU *Gene-*

[1]https://github.com/alexandrekaihara/lst2.0

ral Public License, versão três. O LST 2.0 objetiva a alta configurabilidade, sendo capaz de gerar diversas topologias programaticamente e em tempo real. Diferentemente da primeira versão, tais topologias podem ser compostas por quaisquer números de *switches*, *hosts* e controladores, além mudança na interface de uso de um arquivo JSON para um conjunto de métodos de linguagem de programação. Além disso, a ferramenta permite manipular diferentes configurações de rede e segurança dos nós de rede (*firewall*, *gateway*, entre outros). Visando necessidades específicas para experimentações em segurança, são disponibilizadas imagens pré-construídas para simular fluxos benignos e maliciosos, além de ferramentas para o monitoramento e a coleta de métricas na rede, facilitando o desenvolvimento de estudos de sistemas de detecção de intrusão em rede, entre outros.

O presente estudo está organizado da seguinte forma. Na Seção 3.2 são apresentados os trabalhos relacionados. Na Seção 3.3, a arquitetura da solução e principais funcionalidades da ferramenta são detalhadas. Em seguida, uma demonstração baseada na solução proposta é descrita na Seção 3.4.

3.2 Trabalhos Relacionados

O Mininet é uma ferramenta de emulação leve amplamente conhecida, que permite gerar topologias de redes SDN com grande facilidade. Embora o Mininet venha sendo utilizado por vários estudos de segurança [18, 13], os nós de rede ali criados não possuem o isolamento necessário em relação ao hospedeiro principal por compartilhar o plano de dados de usuários e pelo lançamento de processos compartilhados. Tais características podem gerar resultados incorretos ou mascarar ataques sendo experimentados, por exemplo, no caso de nós terem acesso *root* e acessarem indevidamente processos de outros nós, gerando um comportamento inesperado no experimento. Além disso, os nós de rede gerados precisam ser configurados um a um a cada execução, dificultando a criação de experimentos de maior complexidade [9].

A utilização da tecnologia de contêineres tem sido explorada em *testbeds* [4, 17, 15] devido à individualização do sistema de arquivos, das interfaces de rede e árvore de processos separada para cada contêiner, além de apresentarem consumo de CPU e memória mais eficientes em relação às máquinas virtuais. O NestedNed explora a construção de um *testbed* voltado para redes SDN hierárquicas, no qual é possível gerar topologias de rede com contêineres Docker aninhados conectados via *switches* Open vSwitch. No entanto, esse *testbed* possui limitações quanto à aplicabilidade em outros domínios, uma vez que os nós de rede são contêineres aninhados voltados para redes SDN hierárquicas.

O *testbed* CyberVAN [17] é proposto com foco em segurança, permitindo gerar diferentes cenários experimentais com nós tanto físicos quanto emulados (*e.g.*, contêineres)

através de uma interface de usuário gráfica ou comandos no console. Entretanto, a fidelidade da reprodução de topologias é limitada devido ao uso de softwares de simulação de topologias de rede (*e.g.*, ns3, QualNet, entre outros) quando comparado às topologias emuladas. Além disso, o CyberVAN não provê suporte a geração de topologias programaticamente, não sendo possível beneficiar-se dos recursos e da automação proporcionada por linguagens de programação. Além disso, a configurabilidade da ferramenta está restrita às limitações das interfaces disponibilizadas, que podem não atender aos requisitos dos diversos estudos em segurança [16, 13, 9].

Soluções com múltiplos controladores têm sido propostas a fim de se contornar as limitações de escalabilidade e de segurança que a centralização do plano de controle de redes SDN trouxeram. Nesse sentido, visando ampliar a aplicabilidade, é essencial que *testbeds* permitam a criação de topologias com múltiplos controladores. O DockSDN [4] é um *testbed* com base em contêineres com suporte a geração de topologias personalizadas com múltiplos controladores. No entanto, apesar de toda configurabilidade que o DockSDN provê, não são disponibilizados *scripts* para a geração de ataques em rede, um recurso importante para permitir que pesquisadores foquem mais nas propostas do estudo e menos na elaboração de cenários experimentais.

O DCLab é um testbed com base no Mininet que possui uma API para a criação de experimentos, permitindo gerar topologias programaticamente. Voltado para o contexto de segurança, tanto [14] quanto [13] também são propostos como *testbeds* baseados no Mininet, montando vários cenários para avaliar a detecção e mitigação de ataques DDoS e técnicas de validação de endereço de origem, respectivamente. No entanto, o DCLab não possui foco em aplicações em segurança, ou seja, não provê scripts de ataque em rede e os estudos de [14, 13] desenvolvem testbeds restritos ao contexto de aplicação para o qual foram desenvolvidos.

No melhor dos conhecimentos dos autores sobre *testbeds* emulados, faltam *testbeds* com alta configurabilidade para redes SDN e segurança capazes de gerar topologias programaticamente em tempo real, com suporte a múltiplos controladores. LST 2.0 trabalha disponibilizando imagens Docker pré-construídas que simulam fluxos de uma pequena organização e capaz de simular ataques como *Denial of Service* (DoS), *Port Scanning* e *Brute Force*. E ainda, a solução proposta monitora e coleta métricas da rede por meio do Netflow, sFlow e IPFIX e gera um relatório de métricas dos fluxos por meio do CICFlowMeter [19], auxiliando na geração de *datasets* em segurança, no treinamento de algoritmos de detecção, no monitoramento de ataques em tempo-real, entre outros. Um resumo de todas as características explanadas, bem como o posicionamento do LST 2.0 frente à literatura é demonstrado na Tabela 3.1.

	Segurança	SDN	Multi Controlador	Topologia Configurável	Gerar Topologia via Linguagem de Programação	Configuração em tempo-real	Geração de Relatório	Scripts de Ataques
Mininet		✓	✓	✓	✓			
CyberVAN	✓	✓	✓	✓		✓		✓
[15]	✓			✓				✓
NestedNet		✓		✓		✓		
[9]	✓	✓					✓	✓
DockSDN		✓	✓	✓	✓	✓		
DCLab		✓	✓	✓	✓	✓		
[14]	✓	✓	✓	✓	✓			✓
[13]	✓	✓	✓	✓	✓			✓
LST 2.0	✓	✓	✓	✓	✓	✓	✓	✓

Tabela 3.1: Características dos *Testbeds* Emulados em Relação às Características do LST 2.0

3.3 *Lightweight SDN Testbed 2.0*

O LST 2.0 foi desenvolvido com o objetivo de atender aos mais diversos cenários de estudos em redes SDN e segurança, via uma interface definida por métodos de linguagem de programação de fácil uso, mas que permitem alta flexibilidade na configuração e geração de topologias personalizadas. Nas seções seguintes têm-se a apresentação da arquitetura da ferramenta e suas principais funcionalidades.

3.3.1 Arquitetura da Ferramenta

O LST 2.0 é baseado principalmente em contêineres para a geração dos nós de rede e as configurações de comportamento e de rede são feitas programaticamente. A ferramenta está organizada por meio de um conjunto de métodos bem definidos pertencente a uma hierarquia de classes. A classe pai, nomeada como Node, possui todos os atributos e métodos mínimos para se instanciar, deletar e configurar um contêiner qualquer. Dessa forma, os usuários podem desenvolver classes próprias herdando os atributos e métodos da classe Node e focando-se apenas no desenvolvimento das configurações específicas para o estudo.

Por padrão, estão implementadas três classes que são especializações da classe Node como na Figura 3.1. A classe Host permite criar contêineres que serão nós comuns que consumirão serviços na rede. A classe Switch permite criar *switches* virtuais, cuja configuração de rede é específica para encaminhar pacotes entre portas de uma única interface de ponte (*bridge*) contida no contêiner e também para se conectar a um controlador em um IP e porta específicos. Caso nenhum controlador seja atribuído, o *switch* realizará apenas as funções básicas de camada de rede e também se nenhum IP for atribuído a ele, será um *switch* de camada de enlace. A classe Controller permite criar contêineres que conseguem instanciar um ou mais controlares.

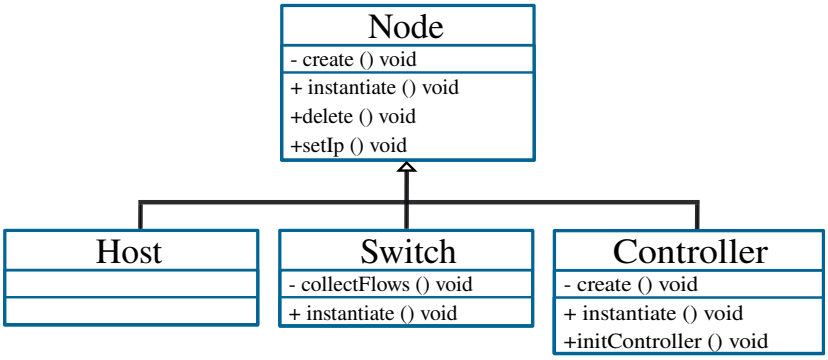

Figura 3.1: Hierarquia de Classes do LST 2.0

Cada nó de rede é representado por uma instância de alguma das especializações da classe Node. Cada instância, por sua vez, é capaz de criar, destruir e configurar o contêiner a qualquer momento, ficando sob responsabilidade do usuário manter as instâncias das classes e destruir os contêineres para finalizar o experimento.

No presente estudo, o LST 2.0 foi implementado utilizando Docker para o gerenciamento dos contêineres. Ainda, o OpenvSwitch foi utilizado como solução de switches virtuais. Os controladores, por sua vez, são implementados com Ryu[2]. Os controladores Ryu são utilizados por padrão para a classe Controller, os quais são capazes de configurar apenas o roteamento e encaminhamento comum dos *switches* na camada de Rede.

3.3.2 Principais Funcionalidades

Visando alta configurabilidade da topologia, o LST 2.0 permite gerar quaisquer números de *hosts*, *switches* virtuais e controladores. Os contêineres podem ser instanciados a partir de quaisquer imagens Docker, sejam as disponibilizadas por este estudo, uma imagem local ou uma imagem hospedada no DockerHub, permitindo a criação de novos comportamentos e perfis de rede nos experimentos. Para conectar os nós de rede, são criados pares virtuais Ethernet Linux inserindo cada ponta diretamente nos *namespaces* dos contêineres. Dessa forma, obtém-se maior configurabilidade de rede e evitam-se limitações de uso dos *drivers* de rede do Docker.

É possível instanciar diferentes tipos de controladores, tais como ONOS, Floodlight, POX, entre outros — o que permite explorar diferentes recursos providos pelos controladores (*i.e., northbound*) ou controladores com melhores desempenhos em determinados cenários de ataques [20]. Tais controladores permitirão avaliar diferentes aspectos da arquitetura SDN, podendo alterar as funções desempenhadas pelos *switches*, introduzir novas políticas na rede e até realizar detecção e mitigação de intrusões em rede. Assim,

[2]Controlador Ryu: https://ryu-sdn.org/

o LST 2.0 é capaz de avaliar soluções que mitigam a centralização do plano de controle da rede, tornando a arquitetura robusta a ataques do tipo DoS, que podem impactar no desempenho ou até bloquear totalmente o uso da rede.

São disponibilizadas imagens Docker para gerar topologias, entre as quais, permitem instanciar *Hosts* com o Sistema Operacional Ubuntu (versão 20:04), *switches* virtuais com o OpenvSwitch e controladores com o controlador Ryu. As demais imagens têm por base o estudo de [9], que simulam fluxos de uma pequena organização. Tais imagens podem ser do tipo Servidor ou do tipo Cliente. Os contêineres do tipo Cliente consomem os serviços na rede, sendo possível atribuir comportamentos específicos para atender às especificidades de cada experimento.

Voltado para o contexto de segurança, os contêineres do tipo Cliente podem realizar ataques na rede como *Port Scan*, *Brute Force* e DoS, representando máquinas infectadas, sendo possível adicionar novos *scripts* de ataques. Os Clientes geram *logs* das execuções dos seus respectivos scripts, permitindo mapear quando uma ação ou ataque foi introduzida na rede.

O LST 2.0 permite que os *switches* Open vSwitch coletem métricas por meio do Netflow, sFlow ou IPFIX. Dessa forma, é possível fazer o monitoramento e coleta de dados da rede de forma local ou remota via *softwares* de monitoramento. Além disso, ao final da execução do experimento é possível gerar um relatório a partir do CICFlowMeter[3] contendo diversas métricas e estatísticas dos fluxos gerados como a duração, número de pacotes, total de *Bytes* transmitidos, comprimento máximo e mínimo dos pacotes, entre outros.

O LST 2.0 se difere de outros emuladores de rede por ser capaz configurar e instanciar os nós de rede programaticamente em tempo real — o que agiliza e torna mais dinâmica a verificação de erros e a validação das configurações durante o próprio desenvolvimento do experimento. Além disso, o LST 2.0 é capaz de gerar diversos perfis de serviços de rede, clientes e de ataques, permitindo o monitoramento e a coleta de dados da rede, trazendo uma ferramenta altamente configurável para os mais diversos cenários experimentais em SDN e segurança, seja isoladamente ou em ambas simultaneamente.

3.4 Demonstração

Foi utilizada uma imagem do Ubuntu Server versão 20.04.2 LTS para a demonstração planejada da ferramenta. A imagem foi instalada em uma máquina virtual (VM) do VirtualBox versão 6.1.26 e configurada com 15 GB de RAM, quatro núcleos de CPU e 32 GB de espaço de memória. Ainda que uma VM tenha sido utilizada, a mesma

[3]https://www.unb.ca/cic/research/applications.html

serve apenas para emular a máquina hospedeira do *testbed* com controle da quantidade de recursos dedicados. Ou seja, poderia ter sido utilizada uma máquina real em vez da VM para executar o LST 2.0.

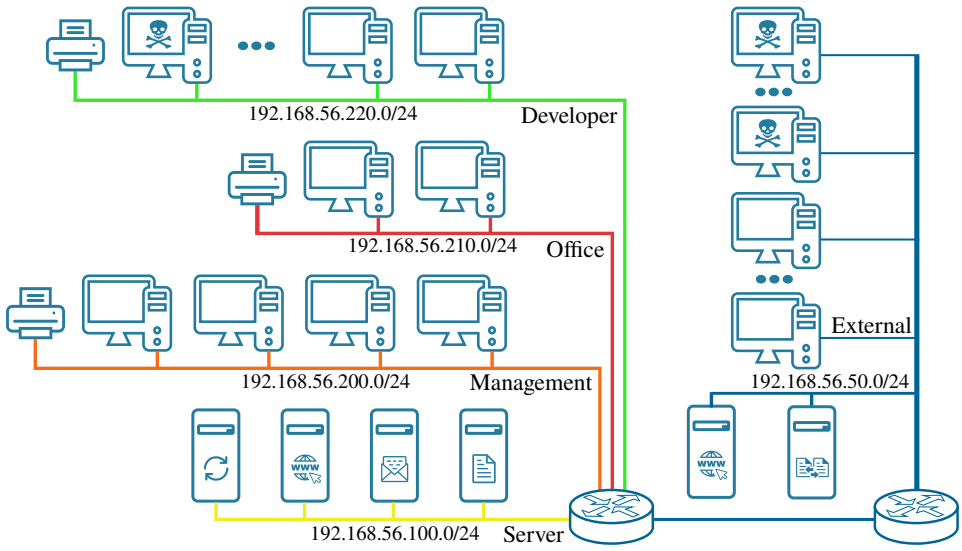

Figura 3.2: Topologia do Experimento Padrão

A demonstração da ferramenta teve por base o experimento do estudo de [9], o qual simula os fluxos de uma pequena organização e de máquinas espalhadas pela rede. Ao executar o experimento, é criada uma topologia composta por quatro sub-redes (Figura 3.2). A sub-rede dos servidores é composta por contêineres provedoras de serviços de *email*, *web*, de arquivos e *backup*. As demais sub-redes são compostas por uma impressora e contêineres do tipo Cliente. Os Clientes possuem comportamentos específicos da sub-rede ao qual pertencem. Parte desses clientes representam máquinas infectadas e geram fluxos maliciosos na rede. A topologia externa é composta por um servidor Seafile (i.e., repositório de arquivos compartilhados), um servidor *web* e várias máquinas do tipo Cliente, que geram tanto fluxos benignos quanto maliciosos na rede.

	Protocol	Duration	Total Fwd Packets	Total Bytes	Max Fwd Pkt Lenght	Min Fwd Pkt Length	Bytes/s	Packets/s	Outras Métricas
Flow 1	TCP	0,481	8	665	137	0	1382,53	16,63	···
Flow 2	TCP	0,074	4	102	39	0	1378,37	54,05	···
Flow 3	TCP	1,140	8	519	142	0	455,26	7,01	···
Flow 4	TCP	1,073	25	11.177	1282	0	10.416,58	23,29	···
Flow 5	UDP	0,362	2	155	47	47	428,17	5,52	···
Flow 6	UDP	0,419	3	156	28	28	372,31	7,15	···
Outros Fluxos	···	···	···	···	···	···	···	···	···

Tabela 3.2: Relatório de Execução do Experimento

Todos os Clientes instanciados geram *logs* das atividades na rede, detalhando o tipo de ação, quando foi realizada, e o status que indica se foi possível realizar ou não a ação. Ao

fim da execução do experimento, a ferramenta criará um relatório com diversas métricas a partir dos fluxos gerados na rede, como demonstrado na Tabela 3.2.

Capítulo 4

Sobre a Ferramenta

4.1 Uso do Lightweight SDN Testbed

Para melhor descrever a utilização da ferramenta, será feita a demonstração da geração de uma topologia linear simples com dois *hosts* conectados a um switch com um controlador (Figura 4.1). Após seguir os passos de instalação das dependências da ferramenta no repositório do projeto[1], é necessário criar um arquivo com o *script* da Figura 4.2 ou iniciar o interpretador do Python e executar cada linha do *script* para reproduzir esta demonstração.

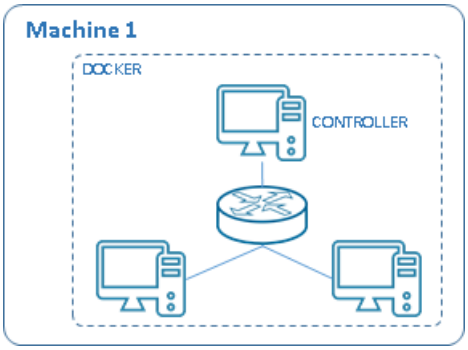

Figura 4.1: Exemplo de Topologia Linear no LST 2.0

Nas linhas um a três, é feito a importação das classes que são especializações da classe Node. Nas linhas quatro a sete são instanciadas as classes. Nas linhas oito a 11 são criados os contêineres efetivamente dos seus respectivos tipos, um *switch* com Open vswitch intalado, um controlador com o controlador Ryu instalado e dois *hosts*, que são imagens Ubuntu 20.04. Nas linhas 12 a 14, são conectados os contêineres através da criação dos pares de rede virtuais Linux que serão colocadas dentro de contêiner. Nas

[1]https://github.com/alexandrekaihara/lst2.0

linhas 16 a 19, são feitas as atribuições de endereço IP e suas respectivas máscaras de sub-rede nas interfaces de cada contêiner.

```
01 from host import Host
02 from switch import Switch
03 from controller import Controller

04 h1 = Host('h1')
05 h2 = Host('h2')
06 s1 = Switch('s1')
07 c1 = Controller('c1')

08 h1.instantiate()
09 h2.instantiate()
10 s1.instantiate()
11 c1.instantiate()

12 h1.connect(s1)
13 h2.connect(s1)
14 c1.connect(s1)

16 h1.setIp('10.0.0.1', 24, s1)
17 h2.setIp('10.0.0.2', 24, s1)
18 s1.setIp('10.0.0.3', 24)
19 c1.setIp('10.0.0.4', 24, s1)

20 c1.initController('10.0.0.4', 9001)
21 s1.setController('10.0.0.4', 9001)

22 s1.connectToInternet('10.0.0.5', 24)

23 h1.setDefaultGateway('10.0.0.5', s1)
24 h2.setDefaultGateway('10.0.0.5', s1)
25 c1.setDefaultGateway('10.0.0.5', s1)
```

Figura 4.2: Script de Topologia Linear no LST 2.0

Após executar o script até a linha 19, já é possível os contêineres comunicarem entre si, uma vez que todos os contêineres do experimento já estão conectados ao *switch*, possuem os seus respectivos endereços de IP e o contêiner *switch* atua como um dispositivo de camada três ou dois quando não conectado a um controlador, permitindo a comunicação dentro da topologia e não exigindo que o usuário tenha que habilitar redes SDN em todas as subredes.

As linhas 20 e 21 iniciam o controlador em um IP e porta específicos e configuram o *switch* a se comunicarem com o controlador, assim, habilitando o SDN na rede. Na linha 22, é criada uma interface de rede entre o contêiner com o hospedeiro principal para

permitir o acesso da topologia à Internet. Nas linhas 23 a 25, é feita a configuração do *gateway* padrão (*i.e.*, destino para os pacotes que não destinam-se à sub-rede local) para a interface com o hospedeiro principal. Dessa forma, é feita a criação de uma pequena topologia linear, porém é possível criar novas especializações mais complexas da classe Node para configurar imagens Docker com serviços específicos na rede e configurá-los e instanciá-los da mesma forma descrita no *script* da Figura 4.2.

4.2 Imagens Pré-Construídas

Esta seção é destinada para descrever detalhes de implementação e utilização das imagens pré-construídas disponibilidas no projeto para poder facilitar o desenvolvimento de ambientes experimentais para redes SDN e segurança.

4.2.1 Especializações da Classe Node

As classes Host, Switch e Controller são exemplos de especializações da classe principal Node (Figura 3.1) que implementam funções para manipular imagens Docker específicas para a criação de exprimentos no LST 2.0.

A classe Host instanciam contêineres que representam máquinas genéricas na rede com sistema operacional Ubuntu 20.04. A classe Switch cria contêineres com o Open vSwitch instalado em que é possível utilizá-lo como *switch* de camada de enlace ou de rede comum, isto é, sem estar habilitado com SDN, e também é possível utilizá-lo como *switch* conectado a um controlador, que definirá o seu comportamento dentro da rede para o encaminhamento e roteamento de pacotes ou quadros. A classe Controller possui o controlador Ryu instalado, cujo comportamento é de permitir o roteamento de camada de enlace e de rede em quaisquer topologias de rede utilizando o cache das informações de roteamento obtidas através dos *switches* que estão conectados ao controlador. O usuário tem a liberdade de inserir novas políticas e novos comportamentos na rede, necessitando apenas alterar a imagem Docker e o *script* em Python que define o controlador.

A partir dessas classes é possível criar experimentos simples com as mais diversas topologias de redes SDN. Entretanto, para a criação de experimentos mais complexos com múltiplos perfis de redes, é possível criar diferentes especializações da classe Node que instanciam e configuram diferentes imagens Docker com diversos serviços e scripts para a reprodução de cenários reais. A exemplificação da criação de um cenário mais complexo será descrita na próxima seção, o qual esclarecerá detalhes de implementação do experimento padrão utilizado nas publicações da ferramenta (Figura 3.2).

4.2.2 Experimento Padrão

O experimento padrão das publicações [1, 11] baseia-se no experimento de [9]. As imagens disponibilizadas consiste em sete imagens que permitem criar uma topologia com vários servidores e clientes que consomem serviços em rede e também realizam ataques na rede, representando máquinas infectadas.

Entre as imagens de servidores disponibilizados tem-se o servidor de arquivos que é basicamente um repositório para o gerenciamento e compartilhamento de arquivos em sistemas operacionais do tipo Linux. O servidor utiliza o *software* Samba para disponibilizar o serviço. Através do comando *mount* torna-se disponível o sistemas de arquivos habilitado no servidor dentro do sistemas de arquivos do usuário, permitindo o acesso para alterar, inserir e deletar arquivos no servidor.

É disponibilizado também uma imagem de um servidor de email, implementado utilizando o Postfix Admin, que é uma ferramenta para a criação de servidores de email. São criados vários endereços de email de usuários no formato "subrede@host", o qual a subrede é o número da sub-rede ao qual o usuário pertence e *host* é o número de IP que identifica o *host* naquela sub-rede. Assim os usuários acessam suas respectivas caixas de email através do IP de cada máquina.

O servidor de *backup* é também um repositório de arquivos compartilhados utilizando Samba, o qual os servidores demais servidores possuem scripts que são executados em horários específicos para fazer uma cópia dos dados de cada aplicação dentro do servidor de *backup* utilizando o Contrab.

O servidor *web* é implementado utilizando o Servidor Web Apache, que é um dos servidores HTTP mais utilizados no mundo. A página disponibilizada é a padrão que vem instalada no próprio Apache que é acessada através da porta 80 no IP que for atribuída ao servidor para os usuários poderem acessar.

O servidor Seafile é um servidor semelhante ao servidor de arquivos, porém é implementado utilizando o Seafile, que é um *software* de hospedagem de arquivos de plataforma cruzada, cujo os arquivos são sincronizados com o usuário através de uma requisição HTTP no IP do servidor e porta 80. Assim, os usuários são capazes de alterar o conteúdo do repositório de arquivos remotamente de uma pasta específica que é compartilhada com múltiplos usuários.

Outra imagem disponibilizada é uma imagem que emula uma impressora que recebe requisições de impressão através da rede. É implementada utilizando Cups PDF, que recebe as requisições de impressão e criar arquivos PDF armazenados separados por usuários na máquina com o Cups PDF.

Por último, é disponibilizado uma imagem que possui scripts em Python que simulam um trabalhador que consome todos os serviços de todos servidores durante o expediente.

Cada cliente possui vários arquivos de configuração que determinam seus comportamentos, como por exemplo, definir quais serviços de rede o cliente pode consumir, o horário de trabalho, se a máquina está infectada (realiza ataques na rede) e se o cliente poderá fazer intervalos para reuniões e *coffee breaks* entre atividades que são definidos em um arquivo de configurações que são copiados dentro dos contêineres[2]. Definido o comportamento do Cliente, a cada tarefa, o cliente escolhe aleatoriamente uma das ações possíveis e as executa e assim por diante, até o fim do expediente do trabalhador.

4.2.3 Ataques

Os contêineres do tipo cliente também podem realizar ataques na rede representando máquinas infectadas. Os ataques disponíveis são *Distributed Denial of Service*(DDos), *Brute Force* e *Port Scan*. O ataque *Brute Force* é realizada através de várias tentativas de conexão SSH com múltiplas senhas a computadores na sub-rede da máquina infectada. O ataque *Port Scan* é feito verificando-se as portas de cada máquina na sub-rede da máquina infectada utilizando a implementação de *Port Scan* do Nmap em Python. O ataque *Distributed Denial of Service* é feita criando vários *sockets* que enviam pacotes à servidores que escutam na porta 80.

Novos scripts de ataques podem ser adicionados à ferramenta adicionando uma nova entrada no arquivo de "attackList.txt" e uma nova chamada no arquivo ´'attacking.py". Assim, quando o cliente iniciar um ataque, será escolhido aleatoriamente um dos ataques implementados.

4.3 Monitoramento e Relatório de Execução

O monitoramento da ferramenta pode ser feito através do uso do Netflow, IPFIX ou sFlow habilitando a o envio de dados dos fluxos dentro dos Open vSwitch *switches*. Assim, em tempo real, é possível obter informações do que está acontecendo na rede. Além disso, é possível definir qual IP e porta esses pacotes serão enviados, podendo ser enviados a um computador remoto para o processamento dos dados de monitoramento da rede.

Para a geração do relatório de execução, é feita a coleta de .PCAP *files* e enviados a um contêiner que tem instalado o CicFlowMeter, que irá processar todos os arquivos e gerará 80 estatísticas a partir dos dados de fluxos coletados durante o experimento. Serão geradas estatísticas como, por exemplo, a duração do fluxo, quantidade de pacotes e bytes transmitidos, vazão em bytes, entre outros. São informações úteis para a criação de *datasets*, treinamento de mecanismos de Inteligência Artificial, entre outros.

[2]https://github.com/alexandrekaihara/lst2.0/tree/main/demonstration/client_behaviour

Capítulo 5

Conclusões 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 do custo de uso, facilitar a portabilidade da ferramenta e flexibilidade para agregar novas funcionalidades à ferramenta. Além disso, priorizou-se a alta configurabilidade da ferramenta para a geração de topologias programaticamente e em tempo real para viabilizar de execução de experimentos em diversos contextos de aplicação.

Como trabalhos futuros, espera-se aprimorar a facilidade de utilização da ferramenta via interface gráfica de usuário, de modo que seja possível executar os experimentos localmente ou em um servidor remoto. Também pretende-se melhorar a coleta de métricas da rede e otimizar o uso de memória das máquinas das imagens pré-construídas que geram fluxos em rede.

Referências

[1] A. Kaihara, et. al.: *Lst: Testbed emulado leve para redes sdn aplicado ao contexto de segurança*. Em *SALÃO DE FERRAMENTAS - Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Sociedade Brasileira de Computação, 2022. 3, 24

[2] M. Khan, O. Rehman, I. Rahman: *Lightweight testbed for cybersecurity experiments in scada-based systems*. Em *020 International Conference on Computing and Information Technology 10 Sep. 2020*, volume 1. IEEE, Tabuk, Saudi Arabia, 2020. 3, 4, 5, 6

[3] M. Šimon, L. Huraj: *Ddos testbed based on peer-to-peer grid*. Em *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*. IEEE, 2016. 3, 4

[4] E. Petersen, Marco Antonio: *Docksdn: A hybrid container-based sdn emulation tool*. Em *2020 IEEE Latin-American Conference on Communications (LATINCOM)*. IEEE, 2020. 3, 4, 5, 6, 13, 14, 15

[5] G. Bonofiglio, et. al.: *Kathará: A container-based framework for implementing network function virtualization and software defined networks*. Em *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018. 3, 4, 6

[6] D. Kreutz, et. al.: *Software-defined networking: A comprehensive survey*. Em *Proceedings of the IEEE*, volume 103. IEEE, 2015. 4, 13

[7] K. Wrona, S. Szwaczyc: *Sdn testbed for validation of cross-layer data-centric security policies*. Em *2017 International Conference on Military Communications and Information Systems (ICMCIS)*. IEEE, 2017. 4, 5, 6

[8] S. Srisawai, P. Uthayopas: *Rapid building of software-based sdn testbed using sdn owl*. Em *2018 22nd International Computer Science and Engineering Conference (ICSEC)*. IEEE, 2018. 4, 5

[9] M. Ring, et. al.: *Flowbased benchmark datasets for intrusion detection*. Em *Proceedings of the 16 th European Conference on Cyber Warfare and Security (ECCWS)*, página 361369. ACPI, 2017. 4, 6, 9, 10, 14, 15, 16, 18, 19, 24

[10] O. Flauzac, E. Robledo: *Is mininet the right solution for an sdn testbed?* Em *IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019. 5, 12, 13

[11] A. Kaihara, et. al.: *Lst 2.0: Testbed emulado baseado em contêineres para redes sdn seguras*. Em *SALÃO DE FERRAMENTAS - SIMPÓSIO BRASILEIRO DE SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS (SBSEG)*. Sociedade Brasileira de Computação, 2022. 12, 24

[12] K. Raghunath, P. Krishnan: *Towards a secure sdn architecture*. Em *9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2018. 13

[13] R. Meena, et. al.: *Ryu sdn controller testbed for performance testing of source address validation techniques*. Em *3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE)*. IEEE, 2020. 13, 14, 15, 16

[14] I. Sumantra, S. Gandhi: *Ddos attack detection and mitigation in software defined networks*. Em *International Conference on System, Computation, Automation and Networking (ICSCAN)*. IEEE, 2020. 13, 15, 16

[15] M. Khan, O. Rehman, I. Rahman: *Lightweight testbed for cybersecurity experiments in scada-based systems*. Em *020 International Conference on Computing and Information Technology 10 Sep. 2020*, volume 1. IEEE, Tabuk, Saudi Arabia, 2020. 13, 14, 16

[16] X. Zhang, N. Prabhu, R. Tessier: *Nestednet: A container-based prototyping tool for hierarchical software defined networks*. Em *International Workshop on Rapid System Prototyping (RSP)*. IEEE, 2020. 13, 15

[17] R. Chadha, et. al.: *Cybervan: A cyber security virtual assured network testbed*. Em *MILCOM 2016 - 2016 IEEE Military Communications Conference*. IEEE, 2016. 13, 14

[18] P. Maity, et. al.: *An effective probabilistic technique for ddos detection in openflow controller*. Em *IEEE Systems Journal*, volume 16. IEEE, 2020. 14

[19] H. Lashkari, et. al.: *Characterization of tor traffic using time based features*. Em *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - ICISSP*. 2017. 15

[20] A. Alashhab, et. al.: *Experimenting and evaluating the impact of dos attacks on different sdn controllers*. Em *IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA*. IEEE, 2021. 17

Apêndice A

Fichamento de Artigo Científico

- Título: LST: Testbed Emulado Leve para Redes SDN Aplicado ao Contexto de Segurança.
  - Kaihara, Alexandre M.; Bondan, Lucas; Gondim, João J. C.; Rodrigues, Gabriel S.; Marotta, Marcelo A.; e Rodrigues, Genaína N.
  - Conferência/Simpósio: Salão de Ferramentas do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2022).
  - Ano: 2022.
- Título: LST 2.0: Testbed Emulado Baseado em Contêineres para Redes SDN Seguras.
  - Kaihara, Alexandre M.; Bondan, Lucas; Gondim, João J. C.; Rodrigues, Gabriel S.; Marotta, Marcelo A.; e Rodrigues, Genaína N.
  - Conferência/Simpósio: XXII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg 2022).
  - Ano: 2022.