Uso de Redes Definidas por *Software* para mitigar ataques DDoS

Trabalho de Conclusão de Curso

Engenharia da Computação

Carlos Antonio Sales de Oliveira Junior
Orientador: Prof. Edison de Queiroz Albuquerque

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

Carlos Antonio Sales de Oliveira Junior

Uso de Redes Definidas de *Software* para mitigar ataques DDoS

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, agosto de 2021

Junior, Carlos Antonio Sales de Oliveira
Uso de Redes Definidas por *Sofware* para mitigar ataques DDoS / Carlos Antonio Sales de Oliveira Junior. – Recife - PE, 2021.
xii, 37 f. : il. ; 29 cm.

Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação) Universidade de Pernambuco, Escola Politécnica de Pernambuco, Recife, 2021.

Orientador: Profª. Drª. Edison de Queiroz Albuquerque.

Inclui referências.

1. Ataque DDoS. 2. Redes definidas por *sofware.* 3. Proteção. I. Título. II Albuquerque, Edison de Queiroz de. III. Universidade de Pernambuco.

Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação
Projeto de Final de Curso

ESCOLA POLITÉCNICA
DE PERNAMBUCO

MONOGRAFIA DE FINAL DE CURSO

Avaliação Final (para o presidente da banca)*

No dia 30/8/2021, às 14h00min, reuniu-se para deliberar sobre a defesa da monografia de conclusão de curso do(a) discente **CARLOS ANTONIO SALES DE OLIVEIRA JUNIOR**, orientado(a) pelo(a) professor(a) **EDISON DE QUEIROZ ALBUQUERQUE**, sob título Uso de Redes Definidas por Software para mitigar ataques DDoS, a banca composta pelos professores:

LUIS CARLOS DE SOUSA MENEZES (PRESIDENTE)
EDISON DE QUEIROZ ALBUQUERQUE (ORIENTADOR)

Após a apresentação da monografia e discussão entre os membros da Banca, a mesma foi considerada:

x Aprovada Aprovada com Restrições* Reprovada

e foi-lhe atribuída nota: 7,5 (sete e meio)

*(Obrigatório o preenchimento do campo abaixo com comentários para o autor)

O(A) discente terá 30 dias para entrega da versão final da monografia a contar da data deste documento.

Luis Carlos de Sousa Menezes
AVALIADOR 1: Prof (a) **LUIS CARLOS DE SOUSA MENEZES**

Edison de Queiroz Albuquerque
AVALIADOR 2: Prof (a) **EDISON DE QUEIROZ ALBUQUERQUE**

AVALIADOR 3: Prof (a)

* Este documento deverá ser encadernado juntamente com a monografia em versão final.

Este trabalho é dedicado à a toda minha família e à todas as pessoas que me ajudaram durante a minha jornada acadêmica.

AGRADECIMENTOS

O primeiro profundo agradecimento vai aos meus pais, ao meu irmão e todos os outros componentes da minha família como tias e tios. Segundo agradecimento vai para o Prof. Edison Queiroz de Albuquerque pelo auxilio, a orientação e o incentivo por todos os passos da construção do projeto. O terceiro a todos os meus professores que me ajudaram na minha formação acadêmica. A todos o meu muito obrigado.

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Autorização de publicação de PFC

Eu, **Carlos Antonio Sales de Oliveira Junior** autor(a) do projeto de final de curso intitulado: **Uso de Redes Definidas por Software para mitigar ataques DDoS;** autorizo a publicação de seu conteúdo na internet nos portais da Escola Politécnica de Pernambuco e Universidade de Pernambuco.

O conteúdo do projeto de final de curso é de responsabilidade do autor.

Carlos Antonio Sales de Oliveira Junior
Carlos Antonio Sales de Oliveira Junior

Orientador(a): **Edison de Queiroz Albuquerque**

Coorientador(a):

Prof, de TCC: **Daniel Augusto Ribeiro Chaves**

Data: 30/8/2021

RESUMO

Os ataques de negação de serviço têm como objetivo o ataque aos serviços que são providos por um servidor. Esse tipo de ataque vem crescendo a cada ano exigindo cada vez mais investimentos financeiros e tecnológicos por partes de empresas e governos. A utilização de redes definidas por *software* permite o controle e o gerenciamento da rede facilitando no planejamento de mecanismos de defesa. Nesse artigo, apresentaremos através de um experimento uma abordagem utilizando o funcionamento de uma rede SDN com um mecanismo para mitigar os ataques DDoS. O experimento foi feito em uma comparação entre dois cenários propostos. O primeiro uma rede sem a proteção e o segundo com a proteção, ou seja, o programa proposto no artigo. A conclusão que foi alcançada é que a solução proposta melhora no combate a perda de pacotes e melhor resposta a um ataque DDoS pela rede.

Palavras-chaves: ataque DDoS; redes definidas por *software*; segurança; proteção.

ABSTRACT

Denial of service services are intended to attack services that are provided by a server. This type of attack has been growing every year, requiring more and more financial and technological investments by companies and governments. The use of software defined networks allows control and management of the network facilitating defense planning. In this article, we will present through an experiment an approach using the functioning of an SDN network with a mechanism to mitigate DDoS. The experiment was carried out in a comparison between two proposed scenarios. The first an unprotected network and the second with protection, that is, the program proposed in the article. The conclusion reached is that the proposed solution improves the fight against packet loss and better response to a DDoS attack over the network.

Keywords: DDoS attack; software defined networks; safety; protection.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	Interface de Programação de Aplicações
DoS	Negação de Serviço
DDoS	Ataque Distribuído de Negação de Serviço
SDN	Redes Definidas por *Software*
TCP	Protocolo de Controle de Transmissão

SUMÁRIO

1 INTRODUÇÃO

Nos dias de hoje é cada vez mais é comum nos depararmos com notícias relacionadas a indisponibilidade de um serviço ou algum site na web. A arquitetura da Internet foi originalmente projetada visando abertura e escalabilidade. Apesar de seu sucesso nesses quesitos, diversos problemas de segurança não foram originalmente previstos (PENG; LECKIE; RAMAMOHANARAO, 2007). Um estudo (INSTITUTE, 2015) mostra que as 641 empresas pesquisadas, que estão envolvidas com prevenção, detecção ou que foram vítimas desses ataques, relataram perdas de 1,5 milhão de dólares de prejuízo em um ano em função desses ataques. Um desses problemas enfrentados é o ataque distribuído de negação de serviço (DDoS ou *Distribuded Denial of Service*). Ele é uma das maiores ameaças a privacidade de dados domésticos e empresariais.

Redes definidas por software (SDN ou *Software Defined Networking*) é uma arquitetura emergente de redes que realiza a separação física do plano de dados do plano de controle de *switches*, que é diretamente programável (ONF, 2012). A arquitetura SDN pode ser explorada para melhorar a segurança de uma rede através de sistemas de monitoração, análise e resposta mais reativos que os sistemas convencionais (SCOTT-HAYWARD; O'CALLAGHAN; SEZER, 2013). A vantagem de uma rede SDN em relação a uma rede no formato tradicional é a sua menor complexidade no gerenciamento, expansão e configuração.

Este artigo está organizado da seguinte maneira. A seção 2 faz a fundamentação teórica apresentando os conceitos sobre DDoS, SDN, o protocolo *OpenFlow* e a segurança. A seção 3 apresenta a metodologia utilizada para analisar o cenário do experimento. Na seção 4 são apresentados os resultados obtidos com a realização do experimento. A seção 5 traz as conclusões e considerações para trabalhos futuros.

1.1 Caracterização do problema

A proteção eficiente de uma rede é de grande importância para a disponibilidade de serviços e recursos. Fazer o monitoramento e a reação a ataques em uma rede requer o emprego de ações e ferramentas para garantir o bom funcionamento dos serviços na rede para que os diferentes tipos de usuários como administradores do serviço, gestores da rede e o usuário final tenha total acesso ao serviço.

O problema a ser discutido nesse artigo é mostrar como ataque DDoS prejudicam a disponibilidade de serviços causando um grande impacto tanto financeiro, como de experiência para o usuário do serviço e como a utilização do SDN pode ser usada para mitigar esse tipo de problema.

1.2 Objetivos

Este artigo tem como objetivo trabalhar em uma solução utilizando uma rede SDN, que vai funcionar em um ambiente controlado e mostrar como essa rede funcionando através de testes, pode prover proteção à ataques DDoS, além de ratificar a sua eficiência no combate. Como resultado do projeto espera-se ter um maior conhecimento tanto teórico como prático sobre ataques DDoS e a utilização de SDN para evitar ou diminuir esse problema.

2 FUNDAMENTAÇÃO TEÓRICA

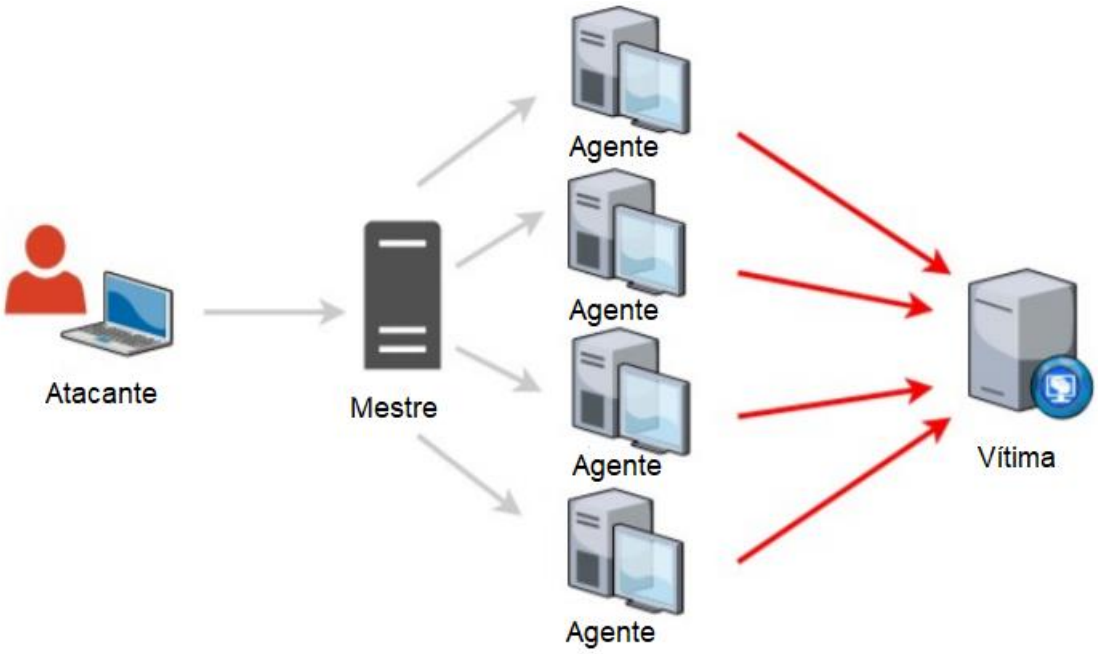
Nesta seção será detalhada a fundamentação teórica com o objetivo de estimular o entendimento dos conceitos e das tecnologias que serão utilizadas. A seção 2.1 será explicado sobre os ataques DDoS. A seção 2.2 mostraremos mais informações sobre o SDN, o protocolo *OpenFlow* e questões da segurança.

2.1 Ataques DDoS

Os ataques de negação de serviço (DoS ou *Denial of Service)* tem como objetivo impossibilitar o fornecimento de serviço prestado por um servidor, portanto esses ataques não possuem como objetivo destruir informação, mas sim, comprometer a disponibilidade dos recursos do sistema. Apesar dos ataques DDoS possuírem o mesmo objetivo dos ataques DoS, eles diferem na quantidade de máquinas necessárias para desempenhar o ataque, visto que em ataques DoS somente um atacante participa do ataque e em ataques DDoS múltiplas máquinas são tipicamente utilizadas (DOULIGERIS; MITROKOTSA, 2004). O primeiro ataque DDoS foi relatado em 1996, e desde então foram relatados ataques em grandes instituições tais como a *webpage* do FBI (2000), *ClickBanck* (2003), *Facebook* (2009), *Twitter* (2009) e *Feedly* (2014) (RADWARE, 2015).

Um ataque DDoS ocorre quando diversos pacotes são enviados simultaneamente por várias máquinas para uma única máquina de destino. A figura 1 ilustra que um ataque é composto por quatro elementos: o atacante, o *handler* ou mestre, o agente ou *zombie* e a vítima. O atacante é a máquina que efetivamente coordena o ataque. O *handler* é um hospedeiro infectado por um programa capaz de controlar múltiplos agentes. O agente é a máquina que realiza o ataque DDoS. Dependendo da aplicação o agente pode recrutar novos agentes e, consequentemente, pode se tornar uma máquina mestre. A vítima é a máquina a ser atacada (DOULIGERIS; MITROKOTSA, 2004).

Figura 1 - Ataque DDoS.



Fonte: Adaptada de Gonçalves (2020).

Os ataques DDoS são classificados em dois tipos: os ataques de esgotamento de largura de banda e os ataques de esgotamento de conexões (recursos). Os ataques que esgotam a largura de banda de um serviço com tráfego indesejado impedem que um cliente legítimo tenha acesso a esse serviço em função do comprometimento do enlace. Os ataques de esgotamento de conexões são capazes de comprometer recursos como a memória e a capacidade de processamento dos sistemas atacados. Esse tipo de ataque torna os serviços alvos incapazes de atender a novas requisições (HUANG; KOBAYASHI; LIU, 2003).

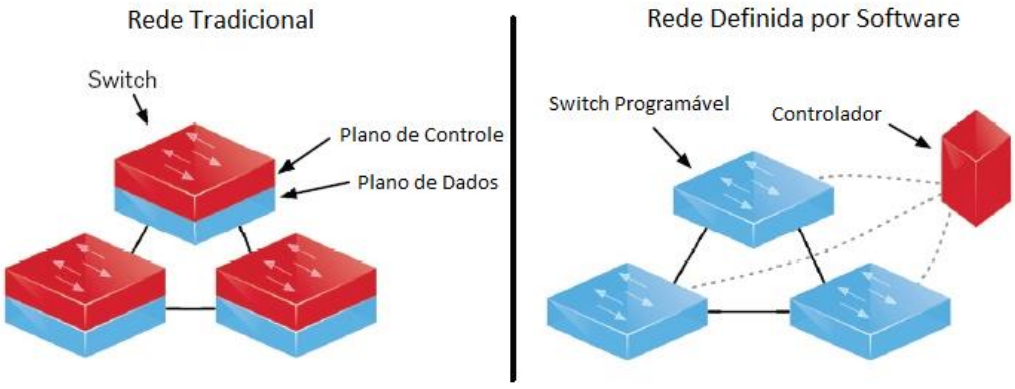
2.2 Redes Definidas por *Software*

As redes de computadores tradicionais por causa de seu tamanho, são redes de difícil manutenção e configuração, por exemplo, a limitação do que pode ser feito com os dispositivos, devido a cada fabricante possuir seu próprio código fechado.

Para fazer o gerenciamento desses dispositivos é necessário o treinamento de uma equipe técnica especializada dificultando a capacidade de implementação de políticas de alto nível. Outra dificuldade é a integração entre planos de controle e de dados da rede em todos os dispositivos que as compõem. Essa integração faz com que cada dispositivo seja responsável por definir e executar suas tarefas de encaminhamento e controle de dados, limitando a flexibilidade da rede (BARBOSA, 2018).

A principal característica de difere as redes tradicionais do SDN é o desacoplamento do plano de controle e do plano de dados. O plano de dados encaminha os pacotes usando as tabelas de encaminhamento geradas pelas informações dadas pelo plano de controle. Toda logica de controle é feita pelo controlador.

Figura 2 - Descrição de uma rede tradicional e uma rede definida por *software.*

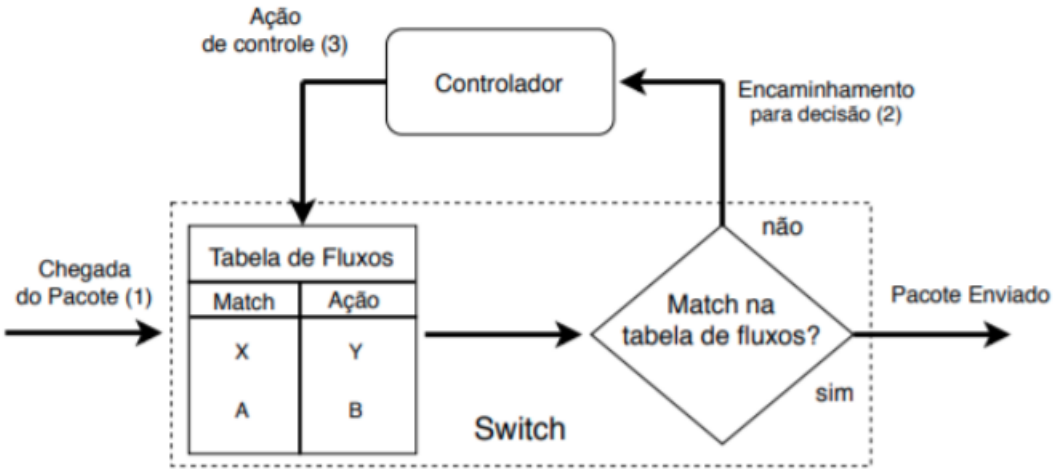


Fonte: Adaptado de (WOLFF, 2019).

Na figura 3 é apresentado um exemplo genérico simplificado do funcionamento de uma rede SDN. O exemplo inicia-se com a chegada de um pacote (1). Se não existe um fluxo para ele na tabela de fluxo o pacote será encaminhado ao controlador (2) que vai decidir o destino dele enviando um novo fluxo para a

Tabela de Fluxos do *switch*. Pela decisão do controlador (3) o pacote pode ser encaminhado pelo comutador ou bloqueado.

Figura 3 - Exemplo funcionamento de uma rede definida por *software.*



Fonte: Adaptado de (SINGH et al., 2017).

Os dispositivos rotadores que componhe a infraestrutura das redes SDN tem funcionalidade determinado pelo controlador e podem ser comportar como *switches*, roteadores, etc.

2.2.1 Protocolo *OpenFlow*

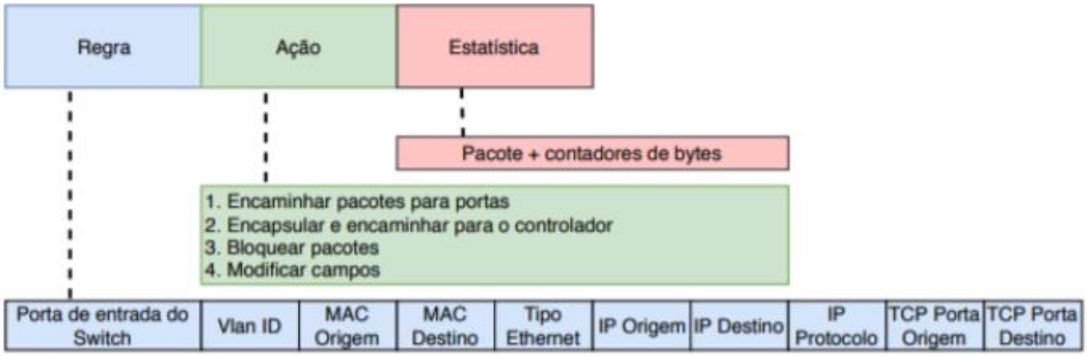
O *OpenFlow* é um protocolo definido para fornecer todas as funcionalidades propostas pelas redes SDN. O *OpenFlow* é um protocolo aberto que permite ao administrador de uma rede programar a tabela de fluxo em diferentes equipamentos. As tabelas de fluxo são utilizadas para executar ações de controle na rede de acordo com as necessidades existentes. Apesar das tabelas de fluxo serem diferentes entre os fabricantes, um conjunto comum de funções executadas por todos os fabricantes, permitiu a exploração das ações de controle pelo *OpenFlow* (MCKEOWN et al., 2008).

O comutador *OpenFlow* é responsável pelo encaminhamento de pacotes entre portas sendo o encaminhamento determinado por um controlador. Cada fluxo

no comutador OpenFlow possui uma ação. As três ações básicas que um comutador sempre suporta são o encaminhamento de pacotes entre portas, o encaminhamento de pacotes para o controlador através de um canal seguro e os fluxos de bloqueio para pacotes. Pode ser usado para contenção de ataques ou redução de trafego (MCKEOWN et al., 2008).

Na figura 4 estão sendo apresentados os padrões que o comutador *OpenFlow* se guia para identificar os fluxos que o plano de dados executa sobre os pacotes em trânsito. A Regra contém os campos do cabeçalho usados na especificação dos fluxos do *OpenFlow*. A ação permite ao comutador saber o que será feito com determinado pacote que “casa” com um determinado fluxo. A estatística contém os contadores que mostram, por exemplo, quantas vezes pacotes foram encaminhados através de um determinado fluxo.

Figura 4 - Campos do cabeçalho, ações e estatísticas do *OpenFlow*.



Fonte: Adaptado de (BRITO, 2019).

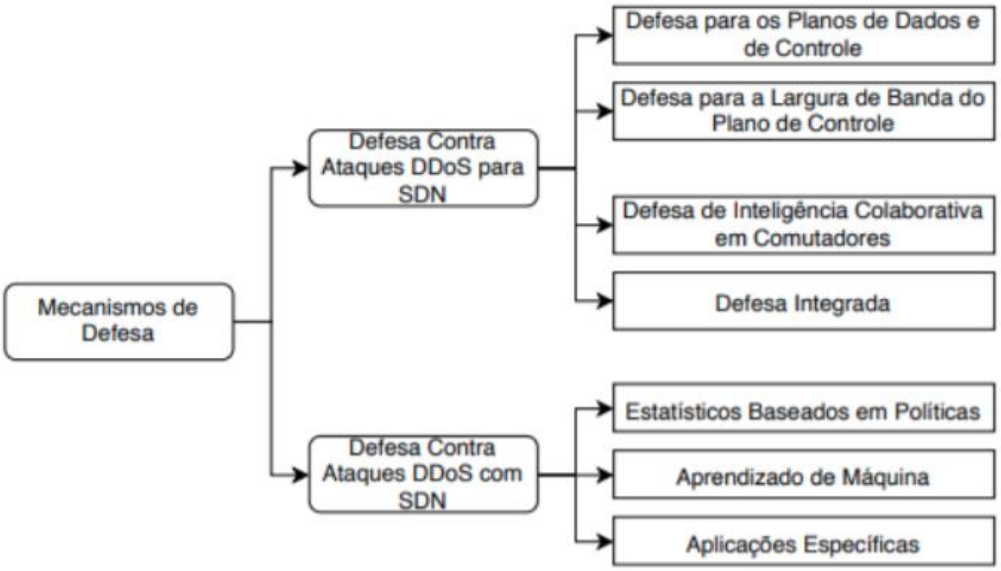
2.2.2 Segurança

O SDN possui muitos desafios de segurança em comum com as redes tracionais. Problemas ligados à questão da escalabilidade, são devido ao controlador ser um ponto único de falha, possibilitando que ataques neste ponto afetem toda a rede. Para aumentar a disponibilidade, vários controladores podem ser ativados, uns fazendo *backup* dos outros (LOPES; FERNANDES; ALBUQUERQUE, 2019).

Ataques de negação de serviço (DoS), inundam a rede com um grande número de pacotes em um curto espaço de tempo, sobrecarregando o controlador. Outra vulnerabilidade introduzida vem do fato de muitas vezes a implementação de SDN se dar através de padrões abertos e conhecidos.

O SDN, na perspectiva da segurança possui dois pontos de vista sobre a proteção de redes contra ataques DDoS (Figura 5). No primeiro ponto, o SDN pode se tornar a própria vítima de ataques por causa do mecanismo de controle centralizado, enquanto que, no segundo ponto, o SDN utiliza seus recursos para proteger redes convencionais (SWAMI; DAVE; RANGA, 2019). As soluções de defesa são classificadas com base nos recursos da estrutura do SDN. Os mecanismos de defesa contra ataques DDoS para SDN foram classificados em quatro categorias: a defesa para os planos de dados e de controle, a defesa para a largura de banda do plano de controle, a defesa de inteligência colaborativa em comutadores e a defesa integrada. Os mecanismos para a defesa contra ataques DDoS com SDN foram classificados em três categorias: os estatísticos e baseados em políticas, o aprendizado de máquina e as aplicações específicas.

Figura 5 - Mecanismos de defesa em SDN.



Fonte: Adaptado de (SWAMI; DAVE; RANGA, 2019).

3 METODOLOGIA

Nessa seção serão apresentados os detalhes sobre o experimento como a escolha do ambiente de emulação para os testes e a construção do protótipo.

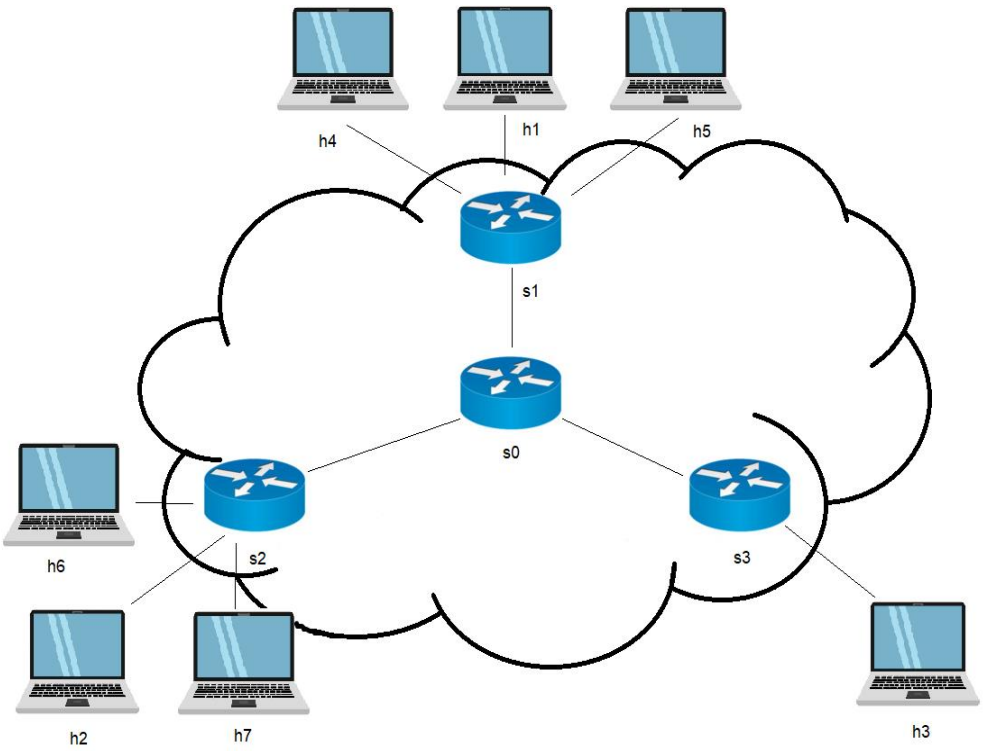
O experimento foi realizado num computador com 8 *gigabytes* de memória e um processador *Intel*(R) *Core*(TM) i7-4720HQ CPU @ 2,60 GHz. O ambiente para teste foi configurado no *VirtualBox* 6.1 (VIRTUALBOX, 2021) que é um *software* de virtualização o qual foi utilizado o sistema Ubuntu na versão 21.04 (UBUNTU, 2021).

Para a construção do protótipo para avaliação foi escolhido a ferramenta *Mininet*. *Mininet* é uma ferramenta de emulação que possibilita a criação de uma rede virtual *OpenFlow* - com controlador, *switches*, hospedeiros e *links* - em uma única máquina real ou máquina virtual (*MININET*, 2021). O controlador escolhido para a rede SDN foi o *Ryu*. *Ryu* é uma estrutura de rede definida por *software* baseada em componentes. *Ryu* fornece componentes de *software* com API bem definida que torna mais fácil para os desenvolvedores criarem novos aplicativos de gerenciamento e controle de rede. Ryu suporta vários protocolos para gerenciar dispositivos de rede, como *OpenFlow* , *Netconf*, *OF-config*, etc. Sobre *OpenFlow*, *Ryu* suporta totalmente 1.0, 1.2, 1.3, 1.4, 1.5 e extensões *Nicira* (RYU, 2021). Todos os programas criados para o experimento estão disponíveis no APÊNDECE A – CÓDIGOS FONTE do artigo e utilizaram a linguagem *Python* que é uma linguagem de programação de alto nível, interpretada de *script*, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte (PYTHON, 2021). A aplicação monitora o trafego no roteador e quando atinge 40% ou 50% de capacidade (HASSIDIN; RAZ; SEGALOV; SHAQED, 2013), ele começa a descarta os novos pacotes entrantes para o alvo e para a porta determinada. Após um intervalo de 10 segundos, os *switches* voltam a encaminhar este tipo de pacote.

Na figura 6 é apresentada a topologia utilizada na rede para a criação do experimento. A rede é formada por sete *hosts* e quatro roteadores e foi utilizado o código topo-7h-3s-1s0. As máquinas h1, h2, h3, h4, h5, h6 e h7 estão conectados aos roteadores s1, s2 e s3 respectivamente os quais funcionam como roteadores de borda. Esses roteadores se conectam com o roteador s0 que funciona como um

roteador de núcleo. O experimento vai ser feito com duas situações. Os testes serão feitos num intervalor de 10 minutos. A primeira situação vai ser simulado um ataque DDoS feito pelos *hosts* h1, h4 e h5 com o programa criado tcpFlood1 e h2, h6 e h7 com o programa criado tcpFlood2 que vão inundar a rede com pacotes em direção ao *host* h3 que vai funcionar como um servidor com o programa TcpServer e com isso vai ser verificado o descarte dos pacotes na rede e o impacto para o seu funcionamento. A segunda situação vai ser feito o mesmo tipo de simulação de ataque, mas utilizando a aplicação *Ryu* DDoS_switch_15v5 criada para mitigar os ataques DDoS e vai ser verificado o comportamento da rede e o descarte de pacotes.

Figura 6 - Topologia da rede.



Fonte: Produzido pelo Autor (2021).

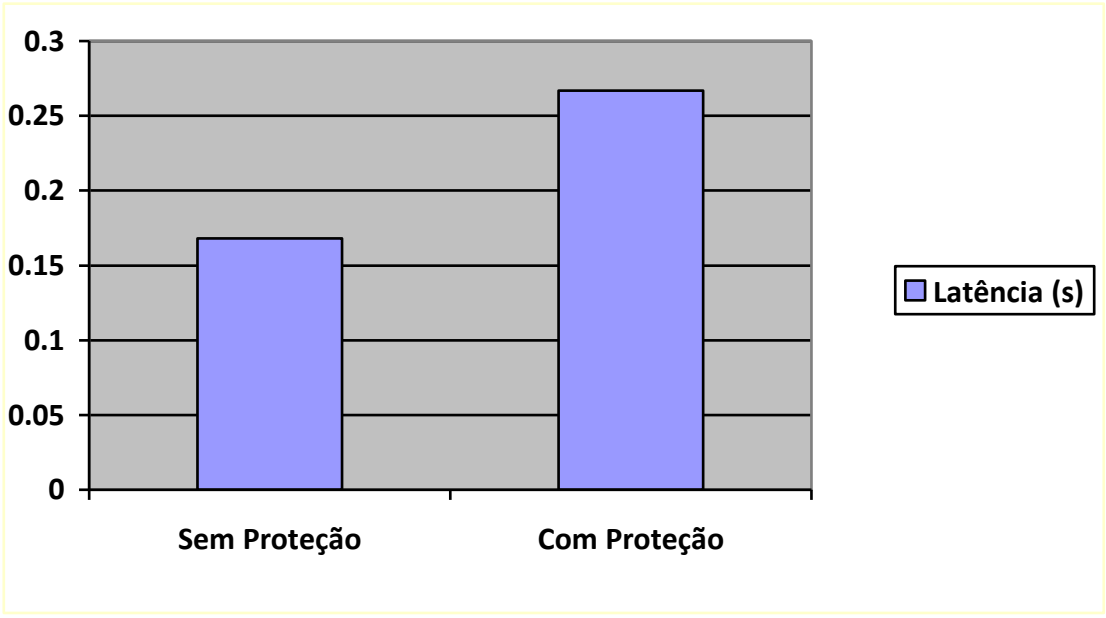
4 RESULTADOS

Nessa sessão são apresentados e exibidos os resultados alcançados com a realização do experimento. A primeira situação proposta é o funcionamento da rede sem proteção contra ataques DDoS utilizando o programa simple_switch_15 que vem como um exemplo no controlador *Ryu*. A segunda situação proposta é o funcionamento da rede com uma proteção.

Na figura 7 é possível verificar um aumento da latência média proporcionada pelas requisições dos *hosts* h1, h2, h4, h5, h6 e h7 ao *host* h3 que está atuando como um servidor. O controlador protege o *host* h3 descartando os pacotes, que excedem o limite configurado, durante 10 segundos. Observamos que o aumento da latência foi bem pequeno. Uma perda pequena para obter a proteção.

Para o cálculo da média da latência foi feita a medição em um intervalo de 10 minutos de teste. Depois foram feitas mais duas execuções e calculado o valor da média desses três valores. Esse procedimento foi repetido para o cálculo do maior valor de latência. As medições foram efetuadas sobre o tráfego entrante no *host* 3 (servidor).

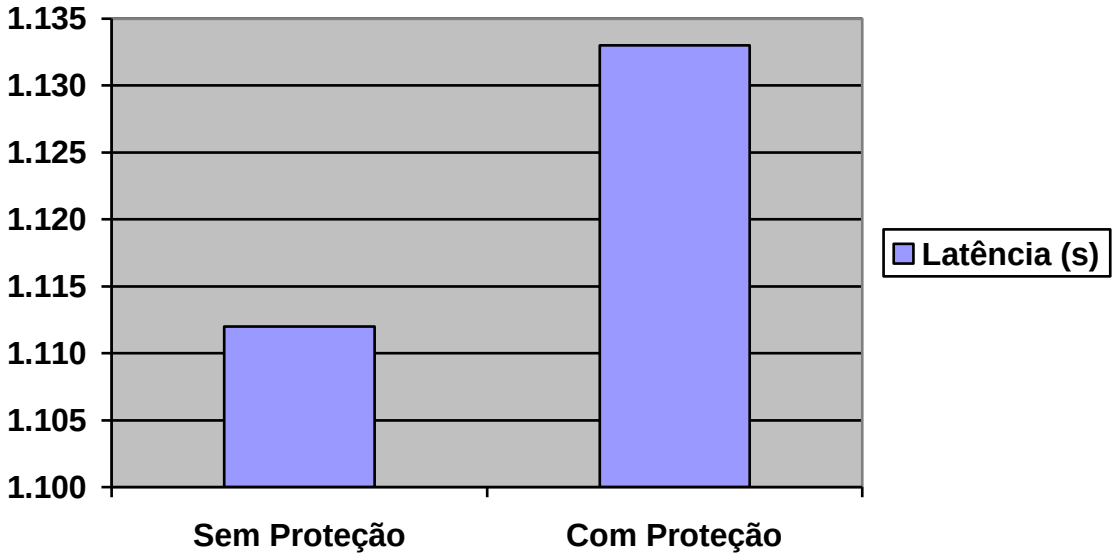
Figura 7 - Média da latência da rede sem e com proteção.



Fonte: Produzido pelo Autor (2021).

Na figura 8 é visto a questão da latência com o maior valor atingido no intervalo de tempo do teste. Como na figura 7 o aumento da latência foi baixo.

Figura 8 - Maior latência alcançada na rede sem e com proteção.

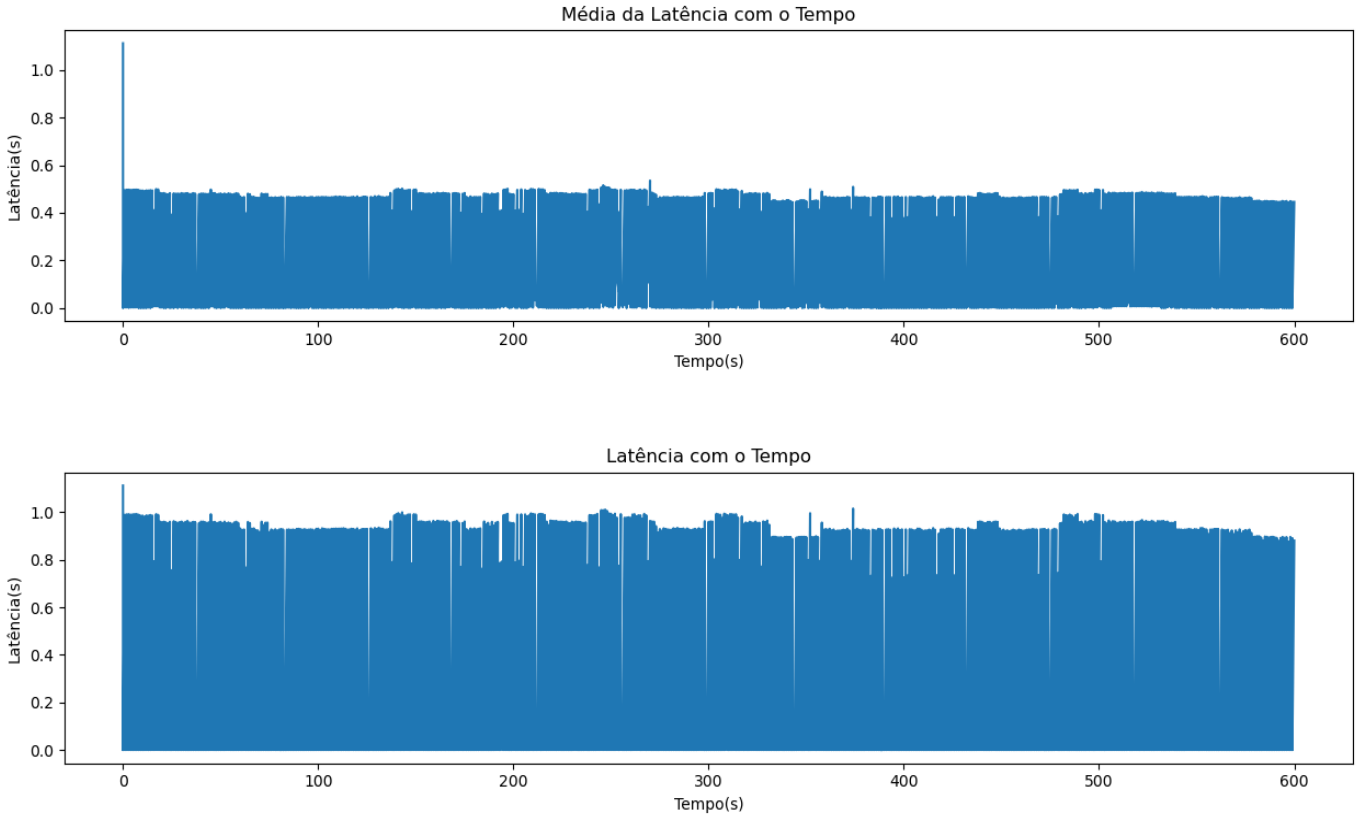


Fonte: Produzido pelo Autor (2021).

Na figura 9 e 10 vemos três gráficos que foram feitos com o auxílio de *Matplotlib*. *Matplotlib* é uma biblioteca abrangente para a criação de visualizações estáticas, animadas e interativas em *Python. Matplotlib* produz figuras com qualidade de publicação em uma variedade de formatos impressos e ambientes interativos em várias plataformas. *Matplotlib* pode ser usado em scripts *Python*, o *shell Python* e *IPython*, servidores de aplicativos da web e vários kits de ferramentas de interface gráfica do usuário (Matplotlib, 2021).

Na figura 9 observamos um aumento de latência em um determinado tempo que é causado pelo nosso programa que quando a rede é saturada ele descarta os pacotes para proteger o serviço. Como estamos utilizando o protocolo de controle de transmissão (TCP ou *Transmission Control Protocol*) no caso se o pacote for perdido ele será novamente enviado.

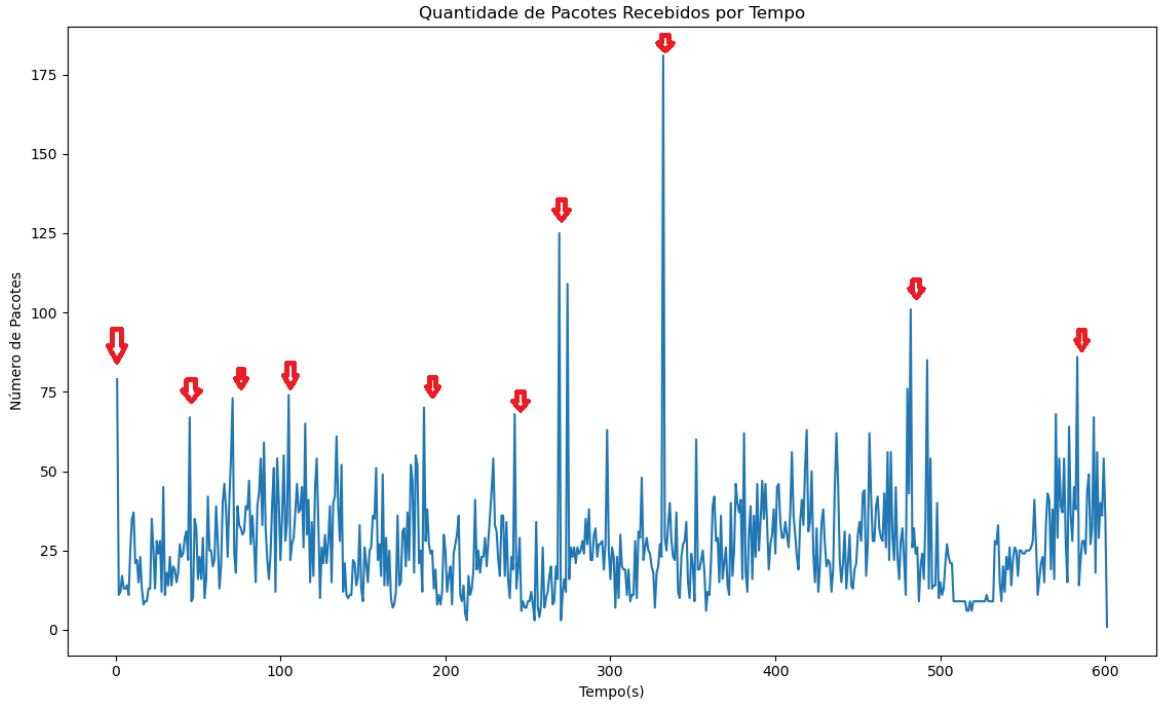
Figura 9 - Variação da latência ao longo do tempo na rede sob ataque com proteção.



Fonte: Produzido pelo Autor (2021).

Na figura 10 é visualizado o início do ataque DDoS ao servidor e a interversão do sistema de proteção que aplica um atraso de 10 segundos que pode ser observado na diminuição do número de pacotes que chegam ao servidor, assim o protegendo. Alguns dos pontos de interversão para proteção são assinalados em vermelho. É observado a grande variação no número de pacotes causada pela grande saturação da rede que ocasiona a perda de pacotes fazendo com o protocolo TCP envie o pacote novamente.

Figura 10 - Variação do número de pacotes que chegam no servidor ao longo do tempo na rede sob ataque com proteção.



Fonte: Produzido pelo Autor (2021).

5 CONCLUSÕES E TRABALHOS FUTUROS

Esse artigo propôs a utilização do SDN para a proteção contra os ataques DDoS. O principal objetivo de se utilizar o SDN com um controlador *Ryu* acrescido do nosso código é cuidar do funcionamento da rede detectando possíveis ataques DDoS e tomando ações como o descarte de pacotes ou atraso nos pacotes para não impactar na experiencia de uso da rede.

Além disso, percebe-se que o sistema é capaz de mitigar um ataque DDoS fazendo com que as latências de resposta do servido sofram um pequeno acréscimo de tempo para requisições legítimas garantindo o funcionamento e proteção do servidor ou serviço.

Para trabalhos futuros podem ser implementadas modificações no nosso programa para um maior desempenho da aplicação. Como exemplo, melhorar a detecção dos pacotes suspeitos, acrescentar novas funcionalidades no nosso programa como um *firewall* e a utilização da nossa solução em um ambiente real como uma empresa e trabalhando em conjunto com uma rede convencional para uma maior e melhor bateria de teste.

Um exemplo do uso da nossa aplicação em um cenário empresarial é a sua utilização como um serviço contratado a parte com uma infraestrutura como controlador e *switch* dedicada para o controle do trafego de pacotes na rede que chegam na empresa ou determinado serviço.

REFERÊNCIAS

[1] BARBOSA, R. R. Migração de redes tradicionais para SDN. Dissertação (Mestre) — Universidade de São Paulo, São Paulo, 2018.

[2] DOULIGERIS, C.; MITROKOTSA, A. {DDoS} attacks and defense mechanisms: classification and state-of-the-art. Computer Networks, v. 44, n. 5, p. 643 – 666, 2004. ISSN 1389-1286.

[3] HASSIDIM, A.; RAZ, D.; SEGALOV, M.; SHAQED, A. Network Utilization: the Flow View. In: IEEE INFOCOM 2013, 2013, Turim. p. 2.

[4] HUANG, Q.; KOBAYASHI, H.; LIU, B. Analysis of a new form of distributed denial of service attack. In: Proc. 37th Ann. Conf. Information Science and Systems (CISS'03). [S.l.: s.n.], 2003.

[5] INSTITUTE, P. The Cost of Denial-of-Services Attacks. Michigan, 2015.

[6] LOPES P. G., FERNANDES S. M. M, ALBUQUERQUE E. Q., Alta DISPONIBILIDADE DOS CONTROLADORES EM REDES DEFINIDAS POR SOFTWARE, Dissertação de Mestrado. 2019.

[7] MATPLOTLIB. Matplotlib, 2021. Disponível em: <https:// https://github.com/matplotlib/matplotlib>. Acesso em: 20 de julho de 2021.

[8] MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, ACM, v. 38, n. 2, p. 69–74, 2008.

[9] MININET. Mininet, 2021. Disponível em: <https://github.com/mininet/mininet>. Acesso em: 20 de março de 2021.

[10] ONF. Software-Defined Networking: The New Norm for Networks. [S.l.], 2012.

[11] PENG, T.; LECKIE, C.; RMAMOHANARAO, K. Survey of network-based defense mechanisms countering the dos and ddos problems. ACM Computing Surveys (CSUR), ACM, v. 39, n. 1, p. 3, 2007.

[12] PYTHON. Python, 2021. Disponível em: <https://www.python.org>. Acesso em: 20 de março de 2021.

[13] RADWARE. DDoS Handbook Radware. 2015. Disponível em: <htps:/security.-radware.com/uploadedfiles/resources_and_content/ddos_handbook.pdf>. Acesso em: 20 de março de 2021.

[14] RYU. Ryu, 2021. Disponível em: <https://ryu-sdn.org/>. Acesso em: 20 de março de 2021.

[15] SCOTT-HAYWARD, S.; O'CALLAGHAN, G.; SEZER, S. Sdn security: A survey. In: IEEE. 2013 IEEE SDN For Future Networks and Services (SDN4FNS). [S.l.], 2013. p. 1–7.

[16] SWAMI, R.; DAVE, M.; RANGA, V. Software-defined networking-based ddos defense mechanisms. ACM Computing Surveys (CSUR), ACM, v. 52, n. 2, p. 28, 2019.

[17] UBUNTU. Ubuntu, 2021. Disponível em <https://www.ubuntu.com>. Acesso em: 17 de março de 2021.

[18] VIRTUALBOX. VirtualBox, 2021. Disponível em <https://www.virtualbox.org>. Acesso em: 18 de março de 2021.

[19] YIN. R. K. Estudo de caso: planejamento e métodos. 3 ed., Porto Alegre: Bookman, 2005.

APÊNDICE A - CÓDIGOS FONTE

Código 1 - Código fonte do tcpFload1 utilizado nos *hosts* h1, h4 e h5.

```
import socket
from datetime import datetime

current_time = datetime.now().minute
waiting = True
while waiting:
     current_time_now = datetime.now().minute
     if current_time_now != current_time:
        waiting = False

while True:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(('10.0.0.3', 5001))
    s.send(b'1')
    s.close()
```

Fonte: Produzido pelo Autor (2021).

Código 2 - Código fonte do tcpFload2 utilizado nos *hosts* h2, h6 e h7.

```
import socket
from datetime import datetime

current_time = datetime.now().minute
waiting = True
while waiting:
     current_time_now = datetime.now().minute
     if current_time_now != current_time:
        waiting = False

while True:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(('10.0.0.3', 5001))
    s.send(b'2')
    s.close()
```

Fonte: Produzido pelo Autor (2021).

Código 3 - Código fonte do tcpServer utilizado no *host* h3.

```
import socket
import time
from datetime import datetime
from datetime import timedelta
import matplotlib.pyplot as plt
import numpy as np

waiting = True
test_minute = 11 #  tempo de 10m por teste

while waiting:
     current_time_now = datetime.now().second
     if 59 == current_time_now:
        waiting = False

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 5001))
s.listen(2)

time_now = time.time()
time_start = True
larger_media = 0.0
service_time = True
current_time = datetime.now() + timedelta(minutes=test_minute) #  tempo de 10m
 por teste
larger_time_msg = 0.0

print('Server iniciado')

x = [] #tempo
x1 = [] #vetor auxiliar
x2 = [] #eixo x da latencia do gráfico da média da latência e latência
x3 = [] #vetor auxiliar
x4 = [] #eixo x do tempo do gráfico de pacotes por segundo
y1 = [] #eixo y da média da latência do gráfico da média da latência
y2 = [] #eixo y da latência do gráfico da latência
y3 = [] #eixo y do número de pacotes do gráfico de pacotes por segundo

while service_time:
    con, cli = s.accept()
    last_time = time.time()
    time_message = last_time - time_now
    msg = con.recv(1024)
    if time_message > 1:
        print('Esperando')
        print('')
```

```
    print('SYN', msg, time_message)
    if time_start: # média
        media = 0.0
        larger_time_msg = time_message
    else:
        media = (time_message + media) / 2
    print('Média: ', media)
    if larger_time_msg <= time_message: # maior tempo de resposta(latência)
        larger_time_msg = time_message
    print('Maior Tempo de Resposta ', larger_time_msg)
    if larger_media <= media: # maior valor que a média atingio
        larger_media = media
    print('Maior Média: ', larger_media)
    print('')
    if time_start: #atribuir o tempo para o proximo calculo de média
        media = time_message
        time_start = False
    time_now = last_time

    con.close()

    current_time_now = datetime.now().minute
    if current_time_now == current_time.minute:
        service_time = False

    x.append(datetime.now().strftime("%M:%S")) #tempo
    y1.append(media) #média latência
    y2.append(time_message) # número da latência

count_x =  len(x)
count_x1 = 0
i = 0
countSeconds = 0 # contador segundos
while count_x > count_x1: #graficos de latências
    x1.append(x[i])
    count_x1 += 1
    if len(x1)-1 != 0 and x1[i-1] != x[i]:
        countSeconds += 1
    i += 1
    x2.append(countSeconds)

count_x3 = 0
i = 0
countPackages = 0 # contador de pacotes
countSeconds = 0  # contador segundos
while count_x > count_x3: # quantidade de pacotes que chegam por segundo
    if  i + 1 < count_x and x[i] == x[i+1]:
```

```
        if x3 == [] or x3[len(x3)-1] != x[i]:
            countSeconds += 1
            x3.append(x[i])
            x4.append(countSeconds)

        countPackages = countPackages + 2
        i = i + 2
        count_x3 = count_x3 + 2

        if i > count_x - 1 or x[i-2] != x[i] and i < count_x:
            y3.append(countPackages)
            countPackages = 0
    else:
        if x3 == [] or x3[len(x3)-1] != x[i]:
            countSeconds += 1
            x3.append(x[i])
            x4.append(countSeconds)

        y3.append(countPackages + 1)
        countPackages = 0
        i = i + 1
        count_x3 = count_x3 + 1

fig, (ax1, ax2) = plt.subplots(2, 1) #grafico de latencias por tempo

ax1.plot(x2,y1) # Dados para cada subplot
ax2.plot(x2,y2)

ax1.set(title='Média da Latência com o Tempo', xlabel='Tempo(s)', ylabel='Latê
ncia(s)')
ax2.set(title='Latência com o Tempo', xlabel='Tempo(s)', ylabel='Latência(s)')
fig.tight_layout()

fig2, ax3 = plt.subplots(1, 1) # grafico de capotes por tempo

ax3.plot(x4,y3) #Dados para o subplot

ax3.set(title='Quantidade de Pacotes Recebidos por Tempo', xlabel='Tempo(s)',
ylabel='Número de Pacotes')


plt.show()
```

Fonte: Produzido pelo Autor (2021).

Código 4 - Código fonte do DDoS_switch_15v5 utilizado no controlador c0.

```
from ryu.base import app_manager
from ryu.app.simple_switch_15 import SimpleSwitch15
from ryu.ofproto import ofproto_v1_5
from ryu.ofproto import ether
from ryu.ofproto import inet
from ryu.controller.handler import set_ev_cls
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER

from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
from ryu.lib.packet import packet
from ryu.lib.packet import ipv4
from ryu.lib.packet import tcp

import time

class DDoS15(SimpleSwitch15):
    # from Parent import *
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        #super(DDoS15, self).__init__(*args, **kwargs)
        self.counter = 0
        self.start_time = time.time()

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler2(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        dpid = datapath.id

        # retrieve OpenFlow message data
        buffer_id = msg.buffer_id
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        data = msg.data

        # retrieve received packet fields
        pkt = packet.Packet(msg.data)
        eth_pkt = pkt.get_protocol(ethernet.ethernet)
        et_dst = eth_pkt.dst
        et_src = eth_pkt.src
        et_typ = eth_pkt.ethertype

        if et_typ == 0x0800:
            ip_pkt = pkt.get_protocol(ipv4.ipv4)
```

```
            ip_dst = ip_pkt.dst
            ip_src = ip_pkt.src
            proto = ip_pkt.proto

            if proto == 6:
                tcp_hdr = pkt.get_protocol(tcp.tcp)
                tcp_dst = tcp_hdr.dst_port
                tcp_src = tcp_hdr.src_port
                flags = tcp_hdr.bits # tcp_flags

                if ip_dst == '10.0.0.3' and tcp_dst == 5001 and flags == 2:
                    if self.counter <= 10: # and delta_time < 1:
                        match = parser.OFPMatch(in_port=1, eth_type=0x0800, ip
v4_dst='10.0.0.3', ip_proto = 6, tcp_dst = 5001, tcp_flags = 2)
                        hard_timeout = 0 # sem timeout
                        priority = 1
                        if dpid == 1:
                            actions = [parser.OFPActionOutput(4), parser.OFPAc
tionOutput(ofproto.OFPP_CONTROLLER)]
                            inst = [parser.OFPInstructionActions(ofproto.OFPIT
_APPLY_ACTIONS, actions)]
                            mod = parser.OFPFlowMod(datapath=datapath, buffer_
id=buffer_id, hard_timeout=hard_timeout, priority=priority, match=match, instr
uctions=inst)
                            datapath.send_msg(mod)
                        elif dpid == 2:
                            actions = [parser.OFPActionOutput(4), parser.OFPAc
tionOutput(ofproto.OFPP_CONTROLLER)]
                            inst = [parser.OFPInstructionActions(ofproto.OFPIT
_APPLY_ACTIONS, actions)]
                            mod = parser.OFPFlowMod(datapath=datapath, buffer_
id=buffer_id, hard_timeout=hard_timeout, priority=priority, match=match, instr
uctions=inst)
                            datapath.send_msg(mod)

                    elif self.counter > 10:
                        print('Mecanismo de Proteção')
                        self.stop_time = time.time()
                        tempo_do_BOOM = self.stop_time - self.start_time
                        print('BOMM!!',tempo_do_BOOM)
                        hard_timeout = 10
                        priority = 65535
                        if dpid == 1:
                            actions = []
                            inst = [parser.OFPInstructionActions(ofproto.OFPIT
_APPLY_ACTIONS, actions)]
                            mod = parser.OFPFlowMod(datapath=datapath, buffer_
id=buffer_id, hard_timeout=hard_timeout, priority=priority, instructions=inst)
```

```
                            datapath.send_msg(mod)
                        elif dpid == 2:
                            actions = []
                            inst = [parser.OFPInstructionActions(ofproto.OFPIT
_APPLY_ACTIONS, actions)]
                            mod = parser.OFPFlowMod(datapath=datapath, buffer_
id=buffer_id, hard_timeout=hard_timeout, priority=priority, instructions=inst)
                            datapath.send_msg(mod)

                    self.counter += 1
                    end_time = time.time()
                    delta_time = self.start_time - end_time
                    if delta_time > 1:
                        self.start_time = end_time
                        print(self.counter, ' SYNs por segundo')
                        self.counter = 0
                else:
                    pass
            else:
                pass
        else:
            pass
```

Fonte: Produzido pelo Autor (2021).

Código 5 - Código fonte do topo-7h-3s-1s0 utilizado para criar a topologia da rede.

```
from mininet.topo import Topo
from mininet.link import Link, TCLink

class MyTopo(Topo):
    def __init__(self):
        Topo.__init__(self)

        h1 = self.addHost('h1', ip='10.0.0.1') #hosts
        h2 = self.addHost('h2', ip='10.0.0.2')
        h3 = self.addHost('h3', ip='10.0.0.3')
        h4 = self.addHost('h4', ip='10.0.0.4')
        h5 = self.addHost('h5', ip='10.0.0.5')
        h6 = self.addHost('h6', ip='10.0.0.6')
        h7 = self.addHost('h7', ip='10.0.0.7')

        s1 = self.addSwitch( 's1', dpid='0000000000000001' ) #switches
```

```
        s2 = self.addSwitch( 's2', dpid='0000000000000002' )
        s3 = self.addSwitch( 's3', dpid='0000000000000003' )
        s0 = self.addSwitch( 's0', dpid='0000000000000004' )

        self.addLink( s1, h1, 1, 10) #links
        self.addLink( s1, h4, 2, 20)
        self.addLink( s1, h5, 3, 30)
        self.addLink( s2, h2, 1, 10)
        self.addLink( s2, h6, 2, 20)
        self.addLink( s2, h7, 3, 30)
        self.addLink( s3, h3, 1, 10)
        self.addLink( s1, s0, 4, 1)
        self.addLink( s2, s0, 4, 2)
        self.addLink( s3, s0, 4, 3)

topos = {'mytopo': MyTopo}
x = vars(MyTopo())
```

Fonte: Produzido pelo Autor (2021).