UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DETECÇÃO DE ATAQUES SYN-FLOODING EM REDES DEFINIDAS POR SOFTWARE

EDUARDO FARIAS BRINDS-LEY FOX

João Pessoa – PB

Janeiro/2019

UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DETECÇÃO DE ATAQUES SYN-FLOODING EM REDES DEFINIDAS POR SOFTWARE

EDUARDO FARIAS BRINDS-LEY FOX

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba, como parte dos requisitos necessários para a obtenção do título de Mestre em Informática.

ORIENTADOR: PROF. DR. VIVEK NIGAM
CO-ORIENTADOR: PROF. DR. IGUATEMI E. DA FONSECA

João Pessoa – PB

Janeiro/2019

Catalogação na publicação
Seção de Catalogação e Classificação

F791d Fox, Eduardo Farias Brinds-Ley.
Detecção de ataques syn-flooding em redes definidas por software / Eduardo Farias Brinds-ley Fox. - João Pessoa, 2019.
60 f. : il.

Orientação: Vivek Nigam.
Coorientação: Iguatemi Fonseca.
Dissertação (Mestrado) - UFPB/CI.

1. Redes definidas por software. 2. Ataques de negação de serviço. 3. Syn-flooding. 4. Segurança da informação. I. Nigam, Vivek. II. Fonseca, Iguatemi. III. Título.

UFPB/BC

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de Eduardo Farias Brinds-Ley Fox, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 31 de janeiro de 2019.

Aos trinta e um dias do mês de janeiro, do ano de dois mil e dezenove, às dez horas, no Centro de Informática da Universidade Federal da Paraíba, em Mangabeira, reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final do Sr. Eduardo Farias Brinds-Ley Fox, vinculado a esta Universidade sob a matrícula nº 20161020167, candidato ao grau de Mestre em Informática, na área de "Sistemas de Computação", na linha de pesquisa "Computação Distribuída", do Programa de Pós-Graduação em Informática, da Universidade Federal da Paraíba. A comissão examinadora foi composta pelos professores: Vivek Nigam (PPGI-UFPB) Orientador e Presidente da Banca, Iguatemi Eduardo da Fonseca (PPGI-UFPB), Examinador Interno, Gustavo Henrique Matos Bezerra Motta (PPGI-UFPB), Examinador Interno, Moises Renato Nunes Ribeiro (UFES), Examinador Externo à Instituição. Dando início aos trabalhos, o Presidente da Banca cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse a exposição oral do trabalho de dissertação intitulado: "Detectando ataques Syn-Flooding em redes definidas por software". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: "***aprovado***". Do ocorrido, eu, Clauirton de Albuquerque Siebra, Coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai assinada por mim e pelos membros da banca examinadora. João Pessoa, 31 de janeiro de 2019.

Prof. Dr. Clauirton de Albuquerque Siebra

Prof. Dr. Vivek Nigam
Orientador (PPGI-UFPB)

Prof. Dr. Iguatemi Eduardo da Fonseca
Examinador Interno (PPGI-UFPB)

Prof. Dr. Gustavo Henrique Matos Bezerra Motta
Examinador Interno (PPGI-UFPB)

Prof. Dr. Moises Renato Nunes Ribeiro
Examinador Externo ao Programa (UFES)

AGRADECIMENTOS

A Deus por ter sido o meu sustento e a razão por qual cheguei até aqui;

Aos meus pais, Maria da Consolação e John, por estarem sempre por perto me apoiando;

À Luciana, minha noiva, que tanto me ajudou com seus conselhos de perseverança;

As minhas irmãs que sempre vibraram com minha felicidade;

Aos amigos da Casa de Evangelização, que me incentivam a dar um bom testemunho de vida em todas as áreas e me ensinam sobre a caridade;

À Ariane, Zenildo, Túlio e João que me ajudaram na troca de conhecimentos sobre SDN;

Ao meu orientador Dr. Vivek Nigam por me ensinar muito com sua paciência e humanidade, sempre me incentivando em cada reunião;

Ao meu co-orientador Dr. Iguatemi Fonseca por também me ajudar, mostrando de forma clara aonde eu precisava melhorar;

A todos os professores do PPGI da UFPB, que eu tive a oportunidade de conhecer e aprender com exemplos de disciplina e comprometimento.

"Pedi e se vos dará. Buscai e achareis. Batei e vos será aberto. Porque todo aquele que pede, recebe. Quem busca, acha. A quem bate, abrir-se-á."

Mt 7,7-8

RESUMO

Com a quantidade de informações disponíveis na Internet, internautas podem facilmente realizar um ataque DoS apenas seguindo um tutorial disponível, sem precisar ter muito conhecimento computacional para esta ação. O *Syn-flooding* é um ataque simples de ser realizado, porém tem consequências desastrosas, podendo impossibilitar o acesso a um site ou outros recursos em uma rede. Em Redes Definidas por *Software* (SDN), este tipo de ataque também pode afetar toda a infraestrutura, podendo parar uma rede por completo, a partir da negação do serviço do próprio controlador. Este trabalho propõe uma detecção de ataques *Syn-flooding*, em uma rede SDN, através da medição da variação da quantidade de fluxos em um intervalo de tempo pré-estabelecido e o monitoramento das portas TCP, auxiliando ao administrador de rede a realizar ações corretivas e preventivas a partir da detecção do ataque. Para a realização da proposta, foi desenvolvida uma ferramenta chamada FindFlows que exibe uma lista de todos os *hosts* ativos em uma rede SDN, informando a quantidade de fluxos de cada *host* em intervalos de tempo diferentes, a variação dos fluxos nesses intervalos e, por fim, a classificação deste *host* como atacante, vítima ou usuário legítimo. Dos testes realizados, o FindFlows conseguiu detectar o ataque *Syn-flooding* em 90% dos casos.

Palavras-chave: Redes Definidas por *Software*, Ataques de Negação de Serviço, *Syn-flooding*, Segurança da Informação

ABSTRACT

With the amount of information available on the Internet, one can easily perform a DoS attack by just following an available tutorial, without having to have much computational knowledge for this action. Syn-flooding is a simple attack to be carried out, but has disastrous consequences, making it impossible to access a site or other resources on a network. In Software Defined Networks (SDN), this type of attack can also affect the entire infrastructure and can stop a network altogether, from the denial of the service of the controller itself. This work proposes the detection of Syn-flooding attacks in an SDN network by measuring the variation of the amount of flows in a pre-established time interval and the monitoring the TCP ports, helping the network administrator to perform corrective and preventive actions from the detection of the attack. To implement the proposal, a tool called FindFlows has been developed that displays a list of all the active hosts in an SDN network, informing the amount of flows of each host in different time intervals, the variation of the flows in those intervals and, finally, the classification of this host as an attacker, victim or legitimate user. Of the tests performed, the FindFlows was able to detect the Syn-flooding attack in 90% of cases.

Keywords: Software Defined Networking, Denial of Service, Syn-flooding, Information Security

LISTA DE FIGURAS

LISTA DE TABELAS

GLOSSÁRIO

ACK – *Acknowledgement*

API – *Application Programming Interface*

CPU – *Central Process Unit*

DDoS – *Distributed Denial of Service*

DoS – *Denial of Service*

FTP – *File Transfer Protocol*

HTTP – *Hypertext Transfer Protocol*

IETF – *Internet Engineering Task Force*

IP – *Internet Protocol*

IPv4 – *Internet Protocol Version 4*

IPv6 – *Internet Protocol Version 6*

NASA – *National Aeronautics and Space Administration*

SDN – *Software Defined Network*

SGBD – *Sistema Gerenciador de Banco de Dados*

SMTP – *Simple Mail Transfer Protocol*

SYN – *Synchronize*

TCAM – *Ternary Content-Addressable Memory*

TCP – *Transport Control Protocol*

TLS – *Transport Layer Security*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Com o crescimento da Internet, as informações ficam cada vez mais fáceis de serem acessadas. Assim também mais facilmente se consegue tutoriais ou *scripts* para realizar ataques pelas Redes de Computadores, sem necessidade de ter um grande conhecimento computacional para isso. Dessa forma, os Ataques de Negação de Serviço (DoS - *Denial of Service*), conhecidos como DoS, passaram a ser algo que um internauta pode realizar com um mínimo de conhecimento, podendo atingir as redes ou servidores que encontram-se mais vulneráveis, ocasionando em consequências desastrosas.

A disponibilidade é um dos pilares da segurança da informação [9] e é este pilar que um ataque de negação de serviço tem o objetivo de quebrar. Ataques do tipo DoS tem como intenção impedir o acesso a um recurso na rede, como um *web site*, ou algum serviço fornecido aos usuários [10], tornando este recurso indisponível. Existem várias formas de realizar esse impedimento de acesso, cada forma caracteriza um ataque diferente dentro da categoria DoS. Quando um ataque deste gênero se origina a partir de vários pontos diferentes, é chamado de Ataque de Negação de Serviço Distribuído (DDoS - (*Distributed Denial of Service*). Nas Olimpíadas do Rio, em 2016, a Arbor Networks[1] identificou uma combinação de diferentes ataques DDoS direcionados à rede utilizada para os jogos, tal ataque atingiu 540 Gbps de tráfego ilegítimo com a intenção de parar as olimpíadas. Dentro destes vários ataques, o *Syn-flooding* era um deles.

O *Syn-flooding* é um tipo de ataque que envia inúmeras solicitações de conexões TCP com a *flag* SYN para a vítima, sendo esta um servidor de alguma aplicação. Para cada uma dessas requisições, o servidor reserva recursos para tratá-las. Porém, centenas ou até milhares de solicitações de estabelecimento de conexões fazem esses recursos se esgotarem. Assim, o objetivo do atacante é esgotar esses recursos para que o servidor não consiga mais responder as cone-

[1] https://asert.arbornetworks.com/ddos-attacks-iot-botnets-dont-mean-game Acessado em: 30/11/2018

xões legítimas [2]. Em Redes Definidas por *Software* (SDN - *Software Defined Networking*) esse ataque gera um novo problema como é descrito a seguir.

SDN é um novo conceito de Redes de Computadores que altera a administração distribuída da rede para uma administração centralizada e de maior flexibilidade para o administrador da rede [11]. Em uma rede convencional cada ativo, como *switches* e roteadores, possui plano de dados e de controle acoplados. Isso quer dizer que para uma mudança em toda parte lógica da rede, é necessário configurar cada um dos dispositivos por vez, sem ter algo central que se comunique com todos os dispositivos e seja responsável por simplificar o trabalho do administrador da rede. Em SDN, o plano de controle é desacoplado dos ativos e colocado a parte, em um controlador. Assim, toda a rede local passa a se tornar programada por meio de um ponto central, facilitando a criação de novas abstrações na rede, proporcionando evolução e inovação às redes de computadores [12].

Portanto, para que o controlador se comunique com os ativos da rede, se faz necessário o uso de um protocolo. OpenFlow é o protocolo mais utilizado em SDN, basicamente ele faz a comunicação entre o controlador da rede (plano de controle) com os *switches* (plano de dados). Cada *switch* SDN possui uma tabela de fluxos que é administrada pelo controlador e nessa tabela existe as regras que indicam que ação tomar com cada novo pacote. Para cada pacote que chega ao *switch*, este verifica se já existe uma entrada na tabela de fluxos informando o que fazer com aquele determinado pacote. Caso não exista nenhuma entrada na tabela informando a ação necessária para o novo pacote, o *switch* encapsula o novo pacote dentro de uma mensagem chamada *packet-in*, do protocolo OpenFlow, e encaminha para o controlador, esperando uma resposta sobre qual a ação a ser tomada [2].

Em uma rede tradicional, um ataque *Syn-flooding* pode saturar os recursos de um servidor de uma aplicação, tornando-o indisponível. Porém, conforme afirma Mohammadi et al. [2], em uma rede SDN o plano de controle é vulnerável a ataque de saturação, como o *Syn-flooding*, fazendo com que este ataque possa tornar o controlador indisponível ou fazê-lo diminuir sua performance em responder requisições de usuários legítimos, podendo ter consequências maiores, como a de tornar toda uma rede indisponível. Isto é possível porque cada pacote deste ataque é uma nova solicitação de conexão, possuindo cabeçalhos diferentes e, consequentemente, gerando inúmeras mensagens *packet-in* com destino ao controlador da rede, fazendo este trabalhar exaustivamente [2]. Assim, o *Syn-flooding*, mesmo sendo um ataque a uma aplicação, afeta a infraestrutura de uma Rede Definda por *Software*.

Considerando a vulnerabilidade que um controlador SDN possui em relação a um ataque de saturação, este presente trabalho contribui na detecção de um dos tipos de ataque DoS em

uma rede SDN, o *Syn-flooding*. Por meio de uma ferramenta desenvolvida nesta pesquisa, chamada FindFlows, é possível detectar o ataque *Syn-flooding*, assim como os *hosts* da rede que possivelmente estejam participando do ataque. A ferramenta FindFlows atua como um monitor de rede (sendo algumas vezes denominada desta forma neste trabalho). A detecção é realizada monitorando o comportamento do crescimento da quantidade de fluxos da rede, dos endereços IP e das portas TCP. Ele pode atuar inserido no próprio controlador SDN ou em um servidor exclusivo para atuar como monitor, ficando esta decisão a cargo do administador da rede, gerando alertas quando é detectado o ataque. Mesmo esta pesquisa considerando a vulnerabilidade do controlador, isto não oculta as características positivas de uma rede SDN, sendo a flexibilidade de uma Rede Definida por *Software* um dos fatores essenciais ao desenvolvimento do detector de ataque *Syn-flooding*.

1.1 Objetivos

1.1.1 Objetivo Geral

Desenvolver uma forma de detectar um ataque DoS *Syn-flooding* em uma rede SDN, por meio do monitoramento do comportamento dos fluxos em um período de tempo, informando assim os *hosts* participantes do ataque.

1.1.2 Objetivos Específicos

- Estudar a comunicação, via protocolo OpenFlow, entre *Switch* e controlador;
- Estudar o comportamento do ataque *Syn-flooding* e como ele age em uma rede SDN;
- Realizar testes de ataques *Syn-flooding* em uma rede SDN em um cenário mais próximo do real;
- Desenvolver uma ferramenta que detecte um ataque *Syn-flooding* em uma rede SDN e informe os *hosts* participantes no ataque;
- Validar a ferramenta de detecção do ataque por meio de emulações de um cenário em uma rede SDN.

1.2 Estrutura da Dissertação

Além deste presente capítulo, o trabalho está dividido em mais 4 capítulos, conforme a descrição de cada um a seguir:

Capítulo 2: Apresenta a fundamentação teórica necessária para compreender a proposta do trabalho, como por exemplo os conceitos de Redes Definidas por *Software*, Ataques de Negação de Serviço, funcionamento do *Syn-flooding* e métodos de identificação;

Capítulo 3: A proposta da dissertação consta neste capítulo. Aqui é apresentada a arquitetura utilizada, a coleta dos dados para detecção do ataque e a comparação com os trabalhos relacionados;

Capítulo 4: Mostra os resultados adquiridos por meio da execução de experimentos realizados nas simulações de cenários reais. Também é neste capítulo que todas as ferramentas utilizadas são apresentadas;

Capítulo 5: Apresenta as considerações finais e os trabalhos futuros.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

Este capítulo visa introduzir o conhecimento necessário para a compreensão da proposta. Para isso, o capítulo encontra-se dividido nas seguintes seções: Um breve histórico até SDN; Fundamentos de SDN; Entendendo os ataques de Negação de Serviço; Funcionamento do ataque *Syn-flooding*; Trabalhos Relacionados.

2.1 Um breve histórico até SDN

Segundo Farhady [13], as Redes tradicionais sofrem com inovação, gerenciamento, flexibilidade e escalabilidade. Isso acontece devido a todos os dispositivos de uma rede tradicional terem plano de dados e controle acoplados, tornando-a distribuída e trazendo grandes problemas de evolução, como no caso da transição entre IPv4 e IPv6 que está em processo há mais de uma década e ainda não foi concluída em diversos países [12].

Antes de chegar até o conceito de Redes Definidas por *Software*, houve um período de estudos, pesquisas e testes. Um passo foi dado de cada vez até chegarmos no que se tem hoje. O estado da arte de cada período do processo evolutivo das redes está dividido em: *Active Networking*; Separação do plano de controle; Surgimento do OpenFlow e SDN; [14].

2.1.1 Active Networking

Na década de 90, a Internet se tornava cada vez mais popular. Isso fazia com que os pesquisadores da época ficassem ansiosos por testar e desenvolver novas ideias para melhorar os serviços de redes. Assim, eles faziam simulações de grandes redes, testando o comportamento das novas soluções. Quando estas novas ideias passavam nos testes de laboratório e continuavam sendo motivadoras, os pesquisadores as apresentavam para a Força Tarefa de Engenharia

da Internet (IETF - *Internet Engineering Task Force*), com o propósito de padronizar os novos protocolos. Porém, essa padronização demorava bastante, desestimulando muitos pesquisadores.

Assim, por mais que de um lado existisse uma comunidade pesquisadores que queriam desenvolver, testar, padronizar novos protocolos e funcionalidades da rede, do outro lado existia uma rede que dificultava a inovação, por não possuir uma interface programável, além de uma entidade que não homologava a padronização desses novos protocolos no tempo desejado. Era necessário ultrapassar essa barreira.

Dessa forma, alguns pesquisadores tiveram uma ideia bastante desafiadora: abrir o plano de controle da rede e reprogramá-lo para tornar o ambiente de rede mais propício a inovação e pesquisa de novas funcionalidades. Assim foi criada as Redes Ativas (*Active Networking*) com o objetivo de criar uma interface de programação que permitisse o acesso aos recursos de cada nó da rede, possibilitando customizações no tratamento dos pacotes e o suporte as novas funcionalidades.

As Redes Ativas tiveram a adoção por parte de uma comunidade de pesquisadores, porém sofreu bastante crítica da própria comunidade da Internet da época, que defendia que a simplicidade do núcleo da rede era crucial para o sucesso da Internet. Assim, as Redes Ativas não tiveram uma adoção geral por parte da comunidade. A falta de um problema imediato a ser solucionado ou até mesmo a falta de clareza no desenvolvimento também contribuiu para a sua não aceitação. As dificuldades encontradas fizeram os pesquisadores estreitar mais o escopo do problema, focando em separar o plano de controle do plano de dados, assim seria possível trabalhar na evolução do plano de controle, por ser normalmente implementado em *software* [14].

2.1.2 Separação do plano de controle

Já nos anos 2000, a vazão das redes aumentavam cada vez mais, possibilitando grandes volumes de tráfego, melhorando nos quesitos de confiabilidade e performance. Isso fez com que os operadores de rede procurassem melhorias para os desafios de engenharia de tráfego que viam surgindo. Porém, perceberam que os protocolos usados para o tratamento dos caminhos do tráfego eram praticamente os mesmos de sempre. Assim, se depararam com um problema de que os atuais protocolos de redes estavam bem atrás dos avanços que a infraestrutura da rede recebia.

Uma comunidade de pesquisadores percebia os desafios que os operadores de redes passavam e buscavam soluções rápidas com os protocolos já existentes. Com um tempo, perceberam

que o acoplamento de plano de controle e dados em cada roteador e *switch* da rede faziam os administradores terem muito trabalho, como: encontrar erros de configurações e prever o comportamento do roteamento. Para resolver esses problemas, iniciou-se os esforços para separar o plano de controle do plano de dados nos ativos de rede. Estes esforços resultaram em um início de uma base para o que seria posteriormente as Redes Definidas por *Software*. O grupo de trabalho, como o ForCES, havia proposto um padrão para o IETF, solicitando uma abertura de uma interface para o plano de dados, com o intuito de garantir a inovação dos *softwares* do plano de controle.

Novamente, a iniciativa ainda não tinha muita força dentro dos grandes fabricantes. Estes tiveram pouco incentivo para adotar o padrão de plano de dados com a API da ForCES. Além do mais que a abertura do plano de dados para essa API poderia incentivar novos participantes no mercado. Assim, para aumentar o incentivo da separação de plano de controle e de dados por parte dos fabricantes, pesquisadores trabalhavam para realizar algo mais atraente. Nesse caminho, um projeto chamado Ethane, criou uma lógica centralizada, a nível de fluxos. Tal projeto reduziu os *switches* para tabela de fluxos as quais eram preenchidas pelo controlador da rede. O Ethane foi testado no Departamento de Ciências da Computação de Stanford, sendo a base para a criação do protocolo OpenFlow, utilizado em SDN [14].

O ForCES e OpenFlow possuiam algo em comum, os dois separaram o plano de controle do plano de dados, padronizando a comunicação entre os dois planos. Porém, segundo um próprio documento do IETF que compara os dois padrões [15], algumas diferenças fizeram estes tomarem diferentes rumos. Na Tabela 2.1 é apresentada uma comparação, destacando as principais diferenças entre os padrões ForCES e OpenFlow.

Tabela 2.1: Comparação entre padrões: ForCES e OpenFlow.

ForCES	**OpenFlow**
Apenas define como os elementos de rede e encaminhamento podem se comunicar entre si;	Os elementos responsáveis pelo plano de dados se tornam simples dispositivos que encaminham pacotes de acordo com as regras do elemento de controle;
Arquitetura de rede inalterada;	Arquitetura de rede modificada;
Elementos lógicos e de controle de rede podem ser descentralizados.	Plano de controle centralizado.

2.1.3 Surgimento do OpenFlow e SDN

Com o OpenFlow, pesquisadores e agências de financiamento mostraram-se bem interessados em realizar mais experimentos. O protocolo se mostrou de fácil implantação, tornando as Redes Definidas por *Software* algo cada vez mais aplicável. Assim, rapidamente surgiu a plataforma de controle NOX, que possibilitou a criação de novas aplicações de controle [14]. A arquitetura do OpenFlow é dividida em três principais componentes: *switch* com OpenFlow compatível, o canal seguro e o controlador. Assim, os *switches* utilizam as tabelas de fluxos para encaminhar os pacotes. Uma tabela de fluxos é uma lista de entrada de fluxos, cada entrada possui campos de correspondência do fluxo, contadores e instruções ou regras. Quando os pacotes chegam no *switch* os campos do cabeçalho destes são comparados com os campos de correspondência de cada entrada existente na tabela de fluxos e, caso exista uma combinação, o pacote é processado de acordo com a ação da entrada combinada. Já os contadores são utilizadas para as estatísticas acerca dos pacotes. Caso não seja encontrada nenhuma entrada na tabela de fluxos, que combine com o novo pacote que chegou ao switch, o pacote é encapsulado e enviado ao controlador [16].

Porém, ainda era necessário realizar testes maiores, em grandes *campi*, para demonstrar a capacidade do protocolo. Assim, o grupo de pesquisa da Universidade de Stanford se esforçava para testá-lo em vários *campi*, para provar que era possível utilizá-lo também em uma grande área. A medida que os testes iam dando certo, SDN passava a ser implantada em redes maiores, como a dos *data-centers*, fazendo com que os fabricantes tivessem cada vez mais interesse em

produzir *switches* que suportassem OpenFlow, por haver um interesse do próprio mercado por essa nova tecnologia.

As Redes Definidas por *Software* trouxeram uma grande autonomia ao administrador da rede, porque não dita como o controlador deve ser projetado para resolver determinado problema. Ao contrário, dá aos pesquisadores e administradores da rede uma plataforma disponível para contribuir na solução dos problemas e desenvolver novos serviços. Por isso, esse novo conceito de rede contribui de forma assídua no quesito inovação [14]. Neste conceito de redes, o sistema operacional da rede é o ponto chave, trazendo a idea de completa abstração das camadas mais baixas e complexas. Tomando como exemplo um sistema operacional de um computador, a abstração deste inclui os componentes de *hardware* da CPU. Assim também é o sistema operacional de uma rede, a abstração esconde a topologia e os ativos da rede [16].

Por outro lado, a programabilidade e centralização de uma Rede SDN, que em muitos aspectos pode ser compreendida como um avanço da tecnologia, não deixa de também ser visto como uma vulnerabilidade e exposição para ataques de redes. Assim, também existe uma preocupação, por parte de outros pesquisadores, com a ascensão das Redes Definidas por *Software* e seus possíveis problemas de segurança [17].

2.2 Fundamentos de SDN

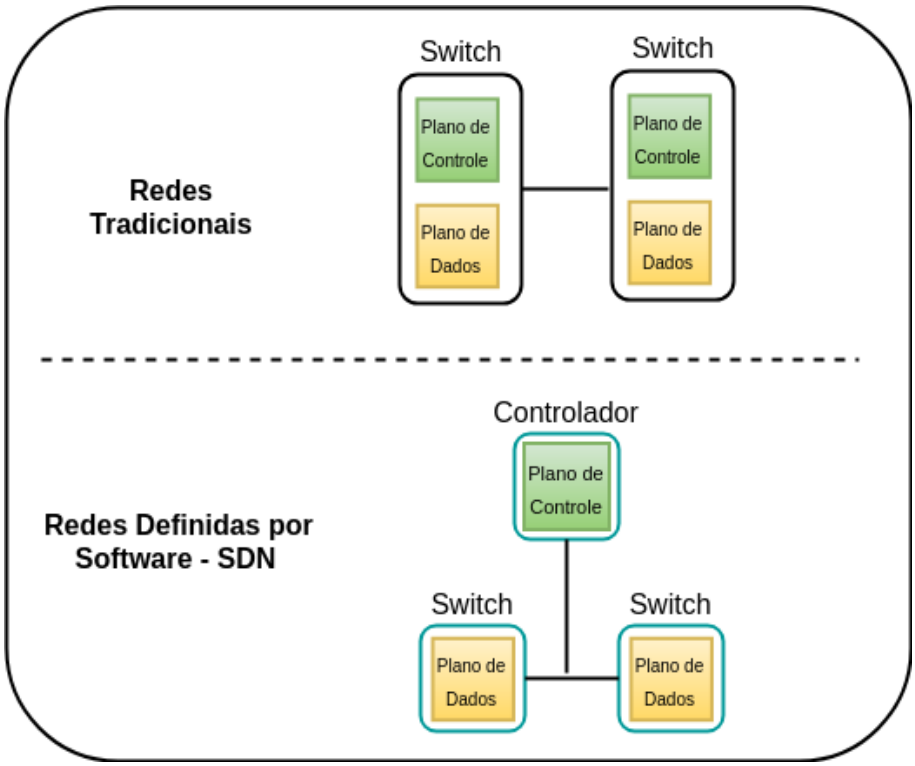

Figura 2.1: Comparativo entre redes tradicionais e SDN.

Na Figura 2.2, percebemos a maior diferença entre uma rede tradicional e uma rede definida por *Software*. Em uma rede tradicional, os *switches* possuem tanto plano de dados como de controle, justificando assim ser uma rede com o controle distribuído. Já em SDN, o plano de controle é separado dos ativos da rede (representados por *switches* na figura) e colocado a parte, em um novo nó, chamado controlador. Essa separação soluciona os problemas de dificuldades em inovação, gerenciamento, flexibilidade e escalabilidade que as redes tradicionais passaram a encontrar.

Segundo Kreutz et al. [12], plano de controle e dados são definidos da seguinte forma:

- **Plano de Controle:** Responsável por decidir como tratar o tráfego da rede e por qual caminho deve ser encaminhado;
- **Plano de Dados:** Responsável por encaminhar o tráfego da rede de acordo com as decisões tomadas pelo plano de controle.

Assim, fazendo uma analogia com uma empresa, a direção, onde se realiza a tomada de decisões, seria o plano de controle. Já o setor administrativo, onde se executa o planejamento e

as decisões da direção, seria o plano de dados. Em outras palavras, em SDN, o controlador da rede decide as ações e, os *switches* ou outros ativos, as executam conforme a decisão realizada pelo controlador.

A inserção de um elemento central na rede, com a função de controlador, traz a esta a característica de centralização de todo o controle lógico da rede. Segundo Kreuz *et al*. esta centralização leva três benefícios adicionais à rede, como:

- Configuração mais simples e menos propensas a erros comparadas com as específicas formas de configuração de cada dispositivo de uma rede tradicional;
- Possibilidade de ser configurada para automaticamente reagir a comportamentos anormais da rede (um ataque DoS, por exemplo);
- Simplificação no desenvolvimento de funções, serviços e aplicações de rede mais sofisticadas.

Cada *switch* SDN possui uma tabela de fluxos onde são armazenadas as regras que informam as ações a serem tomadas para cada pacote, conhecidas como regras de fluxo, podendo ser proativas ou reativas. As regras proativas são aquelas que já são inseridas antes da inicialização do *switch*, sem precisar de uma comunicação com o controlador. Ao contrário, as regras reativas são inseridas através do controlador, de forma dinâmica, de acordo com os novos pacotes que são recebidos pelo *switch*. Segundo Moshref *et al*. [18], as regras reativas possuem uma performance mais baixa, devido a necessidade de um tempo de espera da resposta do controlador da rede. Por exemplo, se um *switch* de uma rede SDN acabou de ser ligado e ainda não recebeu nenhum pacote, a tabela de fluxos estará sem nenhuma regra de fluxo recebida do controlador, ou seja, de modo reativa. No primeiro pacote que o *switch* receber, ele verificará se existe alguma regra correspondente para aquele pacote, na tabela de fluxos. Assim, não encontrando uma regra correspondente, o *switch* perguntará ao controlador que ação ele deverá realizar com o determinado pacote. Em seguida, o controlador responderá com o envio da regra em que estará a ação necessária a se aplicar com o pacote em questão e o *switch* armazenará a regra em sua tabela de fluxo. Caso um pacote, com o cabeçalho semelhante ao já enviado, chegue ao *switch* novamente, este não precisará mais solicitar uma nova regra ao controlador, por já possuir esta regra adicionada em sua tabela de fluxos. Porém, caso chegue um novo pacote, com o cabeçalho completamente diferente dos pacotes anteriores, um novo procedimento de consulta ao controlador deverá ser realizado, sendo necessária adicionar uma nova regra de forma reativa.

O OpenFlow é o protocolo que realiza a comunicação entre o Controlador e os *switches* SDN. Toda troca de pacotes é realizada de forma segura, devido a conexão ser realizada por meio do protocolo TLS (*Transport Layer Secure*) que prover autenticação e confidencialidade na troca de informações. Por padrão, o estabelecimento da conexão TLS é iniciada por parte do *switch* em direção ao controlador, que escuta na porta 6653 do protocolo da camada de transporte TCP. Depois disso, é realizado todo o procedimento de troca de certificados para garantir a segurança na comunicação. Só após esses procedimentos é que as informações de controle da rede podem ser trocadas entre o controlador e os *switches* [19]. Segundo He *et al*. [20], a comunicação entre o *switch* e o controlador ainda possui uma alta latência devido a questões de *hardware* do *switch* SDN.

Três principais componentes são verificados em um *switch* SDN, são eles: Comunicação, Sistema Operacional e Plano de Dados. O componente Comunicação é responsável pela interface de comunicação entre o *switch* e o controlador, atualmente o protocolo padrão que desempenha esta função é o OpenFlow [1] (ver Figura 2.2).

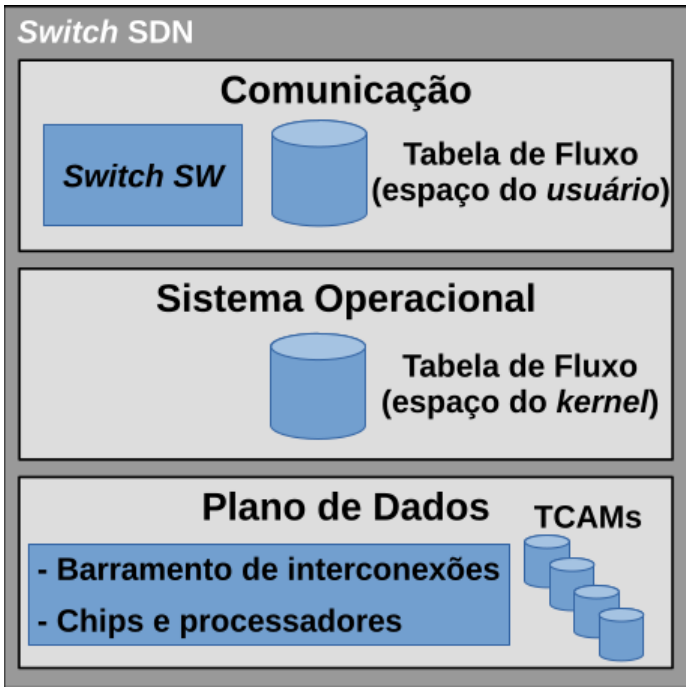

Figura 2.2: Arquitetura de um switch SDN. Adaptado de Mafioletti [1].

O Sistema Operacional é o responsável por fazer a conversão entre a linguagem de alto nível advinda do próprio protocolo OpenFlow para a linguagem de máquina (baixo nível) do ativo de rede. Ele também abstrai toda a camada de *hardware*, para facilitar o desenvolvimento de *drivers* na camada de comunicação [1].

O responsável pelos encaminhamentos entre as portas do *switch* é o componente Plano de

Dados. Em *switches* implementados com *hardware* para SDN, uma memória chamada TCAM (*Ternary Content-Addressable Memory*) armazena a tabela de fluxos. Esta memória é de alto custo devido a poder consultar todas as entradas da tabela de fluxo em alta velocidade, com apenas um ciclo de *clock* do processador. Devido ao alto custo, nem todos os *switches* possuem esta memória e, os que possuem, em sua grande maioria apresentam um tamanho da TCAM relativamente pequeno. Em *switches* que não possuem memória TCAM, a tabela de fluxos pode ser implementada tanto no espaço do usuário, como no espaço do *kernel*, sendo que este último garante uma melhor performance em comparação com as tabelas armazenadas no espaço do usuário [1].

No protocolo OpenFlow existem vários tipos de mensagens que são utilizadas para a comunicação entre os nós de uma rede SDN. As mensagens de necessário conhecimento para a compreensão deste trabalho, são as seguintes:

- **Packet-in:** Utilizada pelo *switch* quando este recebe um novo pacote. Esta mensagem é enviada ao controlador, a fim de se obter a ação necessária para o novo pacote;
- **Packet-out ou Flow-mod:** Resposta do controlador a mensagem de packet-in enviada pelo *switch*. Nesta mensagem, o controlador responde com uma lista de ações que dita o que o *switch* deve fazer com o novo pacote. O *switch*, ao receber esta mensagem, armazena em sua tabela de fluxos a nova regra informada pelo controlador;
- **Flow-removed:** Enviada pelo *switch* ao controlador com o intuito de informar sobre a eliminação de uma regra de fluxo de sua tabela, por motivos de expiração de *timeout*.

Com o alto custo dos *switches* implementados com *hardware* próprio para SDN, a utilização de simuladores ou emuladores para prototipação de Redes Definidas por Software é algo frequentemente realizado para testar novas funcionalidades, de forma rápida e barata, neste tipo de rede.

2.2.1 O emulador Mininet

O Mininet é um emulador que possibilita o desenvolvimento de grandes redes, mesmo quando se tem recursos limitados, como de um simples computador ou de uma máquina virtual [21]. Criado especificamente para suportar pesquisas em SDN e OpenFlow, este emulador prover simplificação e realismo a um custo muito baixo, além de oferecer uma boa performance e escalabilidade [21].

Um grande benifício de se utilizar um emulador como Mininet é que, ao contrário dos simuladores, o mesmo código poderá ser implantado em rede real, após ter sido testado no Mininet [22]. Isso também poderia ser feito se fosse utilizado apenas máquinas virtuais, mas o grande problema neste caso está na escalabilidade, porque seria necessário criar uma máquina virtual para cada dispositivo ou *host* de rede, chegando ao limite rapidamente devido ao limitado recurso computacional que um simples *laptop* oferece [22].

Para criar uma rede SDN, o Mininet emula o controlador, os *hosts*, os *switches* e até mesmo a conexão entre estes dispositivos. Faz isso utilizando um leve mecanismo de virtualização em cima de um sistema operacional Linux [22].

Neste trabalho, foi utilizado o Mininet para prototipação da rede SDN e execução dos testes para a validação da pesquisa. Para os tipos de testes realizados e, sendo configurado com um bom *hardware*, o Mininet possibilita resultados fiéis aos resultados que seriam aplicados em um cenário de rede real. É um emulador utilizado por mais de 100 pesquisadores espalhados por mais de 18 instituições, sendo muitas renomadas como a Universidade de Stanford e a NASA [22]. Assim, muitas pesquisas em SDN, no atual estado da arte, tem sua validação neste emulador, sendo justificada a sua utilização para este trabalho.

2.2.2 Controladores SDN

Ao contrário de uma rede tradicional em que o plano de controle é distribuído entre os dispositivos da rede, em SDN a parte inteligente fica em um nó central, chamado controlador. Este é o responsável pelo plano de controle de toda a rede. Por ter sido desacoplado dos ativos da rede e colocado em um nó central, o plano de controle agora é gerenciado por um sistema operacional (controlador) que é o responsável pela interação do plano de controle com o plano de dados. Essa interação requer uma simplificação e abstração do que antes era baixo nível, oferecendo uma linguagem de programação de alto nível para que o administrador da rede possa programá-la da forma que bem entender.

Assim, em um arquitetura SDN, existem vários tipos de controladores, cada um respondendo por características e funcionalidades específicas de acordo com a rede desejada, como redes de grande porte ou até mesmo redes de testes em cenários de pesquisas. Logo abaixo, alguns dos principais controladores e suas caraterísticas são listados.

O NOX [23] é um controlador robusto, voltado para alto desempenho, desenvolvido na linguagem C++, se utilizando também desta linguagem para o desenvolvimento de aplicações. Foi o primeiro controlador SDN desenvolvido e oferece suporte para a versão 1.0 do OpenFlow.

Porém, em uma versão modificada pelo CPqD (Centro de Pesquisa e Desenvolvimento em Telecomunicações), passou suportar parcialmente a versão 1.3 do OpenFlow [24].

O POX [25] é uma versão do controlador NOX com uma interface mais acessível, por utilizar Python como linguagem de programação. Assim, é utilizado como mais uma alternativa ao NOX. Normalmente é aplicado em cenários de testes, como prototipação de Redes SDN. Atualmente suporta a Versão 1.0 do OpenFlow.

Os controladores OpenDayLight, Beacon, Floodlight e Maestro possuem objetivos semelhantes de aumentar a popularização do uso de SDN. São escritos na linguagem Java [12]. Com excessão do OpenDayLight e do Floodlight, que suportam a Versão 1.3 do OpenFlow, todos os demais dão suporte apenas ao OpenFlow 1.0.

Desenvolvido por um grupo japonês da NTT Lab's, o Ryu [26] é um controlador *Open Source* desenvolvido em Python. Uma forte vantagem deste controlador é que ele não apenas dá suporte ao protocolo OpenFlow, mas também ao NetConf e OF-Config. Outro lado positivo é que por ter um desenvolvimento constante por parte da comunidade, dá suporte sempre as versões mais atuais do OpenFlow. Atualmente, está dando suporte de forma completa a todas as versões do OpenFlow, da 1.0 até a Versão 1.5. Também possui uma API que facilita a criação de aplicações de controle e gerenciamento de uma rede SDN.

A Tabela 2.2 apresenta uma lista com os controladores apresentados e as suas principais características.

Tabela 2.2: Resumo das características dos principais Controladores. Adaptado de Corrêa [8].

Controlador	**Suporte OpenFlow**	**Licença**	**Linguagem API**
NOX	1.0	GPLv3	C++
POX	1.0	GPLv3	Python
OpenDayLight	1.0 e 1.3	EPL v1.0	Java
Beacon	1.0	GPLv2	Java
FloodLight	1.1 e 1.3	Apache	Java
Maestro	1.0	LGPLv2.1	Java
RYU	1.0, 1.1, 1.2, 1.3, 1.4 e 1.5	Apache 2.0	Python

Este trabalho utilizou o controlador Ryu. A escolha se justifica devido aos vários benefícios explicitados.

2.3 Entendendo os Ataques de Negação de Serviço

Ataques DoS possuem a intenção de fazer com que usuários legítimos não consigam acessar algum recurso específico da rede [27]. Quando um servidor possui uma disponibilidade muito alta de recursos (processamento, memória ou banda larga), é necessário um maior esforço por parte do atacante. Assim, este faz um recrutamento de máquinas infectadas para aumentar a eficiência do ataque. A essas máquinas infectadas, dá-se o nome de *zombies*. O objetivo então é fazer com que essas máquinas enviem, de forma coordenada pelo atacante, um enorme número de pacotes para o servidor de destino (ver Figura 2.3), causando uma sobrecarga, fazendo com que os usuários legítimos não consigam acessar os recursos do servidor [28]. A este tipo de ataque com mais de uma origem, dá-se o nome de DDoS, sendo utilizado com maior frequência, devido a ter uma maior eficácia.

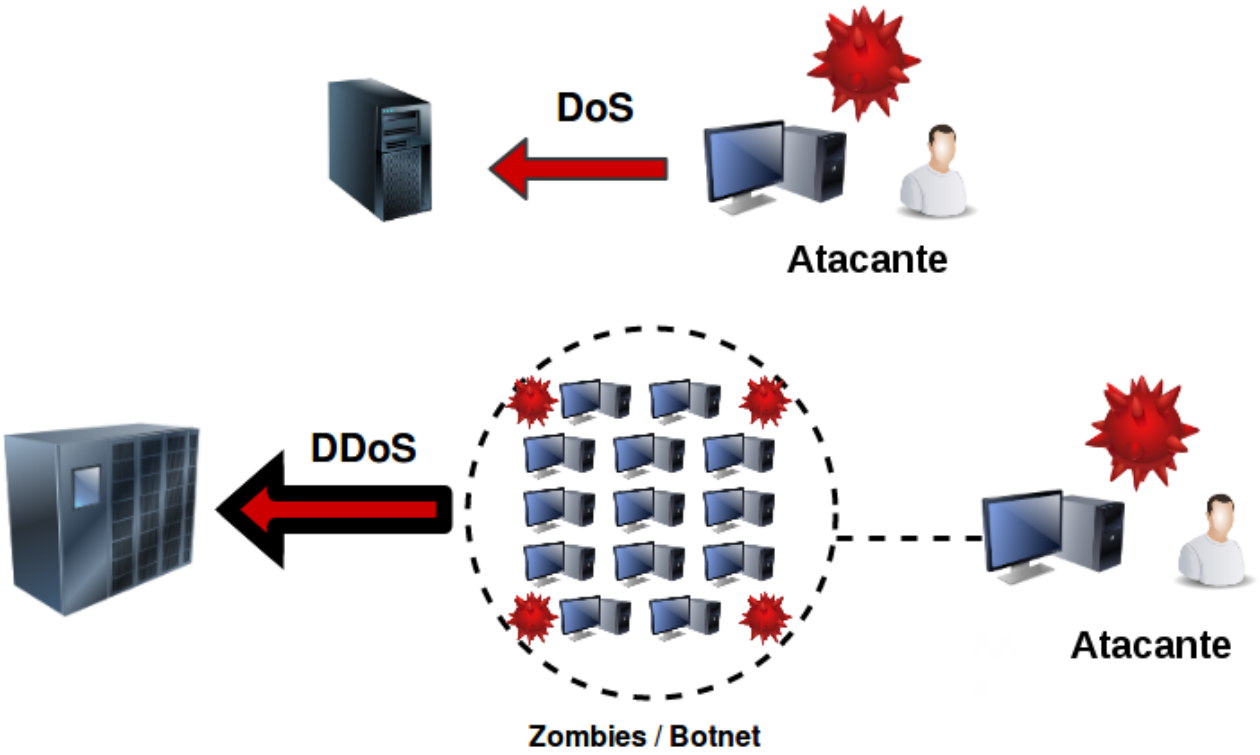

Figura 2.3: Comparação entre DoS e DDoS.

A utilização de *botnets* domina os mecanismos para realização de ataques às redes e aplicações. Existem duas razões para isso, a primeira é devido ao seu uso facilitar o sucesso em ataques de larga escala e a segunda é porque a utilização de máquinas *zombies* oferece, ao atacante, o anonimato do seu IP, devido ao ataque chegar no destino com o endereço IP das máquinas participantes da *botnet* [27].

Porém, além da utilização das *botnets*, o atacante pode esconder seu IP de origem por

meio de uma técnica conhecida de IP *Spoofing*. Esse mecanismo mascara o IP do atacante, fazendo com que, mesmo sem utilizar máquinas *zombies*, possa ainda assim permanecer no anonimato, realizando seus ataques sem ser descoberto devido a falsificação do seu IP. O *Syn-flooding*, ataque utilizado como proposta de detecção deste trabalho, pode utilizar a técnica de IP *Spoofing*.

2.3.1 Funcionamento do *Syn-flooding*

O protocolo TCP (*Transport Control Protocol*) faz parte da camada de transporte do modelo TCP/IP, é orientado a conexão e prover confiabilidade fim a fim numa conexão entre dois *hosts* remotos. Vários serviços de aplicações atuais o utilizam para o estabelecimento de conexão, como: SMTP, HTTP, TELNET, FTP, etc [29].

Conexões TCP são realizadas por meio de uma metodologia conhecida como *Three-Way Handshake*. Que funciona com o cliente enviando um pacote com a *flag* SYN, o servidor respondendo com um SYN-ACK e ficando em espera do pacote final do requisitante para o estabelecimento da conexão, o ACK [3] (ver Figura 2.4).

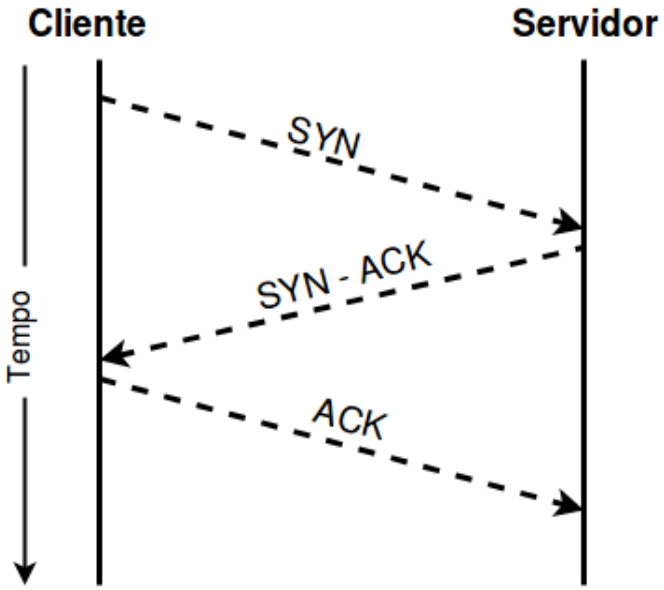

Figura 2.4: Three-Way Handshake.

No caso do *Syn-flooding*, o atacante não envia a última *flag* ACK, fazendo o servidor se manter em uma espera interminável, não completando o processo de estabelecimento de conexão. Uma conexão incompleta aberta consome os recursos do servidor. Várias dessas conexões incompletas abertas fará se esgotar os recursos do servidor, que é a consequência do *Syn-flooding* [3](ver Figura 2.5).

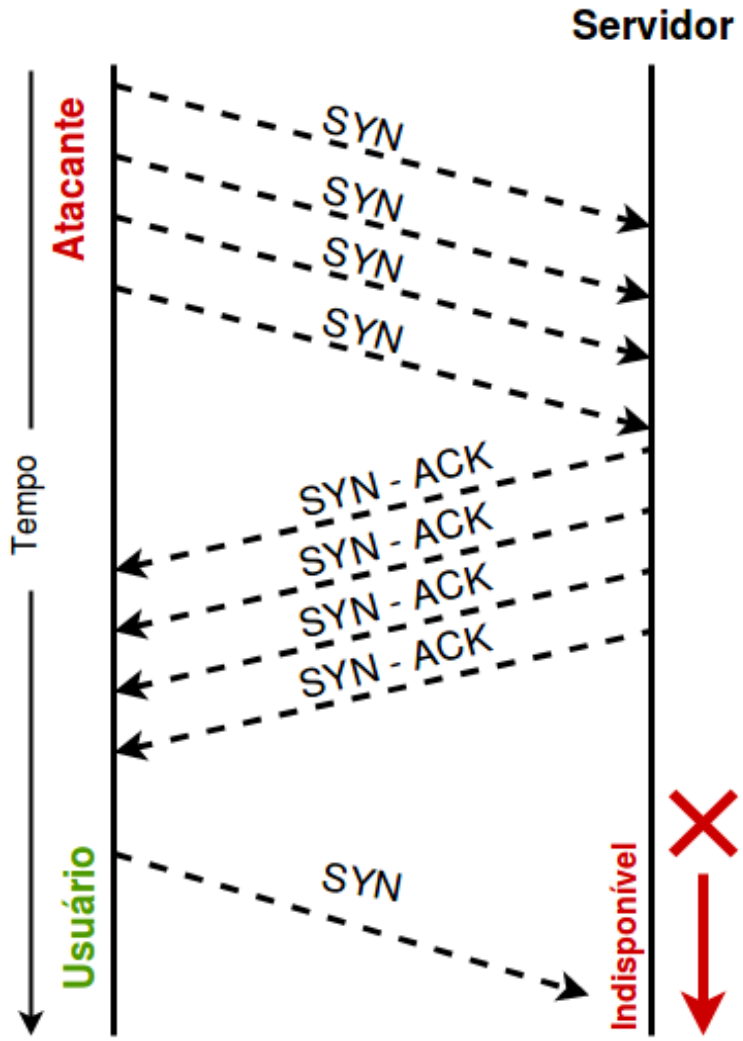

Figura 2.5: Exemplo de Syn-flooding.

Para dificultar a detecção da origem do ataque, alguns atacantes realizam o *Syn-flooding* com técnica de IP *spoofing*, a fim de mascarar o seu IP [2] e não serem detectados.

Atualmente, nos Sistemas Operacionais Windows e Linux, já existe proteção para o ataque *Syn-flooding*. A proteção do Windows se chama "TCP SYN attack protection"e, atualmente, já se encontra ativada nas configurações padrão [30]. No Linux, a proteção se chama "SYN Cookies", porém ainda não vem ativada nas configurações padrão do sistema [30]. Assim, os serviços de aplicação configurados nesses já podem evitar as consequências de um ataque *Syn-flooding*.

Mesmo que algum serviço de aplicação já possua proteção ao *Syn-flooding*, em uma Rede SDN este ataque ainda pode ter consequências desastrosas antes mesmo do tráfego malicioso alcançar o servidor. Isto se justifica devido ao motivo que para cada novo pacote que encapsula a flag SYN de uma solicitação de estabelecimento de conexão TCP, pode ser gerada uma nova

mensagem *packet-in* para o controlador [2]. Assim, um *flood* de SYN, também poderá ocasionar em um *flood* de *packet-in*. Devido a isto, este ataque ocasiona na saturação do próprio controlador SDN, podendo chegar a negar o serviço deste [2] [5] [31] [32] [33]. Assim, um ataque *Syn-flooding*, que em uma rede tradicional tem objetivo de apenas negar o serviço da aplicação, pode afetar a própria infraestrutura em SDN.

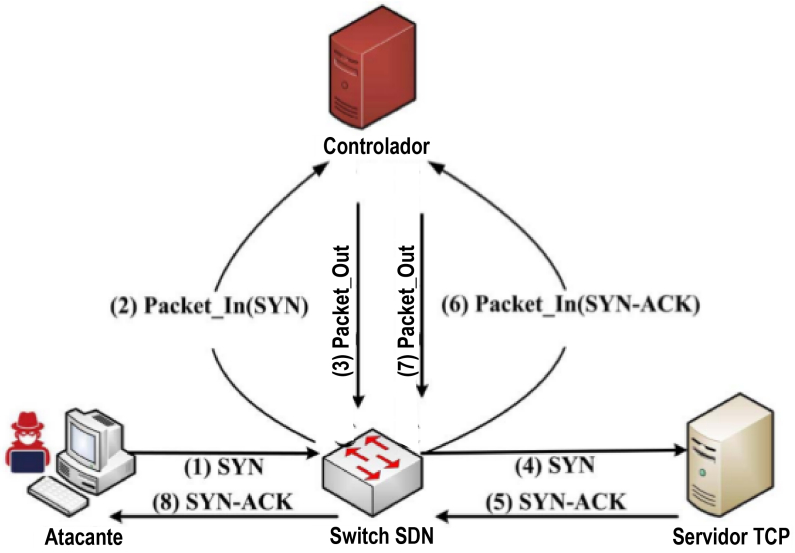

Figura 2.6: Funcionamento do *Syn-flooding* em SDN. Adaptado de Mohammadi *et al.* [2].

Na Figura 2.6, Mohammadi *et al.* [2] apresenta que, caso seja utilizado a técnica de IP *Spoofing*, além da *flag* SYN (1) gerar um *packet-in* para o controlador, também através da resposta do próprio servidor (SYN-ACK - 5), será gerado um novo *packet-in*, congestionando ainda mais o controlador SDN.

2.4 Trabalhos Relacionados

Nesta seção são apresentados os trabalhos de grande relevância no atual estado da arte, todos relacionados com a presente proposta deste artigo. Primeiramente é apresentada cada proposta selecionada do estado da arte e, ao final da seção, é demonstrada as semelhanças e diferenças com a proposta deste presente trabalho, realizando as devidas comparações.

O SLICOTS [2] é uma proposta de detecção e mitigação de *Syn-flooding* em uma rede SDN. Funciona com o monitoramento de todo o processo do TCP *Three-way Handshake* entre dois *hosts*. Durante este processo, é instalada regras temporárias enquanto a conexão não é estabelecida. Só após o estabelecimento das conexões, é instalada uma regra permanente entre cliente

e o servidor. Caso a conexão não chegue a se estabelecer, a solução a denomina de *half-open*. Depois de certa quantidade de *half-open*, é instalada uma regra para bloquear futuras tentativas de conexões do *host* suspeito (a Figura 2.7 explicita esse funcionamento). A tecnologia foi implementada em um controlador OpenDayLight, que utiliza a linguagem de programação Java.

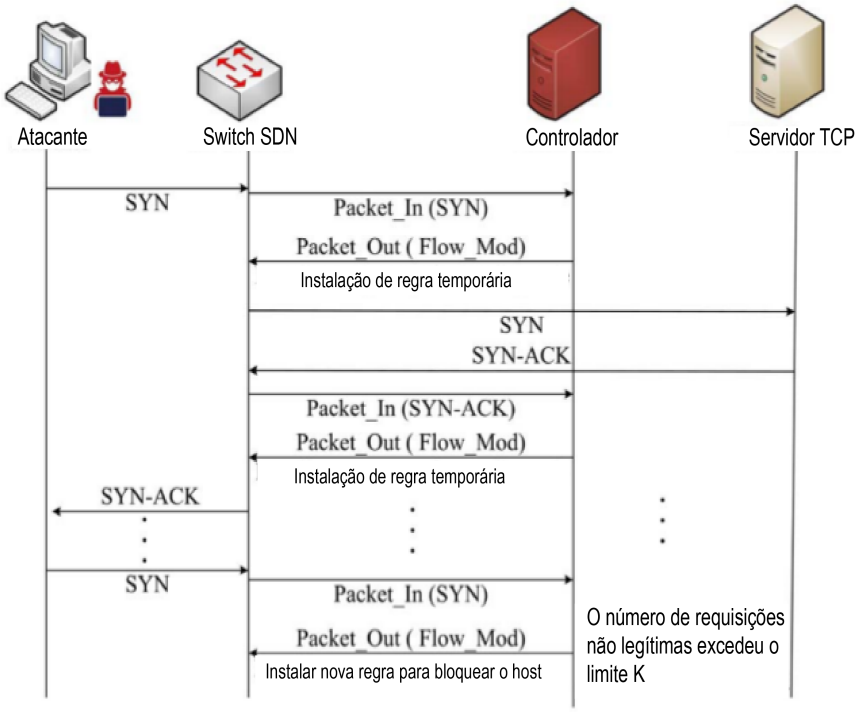

Figura 2.7: Funcionamento do SLICOTS. Adaptado de Mohammadi *et al.* [2].

O OPERETTA [3] também é uma proposta de detecção e mitigação do *Syn-flooding* em SDN. Funciona como um módulo instalado no controlador que faz este atuar como um *proxy* para validar as conexões entre cliente e servidor. Em uma conexão TCP, o cliente envia um SYN para o servidor WEB e este é quem responde com um SYN-ACK. Porém, nesta solução, é o controlador quem responde com um SYN-ACK. Se o cliente responder novamente com um ACK, o controlador percebe que se trata de uma conexão legítima e envia um pacote com a *flag* RST para o cliente, a fim de que ele reestabeça a tentativa de conexão, mas dessa vez liberando a instalação de uma nova regra entre o cliente e o servidor. A solução define um limiar de conexões, não estabelecidas, que cada *host* pode fazer. Ultrapassando o limiar definido, o controlador bloqueia o *host* por tê-lo como atacante (ver Figura 2.8). Foi implementado no controlador POX, que é desenvolvido em Python.

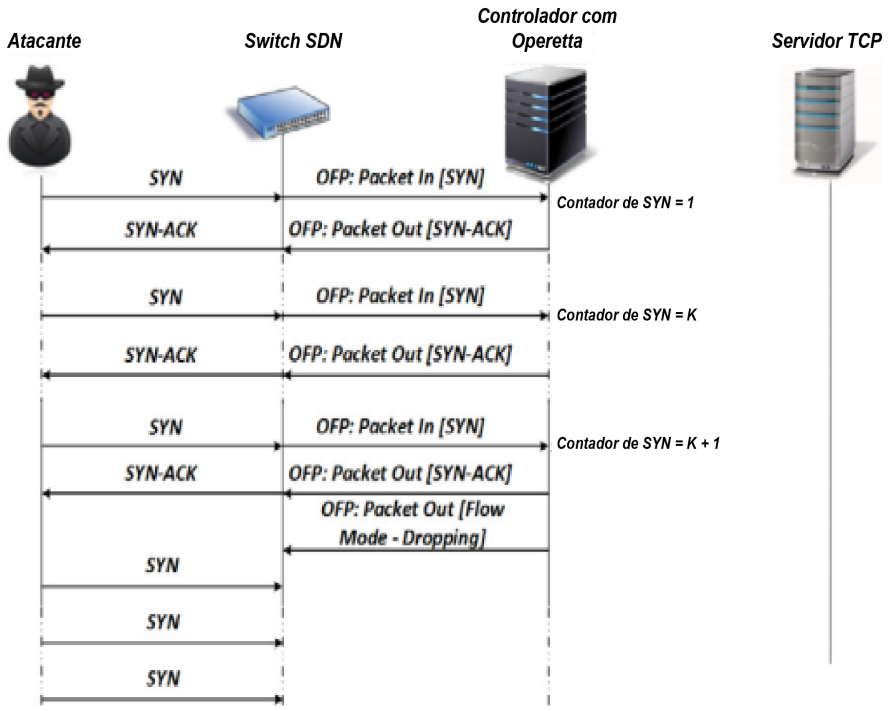

Figura 2.8: Funcionamento do OPERETTA. Adaptado de Fichera *et al.* [3].

O SPHINX [4] é um *Framework* para detecção de ataques em redes SDN. Ele detecta um ataque *Syn-flooding* por meio de 3 etapas: monitoramento das principais mensagens do OpenFlow enviadas para o controlador, mapeamento da rede de acordo com as mensagens do OpenFlow e, finalmente, detecção do ataque através do comportamento de fluxos diferentes do padrão estabelecido pelas políticas. A construção das políticas é dado através do monitoramento do tráfego legítimo na rede. Caso seja encontrado algum tráfego fora do padrão, por exemplo, se a quantidade das mensagens *packet-in* estiver passando de um limiar estabelecido pelo administrador da rede, é gerado um alerta de possível ataque (ver Figura 2.9). Os controladores utilizados para a validação desta solução foram o OpenDaylight e o Floodlight, ambos utilizam a linguagem de programação Java para o desenvolvimento das aplicações de rede.

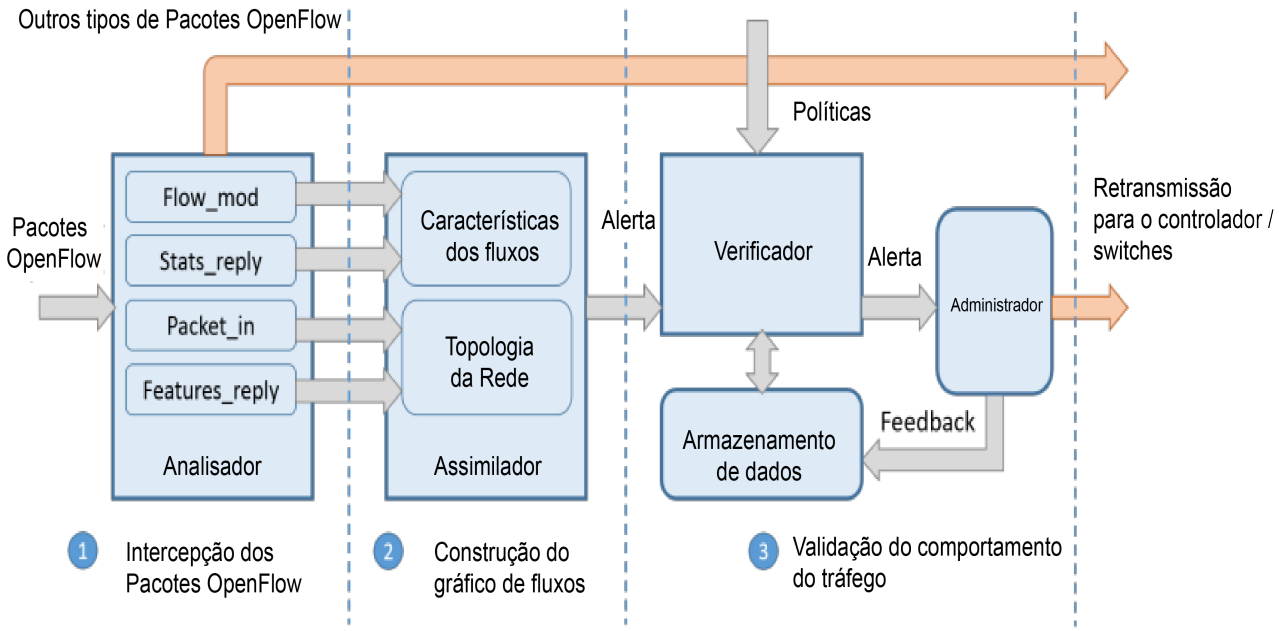

Figura 2.9: Funcionamento do SPHINX. Adaptado de Drawan *et al.* [4].

O AVANT-GUARD [5] faz com que cada dispositivo responsável pelo plano de dados (*switch* ou roteador), em uma rede SDN, atue como proxy para requisições SYN. Ele funciona enviando as mensagens *packet-in* para o controlador somente depois que o *switch* (ou roteador) SDN estabelece uma conexão com o cliente requisitante. Desta forma, a solução não permite que o ataque *Syn-flooding* alcance o controlador e o servidor alvo, devido a este ataque ter a característica de não completar o estabelecimento de conexão TCP, por não enviar o último ACK de confirmação (ver Figura 2.10). O controlador utilizado nesta solução foi o POX.

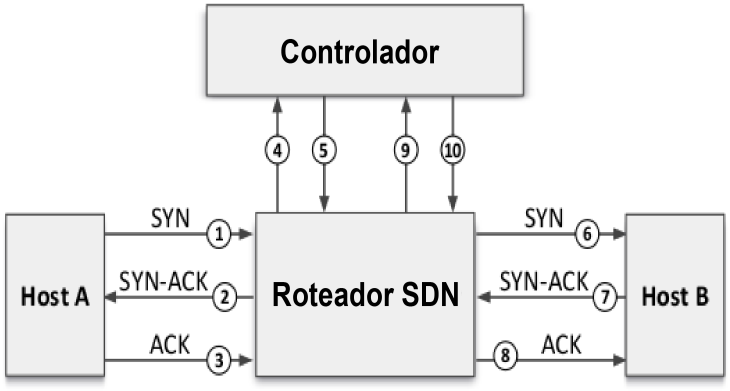

Figura 2.10: Funcionamento do AVANT-GUARD. Adaptado de Shin *et al.* [5].

O LineSwitch [34] é semelhante a solução do AVANT-GUARD, devido a cada *switch* tam-

bém atuar como proxy para requisições SYN. A diferença é que esta proposta utiliza uma *blacklist* com os *hosts* suspeitos e assim já bloqueia o tráfego advindos destes *hosts*, antes de precisar aguardar a verificação de que suas conexões não serão legítimas. Também foi utilizado o controlador POX nesta solução.

A *Selective Packet Inspection* [6] é uma proposta colaborativa de detecção de ataques *Syn-flooding*. Monitora continuamente os fluxos de SYN *packets* com endereços IP diferentes (para detectar IP *Spoofing*). Faz isso por meio de uma base que armazena o IP relacionado a cada porta do *switch* e, caso encontre um endereço IP diferente do esperado para determinada porta, considera o tráfego como vindo de um ataque e bloqueia o *host* malicioso (ver Figura 2.11). POX também foi o controlador utilizado para a validação.

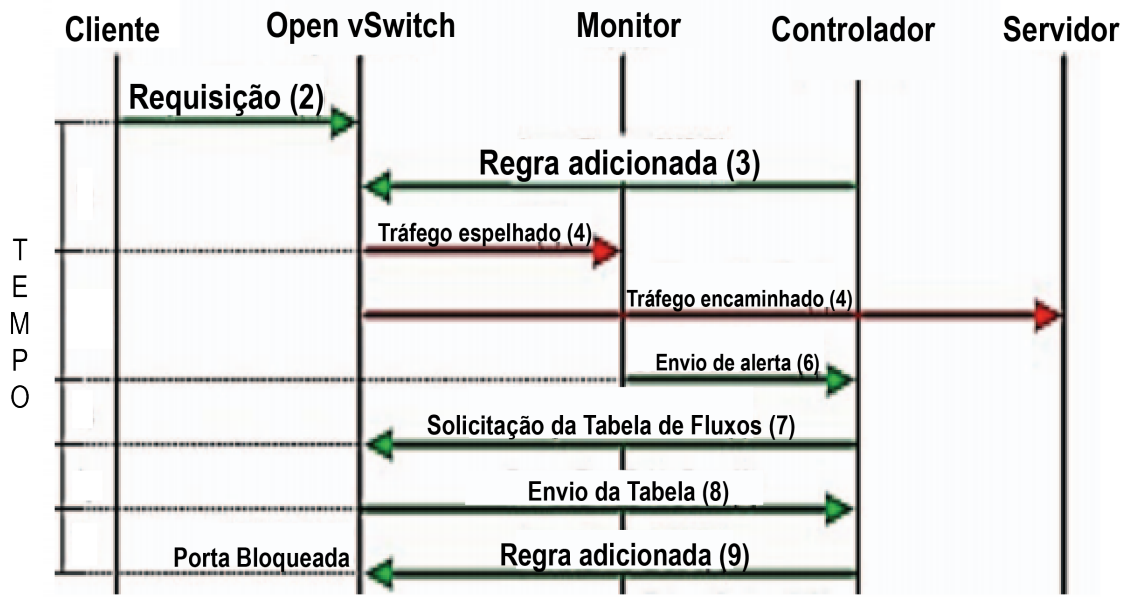

Figura 2.11: Funcionamento do Selective Packet Inspection. Adaptado de Chin *et al.* [6].

O Bloom Filter [7] é um sistema de detecção de um tipo de ataque DDoS chamado *link flooding*. Ele utiliza 2 módulos: Coletor e Detector; ambos podendo ser implementado no próprio controlador. O coletor verifica os fluxos considerados anormais (classificados assim quando os campos *packetCount*, *byteCount*, *durationSeconds* passarem de um limite pré-estabelecido) e armazena o IP de origem desses fluxos em uma estrutura de dados chamada Bloom Filter. O detector realiza a captura de pacotes da rede e compara o tráfego atual com os endereços IP existentes no Bloom Filter e, caso verifique uma correspondência, classifica o tráfego como um ataque, realizando a detecção (ver Figura 2.12). O controlador utilizado para atestar esta solução foi o Floodlight.

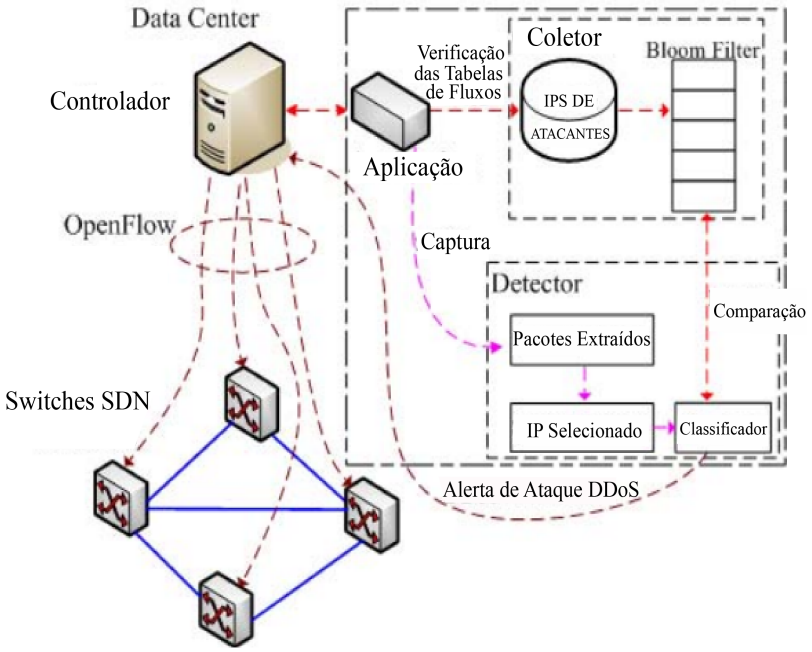

Figura 2.12: Funcionamento do Bloom Filter. Adaptado de Xiao *et al.* [7].

O FlowSec [35] estabelece um limite de pacotes a serem enviados ao controlador para este não ser vítima de um ataque DoS. Utiliza o *meter* do OpenFlow para limitar a quantidade de pacotes. Se a quantidade de pacotes chegar a 80% do máximo, o controlador envia uma mensagem ao *switch* para reduzir a quantidade de pacotes pela metade. A tecnologia utilizou o controlador FloodLight.

Uma grande contribuição é dada por Xu et al [11]. O trabalho tem o propósito detectar ataques DDoS volumétricos. Faz isso medindo a taxa de assimetria dos fluxos de uma subrede, ou seja: compara a quantidade de fluxos originados por determinada subrede com a quantidade de fluxos destinados a essa mesma subrede. Percebendo uma taxa assimétrica maior que um limiar estabelecido em determinado período de tempo, divide a subrede em mais duas subredes menores, verifica onde continua a assimetria, divide novamente até encontrar o *host* responsável (correspondente máscara /32), tanto da vítima como do atacante. O controlador utilizado foi o OpenDayLight.

Yan *et al*. [33] utilizou lógica Fuzzy para realizar a detecção de três tipos de ataques DDoS em uma rede SDN: *Forged Source IP TCP FLOOD*, *Forged Destination Port UDP FLOOD* e *Malformed Packets FLOOD*. O chamado *Forged Source IP TCP FLOOD* é um *Syn-flooding* que utiliza o método IP *Spoofing*. Para a detecção, utilizou o nível de verdade de 0 a 1 (lógica Fuzzy) sobre a existencia do ataque, onde quanto mais próximo de 1, maior a probabilidade da existência do ataque. Para o cálculo da pontuação, esta contribuição utilizou a taxa de pacotes,

o cálculo da entropia entre os endereços IP de origem e destino, e também o mesmo cálculo entre as portas TCP de destino. Para a prototipação do experimento, o Mininet foi o escolhido, junto com o controlador POX.

Tabela 2.3: Resumo das Soluções Apresentadas

Solução	**Controlador**	**Prototipação**	**Ataque**
SLICOTS [2]	OpenDayLight	Mininet	*Syn-flooding*
OPERETTA [3]	POX	Mininet	*Syn-flooding*
SPHINX [4]	OpenDayLight e Floodlight	Mininet	*Syn-flooding*
AVANT-GUARD [5]	POX	Cenário Real	*Syn-flooding*
LineSwitch [34]	POX	Mininet	*Syn-flooding*
Selective Packet Inspection [6]	POX	VMs em Xen-Server	*Syn-flooding*
Bloom Filter [7]	Floodlight	Mininet	*Link flooding*
FlowSec [35]	Floodlight	Mininet	DoS Volumétricos
Xu *et al.* [11]	OpenDayLight	Internet2 [36]	DoS Volumétricos
Yan *et al*. (*Fuzzy*) [33]	POX	Mininet	*Syn-flooding*

Conforme mostra a Tabela 2.3, das soluções apresentadas, as que tratam da detecção e mitigação do ataque *Syn-flooding* em SDN são: SLICOTS [2], OPERETTA [3], SPHINX [4], AVANT-GUARD [5], LineSwitch [34], Selective Packet Inspection [6] e Yan *et al.* [33].

O AVANT-GUARD [5], LineSwitch [34] e OPERETTA [3] são semelhantes por utilizarem *proxy* para requisições TCP, com a diferença de que o AVANT-GUARD necessita de alteração no protocolo OpenFlow. O grande problema da utilização *proxy* está em não garantir o princípio fundamental da Internet, que é permitir conexões fim-a-fim livre de intermediações. Com o uso do *proxy* quem recebe as conexões TCP é o próprio *proxy* e, apenas em seguida, repassa para os *hosts* de destinos, ocasionando em um certo *delay* no estabelecimento de conexões legítimas. O SLICOTS [2] é eficiente na mitigação do *Syn-flooding*, porém requer um alto custo de processamento por abrir o pacote para verificar as *flags* TCP. Por quebrar a flexibilidade do uso de endereço IP em qualquer porta dos *switches*, a solução de inspeção seletiva de pacotes [6] não seria adequada em alguns ambientes, como em um campus universitário que possui laboratórios de informática em que os professores e alunos realizam testes na rede.

Uma solução semelhante a proposta deste artigo é o SPHINX [4], por também estabelecer

o limiar da quantidade de fluxos, não necessitar a abertura do pacote para verificação do ataque, não atuar como *proxy* e não precisar de alteração do protocolo OpenFlow. Pelas características apresentadas, esta solução mostrou-se com uma boa performance e baixo *overhead*, conseguindo detectar o ataque na casa dos milissegundos. Ao contrário do FindFlows, o SPHINX necessita da construção de gráficos de fluxos para avaliar se houve ou não um ataque, detectando através de variações dos valores demonstrados nos gráficos. Esta solução não chega identificar o endereço IP do *host* do atacante e da vítima do ataque *Syn-flooding*.

A solução de Yan *et al.* [33] utilizou a lógica *Fuzzy* para a detecção do ataque. Segundo Ashraf e Latif [37] a detecção de ataques DDoS em redes SDN por meio da lógica *Fuzzy* tem o ponto positivo de que utiliza valores de aproximação de algo verdadeiro e não valores precisos, sendo eficaz quando se monitora as portas do protocolo TCP. Da mesma forma, os mesmos autores também apresentam o lado negativo de se utilizar *Fuzzy Detection*, que é o motivo do alto consumo de recursos para realizar a detecção.

As diferenças das soluções apresentadas até aqui e a proposta atual deste trabalho está em o FindFlows conseguir detectar o atacante e a vítima do *Syn-flooding* ao mesmo tempo que garante a conectividade fim-a-fim sem interceptação e sem precisar alterar o protocolo OpenFlow. Também não necessita abrir os cabeçalhos dos pacotes para analisar as *flags* TCP, reduzindo o custo computacional e utilizando um baixo consumo de recursos. Além disso, a proposta permite a contínua flexibilidade dos dispositivos da rede, sem necessidade de fazer com que cada *host* precise estar de forma permanente em uma porta do *switch*.

Capítulo 3

FINDFLOWS: UM DETECTOR DE ATAQUE SYN-FLOODING EM SDN

Neste capítulo é apresentado, de forma detalhada, a proposta do trabalho, a arquitetura utilizada, bem como o que se está coletando.

É típico de um ataque DoS *Syn-Flooding* abrir centenas ou milhares de requisições TCP com a flag SYN. Considerando que para cada nova requisição, é utilizada uma nova porta de origem, um controlador SDN pode também identificar cada nova requisição como um novo fluxo. Assim, um crescimento exorbitante, da quantidade de fluxos, em um curto intervalo de tempo, pode caracterizar em um ataque deste gênero.

O objetivo da proposta é detectar ataques DoS *Syn-flooding* em Redes SDN, identificando os *hosts* (por meio de seus endereços IP) que participam do ataque.

3.1 O Monitor FindFlows

Para a detecção do ataque, foi criado um monitor chamado FindFlows que atua em conjunto com o controlador. O controlador escolhido foi o Ryu, devido a este fornecer atualizações com frequência, já suportando até a última versão do OpenFlow (1.5) conforme apresentado na Tabela 2.2 do capítulo anterior, e por utilizar a linguagem de programação Python como código do controlador, que é uma linguagem de sintaxe relativamente fácil, com diversas funcionalidades e é bastante utilizada atualmente, sendo a terceira linguagem de programação mais utilizada no mercado [38], possuindo vários fóruns na *Internet*. O FindFlows também foi desenvolvido em Python, a escolha desta linguagem se justifica pelos pontos positivos apresentados sobre a linguagem e devido ao próprio controlador escolhido a utilizar para programação da rede.

O FindFlows identifica cada *host* da rede e analisa a variação da quantidade de fluxos de cada um em dois instantes diferentes, fazendo isso através do cálculo da derivada. A seguir é mostrado um fluxograma da proposta.

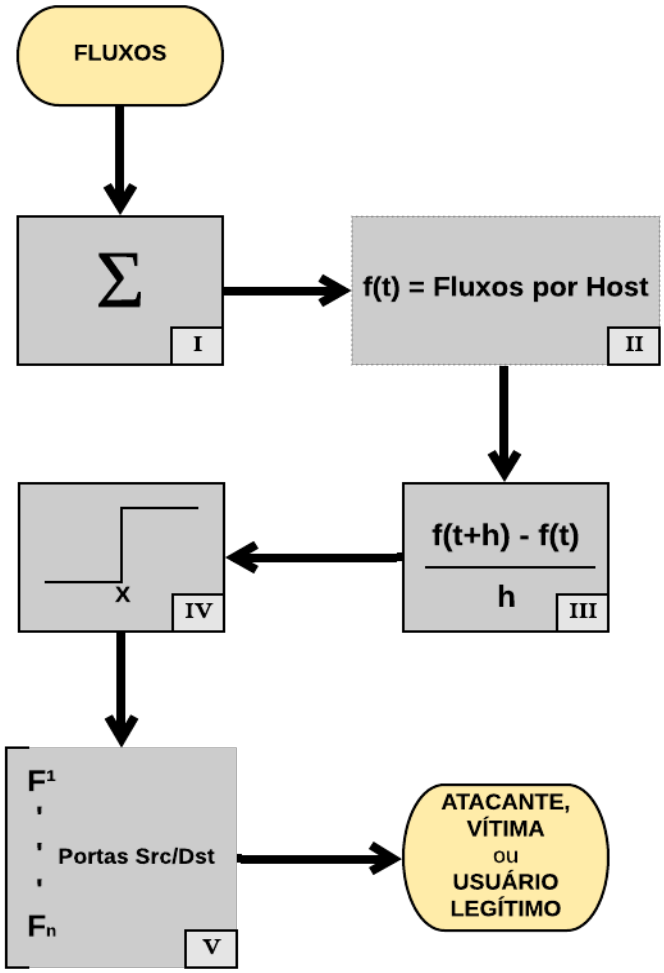

Figura 3.1: Fluxograma do FindFlows

Inicialmente é realizado o somatório dos fluxos, separando-os em seguida por *hosts* de origem, obtendo a quantidade de fluxos por *host*. Assim, por capturar esses valores em dois momentos distintos, é realizado o cálculo da derivada, onde ***t*** é o tempo da verificação da quantidade de fluxos no primeiro instante e ***h*** é o intervalo entre os dois valores. Caso o resultado seja maior que o valor de *threlshold* ***X***, cairá na verificação das portas TCP, categorizando o *host* em seguida. Uma explicação detalhada de cada passo é realizada a seguir:

I. **Somatório dos fluxos:** O OpenvSwitch permite a captura dos fluxos por meio do comando "ovs-ofctl -O OpenFlow13 dump-flows". Assim o primeiro procedimento que a

ferramenta FindFlows realiza é capturar todos os fluxos armazenados na memória de *buffer* do *switch* SDN e colocá-los em um arquivo provisório denominado "flows-SW-X", sendo 'X' a identificação do *switch* da rede, contabilizando assim a quantidade de fluxos da captura. Os arquivos criados durante o processo de detecção, são apenas para o armazenamento das capturas dos fluxos, sendo utilizados de forma provisória, para os cálculos necessários de serem realizados pelo FindFlows. O resultado de todo processamento realizado em cima dos arquivos vão sendo armazenados em uma tabela que é salva na memória. Vale salientar que esse tempo que os fluxos podem ficar na memória do *switch* é configurável pelo controlador da rede SDN. O controlador Ryu foi configurado com um *timeout* de 25 segundos para o armazenamento dos fluxos no *buffer* do *switch*, este tempo foi escolhido por ser suficiente para coletar os fluxos de tráfego legítimo e de ataque. Por padrão, o controlador Ryu vem sem tempo de armazenamento no *buffer*. A entrada desta etapa são os fluxos SDN e a saída é o somatório destes;

II. **Separação da quantidade de fluxos por *host*:** O FindFlows contabiliza a quantidade de fluxos por cada IP de origem, ou seja, por cada *host*. Faz isso selecionando o IP de origem de cada fluxo, do arquivo separado no início, e o comparando se ele já existe na tabela do monitor. Caso na tabela não exista nenhuma entrada para o IP de origem do fluxo em questão, é criada uma nova linha com o novo *host* e a quantidade de fluxos igual a 1. Caso já exista, é apenas incrementado o valor da quantidade de fluxos daquele IP selecionado. Quando a contabilização é finalizada, a tabela estará preenchida de acordo com a quantidade de *hosts* verificados na captura dos fluxos. Em um exemplo, se houver 5000 fluxos de apenas 2 endereços IP de origem, a tabela estará preenchida com apenas 2 linhas, informando a quantidade de fluxos correspondente a cada IP de origem, ou seja, a cada *host* da rede. A entrada deste processo é o somatório total dos fluxos e a saída é a quantidade de fluxos por *host*;

III. **Cálculo da derivada:** Os valores das capturas dos fluxos são realizados em dois momentos distintos para realizar o cálculo da derivada. Assim, ***t*** é o tempo da verificação da quantidade de fluxos no primeiro instante e ***h*** é o intervalo entre os dois valores. A coluna "VARIACAO"da tabela será preenchida com o valor da derivada, sendo esse valor crucial para a detecção do *Syn-flooding*. A entrada desta etapa é a quantidade de fluxos por *host* (em dois instantes distintos) e a saída é o resultado do cálculo da derivada;

IV. **Comparação com o valor de *threshold*:** Um valor de *threshold* **X** limitará a variação da quantidade de fluxos em dois instantes distintos. O limite escolhido foi de 1000 fluxos, devido a esse valor apenas ser alcançado nos testes de ataques realizados, não chegando

a ser um valor de variação alcançado quando se utilizou apenas tráfego legítimo, mesmo assim pode ser adaptado de acordo com o perfil de cada rede. Desta forma, se a variação da quantidade de fluxos for maior que o valor do *threshold*, o *host* que gerou tais fluxos será considerado como suspeito. Aqui, o valor de entrada será a derivada dos dois instantes de captura dos fluxos do *host* e a saída será 1 ou 0, caso atingiu o valor do *threshold* ou não, respectivamente;

V. **Análise das portas TCP de origem e destino**: Mesmo o *host* considerado suspeito, uma segunda verificação dos seus fluxos é realizada para saber se o *host* trata-se um atacante ou uma vítima. O FindFlows faz isso analisando se os fluxos deste *host* possuem uma única porta TCP de destino, em caso positivo este é classificado como um atacante e a vítima também é descoberta pela análise do endereço IP de destino dos fluxos do atacante. Nos casos em que a segunda verificação constatar que o *host* possui portas TCP de destinos diferentes, este é identificado como vítima, visto que uma vítima de um ataque *Syn-flooding* tenta responder as milhares de requisições de conexões advindas com portas de origem diferentes. Neste nível, a entrada será 1 ou 0 (verificação do *threshold*) e a saída será a categorização do *host* como: ATACANTE, VÍTIMA ou USUÁRIO LEGÍTIMO. Caso a entrada seja 0, o *host* já será considerado usuário legítimo, por não ultrapassar o valor de *threshold*, não passando pela verificação das portas.

O fluxograma da Figura 3.2 apresenta de forma lógica a ordem dos níveis de atividades do FindFlows, facilitando o entendimento.

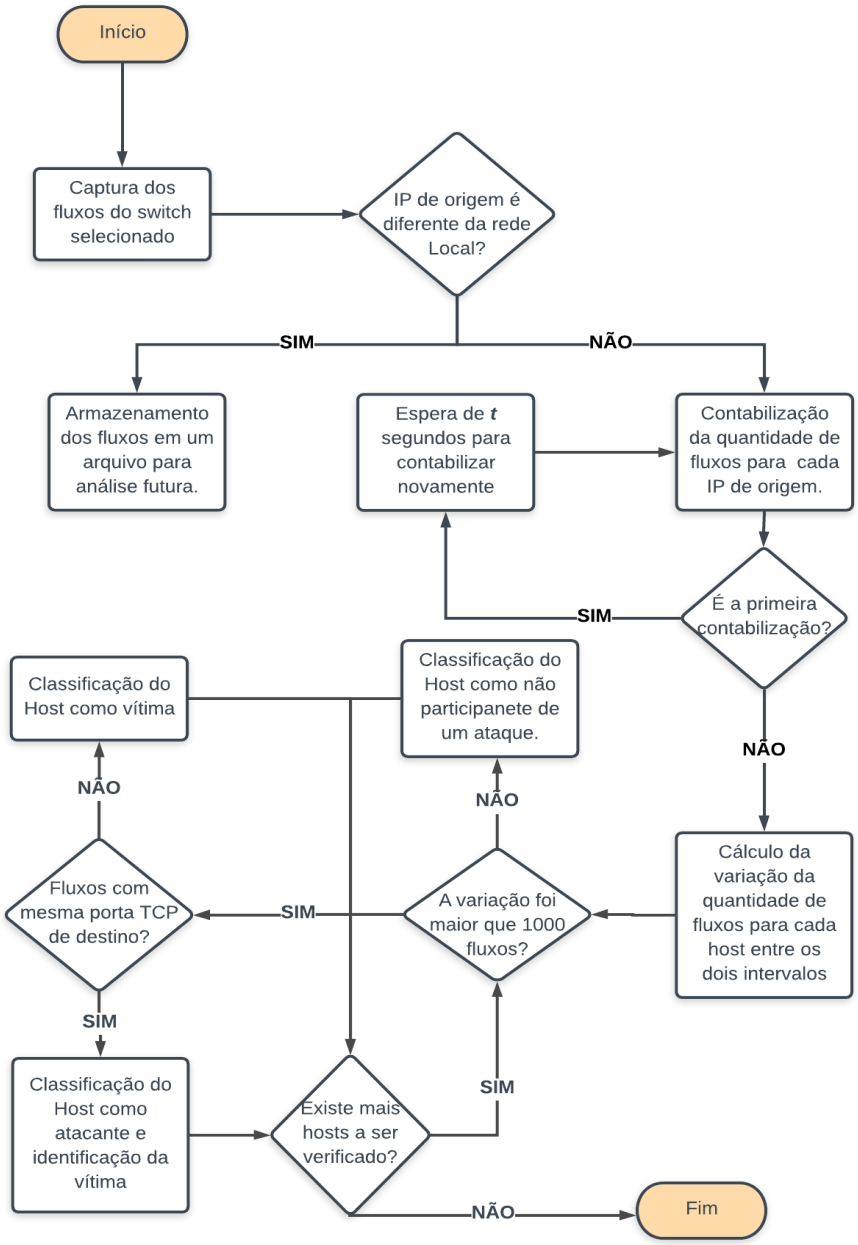

Figura 3.2: Fluxograma lógico do FindFlows

3.1.1 Estrutura de armazenamento dos dados

A ferramenta funciona utilizando uma tabela armazenada na própria memória do programa, preenchendo-a com os resultados de suas consultas. Se a tabela fosse utilizada em SGDB (Sistema Gerenciador de Banco de Dados), as consultas seriam mais lentas, aumentando o tempo de detecção (esse comparativo é apresentado na demonstração dos testes no próximo capítulo).

A tabela possui 5 colunas: "SRC-IP"(varchar), "Q-FLUXOS-A"(int), "Q-FLUXOS-D"(int),

"VARIACAO"(int), "SYN-FLOODING"(varchar).

Segue a descrição de cada coluna:

- **SRC-IP:** possui o IP de origem capturados por meio do fluxo;
- **Q-FLUXOS-A:** possui a quantidade de fluxos de um mesmo IP de origem, no instante da captura;
- **Q-FLUXOS-D:** possui a quantidade de fluxos de um mesmo IP de origem, (***t*** segundos depois da captura de Q-FLUXOS-A;
- **VARIACAO:** possui a variação da quantidade fluxos entre o valor de Q-FLUXOS-A e Q-FLUXOS-D;
- **SYN-FLOODING:** Irá possuir a palavra "NO"para não suspeito de algum ataque *Syn-flooding*, "ATTACKER"para o *host* identificado como suspeito de realizar ataque e "VICTIM"para o *host* que é identificado como o alvo do ataque.

3.1.2 Exemplo de Funcionamento

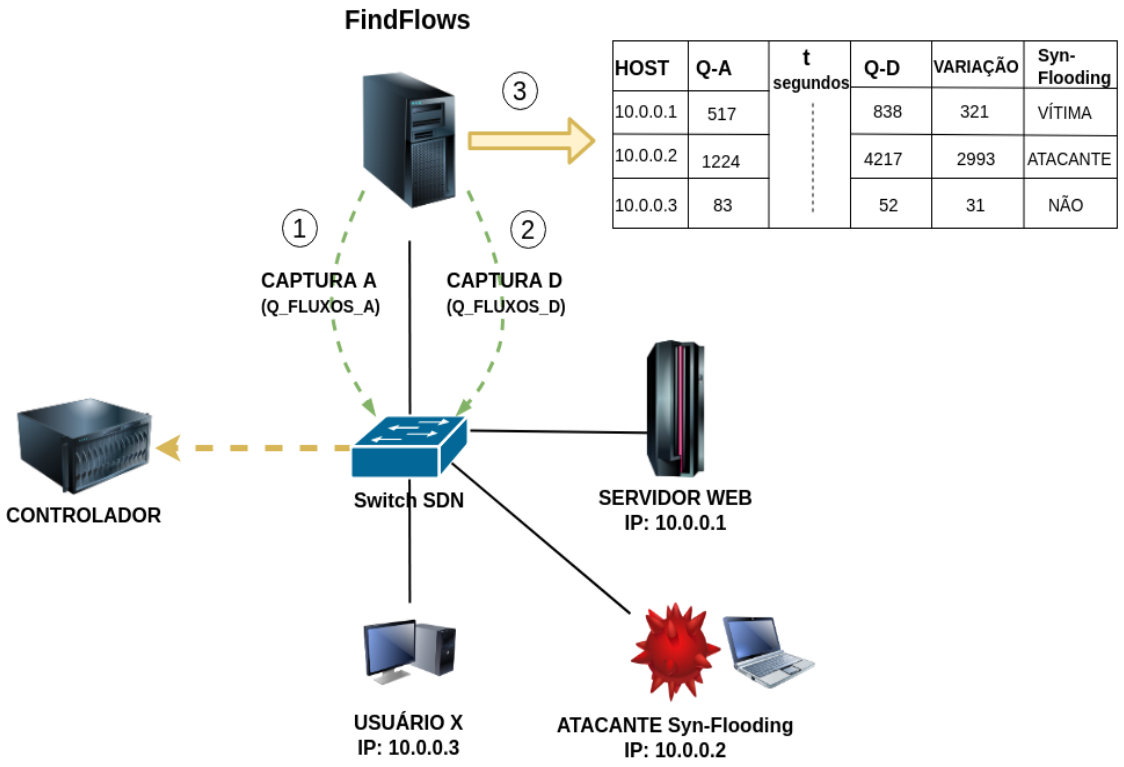

Figura 3.3: Exemplo de funcionamento do FindFlows

A Figura 3.3 exemplifica o funcionamento do FindFlows. O exemplo consta com um *switch* SDN, Controlador, Servidor WEB, Atacante (*Syn-flooding*), Usuário legítimo (Usuário X) e o Monitor FindFlows. Nesse caso, o destino do tráfego do usuário legítimo e do atacante é o Servidor WEB (IP: 10.0.0.1). Assim, o FindFlows realiza a detecção através de 3 processos, conforme a figura.

Em um primeiro momento (1º processo), o FindFlows realiza a 'Captura A', que é a captura de todos os fluxos do *switch* no instante 'A', separando a quantidade de fluxos por *host* nesse instante. No 2º processo, o FindFlows realiza novamente uma captura dos fluxos (Captura B), $\boldsymbol{t}$ segundos após a 'Captura A', também separando a quantidade de fluxos por *host* nesse instante. Após os processos 1 e 2, o FindFlows terá informação suficiente para realizar a detecção.

No 3º processo, o FindFlows irá calcular a variação da quantidade de fluxos entre a Captura A e B. Verificando que esta variação ultrapassou um limiar estabelecido pelo administrador (*threshold*), a ferramenta classificará os fluxos como suspeitos e irá verificar houve repetição nas portas TCP de destino, que é uma característica do *Syn-flooding*. Em caso positivo, o FindFlows classificará o *host* responsável pelos fluxos suspeitos como atacante e, através do IP de destino desses fluxos, também detectará a vítima, preenchendo toda a tabela e detectando o ataque.

O FindFlows realiza a Captura A e B entre um espaço de tempo que pode ser determinado pelo administador da rede de acordo com o perfil de tráfego da rede. O tempo de espera padrão, escolhido entre as duas capturas, foi 1 segundo. Para a escolha desse tempo, foram realizados testes com o $\boldsymbol{t}$ igual a 5; 2; 1; 0,5; 0,25; 0,1 e 0,05 segundos. O tempo de 1 segundo foi o tempo mínimo e suficiente para coletar uma alteração na rede que pudesse caracterizar um possível ataque. A Figura 3.4 exemplifica as capturas separadas por um intervalo de tempo, simbolizado pelo $\Delta\boldsymbol{t}$.

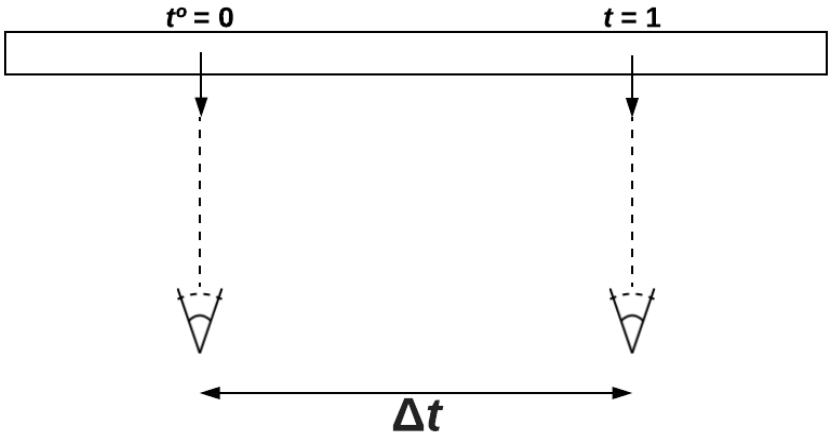

Figura 3.4: Espaço de tempo entre as capturas

Capítulo 4

RESULTADOS EXPERIMENTAIS

Este capítulo descreve as tecnologias e cenários utilizados para as simulações do ataque e preparação do ambiente para proporcionar um teste mais próximo a um cenário real.

4.1 Tecnologias utilizadas

Utilizou-se do emulador Mininet para a emulação de uma rede SDN real. A escolha do Mininet foi considerada como mais viável devido a possuir as características cabíveis e aplicáveis ao cenário proposto: flexibilidade, interatividade e escalabilidade [22]. Este *software* ainda traz uma realidade mais próxima do real, por utilizar o próprio *kernel* para emular *switches* e *hosts* da rede, não sendo uma simulação, mas uma emulação de um cenário real [22].

Para criação do cenário, utilizou-se a virtualização de um servidor PowerEdge T430 do fabricante Dell. Foram utilizadas 3 máquinas virtuais: uma para o controlador Ryu, outra para o emulador Mininet e outra para o monitor FindFlows. Cada máquina virtual possui dois Cores virtualizados do processador Intel Xeon CPU E5-2620 v3 de 2.40 GHz, 3 GB de memória RAM e 10 GB de HD.

Dentro do Mininet, foram utilizadas as seguintes tecnologias:

- **Apache2:** Servidor Web escolhido para ser utilizado no cenário. Foi realizada a escolha do Apache devido a ser o serviço responsável pelo maior número de hospedagem de páginas WEB ativas no mundo, estando com uma porcentagem de 29,95%, seguida de 21,96% do NGINX, segundo uma pesquisa realizada em outubro de 2018 pela Netcraft[1];

[1] https://news.netcraft.com/archives/2018/10/29/october-2018-web-server-survey.html Acessado em: 06/12/2018

- **Hping3:** Essa ferramenta permite fazer um *flood* de conexões TCP com a flag SYN, abrindo milhares de conexões em poucos segundos. Foi utilizada a Versão 3.0.0 para realizar o ataque *Syn-Flooding*. Também possui a opção de realizar conexões com o mascaramento do IP de origem, realizando *IP spoofing*. Por meio desta ferramenta, foi simulado o atacante;

- **Siege:** Ao contrário do *hping3*, essa ferramenta foi utilizada para simular o tráfego de clientes legítimos. Por meio desta ferramenta é possível simular requisições HTTP partindo de uma quantidade de clientes pré-determinada pelo usuário. A versão utilizada foi a 3.0.5.

Para relembrar os passos utilizado pelo FindFlows para realizar a detecção, a Figura 4.1 apresenta um pequeno fluxograma que resume os passos necessários.

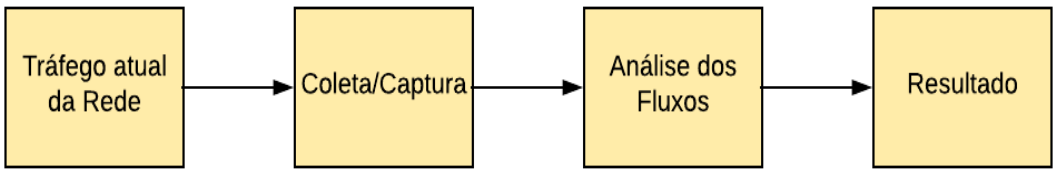

Figura 4.1: Resumo dos passos do FindFlows.

4.2 Cenário Principal

Conforme a Figura 4.2, a rede foi composta por um Servidor Web Apache2, um atacante DoS *Syn-flooding*, quatro *hosts* espalhados por 4 *switches* SDN, o controlador Ryu e o monitor FindFlows. Os 4 *hosts* são responsáveis por emular um total de 50 usuários legítimos da ferramenta Siege (Tabela 4.1).

Tabela 4.1: Quantitativo de usuários legítimos emulados por cada *host* - Cenário Principal

HOST:	**Usuários Legítimos:**
H3	5 usuários
H4	10 usuários
H5	15 usuários
H6	20 usuários
TOTAL:	**50 usuários**

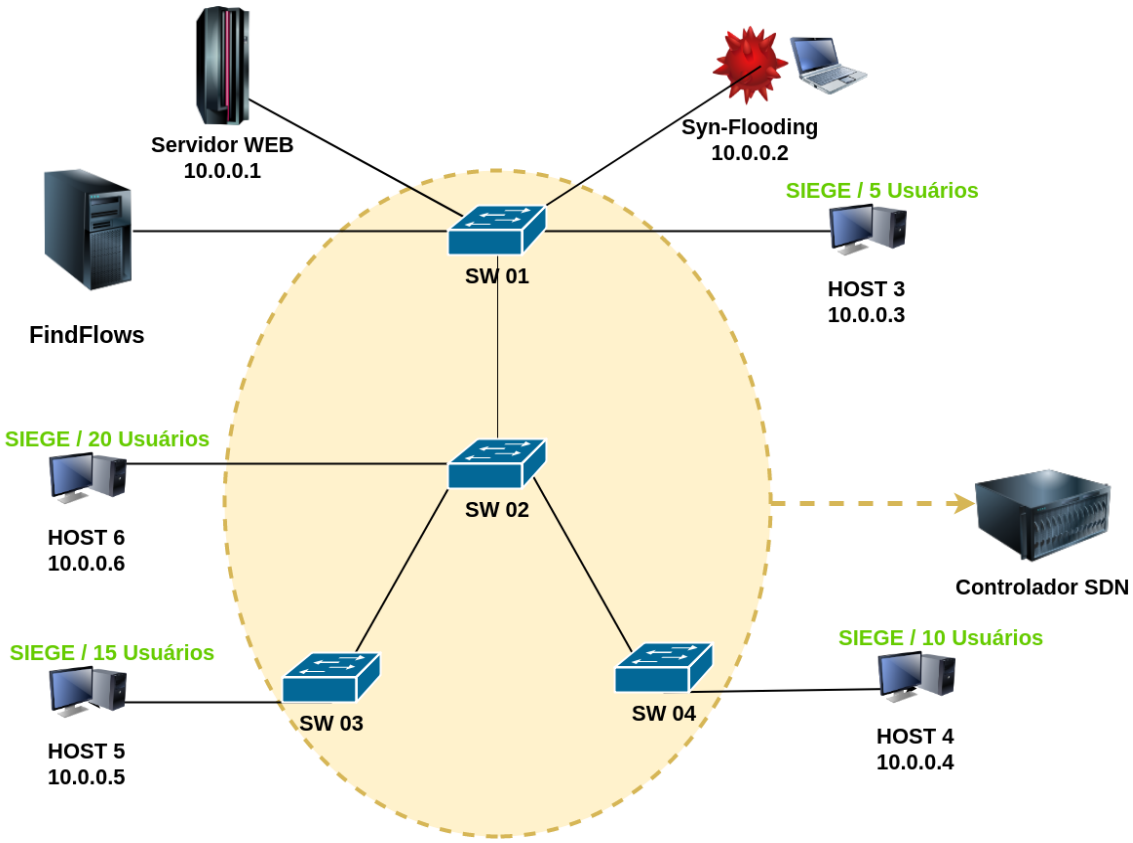

Figura 4.2: Cenário Principal.

4.2.1 Cenário Secundário

Também foi criado um cenário secundário, diminuindo para um *switch* e um *host* (ver Figura 4.2). A justificativa deste cenário, com uma menor quantidade de *switches* e *hosts*, se dá devido a necessidade de executar testes de maneira mais rápida, conforme será explicado na Seção "Testes Realizados".

Conforme a Tabela 4.2, no cenário secundário o *host* 3 teve a quantidade de usuários legítimos simulados aumentada para 50 usuários. Assim, a quantidade de fluxos legítimos concomitante com os fluxos do ataque foi a mesma do Cenário Principal, diferenciando porque os fluxos são gerados através de apenas um *host* e não de forma distribuída.

Tabela 4.2: Quantitativo de usuários legítimos emulados pelo *host* 3 - Cenário Secundário

HOST:	**Usuários Legítimos:**
H3	50 usuários

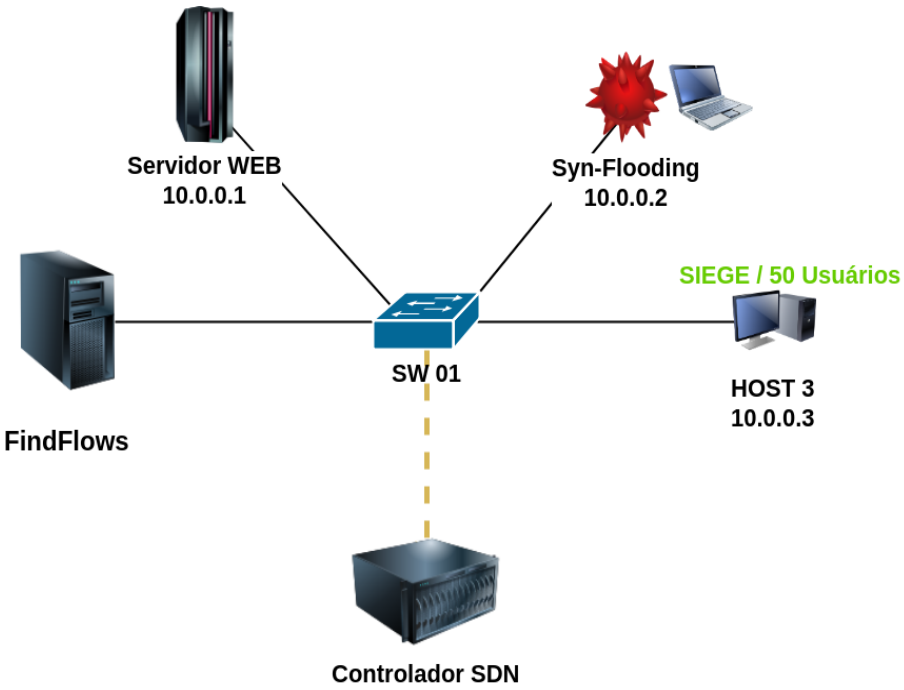

Figura 4.3: Cenário Secundário.

4.3 Testes Realizados

Para a validação da tecnologia, foram implementados 2 grupos de testes, denominados como Grupo A e Grupo B. O Grupo A teve o Cenário Principal (Figura 4.1) como ambiente de testes e, o Grupo B, o Cenário Secundário (Figura 4.2). Nos dois grupos foram realizados 3 variações de testes com todo o tráfego dos *hosts* possuindo, como destino, o Servidor WEB, com IP 10.0.0.1. A descrição dos testes de cada grupo se encontra na próxima seção.

4.3.1 Testes do Grupo A

No Grupo A foi utilizado o Cenário Principal como ambiente de testes. Foram realizados 3 variações de testes nesse cenário, alterando a duração de tráfego legítimo e ataque. Nas 3 variações, inicia-se com o tráfego legítimo e este continua existindo concomitantemente com o tráfego do ataque, que é gerado logo em seguida.

- **1º Teste:** 10 segundos apenas com tráfego legítimo (Siege) + 10 segundos de tráfego legítimo e Ataque *Syn-flooding* ao mesmo tempo;
- **2º Teste:** 25 segundos apenas com tráfego legítimo (Siege) + 25 segundos de tráfego legítimo e Ataque *Syn-flooding* ao mesmo tempo;

- **3º Teste:** 50 segundos apenas com tráfego legítimo (Siege) + 50 segundos de tráfego legítimo e Ataque *Syn-flooding* ao mesmo tempo.

Como cada fluxo passa apenas 25 segundos na tabela de fluxos de cada *switch* (sendo esse tempo configurável no controlador), os testes acima englobam a grande maioria das situações possíveis: antes de zerar os fluxos (1º teste), depois de zerar os fluxos (2º teste) e um múltiplo do 2º teste (3º teste), tendo a intenção de representar a maioria das situações de tráfego legítimo e ataque.

4.3.2 Testes do Grupo B

No Grupo B foi utilizado o Cenário Secundário como ambiente de teste. Foram realizados 3 variações de testes nesse cenário, alterando a duração de tráfego legítimo e ataque. Os testes deste grupo foram realizados com o objetivo de verificar se o FindFlows consegue detectar o ataque *Syn-flooding* mesmo em pouco tempo de ataque. Também, nas 3 variações, inicia-se com o tráfego legítimo e este continua existindo concomitantemente com o tráfego do ataque, que é gerado posteriormente.

- **1º Teste:** 1 segundo apenas com tráfego legítimo (Siege) + 1 segundo de tráfego legítimo e Ataque *Syn-flooding* ao mesmo tempo;
- **2º Teste:** 2 segundos apenas com tráfego legítimo (Siege) + 2 segundos de tráfego legítimo e Ataque *Syn-flooding* ao mesmo tempo;
- **3º Teste:** 5 segundos apenas com tráfego legítimo (Siege) + 5 segundos de tráfego legítimo e Ataque *Syn-flooding* ao mesmo tempo.

Em todos os testes, a ferramenta FindFlows foi executada logo em seguida do tempo do ataque, com o objetivo de detecção do *Syn-flooding*. Cada teste foi realizado 5 vezes, somando 15 amostras de resultados por cada grupo de teste, totalizando 30 amostras. A próxima seção (Seção 4.4) descreve detalhadamente os resultados destes testes.

4.4 Resultados

A ferramenta da proposta, FindFlows, preenche uma tabela para exibição dos resultados. Os dados das tabelas seguintes correspondem as amostras de cada teste realizado.

4.4.1 Resultados dos Testes do Grupo A

Conforme apresentado na Seção 4.3, os testes do Grupo A foram realizados com o objetivo de englobar o maior número de situações possíveis de um ataque *Syn-flooding*. Esses testes foram realizados no Cenário Principal (Figura 4.1).

Tabela 4.3: Amostra do 1º teste do Grupo A - *switch* 1.

SRC-IP	Q-FLUXOS-A	Q-FLUXOS-D	VARIACAO	SYN-FLOODING
10.0.0.3	102	48	54	NO
10.0.0.1	965	543	422	VICTIM
10.0.0.4	200	109	91	NO
10.0.0.5	260	135	125	NO
10.0.0.6	404	219	185	NO
10.0.0.2	64	2937	2873	ATTACKER

Na Tabela 4.3, a coluna "Q-FLUXOS-A"apresenta a quantidade de fluxos no instante 0. Já a coluna Q-FLUXOS-D contém a quantidade de fluxos 1 segundo depois. Em "VARIACAO"encontra-se a variação dos valores em "Q-FLUXOS-A"e "Q-FLUXOS-D", este valor é o que é utilizado na 1ª verificação para a detecção do *Syn-flooding*. A tabela mostra que foi detectado o *host* atacante.

Abaixo segue o resultado de uma das amostras do segundo teste:

Tabela 4.4: Amostra do 2º teste do Grupo A - *switch* 1.

SRC-IP	Q-FLUXOS-A	Q-FLUXOS-D	VARIACAO	SYN-FLOODING
10.0.0.4	3	0	3	NO
10.0.0.3	2	0	2	NO
10.0.0.1	19	34	15	VICTIM
10.0.0.6	7	0	7	NO
10.0.0.5	8	1	7	NO
10.0.0.2	1956	3956	2000	ATTACKER

Na Tabela 4.4, é perceptível que, aumentando o tempo de ataque antes da execução do FindFlows, a quantidade de fluxos processados de tráfego legítimo diminui já em "Q-FLUXOS-A", isso pode se explicar devido ao ataque estar em execução por mais tempo. Também, neste caso, foi detectado o *host* atacante.

Tabela 4.5: Amostra do 3º teste do Grupo A - *switch* 1.

SRC-IP	Q-FLUXOS-A	Q-FLUXOS-D	VARIACAO	SYN-FLOODING
10.0.0.2	1995	3941	1946	ATTACKER
10.0.0.1	0	50	50	VICTIM

Atacando por mais tempo (3º teste, 50 segundos), foi verificado que os fluxos processados foram apenas do atacante e da vítima, devido ao ataque atingir o seu objetivo de negar o serviço para as conexões legítimas dos outros *hosts* (ver Tabela 4.5).

Durante os testes, também foi executada a ferramanta FindFlows no *switch* 2, a fim de verificar se realmente não seria detectado nenhum *host*, como atacante, incorretamente. As tabelas seguintes (4.6; 4.7 e 4.8) apresentam os resultados de cada teste.

Tabela 4.6: Amostra do 1º teste do Grupo A - *switch* 2.

SRC-IP	Q-FLUXOS-A	Q-FLUXOS-D	VARIACAO	SYN-FLOODING
10.0.0.5	235	16	219	NO
10.0.0.4	140	12	128	NO
10.0.0.1	652	5	647	NO
10.0.0.6	322	22	300	NO

Tabela 4.7: Amostra do 2º teste do Grupo A - *switch* 2.

SRC-IP	Q-FLUXOS-A	Q-FLUXOS-D	VARIACAO	SYN-FLOODING
10.0.0.5	17	16	1	NO
10.0.0.1	13	11	2	NO
10.0.0.4	17	16	1	NO
10.0.0.6	24	23	1	NO

Tabela 4.8: Amostra do 3º teste do Grupo A - *switch* 2.

SRC-IP	Q-FLUXOS-A	Q-FLUXOS-D	VARIACAO	SYN-FLOODING
10.0.0.6	23	20	3	NO
10.0.0.5	16	15	1	NO
10.0.0.4	11	10	1	NO
10.0.0.1	5	0	5	NO

Conforme esperado, os *hosts* que não estavam envolvidos no *Syn-flooding*, sendo atacante ou vítima, não foram detectados como tal.

Como cada teste foi realizado por 5 vezes, 5 amostras para cada teste foram geradas do valor da variação de quantidade de fluxos. Assim, foi calculado a média aritmética dessas amostras, para cada *host*, em cada um dos testes. Os dados foram armazenados em um gráfico histograma, para melhor visualização e comparação dos resultados obtidos (ver Figura 4.4).

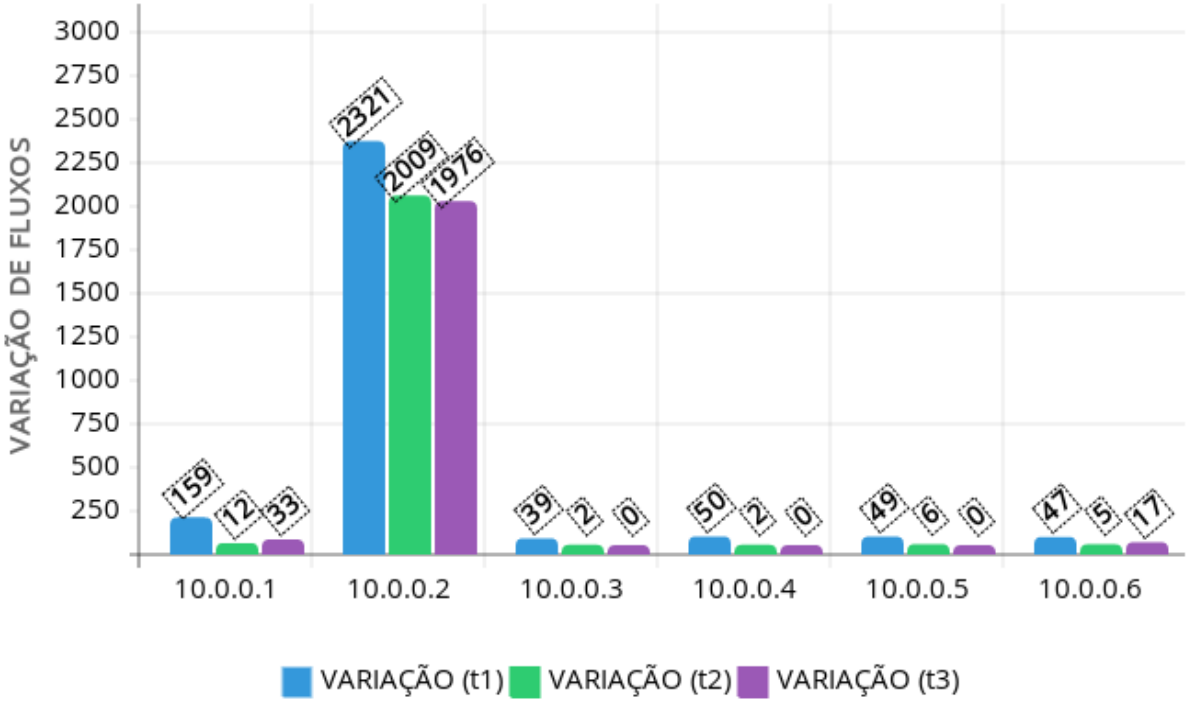

Figura 4.4: Comparação do valor da variação da quantidade de fluxos para cada *host*

Em azul, encontra-se a média dos valores da variação da quantidade de fluxos para o 1º teste. Já na cor verde e roxa, para o 2º e 3º teste, respectivamente. Foi observado que os maiores valores dessa variação vem do IP do atacante (10.0.0.2), acompanhado em seguida do IP da vítima.

Inicialmente, as informações utilizadas para a detecção do ataque eram armazenadas em um SGBD (Sistema Gerenciador de Banco de Dados) MySQL. Porém, as consultas e alterações nas tabelas do banco se mostravam lentas, atrasando também a própria detecção do ataque. Após alterar a tabela para que fosse armazenada em memória, uma melhora considerável foi verificada na detecção do ataque, conforme tabela comparativa abaixo.

Como mostrado na Tabela 4.8, o tempo de detecção aumentou de acordo com o aumento do tempo de execução do ataque.

Tabela 4.9: Comparação do tempo de detecção da tabela em um SGBD e em memória

Testes	SGBD - Tempo (min)	Memória - Tempo (min)
1º Teste	1:48	0:02
2º Teste	2:16	0:03
3º Teste	4:10	0:04

Com mais tempo de ataque, mais fluxos foram gerados, aumentando o tempo que o FindFlows levou para processar a quantidade de fluxos e realizar as operações devidas. Esse aumento do tempo ficou bem perceptível quando a tabela estava sendo armazenada em um SGBD MySQL. Porém, quando a tabela utilizada passou a ser em memória, o tempo de detecção caiu para casa dos segundos e o crescimento, de acordo com o tempo de ataque, se mostrou menos ascendente.

4.4.2 Resultados dos Testes do Grupo B

Conforme apresentado na Seção 4.3, os testes do Grupo B foram realizados com o objetivo de verificar a eficiência do FindFlows em detectar o ataque *Syn-flooding* em pouco tempo de ataque. Esses testes foram realizados no Cenário Secundário (Figura 4.2).

Tabela 4.10: Amostra do 1º teste do Grupo B

SRC-IP	Q-FLUXOS-A	Q-FLUXOS-D	VARIACAO	SYN-FLOODING
10.0.0.3	118	120	2	NO
10.0.0.1	112	144	32	VICTIM
10.0.0.2	1765	3727	1962	ATTACKER

O 1º teste do Grupo B é o de menor tempo de ataque, apenas 1 segundo. A Tabela 4.10 apresenta que houve detecção de ataque mesmo em pouco tempo de *Syn-flooding*. Porém, nesse teste, das 5 amostras, apenas 2 conseguiram detectar o ataque. As outras 3 amostras não exibiram a tabela com os *hosts*, não detectando o ataque. A explicação de não conseguir detectar o ataque nas demais amostras pode ser dada devido ao tempo ser extramente curto para os fluxos serem gerados depois da resposta do Controlador ao *Switch* SDN do Mininet.

Tabela 4.11: Amostra do 2º teste do Grupo B

SRC-IP	Q-FLUXOS-A	Q-FLUXOS-D	VARIACAO	SYN-FLOODING
10.0.0.3	266	266	0	NO
10.0.0.1	283	339	56	VICTIM
10.0.0.2	1446	3386	1940	ATTACKER

O 2º teste é o de 2 segundos de ataque. Neste teste em todas as 5 amostras foi detectado o ataque, não havendo o problema de não ter coleta para realizar a análise dos fluxos, problema que aconteceu no 1º teste. A Tabela 4.11 apresenta o resultado. Como visto, o *host* com IP 10.0.0.3 é apresentando como não atacante, isso mostra que, mesmo sendo o *host* que simula uma quantidade de 50 usuários legítimos, o FindFlows não cai em falso positivo por não verificar uma variação de fluxos que apresente uma anormalidade.

Tabela 4.12: Amostra do 3º teste do Grupo B

SRC-IP	Q-FLUXOS-A	Q-FLUXOS-D	VARIACAO	SYN-FLOODING
10.0.0.3	540	540	0	NO
10.0.0.1	539	567	28	VICTIM
10.0.0.2	916	2884	1968	ATTACKER

O 3º e último teste do Grupo B foi feito com 5 segundos de ataque *Syn-flooding*. Quanto maior maior é o tempo de ataque, maior será o tamanho da coleta dos fluxos e, assim, a detecção é realizada mais facilmente. A Tabela 4.12 apresenta o resultado de uma das amostras do 3º teste. Nesse teste, o ataque também foi detectado em todas as 5 amostras.

Tabela 4.13: Média do tempo de detecção dos testes do Grupo B

Testes	Tempo
1º Teste	1,3 segundos
2º Teste	1,5 segundos
3º Teste	1,8 segundos

A Tabela 4.13 apresenta o tempo médio de detecção dos testes do Grupo do B. Com menos tráfego de coleta, o tempo de detecção foi menor do que os testes no grupo A.

Capítulo 5

CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

De acordo com o que foi informado na seção dos trabalhos relacionados (seção 2.4), as diferenças das soluções do atual estado da arte e a proposta deste trabalho está em o FindFlows conseguir detectar o atacante e a vítima do *Syn-flooding* ao mesmo tempo que garante a conectividade fim-a-fim sem a interceptação que ocorreria no uso de *proxy* e sem necessitar alterar o protocolo OpenFlow. Devido a isso, não necessita abrir os cabeçalhos dos pacotes para analisar as *flags* TCP, reduzindo a utilização dos recursos computacionais. Por não precisar associar, de forma permanente, as portas de um *switch* aos *hosts* conectados, a proposta ainda permite a contínua flexibilidade e movimentação dos dispositivos da rede. Assim, por apenas analisar a variação da quantidade de fluxos em dois instantes diferentes e as portas TCP de destino, a ferramenta se mostra leve, fazendo a atual proposta não apresentar problemas de atraso e nem de alto consumo dos recursos computacionais para detecção do ataque. Estas características do FindFlows em relação ao atual estado da arte o coloca como proposta relevante para a contribuição e evolução desta área de pesquisa.

Este trabalho teve como o seu objetivo a detecção de um tipo de ataque DoS em um rede SDN. Para isso, foi apresentado os benefícios de uma rede SDN no contexto de flexibilidade, escalabilidade e inovação que ajudaram no desenvolvimento da proposta. Também foi mostrado que, por ser uma tecnologia emergente, que tem como característica a centralização do plano de controle, as Redes Definidas por *Software* tem sido um grande cenário de pesquisa na área de segurança das Redes de Computadores. Assim, apresentando os riscos que um controlador SDN pode estar exposto, foi demonstado as atuais soluções no estado da arte para solucionar problemas de vulnerabilidades a ataques DoS. Dentro dos vários ataques DoS existentes, esta pesquisa visou especificamente a detecção do *Syn-flooding* em uma rede SDN.

Um ataque do tipo *Syn-flooding* tem o objetivo de ocasionar a negação do serviço de um Servidor de aplicação, mais comumente utilizado para atacar serviços WEB. Porém, conforme

descrito na introdução e fundamentação teórica deste trabalho, em uma rede SDN esse ataque pode ter consequências ainda maiores por ter a possibilidade de negar o serviço do próprio controlador, através das inúmeras mensagens *packet-in* geradas pela quantidade de requisições diferentes para o servidor, afetando também a infraestrutura, demonstrando assim o problema existente e a necessidade de uma pesquisa que atue nesta solução.

A proposta deste trabalho foi criar um monitor de fluxos SDN, denominado FindFlows, para detectar o ataque DoS *Syn-flooding*, fazendo isso através dos passos abaixo:

- Captura dos fluxos;
- Contabilização da quantidade de fluxos por *host*;
- Armazenamento da variação da quantidade de fluxos por cada *host*;
- Categorização dos *hosts* da rede.

Como visto na seção "FindFlows: Um detector de ataque Syn-Flooding", a detecção acontece em duas etapas:

1. Verificação da variação da quantidade de fluxos entre dois instantes diferentes;
2. Análise das portas TCP de destino dos fluxos.

Para alcançar o objetivo, fez-se necessário realizar as etapas seguintes:

- Análise das mensagens trocadas entre *switch* e controlador, através do protocolo *OpenFlow*;
- Estudo do ataque *Syn-flooding* em uma rede SDN;
- Criação de uma ferramenta de detecção *hosts* participantes do ataque *Syn-flooding*;
- Validação da ferramenta.

Para a validação, foi criado um cenário próximo ao que seria uma rede local SDN de uma empresa. Utilizou-se do Mininet para o emular o cenário da rede SDN junto com outras ferramentas para validar a detecção. Dividido entre dois grupos de testes, cada um com 15 testes realizados, totalizando 30 amostras, a detecção do ataque *Syn-flooding* pôde ser realizada em 27 amostras, ou seja, 90%. O único teste que não foi possível detectar o ataque nas 5 amostras,

foi o 1ª teste do Grupo B, caracterizado por apenas 1 segundo de ataque. Neste teste, o FindFlows não teve coleta suficiente para detectar o ataque, ainda conseguindo detectar em 2, das 5 amostras.

Assim, a ferramenta FindFlows foi desenvolvida para a detecção do *host* autor do ataque *Syn-flooding* em uma rede SDN. Mostrando ao administrador da rede uma tabela com todos os *hosts* da rede, a quantidade de fluxos em dois instantes diferentes, a variação destes dois instantes e a classificação dos *hosts* como usuários legítimos, vítima ou atacante. Os *hosts* que geraram fluxos com IP não correspondente ao da sua subrede local, tiveram seus fluxos separados em um arquivo a parte por serem suspeitos de utilizarem IP *Spoofing*, possibilitando ao analista da rede uma análise mais cautelosa destes fluxos, em um tempo posterior.

Foi verificado que é necessário avançar em alguns pontos para a continuação evolutiva da presente pesquisa. Os trabalhos futuros estão listados abaixo:

1. Melhorar a detecção da falsificação de IP de Origem;
2. Detectar outros tipos de ataques DoS com características semelhantes ao *Syn-flooding*;
3. Mitigar o ataque *Syn-flooding* e outros ataques, além de detectar.

A escolha destes itens para os trabalhos futuros se justifica devido o FindFlows precisar avançar na detecção de falsificação de IP de Origem (IP *Spoofing*) de forma mais eficaz, já que atualmente ele somente separa os fluxos que podem ter seus endereços IP falsificados. Além disso, é interessante que ele seja utilizado para também detectar outras formas de ataques DoS que tenham características semelhantes ao *Syn-flooding*. Uma contribuição a mais também seria que esta mesma tecnologia também mitigasse o ataque, além da função de detectar. Todos esses avanços poderão ser contemplados em trabalhos futuros desta presente contribuição.

REFERÊNCIAS

[1] Diego Rossi Mafioletti. Crops: uma proposta de comutador programável de código aberto para prototipação de redes. Master's thesis, 2016.

[2] Reza Mohammadi, Reza Javidan, and Mauro Conti. Slicots: An sdn-based lightweight countermeasure for tcp syn flooding attacks. *IEEE Transactions on Network and Service Management*, 2017.

[3] Silvia Fichera, Laura Galluccio, Salvatore C Grancagnolo, Giacomo Morabito, and Sergio Palazzo. Operetta: An openflow-based remedy to mitigate tcp synflood attacks against web servers. *Computer Networks*, 92:89–100, 2015.

[4] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. Sphinx: Detecting security attacks in software-defined networks. In *NDSS*, 2015.

[5] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 413–424. ACM, 2013.

[6] Tommy Chin, Xenia Mountrouidou, Xiangyang Li, and Kaiqi Xiong. Selective packet inspection to detect dos flooding using software defined networking (sdn). In *Distributed Computing Systems Workshops (ICDCSW), 2015 IEEE 35th International Conference on*, pages 95–99. IEEE, 2015.

[7] Peng Xiao, Zhiyang Li, Heng Qi, Wenyu Qu, and Haisheng Yu. An efficient ddos detection with bloom filter in sdn. In *Trustcom/BigDataSE/I SPA, 2016 IEEE*, pages 1–6. IEEE, 2016.

[8] João Henrique Gonçalves Medeiros Corrêa. Mitigando ataques de negação de serviço usando listas brancas dinâmicas baseadas em assinaturas. Master Thesis, 2017.

[9] Marcos Sêmola. *Gestão da segurança da informação*, volume 2. Elsevier Brasil, 2014.

[10] D Sophia Navis Mary and A Thasleema Begum. An algorithm for moderating dos attack in web based application. In *Technical Advancements in Computers and Communications (ICTACC), 2017 International Conference on*, pages 26–31. IEEE, 2017.

[11] Yang Xu and Yong Liu. Ddos attack detection under sdn context. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

[12] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[13] Hamid Farhady, HyunYong Lee, and Akihiro Nakao. Software-defined networking: A survey. *Computer Networks*, 81:79–95, 2015.

[14] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.

[15] Tina Tsou, Xingang Shi, Jing Huang, Zhiliang Wang, and Xia Yin. Analysis of comparisons between openflow and forces. *Analysis*, 2012.

[16] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. Network innovation using openflow: A survey. *IEEE communications surveys & tutorials*, 16(1):493–512, 2014.

[17] Sandra Scott-Hayward, Gemma O'Callaghan, and Sakir Sezer. Sdn security: A survey. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For*, pages 1–7. IEEE, 2013.

[18] Masoud Moshref, Apoorv Bhargava, Adhip Gupta, Minlan Yu, and Ramesh Govindan. Flow-level state transition as a new switch primitive for sdn. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 61–66. ACM, 2014.

[19] OpenFlow Switch Specification Version. 1.5. 1 (protocol version 0x06), december, 2014.

[20] Keqiang He, Junaid Khalid, Aaron Gember-Jacobson, Sourav Das, Chaithan Prakash, Aditya Akella, Li Erran Li, and Marina Thottan. Measuring control plane latency in sdn-enabled switches. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 25. ACM, 2015.

[21] Karamjeet Kaur, Japinder Singh, and Navtej Singh Ghumman. Mininet as software defined networking testing platform. In *International Conference on Communication, Computing & Systems (ICCCS)*, pages 139–42, 2014.

[22] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[23] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[24] Eder Leão Fernandes. Nox 1.3 oflib. https://github.com/CPqD/nox13oflib. Acesso em 22 de Outubro de 2018.

[25] 2013 POX. Noxrepro. Pox controller. https://noxrepo.github.io/pox-doc/html/. Acesso em 22 de Outubro de 2018.

[26] 2016 RYU Controller. Build sdn agilely. http://osrg.github.io/ryu/. Acesso em 22 de Outubro de 2018.

[27] Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE communications surveys & tutorials*, 15(4):2046–2069, 2013.

[28] Yuri Gil Dantas, Vivek Nigam, and Iguatemi E Fonseca. A selective defense for application layer ddos attacks. In *Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint*, pages 75–82. IEEE, 2014.

[29] Mitko Bogdanoski, Tomislav Shuminoski, and Aleksandar Risteski. Analysis of the syn flood dos attack. *International Journal of Computer Network and Information Security*, 5(8):1, 2013.

[30] Wesley Eddy. Tcp syn flooding attacks and common mitigations. Technical report, 2007.

[31] Haopei Wang, Lei Xu, and Guofei Gu. Floodguard: A dos attack prevention extension in software-defined networks. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 239–250. IEEE, 2015.

[32] Syed Akbar Mehdi, Junaid Khalid, and Syed Ali Khayam. Revisiting traffic anomaly detection using software defined networking. In *International workshop on recent advances in intrusion detection*, pages 161–180. Springer, 2011.

[33] Qiao Yan, Qingxiang Gong, and Fang-an Deng. Detection of ddos attacks against wireless sdn controllers based on the fuzzy synthetic evaluation decision-making model. *Adhoc & Sensor Wireless Networks*, 33, 2016.

[34] Moreno Ambrosin, Mauro Conti, Fabio De Gaspari, and Radha Poovendran. Lineswitch: Efficiently managing switch flow in software-defined networking while effectively tackling dos attacks. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 639–644. ACM, 2015.

[35] Mutalifu Kuerban, Yun Tian, Qing Yang, Yafei Jia, Brandon Huebert, and David Poss. Flowsec: Dos attack mitigation strategy on sdn controller. In *Networking, Architecture and Storage (NAS), 2016 IEEE International Conference on*, pages 1–2. IEEE, 2016.

[36] 2018 Internet2. The internet2 community: enabling the future. https://www.internet2.edu/about-us/. Acesso em 29 de Outubro de 2018.

[37] Javed Ashraf and Seemab Latif. Handling intrusion and ddos attacks in software defined networks using machine learning techniques. In *Software Engineering Conference (NSEC), 2014 National*, pages 55–60. IEEE, 2014.

[38] DEVMEDIA. Top 10 linguagens de programação mais usadas no mercado. https://www.devmedia.com.br/top-10-linguagens-de-programacao-mais-usadas-no-mercado/39635. Acesso em 5 de Novembro de 2018.