UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

COORDENAÇÃO DE CIÊNCIA DA COMPUTAÇÃO

ALLAINN CHRISTIAM JACINTO TAVARES

DETECÇÃO DE ATAQUES DDOS ATRAVÉS DA LÓGICA PARACONSISTENTE ANOTADA

SANTA HELENA

2021

ALLAINN CHRISTIAM JACINTO TAVARES

DETECÇÃO DE ATAQUES DDoS ATRAVÉS DA LÓGICA PARACONSISTENTE ANOTADA

DDoS Attack Detection Using Annotated Paraconsistent Logic

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador: Euclides Peres Farias Junior

SANTA HELENA

2021

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Santa Helena

ALLAINN CHRISTIAM JACINTO TAVARES

DETECÇÃO DE ATAQUES DDoS ATRAVÉS DA LÓGICA PARACONSISTENTE ANOTADA

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Segurança de redes de computadores.

Data da aprovação: 20 de Maio de 2021

Prof. Euclides Peres Farias Junior, Mestrado – Universidade Tecnológica Federal do Paraná

Profª. Michele Nogueira Lima, Doutorado – Universidade Federal de Minas Gerais

Prof. Franck Carlos Vélez Benito, Doutorado – Universidade Tecnológica Federal do Paraná

Dedicatória: Dedico este trabalho à minha família, pelos momentos de ausência, aos meus professores pelos ensinamentos e ao meu orientador por me guiar.

AGRADECIMENTOS

Primeiramente agradeço aos meus pais por me apoiarem nessa jornada. Mesmo que distantes, estiveram perto alimentando o meu sonho de ser um cientista da computação. Agradeço a minha irmã e aos meus sobrinhos por me incentivarem a continuar para que um dia eu possa estar por perto. Agradeço aos meus antigos(as) e novos(as) amigos(as) que ouviram meus desabafos, me aturaram nos momentos difíceis, me trouxeram momentos felizes e me deram força para levantar e continuar essa caminhada que me trouxe até aqui. Não posso deixar de agradecer todos os meus professores que trouxeram a sabedoria e o conhecimento, que me apoiaram e me incentivaram a querer aprender mais e mais, que me fascinaram com as teorias e práticas e que me desafiaram a descobrir novos "mundos" e "universos". Sou realmente grato aos professores que tive dentro e fora de sala de aula. Quero agradecer a um professor em especial, o Prof. Me. Euclides P. F. Junior, que me deu uma oportunidade em um momento difícil, depositou a sua confiança e me guiou no decorrer do curso. Por fim, agradeço ao Sr. Clemente por sempre receber com um sorriso caloroso os alunos na portaria da universidade.

"Não há fatos eternos, como não há
verdades absolutas."

Friedrich Wilhelm Nietzsche

RESUMO

TAVARES, Allainn C. J. Detecção de ataque DDoS através da lógica paraconsistente anotada. 2021. 121f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Tecnológica Federal do Paraná. Santa Helena.

O desenvolvimento computacional inovou a rede de comunicação, de tal modo criou-se a rede de computadores. A Internet (rede mundial de computadores) é a maior rede de computadores, no qual permite a troca de mensagens em tempo real. Com o aumento de dispositivos conectados a Internet , na qual na maioria das vezes os mesmos não têm em seus interiores a implementação essencial de segurança, que acarreta em vulnerabilidades suscetíveis a invasões para se formar uma *botnet* com o intuito de realizar um ataque de Negação de Serviço Distribuído (DDoS). Os ataques DDoS são divididos em categorias. Dentre as categorias encontra-se os ataques de inundação (do inglês, *flood*), no qual acarreta a paralisação do serviço ao esgotar os recursos de rede/transporte ou os recursos do servidor. Existem diversos tipos de ataque DDoS, dentre eles destaca-se o ataque do tipo SYN (*Synchronize*) *flood*. A liberdade na pesquisa científica de buscar métodos e/ou recursos multidisciplinares para resolver problemas é uma das raízes e virtudes aplicáveis para a Ciência. Desta forma, a Lógica Paraconsistente (LP) foi uma das técnicas que despertou interesse no estudo da detecção de ataques, em função da sua característica e aplicação. Logo, este trabalho tem como objetivo provar a viabilidade do uso da Lógica Paraconsistente, especificamente a Lógica Paraconsistente Anotada (LPA), na segurança em redes de computadores para detectar ataques, com o foco em ataques DDoS do tipo SYN flood. Diante deste desafio, o presente trabalho apresenta a implementação do LPAProg-DDoS, de modo a contribuir com às áreas de Redes de Computadores e em especial a Segurança em Redes de computadores para tomada de decisão. O LPAProg-DDoS realiza a captura de dados da rede ou realiza a abertura de um arquivo de captura de rede, para então realizar o pré processamento dos dados de rede, calcular os graus de crenças e descrenças, computar com a LPA e exibir os resultados. Com a finalidade de realizar a comparação dos resultados, é apresentado a implementação do sistema FuzProg-DDoS, na qual difere do LPAProg-DDoS na etapa de computar, sendo que o FuzProg-DDoS faz o uso da Lógica *Fuzzy*. Os testes são realizados sobre a base de dados *Bot IoT* e de um cenário real criado na UTFPR. Os sistemas LPAProg-DDoS e FuzProg-DDoS obtiveram os resultados satisfatórios na base de dados *Bot IoT* para todas as métricas de avaliação. No cenário real, ambos os sistemas conseguem detectar os ataques DDoS do tipo SYN *flood* com a acurácia geral superior a 80%, mas nos testes realizados o FuzProg-DDoS obteve melhor assertividade. Desta forma, de acordo com os resultados obtidos, em um estágio considerado ainda preliminar, foi possível constatar que a LPA tem muito potencial de aplicação prática em segurança de redes.

Palavras-chave: Segurança de Redes e Computadores. Detecção de ataques DDoS. Lógica Paraconsistente..

ABSTRACT

TAVARES, Allainn C. J. DDoS Attack Detection Using Annotated Paraconsistent Logic. 2021. 121p. Work of Conclusion Course (Graduation in Computer Science) – Federal Technol-ogyUniversity – Paraná. Santa Helena.

Computational development has innovated the communications network, which is why the computer network was created. The Internet (global computer network) is the largest computer network, in which it allows the exchange of messages in real time. With the increase of devices connected to the Internet, in which most of the time they do not have the essential security implementation inside, which generates vulnerabilities susceptible to invasion to form a botnet in order to carry out a Distributed Denial Attack Service (DDoS). DDoS attacks are divided into categories. Among the categories are flood attacks, which causes the service to stop when network / transport resources or server resources are exhausted. There are several types of DDoS attacks, among them the SYN (Synchronize) flood attack. The freedom in scientific research to seek multidisciplinary methods and / or resources to solve problems is one of the roots and virtues applicable to science. Thus, Paraconsistent Logic (LP) was one of the techniques that aroused interest in the study of attack detection, depending on its characteristic and application. Therefore, this work aims to verify the viability of using Paraconsistent Logic, specifically Annotated Paraconsistent Logic (LPA), in security in computer networks for the detection of attacks, with a focus on SYN flood-type DDoS attacks. Faced with this challenge, this paper presents the implementation of LPAProg-DDoS, in order to contribute to the areas of Computer Networks and in particular Computer Network Security for decision-making. LPAProg-DDoS captures network data or opens a network capture file, then pre-processes the network data, calculates degrees of belief and disbelief, calculates with the LPA, and displays the results. In order to compare the results, the implementation of the FuzProg-DDoS system is presented, in which it differs from LPAProg-DDoS in the computational step, with FuzProg-DDoS making use of Fuzzy Logic. The tests are carried out in the Bot IoT database and in a real scenario created in UTFPR. The LPAProg-DDoS and FuzProg-DDoS systems performed satisfactorily in the Bot IoT database for all evaluation metrics. In the real world, both systems are capable of detecting SYN flood type DDoS attacks with an overall accuracy greater than 80%, but in the tests carried out, FuzProg-DDoS obtained better assertiveness. Thus, according to the results obtained, in a stage considered still preliminary, it was found that the LPA has a lot of potential for practical application in network security.

Keywords: Computer and network security. DDoS attack detection. Paraconsistent Logic.

LISTA DE ILUSTRAÇÕES

1 Estrutura de camadas utilizado pelo TCP/IP. 28

2 Encapsulamento de pacote TCP/IP. 29

3 Estrutura cabeçalho TCP. 29

4 *three-way handshaking* usado para criar uma conexão TCP. 31

5 *three-way handshaking* usado para encerrar uma conexão TCP. . . 32

6 Estrutura do ataque DDoS 36

7 Diferentes tipos de ataques DDoS 37

8 Ataque SYN *Flood* 39

9 Sistema básico LPA 41

10 Quadro QUPC com $N_{exig} = 0,60$ 41

11 Funções de pertinência para a variável equação 46

12 Sistema de inferência *Fuzzy* 48

13 Exemplo de inferência a partir da nota dada ao serviço do garçom 49

14 Exemplo de inferência da nota dada a qualidade da comida 49

15 Exemplo de deffuzzificação com o método centroide *Fuzzy* 50

16 Fases do trabalho. 55

17 Componentes do módulo LPAProg. 56

18 Exemplo da transformação da operação lógica em RPN. 60

19 Exemplo do *log* gerado pelo módulo LPAProg. 62

20 Exemplo do arquivo de saída gerado pelo módulo LPAProg. . . . 62

21 Exemplo do QUPC gerado pelo módulo LPAProg. 63

22 Diagrama de sequência do módulo LPAProg. 64
23 Dados utilizados para comparação. 65
24 QUPC utilizado para comparação. 65
25 Componentes do módulo LPAProg-DDoS. 67
26 Arquitetura do sistema LPAProg-DDoS. 68
27 Exemplo de representação de pacote no *Scapy*. 70
28 Exemplo do arquivo CSV com a contagem. 71
29 Componentes do módulo FuzProg-DDoS. 78
30 Exemplo do conjunto de entrada com o nível de exigência igual a 0,65. 79
31 Exemplo do conjunto de saída com o nível de exigência igual a 0,65. 80
32 Exemplo da saída *deffuzificada* com o nível de exigência igual a 0,65 e o resultado tende a ser tráfego normal. 82
33 Exemplo da saída *deffuzificada* com o nível de exigência igual a 0,65 e o resultado tende a ser ataque. 83
34 Ambiente de teste da base de dados Bot-IoT. 85
35 Ambiente de teste do cenário real. 87
36 Nova distribuição dos componentes do módulo LPAProg-DDoS. . 90
37 Nova distribuição dos componentes do módulo LPAProg-DDoS. . 91
38 Exemplo da matriz de confusão. 92
39 Gráfico de tempo das janelas geradas pelo sistema. 95
40 *Boxplot* da variação do tempo das janelas. 95
41 Gráfico de tempo gasto para abertura dos arquivos. 96

42 *Boxplot* da variação do tempo para abrir os arquivos. 96

43 Gráfico de tempo gasto para pré processar as janelas. 97

44 *Boxplot* da variação do tempo para pré processar as janelas. . . . 97

45 Gráfico de tempo gasto para calcular os graus das janelas. 98

46 *Boxplot* da variação do tempo para calcular os graus das janelas. 99

47 Gráfico de tempo gasto para computar e exibir os resultados das janelas. 100

48 *Boxplot* da variação do tempo para computar as janelas e exibir os resultados. 100

49 Alguns dos gráficos gerados pelo LPAProg-DDoS ao ser executado na base de dados *Bot IoT* 101

50 Alguns dos gráficos gerados pelo FuzProg-DDoS ao ser executado na base de dados *Bot IoT* 101

51 Gráfico de tempo gasto para capturar os pacotes e gerar as janelas. 104

52 Gráfico de tempo das janelas geradas pelo sistema. 104

53 Gráfico de tempo gasto para pré processar as janelas. 105

54 Gráfico de tempo das janelas geradas pelo sistema. 105

55 Gráfico de tempo gasto para calcular os graus das janelas. 106

56 *Boxplot* da variação do tempo para calcular os graus das janelas. 107

57 Gráfico de tempo das janelas geradas pelo sistema. 107

58 Gráfico de tempo das janelas geradas pelo sistema. 108

59 Alguns dos gráficos gerados pelo LPAProg-DDoS ao ser executado no cenário real 109

60 Alguns dos gráficos gerados pelo FuzProg-DDoS ao ser executado no cenário real 110

LISTA DE TABELAS E QUADROS

1 Tabela de Regiões 43

2 Tabela de dependências do LPAProg-DDoS 66

3 Descrição das variáveis do cálculo de obtenção dos graus. 74

4 Tabela de dependências do FuzProg-DDoS 77

5 Descrição dos intervalos de tempo da base de dados do cenário real. 88

6 Matriz de confusão da saída dos sistemas LPAProg-DDoS e FuzProg-DDoS 102

7 Resultados dos sistemas LPAProg-DDoS e FuzProg-DDoS 102

8 Matriz de confusão da saída do teste com o sistema LPAProg-DDoS 109

9 Resultados do teste com o sistema LPAProg-DDoS 109

10 Matriz de confusão da saída do teste com o sistema FuzProg-DDoS 110

11 Resultados do teste com o sistema FuzProg-DDoS 110

LISTA DE ABREVIATURAS E SIGLAS

ACK	Acknowledgement
DoS	Denial of Service
DDoS	Distributed Denial of Service
FIN	Finish
HD	Hard Disc
HTTP	Hypertext Transfer Protocol
IA	Inteligência Artificial
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
LAN	Local Area Network
LP	Lógica Paraconsistente
LPA	Lógica Paraconsistente Anotada
MATA	Modified Adaptive Threshold algorithm
PSH	Push
QUPC	Quadro Unitário de Plano Cartesiano
RAM	Random Access Memory
RPN	Reverse Polish Notation
RST	Reset
SCTP	Stream Control Transmission Protocol
SDN	Software Defined Network
SIP	Session Initiation Protocol
SYN	Synchronization

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URG	Urgent
WAN	Wide Area Network
WWW	World Wide Web

LISTA DE SÍMBOLOS

μ_1	Grau de crença
μ_2	Grau de descrença
λ_1	Grau de crença
λ_2	Grau de descrença
μ_{1R}	Grau de crença resultante
μ_{2R}	Grau de descrença resultante
G_{contr}	Grau de contradição
H_{cert}	Grau de certeza
N_{exig}	Nível de exigência
W	Baricentro
V	Verdade
F	Falso
$\top$	Inconsistência
$\bot$	Paracompleteza ou Indeterminação
NOT	Negação
OR	Disjunção
AND	Conjunção
max	Máximo
min	Mínimo
S	Quantidade de pacotes que tem somente a *flag* SYN ativa
SA	Quantidade de pacotes que tem somente as *flag* SYN e a *flag* ACK ativas
F	Quantidade de pacotes que tem a *flag* FIN ativa

A	Quantidade de pacotes que tem a *flag* ACK ativa
R	Quantidade de pacotes que tem a *flag* FIN ativa
P	Quantidade de pacotes que tem a *flag* PSH ativa
TCP	Quantidade de pacotes do que são do protocolo TCP
d	Quantidade de dígitos do valor TCP

SUMÁRIO

1 INTRODUÇÃO **19**

1.1 OBJETIVOS 21

1.1.1 Geral 21

1.1.2 Específicos 21

1.2 CONTRIBUIÇÕES DO TRABALHO 21

1.3 JUSTIFICATIVA 22

1.4 DELIMITAÇÕES DO TRABALHO 23

2 REVISÃO DA LITERATURA **24**

2.1 COMUNICAÇÃO DE DADOS 24

2.2 REDES DE COMPUTADORES 26

2.3 SEGURANÇA DE REDES E COMPUTADORES 32

2.3.1 Ataques DDoS 35

2.4 LÓGICA PARACONSISTENTE 39

2.5 LÓGICA FUZZY 44

2.6 ESTADO DA ARTE 50

3 METODOLOGIA **54**

3.1 PROPOSTA DO TRABALHO 55

3.2 LPAProg 55

3.3 LPAProg-DDoS 66

3.3.1 Coleta e pré-processamento dos dados 68

3.3.2 Calcular os graus de crenças e descrenças 71
3.3.3 Computar com a LPA e exibir os resultados 75
3.4 FUZPROG-DDOS 76
3.5 AMBIENTES DE TESTES 83
3.5.1 Ferramentas 83
3.5.2 Base de dados Bot-IoT 85
3.5.3 Cenário Real 86
3.5.4 Aplicação dos Testes 89
3.6 MELHORIAS NOS SISTEMAS 89

4 ANÁLISE DE RESULTADOS 92

4.1 BASE DE DADOS BOT-IoT 94
4.2 CENÁRIO REAL 103

5 CONSIDERAÇÕES FINAIS 112

1 INTRODUÇÃO

O desenvolvimento computacional inovou a rede de comunicação, de tal modo criou-se a rede de computadores. As redes de computadores são formadas por dispositivos interconectados com a finalidade de comunicar e trocar informações entre si (TANENBAUM, 2011). A Internet — rede mundial de computadores — é a maior rede de computadores, no qual permite a troca de mensagens em tempo real — como texto, voz ou vídeo — entre qualquer indivíduo que esteja conectado. Além disso, a Internet permite acessar diversos serviços como: rede social, compra, venda, editor de texto, busca, videoconferência, programas de gerenciamento, dentre outros.

Os dispositivos conectados em redes de computadores, na maioria das vezes, são equipamentos computacionais autônomos que são controlados e gerenciados com o auxílio do sistema operacional (TANENBAUM, 2011). De tal modo, é possível realizar o controle de um equipamento de forma remota, ou seja, à distância. No entanto, esta conectividade sugere que haja segurança para que terceiros (atacantes) não autorizados tenham acesso ao dispositivo. Portanto, um usuário mal intencionado pode utilizar de métodos para descobrir vulnerabilidades de um sistema com finalidades ilícitas, como roubar informações, invadir o equipamento ou efetuar ações capazes de trazer a indisponibilidade dos serviços.

Dentre os tipos de ataques existentes, um dos que se despontam e até o momento não há uma solução definitiva, é o ataque denominado Negação de Serviço (do inglês, *Denial of Service* – DoS). Este ataque tem como finalidade inibir que a informação do remetente chegue ao destinatário ou impedir que o destinatário leia a informação, de tal modo que o destinatário não possa responder. Para tal, o atacante utiliza-se de um dispositivo para enviar mensagens falsas de modo a congestionar o meio de comunicação ou sobrecarregar o processamento da vítima. Com o propósito de intensificar o ataque, atacantes invadem dispositivos — vítimas secundárias — de modo a controlá-los e utilizá-los para realizar diversos ataques DoS à vítima principal. Tal ataque é conhecido como Negação de Serviço Distribuído (do inglês, *Distributed Denial of Service* – DDoS).

No entanto, os ataques DDoS são divididos em categorias. Dentre as categorias encontra-se os ataques de inundação (do inglês, *flood*), no qual acarreta a paralisação do serviço ao esgotar os recursos de rede/transporte ou os recursos do servidor. Existem diversos tipos de ataque DDoS, dentre eles destaca-se o ataque do tipo SYN (*Synchronize*) *flood*, o qual explora uma vulnerabilidade do processo de estabelecimento de conexão do protocolo TCP/IP (DA SILVA, 2018).

De acordo com o Centro de Estudos, Respostas e Tratamento de Incidentes de Segurança no Brasil (CERT.br), o ataque DDoS ocupou a segunda posição de incidentes reportados a ela no ano de 2019 (CERT.BR, 2020). Além disso, de acordo com Kupreev, Badovskaya e Gutnikov (2019) o ataque DDoS tipo SYN *flood* obteve a primeira posição, com 82,43%, entre os tipos de ataques DDoS utilizados no segundo trimestre de 2019. No mais, os ataques DDoS acarretam diversos prejuízos às empresas, no qual podem ser financeiro, econômico, jurídico, de confiabilidade, dentre outros. Em alguns casos o prejuízo pode envolver também os usuários. Com isso, é de suma importância detectá-lo e impedi-lo antes de intensificar a perda. Deste modo, pesquisadores estão trabalhando pra desenvolver métodos que auxiliam os administradores de rede a detectar ataques DDoS. Há diversas técnicas utilizadas para detecção, como: aprendizado de máquina (DOSHI; APTHORPE; FEAMSTER, 2018), rede neural (LIU, 2020), mineração de dados (BHAYA; EBADYMANAA, 2017), teoria dos grafos (FERREIRA; NOGUEIRA, 2018), série temporal (DA SILVA, 2018), lógica *fuzzy* (FALCÃO, 2019), análise da quantificação da recorrência (RIGHI, 2017), dentre outras.

A liberdade na pesquisa científica de buscar métodos e/ou recursos multidisciplinares para resolver problemas é uma das raízes e virtudes aplicáveis para a Ciência. Desta forma, a Lógica Paraconsistente (LP) foi uma das técnicas que despertou interesse no estudo da detecção de ataques, em função da sua característica e aplicação. Pois a LP é uma lógica não clássica fundamentada na revogação do princípio da **Não Contradição** (SILVA FILHO, 2010). Esta lógica admite o tratamento de informações contraditórias na sua estrutura teórica, sem trivialização. Entre as classes da lógica paraconsistente encontra-se a Lógica Paraconsistente Anotada (LPA), na qual utiliza dos sinais de graus de crença e descrença de uma proposição para obter

o resultado (SILVA FILHO, 2010). Pelo fato da detecção de ataque DDoS ser um problema não trivial e com intuito de auxiliar nas pequisas, este trabalho utiliza da LPA para detectar ataque DDoS. Além disso, este trabalho apresenta um método de detecção de ataque DDoS do tipo SYN *flood* através da Lógica *Fuzzy*, com o intuito de comparar os resultados com o método que utiliza a LPA.

1.1 OBJETIVOS

1.1.1 Geral

Provar a viabilidade do uso da Lógica Paraconsistente na segurança em redes de computadores para detectar ataques.

1.1.2 Específicos

- Detectar ataques DoS e DDoS do tipo SYN *Flood*;
- implementar um *framework* da LPA;
- desenvolver o módulo de detecção de ataques DDoS;
- criar topologias de redes;
- criar ambientes de ataques para as topologias criadas;
- simular tráfego de rede legítimo e de ataque;
- aplicar o *framework* em diversos tipos de redes;
- avaliar os resultados através de métodos estatísticos.

1.2 CONTRIBUIÇÕES DO TRABALHO

Este trabalho propõe diversas contribuições para às áreas de Redes de Computadores e em especial a Segurança em Redes de computadores, na qual a principal

contribuição advém da exploração da LPA aplicada na detecção de ataques e na tomada de decisão para o gerenciamento da segurança de redes. As contribuições secundárias, tratam da possibilidade da construção de um *framework* para detecção de ataques DDoS, bem como a aplicação em redes de diversas topologias e tipos, por exemplo: SDN, Redes Vanets, aplicações em redes IoT, dentre outras. Outra contribuição, decorre das aplicações inteligentes que exigem tomada de decisão, pois a lógica paraconsistente é eficiente neste processo.

Desta forma, este trabalho tem a característica de conhecimento aprofundado na arquitetura de redes e seus protocolos, como TCP/IP, UDP, sistemas operacionais, redes e segurança de redes, de forma que este conhecimento proverá melhor usabilidade das principais técnicas de coleta de dados e monitoramento dos ativos de redes, o que força o domínio completo em gerenciamento e segurança de redes. Além disso, concede a oportunidade de trabalhos futuros aplicados em segurança e gerenciamento de redes, aplicação e exploração da LPA na área de segurança de redes de forma inteligente.

1.3 JUSTIFICATIVA

A segurança em redes está cada vez mais em evidência, em especial os ataques DDoS que estão se tornando cada vez mais poderosos em função dos recursos de comunicação distribuída conforme descrito no CERT.br (2020). Além disso, ataques desta natureza acarretam prejuízos financeiro, econômico, científico, jurídico e até mesmo há imagem de uma empresa. Desta forma, a necessidade de se propor novas tecnologias ao combate aos crimes virtuais, faz com que a comunidade científica busque em outras áreas do conhecimento, possíveis soluções para auxiliar ou aprimorar sistemas inteligentes que possam suportar e combater de forma eficiente os ataques e crimes cibernéticos, como é o caso da LPA, uma vez que faz parte das chamadas lógicas não clássicas, pois contém disposições contrárias a alguns dos princípios básicos da Lógica Aristotélica (ABE, 2013). Outra justificativa plausível para este trabalho é a finalidade acadêmica, no qual promove o estudo aprofundado das disciplinas de redes e segurança de computadores, cuja a solução proposta está

alinhada à inovação científica e tecnológica.

1.4 DELIMITAÇÕES DO TRABALHO

Este trabalho tem a finalidade de desenvolver uma aplicação baseada na LPA aplicada na segurança de redes de computadores. Desta forma, uma vez que a segurança de redes é uma área muito vasta, este trabalho centra-se, exclusivamente, na detecção de ataque de negação de serviço de forma individual e distribuída (DoS e DDoS). Em função de existir diversos tipos de ataque DDoS, este trabalho centra-se especificamente nos ataques do tipo *SYN Flood*, empregados no protocolo TCP/IP.

2 REVISÃO DA LITERATURA

Neste capítulo estão presentes os principais conceitos e fundamentos abrangendo: comunicação de dados, redes de computadores, segurança em computadores, segurança em redes, ataque de negação de serviço distribuídos e a Lógica paraconsistente. Tais itens servem como base para o entendimento dos métodos utilizados para realizar a detecção de ataque DDoS com o LPAProg-DDoS e o FuzProg-DDoS. A seção 2.1 apresenta a definição e os princípios para ocorrer a comunicação de dados. A seção 2.2 aborda a base do funcionamento da redes de computadores com os protocolos utilizados para que se possa ter uma comunicação entre os dispositivos. A seção 2.3 descreve os conceitos de segurança, abrange as formas de ataques DDoS e aprofunda no ataque DDoS do tipo SYN *Flood*. A seção 2.4 denota a LP e como é o processo de obtenção e validação do resultado. A seção 2.5 descreve os conceitos teóricos sobre a Lógica *Fuzzy* e como é obtido e validado os resultados a partir dos dados de entradas. Por fim, a seção 2.6 apresenta os trabalhos considerados estado da arte sobre a detecção de ataques DDoS.

2.1 COMUNICAÇÃO DE DADOS

A comunicação é a troca de informações entre os indivíduos, de modo que ela possa ocorrer de forma local ou remota (FOROUZAN, 2009). A diferença entre a comunicação local e remota é a distância que a mesma é realizada, na qual a comunicação local ocorre normalmente frente a frente, enquanto a comunicação remota acontece a distância.

Dados se refere a registros, que em conjunto pode gerar uma informação. Há diversas formas de representação dos dados, na qual entre as principais estão: textos, números, imagens, áudio e vídeo.

A comunicação de dados é considerada como sendo a troca de dados entre dois dispositivos por um meio de transmissão. Para que a comunicação de dados ocorra é necessário quatro elementos fundamentais: entrega, precisão, sincronização e *jitter*

(FOROUZAN, 2009). Na entrega o sistema deve garantir que os dados sejam entregues para o dispositivo ou usuário de destino correto. Na precisão o sistema deve realizar a entrega dos dados sem alterações, de modo que se os dados forem alterados eles se tornam inúteis. Na sincronização o sistema deve garantir que os dados sejam entregues no tempo desejado, ou seja, que não aconteça atraso na entrega, no qual um atraso pode tornar os dados inúteis. E no *jitter*[1] o sistema deve ser capaz de garantir intervalos de tempo equidistantes entre as entregas dos dados.

A comunicação de dados é realizada a partir de cinco componentes: mensagem, emissor, receptor, meio de transmissão e protocolo (FOROUZAN, 2009). A mensagem é o conjunto de dados que o dispositivo emissor quer repassar para o dispositivo receptor, no qual é necessário um meio de transmissão (por exemplo cabo de par trançado, fibra ótica, *Wi-Fi*) e uma forma de protocolar a mensagem para que o receptor receba-a e seja capaz de abri-la, interpretá-la e entendê-la. De forma análoga a entrega de uma carta do remetente ao destinatário por meio de uma empresa que utiliza de um transporte e as rodovias para que a carta selada chegue ao destinatário.

O protocolo é a regra que o remetente, o destinatário e todos os dispositivos intermediários precisam seguir para que seja possível se comunicarem de forma efetiva entre si (FOROUZAN, 2009).

Para que um sistema de comunicação de dados troquem informações entre si é preciso primeiramente preparar as informações da fonte e transmitir por um meio físico, então as informações são extraídas para serem entregues (COMER, 2016). No qual é necessário uma fonte de informação para que se tenha a representação das informações de forma digital ou analógica.

A comunicação realizada por um sistema pode ocorrer a partir de várias fontes de envio ou destino, ou seja, um dispositivo pode enviar dados para diversos dispositivos que estejam conectados por um meio de transmissão. Logo um usuário também pode receber informações de diversos indivíduos. As fontes de informações vão além dos periféricos computacionais como teclados e mouses, mas também incluem microfones, câmeras de vídeo, sensores e dispositivos de medição (COMER, 2016).

[1] Medida de variação do atraso entre os pacotes.

Uma rede de comunicação é considerada um ambiente no qual um grupo de dispositivos, *links* de comunicação e pacotes de *software* permitem às pessoas e equipamentos trocarem informações entre si e entre outros dispositivos(DANTAS, 2010). Com a evolução computacional e a necessidade de comunicação entre equipamentos foi possível a criação das redes de computadores. Do qual, a fusão entre computadores e a comunicação modificou a organização dos sistemas computacionais (TANENBAUM, 2011).

2.2 REDES DE COMPUTADORES

Redes de computadores é um grupo de dispositivos autônomos interconectados por uma única tecnologia, o qual deve haver a necessidade da troca de informações entre dois ou mais dispositivos (TANENBAUM, 2011). Em uma rede de computadores os dispositivos podem ser considerados como um dispositivo final (*host*) ou um dispositivo de conexão. Os *hosts* são os responsáveis em preparar a mensagem (informações) e enviar para um *host* destinatário. Alguns exemplos de *hosts* são os computadores, *smarthphones*, *notebooks* e servidores. Para que se possa comunicar diversos *hosts*, ou até mesmo comunicar diversas redes, pode-se utilizar os dispositivos de conexão. Estes dispositivos auxiliam na entrega dos pacotes. Entre os principais dispositivos de conexão estão: os *switchs*, os *hubs* e os roteadores.

O pacote em uma rede de computadores é uma estrutura utilizada para que a informação possa ser trafegada da origem para o destino. Na troca de informações, as mesmas são quebradas em diversos pacotes para então serem transmitidas. Além da mensagem, um pacote é composto pelos cabeçalhos que são encapsulados no momento de envio. Os cabeçalhos recebem informações como o endereço de destino, o endereço de origem, a porta de origem, o protocolo utilizado e entre outras (FOROUZAN, 2009).

As redes de computadores podem ser categorizadas pelo seu tamanho e distância de alcance. Entre as mais comuns estão a rede local (do inglês *Local Area Network* – LAN) que conecta alguns *hosts* em um mesmo local e a rede de longa distância

(do inglês *Wide Area Network* – WAN) que conceta dispositivos localizados geograficamente em locais distintos entre si. Quando há conexão entre duas redes ou mais, pode-se simplificar como uma *internetwork* ou internet. A internet é uma rede comutada por se tratar de uma combinação de enlaces e *switches*, no qual um *switch* é conectado em pelo menos dois enlaces (FOROUZAN; MOSHARRAF, 2013).

A Internet é a rede das redes, na qual é a maior rede existente que conecta o mundo. A rede mundial de computadores, como também é conhecida a Internet, permite que qualquer usuário faça parte dela, necessitando que o mesmo esteja conectado fisicamente a um provedor de serviço Internet (do inglês, *Internet Service Provider* – ISP) (FOROUZAN; MOSHARRAF, 2013). De tal modo, a Internet permite que diversos dispositivos se comuniquem e troquem informações entre si, sendo que um dispositivo conectado à Internet pode ser visualizado por outros indivíduos conectados.

Para a criação da Internet foi preciso desenvolver um conjunto de padrões para que fosse possível realizar a interconexão entre diversas tecnologias de comutação de pacotes com a finalidade do funcionamento global. Com isso, foi proposto um modelo para definir um conjunto de protocolos de comunicação entre computadores em rede, denominado TCP/IP (COMER, 2016) (FOROUZAN, 2009). Este conjunto vem de dois protocolos: Protocolo de Controle de Transmissão (do inglês *Transmission Control Protocol* – TCP) e Protocolo de Internet (do inglês *Internet Protocol* – IP).

O TCP/IP é constituído por um conjunto de protocolos, no qual é possível distinguir os protocolos por hierarquia de nível superior ou inferior. Tal modelo, abrange os protocolos em camadas, na qual originalmente foram definidas quatro camadas: interface de rede, internet, transporte e aplicação (FOROUZAN; MOSHARRAF, 2013) (COMER, 2016). A Figura 1 mostra uma representação do TCP/IP.

A camada interface de rede é responsável em especificar os detalhes do meio de transmissão e do hardware associado na comunicação. Além disso, na camada de interface de rede, localiza-se os detalhes relativos a comunicação, como o endereço físico do dispositivo, o tamanho máximo do pacote que a rede pode suportar e os protocolos usados para acessar o meio (COMER, 2016). Na camada de internet

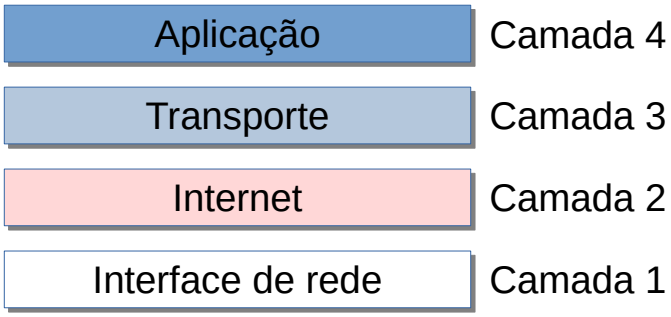

Figura 1: Estrutura de camadas utilizado pelo TCP/IP.
Fonte: Adaptado de (FOROUZAN; MOSHARRAF, 2013)

encontra-se o protocolo IP, na qual fica responsável em realizar a ligação entre redes (FOROUZAN, 2009). Além disso, a camada de internet utiliza de um endereço IP para identificar a rede e o dispositivo que encontra-se nesta rede. Na camada de transporte localiza-se os protocolos TCP, UDP (do inglês *User Datagram Protocol*) e SCTP (do inglês *Stream Control Transmission Protocol*), na qual são utilizados para viabilizar a comunicação de uma aplicação de um dispositivo com a aplicação em outro dispositivo (COMER, 2016). Ou seja, está camada é responsável em prover serviços para a camada de aplicação (FOROUZAN; MOSHARRAF, 2013). A última camada do TCP/IP, a camada de aplicação, define como um par de aplicações interagem na comunicação (COMER, 2016). Nesta camada encontra-se os dados que são trafegados. Este trabalho concentra-se na camada de transporte, principalmente no protocolo TCP.

Um pacote é formado pelos cabeçalhos que são encapsulados ao passar em cada camada, iniciando da camada de aplicação até a camada de enlace. Ao ser enviado o pacote, o destinatário realiza o desencapsulamento do pacote iniciando da camada de enlace e obtendo os dados na camada de aplicação. A figura 2 exemplifica como é um processo de encapsulamento e desencapsulamento de um pacote em cada camada.

A camada de transporte tem a finalidade de garantir uma comunicação processo a processo, ou seja, além do pacote ser enviado e recebido por um dispositivo final, o protocolo da camada de transporte deve garantir que a mensagem seja entregue ao processo correto (FOROUZAN; MOSHARRAF, 2013).

[2]Disponível em: <http://infotecnews.com.br/modelo-tcpip/>. Acesso em: 22 set. 2020

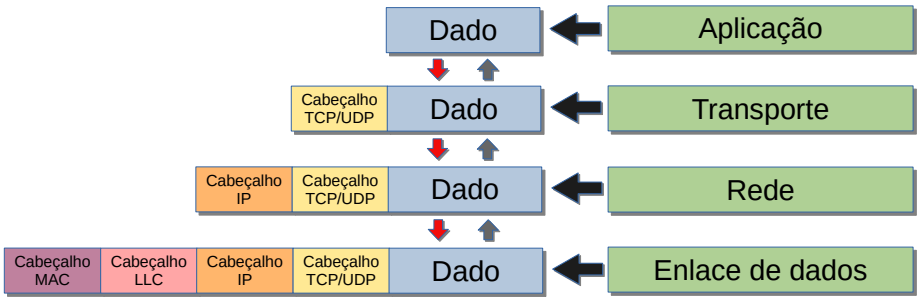

Figura 2: Encapsulamento de pacote TCP/IP.
Fonte: Adaptado de publicação na InfoTec News[2]

O protocolo TCP é o principal protocolo da camada de transporte utilizado na Internet, no qual é responsável em estabelecer a conexão lógica entre as camadas de transportes dos dispositivos antes de transferir os dados e também garantir uma transferência confiável (FOROUZAN; MOSHARRAF, 2013) (COMER, 2016). Além disso, o protocolo TCP compensa a perda, o atraso, a duplicação e a entrega de pacotes fora de ordem, de tal modo que não haja sobrecarga na rede e nos roteadores (COMER, 2016). O cabeçalho do pacote com o protocolo TCP utiliza de uma estrutura (Figura 3) que contém: endereço da porta de origem, endereço da porta de destino, número de sequência, número de confirmação, comprimento do cabeçalho (HLEN), controle, tamanho da janela, soma de verificação, ponteiro de urgência e opções. Para este trabalho, é importe destacar somente o campo de controle, que serão fundamentais na metodologia proposta para detecção de ataque DDoS.

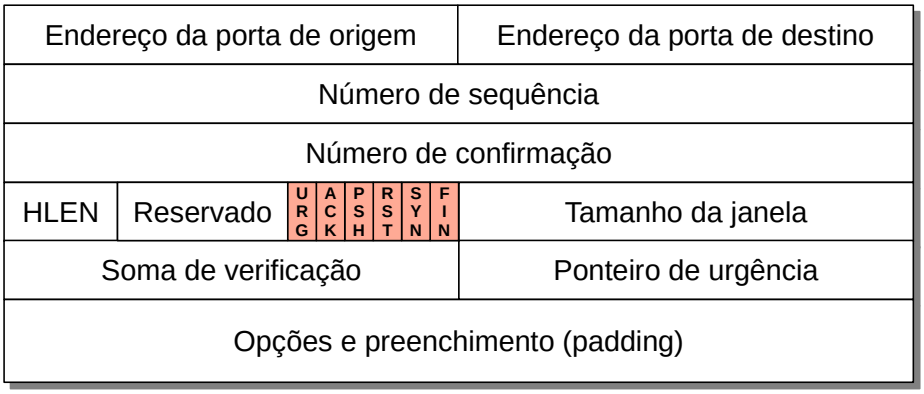

Figura 3: Estrutura cabeçalho TCP.
Fonte: Adaptado de (FOROUZAN; MOSHARRAF, 2013)

O endereço da porta de origem é um campo que define o número da porta do aplicativo no momento do envio do pacote. O endereço da porta de destino é um

campo que define o número da porta do aplicativo que receberá o pacote. O número de sequência representa a sequência dos pacotes que trafegam para o destinatário. Número de confirmação é um campo que define um número esperado pelo receptor para confirmar a troca de mensagem. O comprimento de cabeçalho indica o número de palavras no cabeçalho TCP. O campo tamanho da janela define o tamanho da janela do TCP para a ser enviada. A soma de verificação é um campo que contém um valor calculado e usado para verificar a integridade dos dados transmitidos. O ponteiro de urgência valida se o pacote é ou não urgente. O campo opções contém informações opcionais e adicionais no cabeçalho TCP.

A área destacada na figura 3 representa os diferentes *bits* de controle. Este campo tem como finalidade definir os marcadores (*flags*) que são utilizados no controle da conexão. As *flags* permitem que a conexão seja estabelecida, finalizada, cancelada ou mesmo definir o modo de conexão como transferência de dados (FOROUZAN; MOSHARRAF, 2013). Cada *flag* tem sua representação, exemplificando: URG (do inglês *urgent*) representa que o ponteiro de urgência é válido; ACK (do inglês *acknowledgement*) que o valor do campo de confirmação é válido; PSH (do inglês *push*) realiza o pedido de empurrar os dados; RST (do inglês *reset*) reiniciar a conexão; SYN (do inglês *synchronization*) sincronizar os números de sequência durante a conexão; e FIN (do inglês *finish*) finalizar a conexão (FOROUZAN, 2009).

O protocolo TCP é orientado a conexão, por isso estabelece um caminho lógico entre a origem e o destino. Deste modo, é acessível realizar os processos de confirmação e retransmissão de pacotes danificados ou perdidos. Além disso, na transmissão orientada à conexão é ordenada em três passos: estabelecer a conexão, transferir os dados e finalizar a conexão (FOROUZAN; MOSHARRAF, 2013).

Para que um dispositivo de origem (cliente) envie mensagem TCP para um de destino (servidor) é preciso primeiramente estabelecer a conexão. Do qual, é imprescindível que o dispositivo de destino aprove a troca de dados com o dispositivo de origem de forma mútua. Para isso, o TCP utiliza da apresentação em três vias (*three-way handshaking*) (COMER, 2016).

Para que seja possível realizar uma conexão TCP, o servidor deve informar ao

TCP que está pronto para aceitar uma conexão. Deste modo, o *three-way handshaking* utiliza das *flags* SYN e ACK para iniciar uma comunicação entre o cliente e o servidor. A conexão ocorre com o cliente enviando um pacote com a *flag* SYN ativa para o servidor, indicando que deseja realizar uma conexão. Então o servidor responde com um pacote com as *flags* SYN e ACK ativas, indicando a sincronização da comunicação com o cliente e a confirmação que foi recebido o primeiro pacote com sucesso. Por fim, o cliente responde ao servidor um pacote com a *flag* ACK confirmando o recebimento do pacote (COMER, 2016) (FOROUZAN; MOSHARRAF, 2013). Após as três etapas os dispositivos podem trocar mensagens entre si. A figura 4 exemplifica os passos realizados pelo *three-way handshaking* para iniciar a conexão TCP entre o cliente e o servidor.

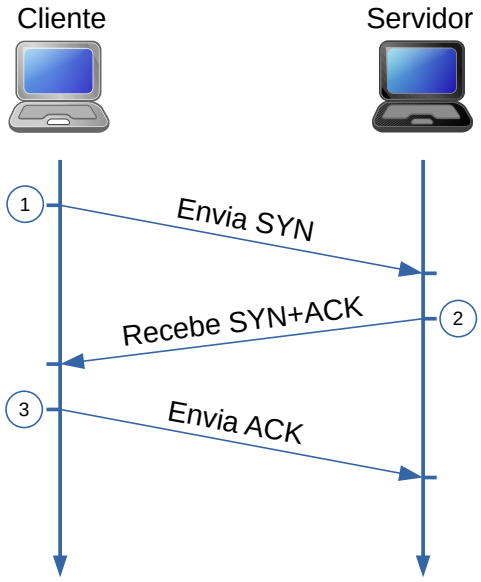

Figura 4: *three-way handshaking* usado para criar uma conexão TCP.
Fonte: Adaptado de (FOROUZAN, 2009)

O *three-way handshaking* também é utilizado para finalizar a conexão. A diferença entre iniciar e finalizar uma conexão é que em vez de utilizar a *flag* SYN, utiliza-se a *flag* FIN. Para isso, o cliente envia para o servidor um pacote com a *flag* FIN ativa, desejando que a conexão seja encerrada. O servidor responde confirmando o recebimento do pacote com as *flags* FIN e ACK ativas, indicando que a conexão pode ser finalizada. Por fim, o cliente envia para o servidor um pacote com a *flag* ACK ativa confirmando o encerramento da conexão (FOROUZAN; MOSHARRAF, 2013). A figura 5 exemplifica os passos realizados pelo *three-way handshaking* para

encerrar a conexão TCP entre o cliente e o servidor.

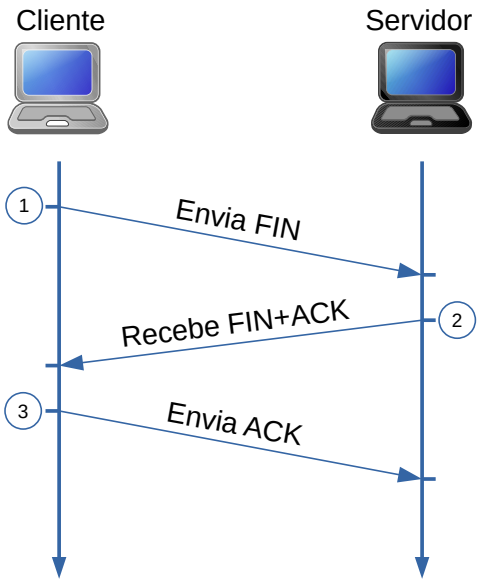

Figura 5: *three-way handshaking* usado para encerrar uma conexão TCP.
Fonte: Adaptado de (FOROUZAN, 2009)

Os fundamentos apresentados nesta seção são base para entender o funcionamento de uma conexão TCP entre dois dispositivos conectados a uma rede. As redes de computadores é uma área extensa, este trabalho se conteve em descrever os principais aspectos que foram utilizados para alcançar os objetivos. Além disso, outros aspectos fogem do escopo da metodologia deste trabalho.

A comunicação em redes não é uma tarefa trivial, uma vez que há a necessidade de administrar e configurar diversos dispositivos interconectados de forma que se possa garantir desempenho e performance da rede. Porém, se faz necessário atender os aspectos de segurança, pois como os dados e as informações são trafegadas e distribuídas não se pode aceitar uma infraestrutura de rede sem uma segurança mínima nas informações.

2.3 SEGURANÇA DE REDES E COMPUTADORES

De acordo com Stallings (2015) a segurança de computadores está relacionada com o conceito de confiabilidade, tendo como objetivo preservar a confidencialidade, integridade e disponibilidade de um sistema informatizado. Entretanto, Guttman e

Roback (1995) definem a segurança de computadores como:

> *Computer Security: The protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications).*[3](GUTTMAN; ROBACK, 1995, p. 5)

A confidencialidade é a garantia que os dados não serão acessados por indivíduos não autorizados, de modo que o sistema seja capaz de controlar os indivíduos que podem coletar e armazenar informações e também diferenciar os indivíduos que podem visualizar as informações.

A integridade é a garantia que os dados não serão alterados por indivíduos não autorizados. Ou seja, o sistema deve ser capaz de desempenhar a função de definir os indivíduos que podem manipular os dados.

A disponibilidade é a garantia que o sistema funcionará prontamente sem a negação de serviço a usuários autorizados. Ou seja, a disponibilidade deve "assegurar que o acesso e o uso das informações seja confiável e realizado no tempo adequado" (BROWN; STALLINGS, 2017, p. 8). A perda de disponibilidade de um serviço pode acarretar prejuízos, da qual, o usuário legítimo é impedido de ter acesso às informações, ou mesmo, ao sistema de informação.

A segurança computacional classifica os níveis de exigência de disponibilidade de um sistema ou serviço. Na qual, um sistema que fornece serviços de autenticação para sistemas, aplicações e dispositivos críticos se enquadra em uma exigência alta da disponibilidade do sistema. Como por exemplo serviços que são acessados por clientes e funcionários para a realização de tarefas críticas. De tal modo, a indisponibilidade de tal serviço pode acarretar grande perda financeira. Além de perda de produtividade dos empregados, também pode acarretar potencial perda de clientes

[3]Segurança de Computadores: A proteção atribuída a um sistema de informação automatizada para alcançar os objetivos apropriados de preservação da integridade, disponibilidade e confidencialidade dos recursos do sistema de informação (inclui *hardware*, *software*, *firmware*, informação/-dados e telecomunicações) (tradução nossa).

(BROWN; STALLINGS, 2017).

Entre as terminologias utilizadas em segurança de computadores, destacam: atacante, ameaça, ataque, contramedida, risco e vulnerabilidade. Na qual, o atacante é um indivíduo que ataca um sistema; a ameaça é uma circunstância, capacidade, ação ou evento que pode vir a infringir a segurança e causar dano, de tal modo que a ameaça é considerada um perigo possível que poderia explorar uma vulnerabilidade; o ataque é a tentativa de violação da segurança do sistema, de modo que são utilizadas técnicas para burlar os serviços de segurança e violar a política de segurança de um sistema; a contramedida é uma ação tomada para reduzir uma ameaça, uma vulnerabilidade ou um ataque, de modo a eliminar, prevenir ou minimizar os danos; o risco é a probabilidade da perda da segurança ao se utilizar uma ameaça para explorar as vulnerabilidades e causar dano; a vulnerabilidade é a falha, defeito ou fraqueza de um sistema que poderia ser explorada para violar a política de segurança do sistema (SHIREY, 2000).

A vulnerabilidade de um sistema pode acarretar em sua indisponibilidade ou torná-lo muito lento, de modo a se tornar impossível ou impraticável o seu uso. As ameças são capazes de explorar as vulnerabilidades. Com isso, pode executar uma ameaça – um ataque – para violar o sistema. Os ataques são divididos em ativo e passivo. De modo que o ativo realiza a tentativa de alterar ativos de sistemas ou afetar sua operação. Em contrapartida, o ataque passivo realiza a tentativa de descobrir ou fazer uso de informações proveniente do sistema, de modo que não afeta seu hardware, software ou sua rede de comunicação. (BROWN; STALLINGS, 2017).

As contramedidas são medidas utilizadas para lidar com um ataque à segurança. Nem sempre é possível impedir o sucesso de um ataque, mas pode-se utilizar como meta a detecção do ataque e recuperação do sistema. As contramedidas não garantem a resolução das vulnerabilidades (BROWN; STALLINGS, 2017).

Entre as consequências de ameaças informadas por Shirey (2000), encontra-se a disrupção. Esta consequência é diretamente ligada a disponibilidade, no qual interrompe ou impede a operação correta de serviços e funções de sistema. Entre

os ataques que ocasiona a disrupção encontra-se a obstrução, que trata-se de um ataque que interrompe a entrega de serviços, atrapalhando a operação do sistema (SHIREY, 2000).

Existem dois modos para obstruir a operação do sistema, no qual o primeiro trata-se de incapacitar os dispositivos de conexão, de modo a impedir a comunicação entre os usuários e o serviço. O segundo trata-se de realizar uma sobrecarga no serviço com a finalidade de congestioná-lo com um tráfego intenso de comunicação ou processamento (BROWN; STALLINGS, 2017). A sobrecarga pode deixar o serviço lento ou impedir que os seus usuários possam usufruir do sistema.

Desta forma, pode-se assumir que a segurança é a área responsável por proteger um sistema das ameaças e vulnerabilidades que as redes de computadores possam ter. Para tanto, é necessário que especialistas em segurança tenham a capacidade e a oportunidade de construir ambientes seguros e resilientes capazes de detectar e mitigar possíveis ataques no ambiente computacional (KUROSE; ROSS, 2013).

2.3.1 Ataques DDoS

O ataque DDoS é uma forma distribuída do ataque negação de serviço (DoS). O DoS inunda um dispositivo final (normalmente um servidor Web) com um fluxo intenso de pacotes (COMER, 2016). O ataque tem como finalidade consumir os recursos da vítima para que o sistema fique lento, ou mesmo impeça, o usuário a usufruir do serviço como um site, serviço Web, sistema de computador, entre outros. Este é um ataque de disrupção contra a disponibilidade do sistema.

O DDoS utiliza de um massivo conjunto de dispositivos conectados a rede para realizar envios de fluxo de pacotes para o servidor – vítima. Neste tipo de ataque há dois grupos de vítima: a secundária e a principal. As vítimas secundarias são os dispositivos utilizados para a realização do ataque. Normalmente, o atacante precisa assumir o controle de dispositivos na rede, com um software malicioso, para então realizar o ataque. O conjunto de vítimas secundárias é denominado *botnet*. A vítima principal é a que recebe todo o fluxo da *botnet*, de tal modo o atacante

não envia pacotes diretamente para a vítima principal (COMER, 2016). A figura 6 exemplifica a estrutura do ataque DDoS.

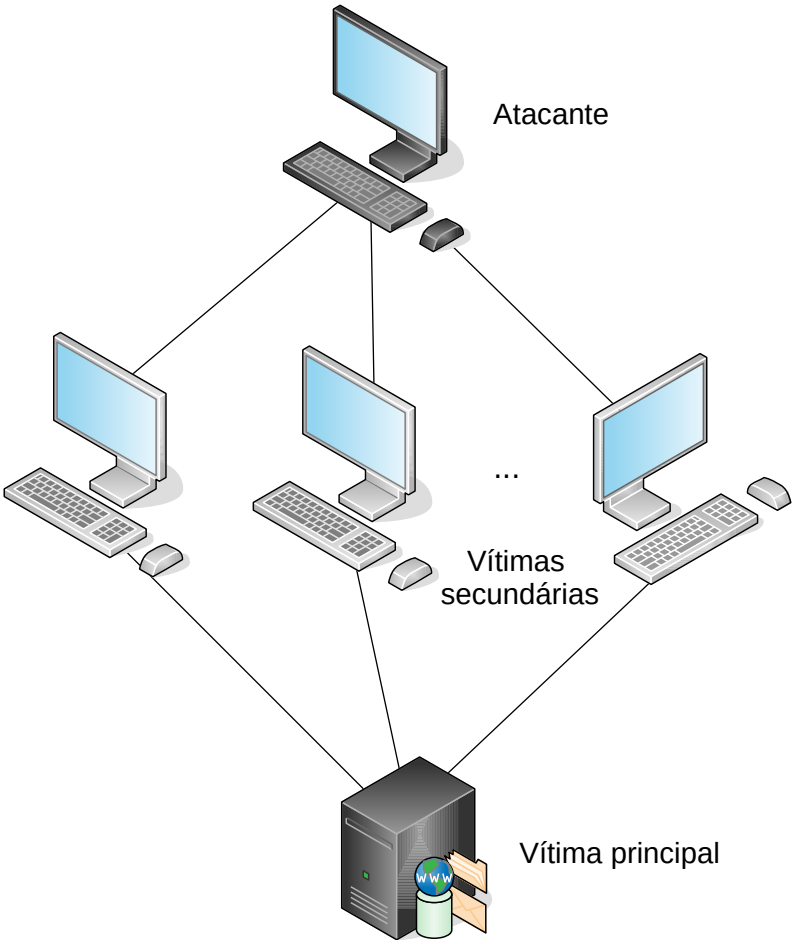

Figura 6: Estrutura do ataque DDoS
Fonte: Adaptado de (MAHJABIN et al., 2017)

O ataque DDoS é dividido em três fases: o recrutamento das vítimas secundárias, a propagação e o ataque. A primeira fase do ataque é gerar a *botnet*, de modo que o atacante utiliza de programas de autopropagação para infectar os dispositivos. A próxima fase é propagar o código do ataque para os dispositivos que foram comprometidos. A última fase é tentar o ataque de modo a prejudicar a vítima principal (MAHJABIN et al., 2017).

Existem várias tipos de ataque DDoS, com isso foram criadas categorias para se enquadrar cada tipo de ataque (MAHJABIN et al., 2017). A figura 7 mostra as categorias e os diferentes tipos de ataque DDoS.

Os tipos de ataques DDoS primeiramente podem ser divididos em ataques de

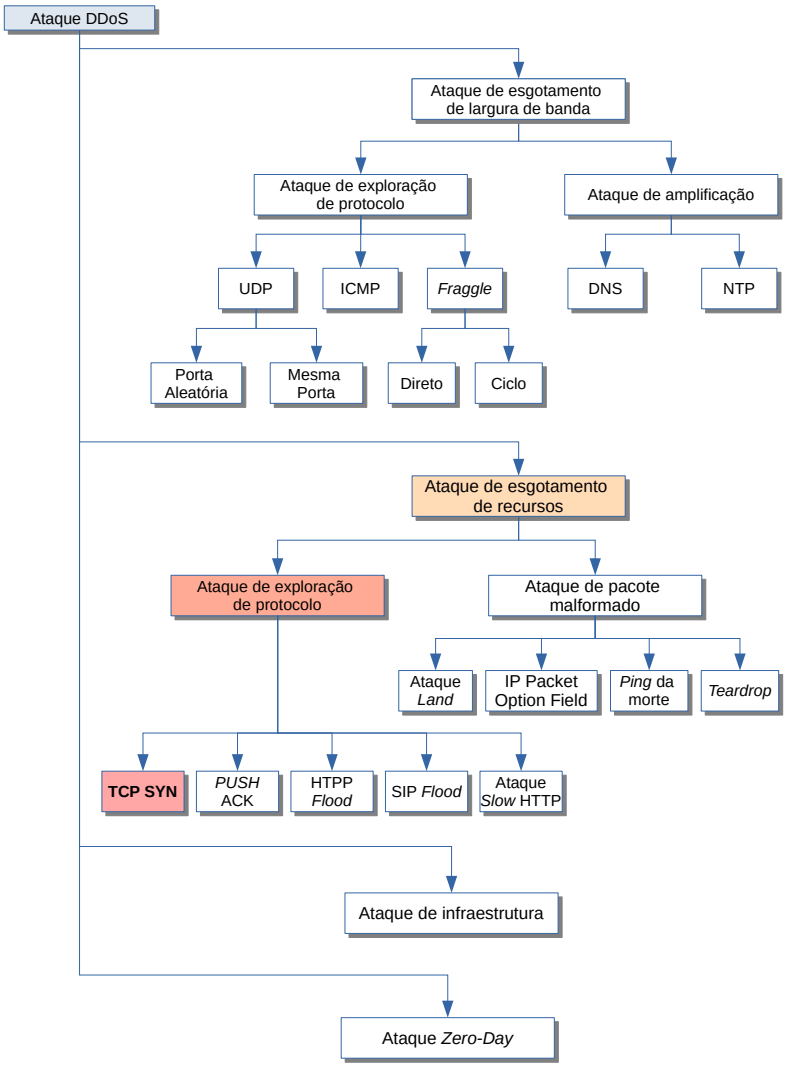

Figura 7: Diferentes tipos de ataques DDoS
Fonte: Adaptado de (MAHJABIN et al., 2017)

esgotamento de largura de banda ou em esgotamento de recursos. De tal modo que o primeiro tem como objetivo congestionar a rede da vítima para que usuários não consigam acessar a rede. Tal ataque afeta também outros sistemas que são dependentes da rede. O ataque de esgotamento de recursos tem a finalidade de congestionar a memória e a unidade central de processamento (do inglês *Central Processing Unit* – CPU) do dispositivo. Com isso, a vítima principal não pode responder as solicitações dos usuários legítimos. Quando há um ataque que esgota a largura de banda e os recursos, ele é denominado ataque de infraestrutura. Os ataques que tem seus impactos desconhecidos são denominados ataque *zero-day* (MAHJABIN et al., 2017). Desta forma, o escopo deste trabalho é sobre os ataques

de esgotamento de recursos.

Entre os principais recursos afetados pelo ataque de esgotamento de recursos estão a memória, os soquetes e a CPU. Existem duas maneirar de realizar este tipo de ataque: ataque de exploração de protocolo e ataque de pacote malformado. Na primeira o atacante explora as vulnerabilidades dos protocolos de rede da camada de transporte e de aplicação. No segundo, como o próprio nome diz, são utilizados pacotes malformados para projetar um ataque (MAHJABIN et al., 2017). Este trabalho investiga os ataques de exploração de protocolos.

Os ataques de exploração de protocolos abrangem quatro tipo de ataques: TCP SYN, *PUSH* ACK, HTTP (*hypertext transfer protocol*) *Flood* (inundação), SIP (*session initiation protocol*) *Flood* e ataque *slow* HTTP. Entre os tipos de ataques apresentados, o trabalho concentra-se no TCP SYN, também conhecido como SYN *flood*.

2.3.1.1 SYN Flood

O ataque SYN *flood* é uma técnica utilizada para bloquear o serviço TCP de um determinado dispositivo (COMER, 2016). Nesse ataque, o atacante utiliza do protocolo de Internet TCP/IP para sobrecarregar o dispositivo da vítima com solicitações SYN (DONG; ABBAS; JAIN, 2019). Ou seja, o atacante tenta enviar vários pacotes contínuos com a finalidade de evitar que o servidor feche a conexão (MALLIKARJUNAN; MUTHUPRIYA; SHALINIE, 2016). De tal modo, várias conexões mantêm-se no processo de abertura de conexão, com isso são consumidos os recursos da vítima, como memória e CPU. Logo, há a dificuldade dos usuários legítimos de estabilizar a conexão com a vítima e a vítima responder as solicitações legítimas.

Este tipo de ataque explora o mecanismo *three-way handshaking* utilizado pelo protocolo TCP para estabelecer uma conexão. Neste processo, o servidor armazena nas pilhas de memória todos os estados intermediários até estabelecer a conexão ou esgotar o tempo limite. De tal modo, o atacante explora esse recurso e inunda a memória da vítima, na qual as solicitações de conexão de usuários legítimos podem ser rejeitadas (MAHJABIN et al., 2017).

Para que o ataque funcione, o atacante envia várias requisições de abertura de conexão, pacote com a flag SYN ativa, e não confirma a abertura de conexão, não enviando o pacote com a flag ACK ativa. Deste modo, o atacante não completa o processo de *handshaking*, entretanto cria-se um grande número de conexões incompletas (MAHJABIN et al., 2017). Para aumentar a chance de sucesso do ataque, atacantes falsificam o endereço de IP de origem para um endereço não existente. De tal modo, a vítima ficará no aguardo da resposta de um dispositivo que não se encontra na rede. Tal técnica é denominada *spoofed* SYN *flood* (MALLIKARJUNAN; MUTHUPRIYA; SHALINIE, 2016). A figura 8 apresenta um exemplo de ataque DDoS SYN *Flood* em um servidor *Web*. Neste exemplo, pode-se observar que não há respostas aos SYN/ACK, no qual esta ação provoca o esgotamento do sistema, forçando então ações corretivas e/ou a disponibilidade do servidor.

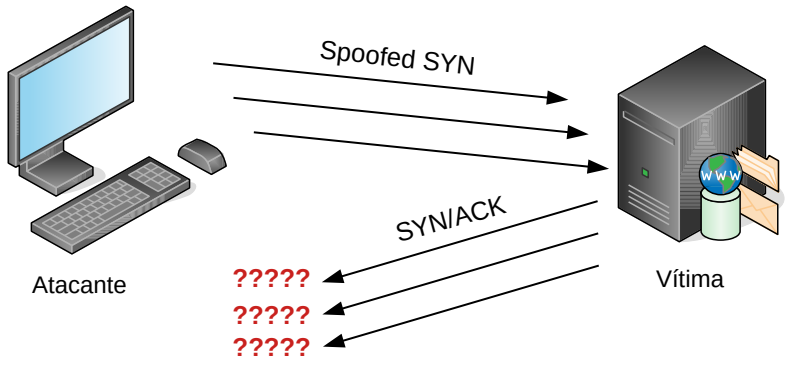

Figura 8: Ataque SYN *Flood*
Fonte: Adaptado de (MAHJABIN et al., 2017)

Nesta seção foi possível descrever algumas informações pertinentes aos comportamentos e formas de ataques do tipo de negação de serviço, bem como ter-se uma ideia do que esta técnica de ataque pode comprometer um ambiente.

2.4 LÓGICA PARACONSISTENTE

A lógica paraconsistente (LP) é uma lógica não clássica heterodoxa das teorias inconsistentes, mas não trivial. Criada de modo independente pelo polonês Stanislaw Jaskowski e o brasileiro Newton C. A. da Costa (DA COSTA; KRAUSE; BUENO, 2007). De tal modo que, um sistema de lógica é chamada de paraconsistente se

puder ser empregado como subjacente a teorias inconsistentes, porém não triviais. Esta lógica é fundamentada na revogação do princípio da Não Contradição (SILVA FILHO, 2010). Na qual, admite o tratamento de informações contraditórias na sua estrutura teórica, sem trivialização.

As LPs são divididas nas classes: lógica deôntica paraconsistente, lógica para inconsistência, lógica discursiva, lógica formal inconsistente e a lógica paraconsistente anotada (LPA). Neste trabalho tratamos apenas a classe da LPA. As outras classes fogem do escopo metodológico do trabalho.

A LPA utiliza os graus de crença (μ_1) e descrença (μ_2) para representar os sinais de entradas. No qual, estes graus variam no intervalo real de 0 a 1 e podem ser obtidos a partir de medições, meios estatísticos ou probabilísticos (NETO; VENSON, 2002). Além disso, os graus representam os sinais lógicos, no qual o grau de crença representa o quanto de veracidade se tem sobre uma determinada preposição, por outro lado, o grau de descrença representa o grau de falsidade da preposição.

Ao analisar os valores dos graus de crença e descrença com a análise paraconsistente, é dado como saída um estado lógico (Figura 9), que pode ser representada em sua forma mais simples:

- **(1;0)** - representa **Verdade** (**V**), a crença total e nenhuma descrença;
- **(0;1)** - representa **Falso** (**F**), nenhuma crença e descrença total;
- **(1;1)** - representa **Inconsistência** ($\top$), ao mesmo tempo a crença e descrença total;
- **(0;0)** - representa **Paracompleteza** ou **Indeterminação** ($\bot$), ausência total de crença e de descrença.

De modo a gerar uma representação visual, a LPA utiliza de um quadro reticulado intitulado Quadro Unitário de Plano Cartesiano (QUPC). O QUPC apresenta valores X e Y variando num intervalo real fechado [0,1], de modo que estes valores representam respectivamente os graus de crença, μ_1 e de descrença, μ_2. O

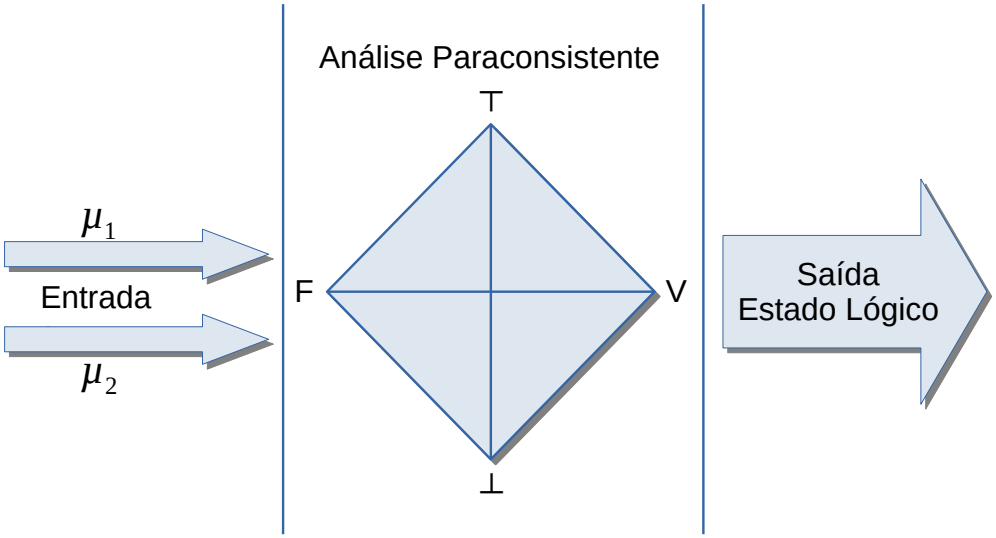

Figura 9: Sistema básico LPA
Fonte: Adaptado de (NETO; VENSON, 2002)

QUPC é dividido em doze regiões, como mostra a figura 10. Além disso, a análise paraconsistente pode ser feita através do QUPC (NETO; VENSON, 2002).

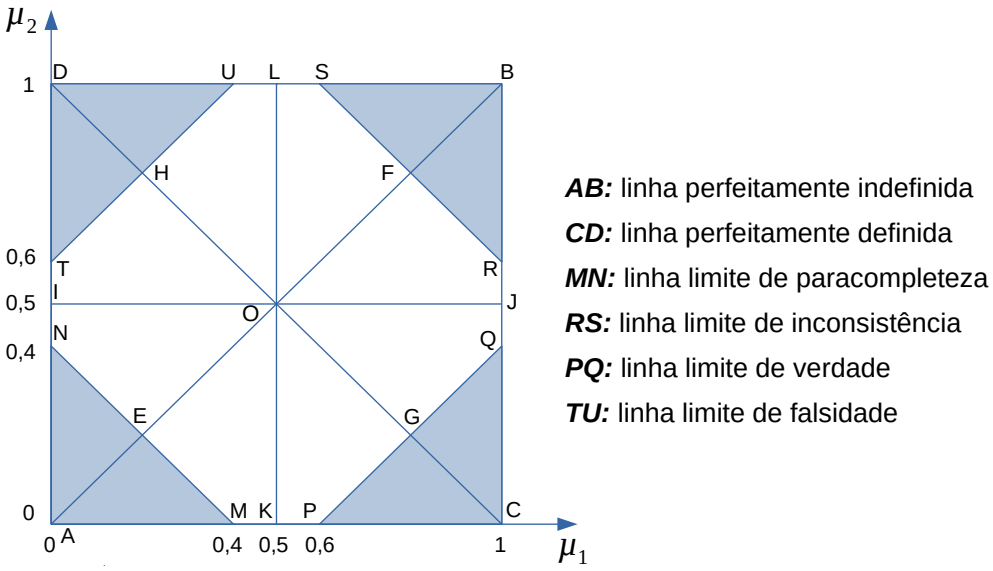

Figura 10: Quadro QUPC com $N_{exig} = 0,60$
Fonte: Adaptado de (CARVALHO; BRUNSTEIN; ABE, 2003)

A linha de limite de paracompleteza e de inconsistência são representadas por z_1, enquanto a linha de limite de falsidade e de verdade são representadas por z_2. Convencionalmente, é adotada a notação $z_1 = z_2 = z$ de forma a gerar uma simetria ao gráfico, como mostra a Figura 10, na qual $z_1 = z_2 = z = 0,60$. Desta forma, o valor de z_2 é chamado de nível de exigência (N_{exig}). Considera z_2 como o nível de

exigência por ser limite de regiões de decisões. Por este motivo, são destacadas na Figura 10 as quatro regiões extremas e uma região central.

As regiões CPQ e DTU são chamadas de regiões de decisões, sendo CPQ uma decisão favorável (ou viabilidade) e o DTU uma decisão desfavorável (ou inviabilidade). Partindo deste princípio, os pontos no plano cartesiano MNTUSRQP representam uma única região que não permite tomada de decisão, ou seja, quando o ponto que traduz o resultado da análise pertence a essa região diz-se que a análise é não conclusiva. Apesar disso, esta região pode ser dividida em regiões menores, na qual essas regiões correspondem a respostas de quase tendendo à, como exemplo a região OFSL que representa quase inconsistência tendendo à falsidade. É importante ressaltar que, se o resultado estiver na região BRS (região de inconsistência), a análise é não conclusiva quanto à viabilidade, mas acusa um alto grau de inconsistência dos dados. Analogamente, se estiver na região AMN (de paracompleteza), significa que os dados apresentam um alto grau de indeterminação. A Tabela 1 apresenta o intervalo de cada região dentro do quadrante QUPC.

A análise paraconsistente pode ocorrer a partir do grau de contradição e do grau de certeza, de modo que é possível calculá-los a partir dos graus de crença e descrença resultantes. De tal modo, o grau de contradição (G_{contr}) pode ser obtido a partir da equação $G_{contr} = \mu_1 + \mu_2 - 1$. E o grau de certeza ($H_{cert}$) a partir da equação $H_{cert} = \mu_1 - \mu_2$. Ambos os resultados variam no intervalo [-1, 1].

A LPA suporta analisar um conjunto de graus de crenças e descrenças, para isso é necessário obter um grau de crença e de descrença resultante (μ_{1R} e μ_{2R} respectivamente). Pelo fato da LPA ser uma lógica, ela também dispõe dos operadores lógicos NOT, OR e AND. A partir deste operadores, ao aplicá-los no conjunto de dados de entrada, é possível obter apenas um grau de crença e descrença, na qual, estes valores são os graus resultantes.

O operador NOT realiza a negação dos graus de crença e descrença, de modo que dado a entrada os valores são invertidos entre eles, como: $NOT(\mu_1; \mu_2) = (\mu_2; \mu_1)$. Com isso, pode-se observar que: $NOT(\top) = \top$, $NOT(\bot) = \bot$, $NOT(V) = F$ e $NOT(F) = V$.

O OR é um operador de disjunção clássica, ou seja, dado dois valores o resultado é o maior entre eles. A LPA resolve o OR da seguinte forma: $(\mu_1; \mu_2)\ OR\ (\lambda_1; \lambda_2) = (max\{\mu_1; \lambda_1\}; max\{\mu_2; \lambda_2\})$.

Por fim, o AND realiza a conjunção entre os graus, ou seja, escolhe o menor valor entre eles. $(\mu_1; \mu_2)\ AND\ (\lambda_1; \lambda_2) = (min\{\mu_1; \lambda_1\}; min\{\mu_2; \lambda_2\})$.

Região	Condicional	Resultante
AMN	$-1 \leq G_{contr} \leq -N_{exig}$	Paracompleteza (ou Indeterminação) ($\bot$)
BRS	$N_{exig} \leq G_{contr} \leq 1$	Inconsistência ($\top$)
CPQ	$N_{exig} \leq H_{cert} \leq 1$	Verdade (V)
DTU	$-1 \leq H_{cert} \leq N_{exig}$	Falsidade (F)
OFSL	$0 \leq G_{contr} < N_{exig}$ e $-0,5 \leq H_{cert} < 0$	Quase inconsistência tendendo à falsidade ($Q\top \rightarrow F$)
OHUL	$0 \leq G_{contr} < 0,5$ e $-N_{exig} < H_{cert} < 0$	Quase falsidade tendendo à inconsistência ($QF \rightarrow \top$)
OHTI	$-0,5 \leq G_{contr} < 0$ e $-N_{exig} < H_{cert} < 0$	Quase falsidade tendendo à paracompleteza ($QF \rightarrow \bot$)
OENI	$-N_{exig} < G_{contr} < 0$ e $-0,5 < H_{cert} < 0$	Quase paracompleteza tendendo à falsidade ($Q\bot \rightarrow F$)
OEMK	$-N_{exig} < G_{contr} < 0$ e $0 \leq H_{cert} < 0,5$	Quase paracompleteza tendendo à verdade ($Q\bot \rightarrow V$)
OGPK	$-0,5 \leq G_{contr} < 0$ e $0 \leq H_{cert} < N_{exig}$	Quase verdade tendendo à paracompleteza ($QV \rightarrow \bot$)
OGQJ	$0 \leq G_{contr} < 0,5$ e $0 \leq H_{cert} < N_{exig}$	Quase verdade tendendo à inconsistência ($QV \rightarrow \top$)
OFRJ	$0 \leq G_{contr} < N_{exig}$ e $0 \leq H_{cert} < 0,5$	Quase inconsistência tendendo à verdade ($Q\top \rightarrow V$)

Tabela 1: Tabela de Regiões
Fonte: Adaptado de (SILVA FILHO, 2010)

Além destas indicações dentro do quadro reticulado faz-se uso do indicador denominado baricentro (W). O baricentro é a média ponderada dos graus resultantes, sendo obtido através da soma ponderada dos graus de crença resultantes e dividido pela soma dos pesos (Equação 2.1), no qual é realizado o mesmo processo para os graus de descrença resultantes (Equação 2.2). Assim obtêm-se os valores médios dos graus de crença e descrença resultantes. O baricentro tem como função a tomada de decisão final, sendo ele a influência conjunta de todas as análises realizadas.

$$W_1 = \frac{\sum_{i=1}^{n}(p_i \cdot \mu_{1i})}{\sum_{i=1}^{n} p_i} \tag{2.1}$$

$$W_2 = \frac{\sum_{i=1}^{n}(p_i \cdot \mu_{2i})}{\sum_{i=1}^{n} p_i} \tag{2.2}$$

A LPA avalia a entrada em vários quadrantes, além de tratar a verdade, falsidade, incerteza e a inconsistência, a LPA oferece os resultados internos, como por exemplo quase inconsistente tendendo à verdade. Com isso, é possível ter várias tomadas de decisões para os valores analisados. Por se tratar de uma lógica inconsistente, é relevante o estuda de sua aplicabilidade em problemas não triviais. Então este trabalho centra-se na usabilidade da LPA na segurança em redes de computadores.

2.5 LÓGICA FUZZY

A Lógica *Fuzzy* (Nebulosa) utiliza de métodos matemáticos para aproximar as inferências ao raciocínio humano, na qual leva em consideração o aspecto de incerteza (PONTES SARAIVA, 2006). Diferente da lógica clássica, que possibilita dois resultados distintos, como 0 ou 1, a *Fuzzy* por outro lado permite infinitos resultados em um intervalo entre 0 e 1. Com o objetivo de tratar informações de caráter impreciso ou vago, a Lógica *Fuzzy* utiliza da Teoria de Conjuntos *Fuzzy* concebida por L.A. Zadeh[4]. Na qual, de acordo com Tanscheit (2004), utiliza de ferramentas matemáticas para traduzir a informação imprecisa, expressa por um conjunto de regras linguísticas, em termos matemáticos que podem ser resolvidos por um computador.

O Conjunto *Fuzzy* utiliza dos graus de pertinência para definir o quanto um elemento é compatível com o conjunto, de tal modo, este elemento pode pertencer a mais de um conjunto *Fuzzy* com distintos graus de pertinência (TANSCHEIT, 2004). Já Simões e Shaw (2007) destacam que "A propriedade fundamental da Lógica *Fuzzy* é que a função de pertinência $\mu_A(x)$ tem todos os valores dentro do intervalo [0,1], um elemento pode ser membro parcialmente de um conjunto, indicado por um valor [...] dentro do intervalo numérico". A equação 2.3 apresenta como é o comportamento da Lógica *Fuzzy* perante a um conjunto A e um elemento x com relação a esses conjunto.

[4]Zadeh, L.A. (1965). "Fuzzy Sets". **Information and Control**, v. 8, p. 338-353.

$$\mu(x) = \begin{cases} 1 & \text{se, e somente se, } x \in A \\ 0 & \text{se, e somente se, } x \notin A \\ 0 \leq \mu(x) \leq 1 & \text{se } x \text{ pertence parcialmente a } A \end{cases} \tag{2.3}$$

Basicamente, os Conjuntos *Fuzzy* são designados por uma variável linguística, na qual são responsáveis em ditar o nome destes conjuntos. A variável linguística aproxima a Lógica *Fuzzy* ao pensamento humano, por proferir característica textual no lugar ou em adição a variáveis numéricas a um determinado conjunto. As variáveis linguísticas podem ter valores como: altura, peso, idade, temperatura, clima e entre outros. No interior destas variáveis podem haver diversos termos, como: baixo, médio e alto; jovem, adulto e idoso; quente, morno e frio; ensolarado, parcialmente nublado e nublado; e dentre outros (TANSCHEIT, 2004). Além disso, Marro et al. (2010) define a variável linguística "como uma entidade utilizada para representar de modo impreciso [...] um conceito ou uma varável de um dado problema".

A função de pertinência é responsável em definir a forma de cada conjunto dado pelas variáveis linguísticas e seus termos. Há diferentes formas de representar um Conjunto *Fuzzy*, entre as principais estão: triangular, trapezoidal, gaussiana e sino generalizado (TANSCHEIT, 2004). Este trabalho foca na representação triangular e trapezoidal, as demais fogem do escopo metodológico e não serão apresentadas.

A representação triangular é representada por um triângulo no gráfico, na qual o seu centro representa o ponto máximo de pertinência ao conjunto e os pontos extremos esquerdo e direito representam o valor mínimo. O trapezoidal, representado por um trapézio no gráfico, é definido por quatro pontos, dos quais, utilizam dos dois pontos centrais para referir ao valor máximo de pertinência, gerando uma reta entre eles. Em contrapartida, os dois outros pontos, o da direita e esquerda, infere o grau mínimo de pertinência (MARRO et al., 2010). A figura 11 apresenta um exemplo de funções de pertinência para a variável equação, na qual os termos baixo e alto utilizam a representação trapezoidal e o termo médio a triangular.

As operações lógicas *NOT*, *OR* e *AND* na Lógica *Fuzzy* são baseadas nas opera-

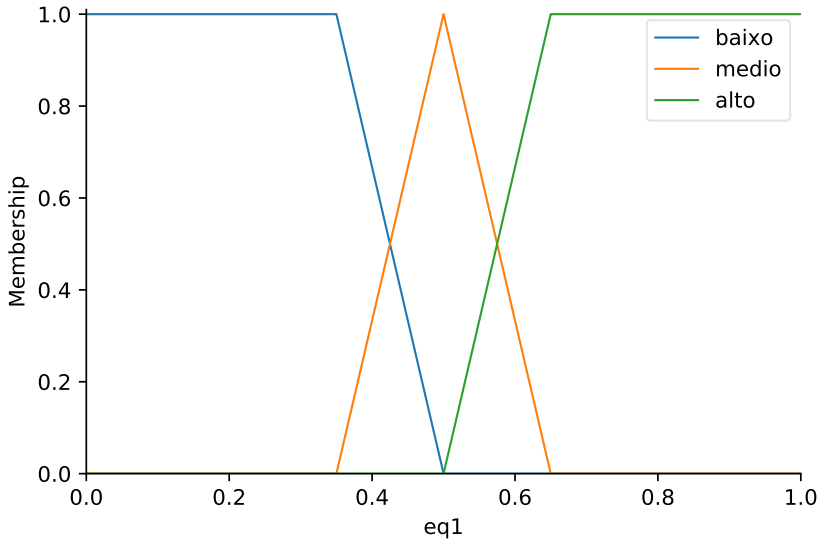

Figura 11: Funções de pertinência para a variável equação
Fonte: Autoria própria (2021)

ções da teoria dos conjuntos, no qual pode-se considerar o NOT como o complemento de um conjunto, o OR como a união entre dois conjuntos e o AND a interseção. Logo, para realizar o NOT em um determinado grau de pertinência basta subtrair pelo valor um (Equação 2.4). Tratando-se dos operadores OR e AND (que são operadores binárias), o OR (Equação 2.5) é o máximo entre os valores e o AND (Equação 2.6) é o mínimo (MARRO et al., 2010).

$$NOT(A) = A' = \{x, \mu_{A'}(x))|x \in U \text{ e } \mu_{A'}(x) = 1 - \mu_A(x)\} \tag{2.4}$$

$$A\ OR\ B = A \cup B = \{(x, max(\mu_A(x); \mu_B(x)))|x \in U\} \tag{2.5}$$

$$A\ AND\ B = A \cap B = \{(x, min(\mu_A(x); \mu_B(x)))|x \in U\} \tag{2.6}$$

Para realizar inferência, a Lógica *Fuzzy* utiliza-se de regras linguísticas para obter o resultado. Essas regras tem o formato de *se ... ento*, que normalmente é denominada de implicação. Deste modo, a regra linguística é formada pelo antecedente e o consequente. No qual, pode haver mais de um antecedente, neste caso, usa-se

os operadores lógicos *AND* (Equação 2.7) e *OR* (Equação 2.8) para combina-los e gerar o resultado. O operador *OR* também pode ser empregado para ligar várias regras (Equação 2.9), no qual terá vários consequentes (TANSCHEIT, 2004).

$$\text{se } (x_1 \text{ é } A_1)\ AND\ (x_2 \text{ é } A_2)\ AND\ \ldots\ AND\ (x_m \text{ é } A_m) \text{ então } (y \text{ é } B) \tag{2.7}$$

$$\text{se } (x_1 \text{ é } A_1)\ OR\ (x_2 \text{ é } A_2)\ OR\ \ldots\ OR\ (x_m \text{ é } A_m) \text{ então } (y \text{ é } B) \tag{2.8}$$

$$\begin{aligned} R_1 &: \text{ se } (x \text{ é } A_1) \text{ então } (y \text{ é } B_1)\ OR \\ R_2 &: \text{ se } (x \text{ é } A_2) \text{ então } (y \text{ é } B_2)\ OR \\ R_n &: \text{ se } (x \text{ é } A_n) \text{ então } (y \text{ é } B_n) \end{aligned} \tag{2.9}$$

No processo de inferência então é realizada a avaliação das entradas a partir das regras estabelecidas, com o objetivo de obter conclusões ao utilizar a teoria dos Conjuntos *Fuzzy*. O método de inferência utilizado neste trabalho é o método Mamdani[5], na qual o processo de raciocínio é dividido em quatro etapas: (1) *fuzzificação*, (2) avaliação das regras *Fuzzy*, (3) agregação das regras *Fuzzy* e (4) *defuzzificação* (MARRO et al., 2010). A figura 12 apresenta o diagrama das etapas de um sistema de inferência *Fuzzy*.

Realizado a entrada dos dados para o Conjunto *Fuzzy* na etapa de *fuzzificação*, na qual também realiza a ativação das regras relevantes para uma determinada situação, é então obtido o Conjunto *Fuzzy* de saída através do processo de inferência (*modus ponens* generalizado). Com isso, é possível realizar o estágio de *defuzzificação*, na qual é efetuada a interpretação dessas informações para então gerar uma saída precisa (TANSCHEIT, 2004).

[5]Professor Ebrahom Mamdani (1942 - 2010), na Universidade de Londres, em 1975 construiu um dos primeiros sistemas Fuzzy, na qual ele aplicou um conjunto de regras fuzzy fornecidor por operadores experientes.

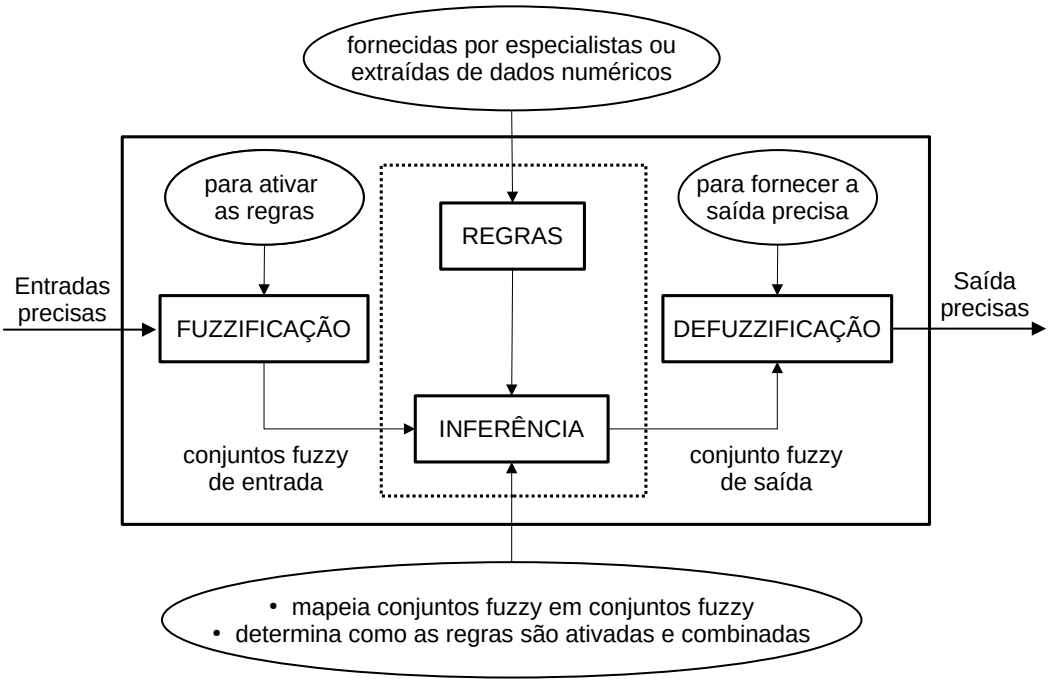

Figura 12: Sistema de inferência *Fuzzy*
Fonte: Adaptado de (TANSCHEIT, 2004)

As figuras 13 e 14 exemplifica graficamente o processo de inferência a partir das entradas dos dados (que são representados pela linha vertical). Nesse exemplo é avaliado o serviço do garçom e a qualidade da comida de um determinado estabelecimento com a finalidade de definir qual será a gorjeta ofertada. Observa-se a partir da área colorida o quanto o determinado valor pertence a cada termo. Para isso é necessário aplicar as regras de inferência para alcançar os resultados.

O processo de *defuzzificação*, a partir dos dados concebido pela etapa de inferência, realiza um determinado método para então obter uma saída numérica. Entre os métodos de *defuzzificação* encontram-se: centro de gravidade, média dos máximos e o maior dos máximos. O presente trabalho centra-se na técnica centro de gravidade, também conhecida como centroide. Na qual, essa técnica obtém o ponto médio que ao traçar uma linha vertical divide o conjunto agregado ao meio (TANSCHEIT, 2004). A equação 2.10 apresenta a fórmula matemática utilizada para obter esse ponto.

A figura 15 exibe um exemplo do processo de *defuzzificação* com o método centroide graficamente, na qual a linha vertical representa a saída numérica do resultado

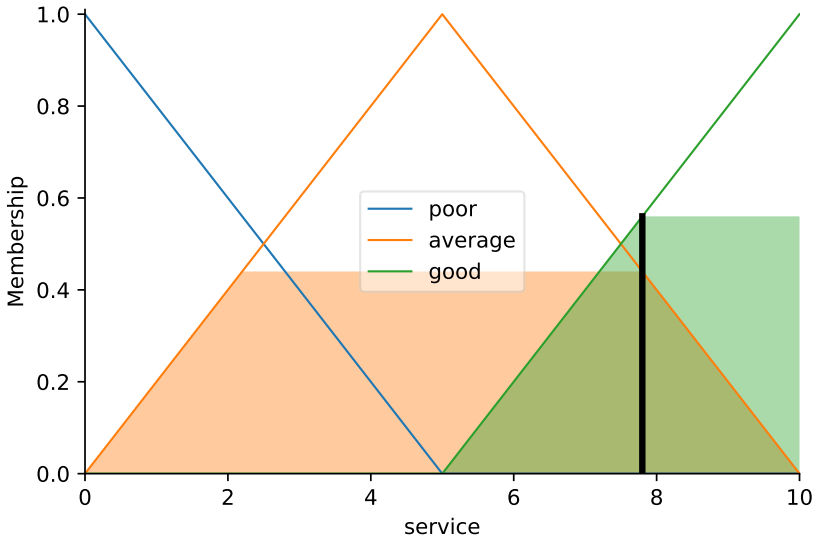

Figura 13: Exemplo de inferência a partir da nota dada ao serviço do garçom
Fonte: Autoria própria (2021)

Figura 14: Exemplo de inferência da nota dada a qualidade da comida
Fonte: Autoria própria (2021)

final. O exemplo realiza a *deffuzzificação* a partir dos dados exibidos anteriormente, nas figuras 13 e 14, no qual, após a aplicação das regras definidas, é obtido a pertinência para cada conjunto de saída. Com isso, ao aplicar o centroide o resultado obtido é uma gorjeta média. A partir do valor numérico da saída é possível realizar a tomada de decisão sobre as entradas.

$$C = \frac{\sum_{x=a}^{b} \mu(x) \cdot x}{\sum_{x=a}^{b} \mu(x)} \tag{2.10}$$

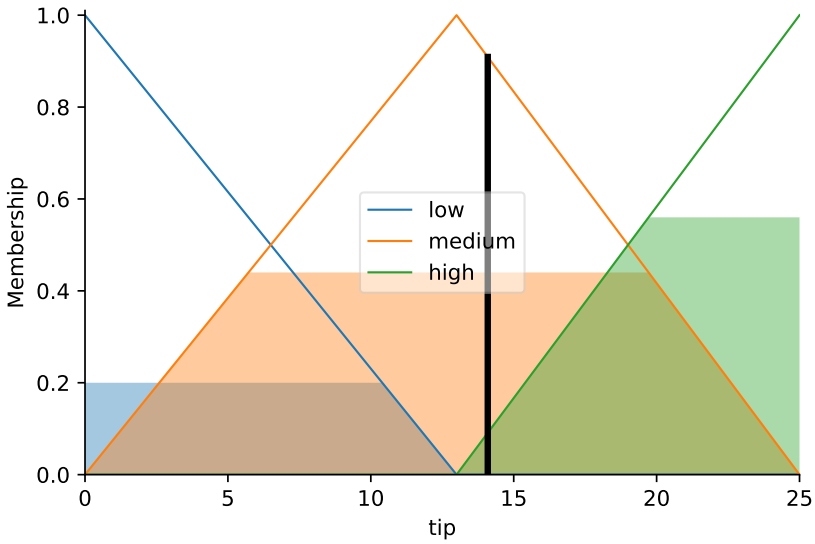

Figura 15: Exemplo de deffuzzificação com o método centroide *Fuzzy*
Fonte: Autoria própria (2021)

A Lógica *Fuzzy* é de suma importância para a tomada de decisão quando se trata de valores incertos. Por isso ela é empregada em diversos aspectos, como em ar condicionado, trem balas, e até mesmo na segurança computacional. Logo, é possível a aplicabilidade da Lógica *Fuzzy* neste trabalho.

2.6 ESTADO DA ARTE

Esta subseção abrange os trabalhos relacionados que utilizam de técnicas para detectar ataques DDoS ou que utilizam da lógica paraconsistente com a finalidade de obter segurança cibernética.

O trabalho de Carvalho (2020) propõe uma solução (DoSSEC) para detectar e mitigar ataques DDoS SYN *Flood* em SDN. A proposta utiliza as estatísticas geradas pelos *switches* para detectar eventuais ataques. De tal modo, são aplicados diferentes métodos estatísticos, como por exemplo qui-quadrado e entropia. Assim, é possível caracterizar eventuais ataques à rede.

A ferramenta *FindFlows* proposta por Fox (2019) tem o intuito de detectar

ataque DDoS SYN *Flood* em SDN. A detecção acontece a partir do procedimento de medição da variação da quantidade de fluxo em um intervalo de tempo pré-estabelecido e o monitoramento das portas TCP. Além disso, a ferramenta é capaz de identificar o IP do atacante.

Uma nova abordagem apontada por Oo e Htein Maw (2019) tem o objetivo de definir o limite estático das técnicas de análise estatística para auxiliar na detecção e mitigação de ataque DDoS em SDN. Deste modo, o autor apresenta o algoritmo de limiar adaptativo modificado (do inglês, *Modified Adaptive Threshold algorithm* – MATA). Este algoritmo reduz a taxa de falsos negativos e aumenta a precisão.

Um mecanismo de detecção adaptativa é sugerido por Hussain et al. (2019). Tal mecanismo utiliza a técnica de inteligência artificial denominada SYN *Flood Attack Detection Based on Bayes Estimator* (SFADBE) para rede móvel *ad hoc* (MANET). No algoritmo cada nó reúne as informações do canal disponível, com isso ao algoritmo seleciona um canal livre e seguro para realizar o tráfego dos pacotes.

O trabalho de Fernandes et al. (2019) apresenta a utilização da lógica paraconsistente para analisar a segurança cibernética em veículos autônomos. A LPA pode auxiliar na tomada de decisão, com isso é possível mitigar inúmeras falhas que causam acidentes e põe em risco vidas humanas.

Uma abordagem que realiza a detecção de ataque DDoS do tipo SYN *Flood* em redes SDN utilizando a LPA é proposta por Junior, Tavares e Nogueira (2019) a partir do estudo da LPA e do *three-way handshake* realizado pelo protocolo TCP para inicializar a conexão entre cliente e servidor. A partir da captura dos pacotes trafegados na rede são extraídas as seguintes informações: quantidade total de pacotes, quantidade de pacotes com *flag* SYN ativa e quantidade de pacotes com *flag* FIN ativa. Com as informações obtidas é realizado um cálculo para obter os graus de crença e descrença, de modo que os valores obtidos são entradas para o módulo responsável por realizar a validação utilizando a LPA. A partir do resultado gráfico e textual é possível identificar se em uma determinada janela houve um ataque. Partindo deste princípio, este trabalho apresenta melhorias provenientes dos primeiros estudos aplicados.

A dissertação da Falcão (2019) propõe a criação de um sistema intitulado *FuzDetect*, no qual tem como objetivo alertar a ocorrência de ataques DDoS e também classificá-los em ataque de exaustão ou volumétrico. Para isso, é realizado a coleta de dados em uma SDN a partir da interface de um *OpenFlow Switch*. Com os dados coletados, o sistema realiza a classificação com a Lógica *Fuzzy*, na qual são analisados os últimos metadados de fluxo coletados, para então classificá-los em subtipos de ataque ou em tráfego legítimo. O sistema também utiliza da Otimização por Enxame de Partículas (do inglês *particle swarm optimization* – PSO) para adaptar-se ao tráfego da rede de forma dinâmica.

O artigo apresentado por Alsirhani, Sampalli e Bodorik (2019) propõe um sistema dinâmico de detecção de ataques DDoS baseado em três componentes: algoritmos de classificação; um sistema distribuído; e um sistema de Lógica *Fuzzy*. A estrutura da Lógica *Fuzzy* é utilizada para selecionar dinamicamente um algoritmo de um conjunto de algoritmos de classificação preparados, que detectam diferentes padrões de DDoS. Os algoritmos de classificação utilizados no artigo foram: *Naive Bayes*, Árvore de Decisão (Entropia), Árvore de Decisão (Gini) e *Random Forest*. O sistema de Lógica *Fuzzy* empregado identifica o melhor algoritmo com base em três parâmetros de entradas: o volume de tráfego, a precisão do algoritmo de classificação e os atrasos do algoritmo de classificação. Além disso, Cada uma das entradas tem um conjunto de funções de pertinência que são representadas por uma função de pertinência triangular. Desta forma, o artigo então avaliou a eficacia de cada algoritmo e o sistema de Lógica *Fuzzy*. Logo, o sistema de Lógica *Fuzzy* pode efetivamente selecionar o algoritmo de classificação correto com base no status do tráfego.

Entretanto, Novaes et al. (2020) apresentam o sistema LSTM-FUZZY com o objetivo principal a caracterização do tráfego de rede, detecção e mitigação de ataques DDoS e *Portscan* em ambiente de SDN. O sistema teve como princípio o conceito de Assinatura Digital de Segmentos de Rede (do inglês *Digital Signature of Network Segments* – DSNS), na qual a primeira fase realiza a previsão do comportamento normal do tráfego da rede e a definição dos limites de normalidade. Na segunda faze é realizado a aplicação da lógica *fuzzy* para determinar se existem anomalias, utilizando como parâmetro o tráfego previsto e os limiares definidos na última etapa.

Por fim, a terceira fase é aplicada as políticas de mitigação com o intuito de tomar contra-medidas contra os ataques detectados, garantindo o funcionamento da rede. O modelo proposto para detecção de anomalias neste trabalho utiliza tráfego passado, o previsto por LSTM e a lógica *Fuzzy*. Os resultados obtidos mostram a eficiência do sistema em auxiliar no gerenciamento da rede, detectar e mitigar a ocorrência dos ataques.

3 METODOLOGIA

Este capítulo apresenta a metodologia, os materiais e processos empregados para a execução do trabalho e como os objetivos foram alcançados. Também é apresentado a solução proposta de detecção de ataque DDoS SYN *Flood* chamada LPAProg-DDoS. Primeiramente é discutido sobre o módulo de resolução da LPA, no qual abrange apenas a parte LPAProg. Então, é tratado os elementos fundamentais da solução proposta, de como é realizado o processo de detecção com o auxílio da LPA.

A figura 16 apresenta a sequência de fases realizadas para a construção do trabalho. Na qual a primeira fase abrange a construção do módulo LPAProg desde a infraestrutura até a validação das saídas. Além disso, esta fase aborda as ferramentas utilizadas para a construção do módulo e os processos da LPA para gerar o resultado. A segunda fase foca na utilização da LPAProg para detecção de ataque. De tal modo, é descrito os processos de construção do módulo LPAProg-DDoS, coleta de dados, pré-processamento e transformação dos dados em entrada para LPA. Além disso, a segunda fase apresenta o sistema FuzProg-DDoS que utiliza da Lógica *Fuzzy* para detectar os ataques, na qual este sistema é utilizado para comparar os resultados com o LPAProg-DDoS. A terceira fase, apoia-se na construção do ambiente para a realização de testes do LPAProg-DDoS e FuzProg-DDoS. Com isso, esta fase abrange as topologias empregadas, as ferramentas utilizadas, a configuração do módulo e as bases dos testes realizados para validação dos sistemas. A quarta fase tem o intuito de apresentar os resultados a partir dos testes, nos mesmos ambientes reportados, em um sistema de detecção de ataque DDoS SYN *Flood* com a LPA e a lógica *Fuzzy*, analisar os resultados, compara-los entre ambas as técnicas e fazer uma estatística completa, no qual fica a cargo de ser apresentada no capítulo 4 com todas as medidas e aplicações de testes. Por fim, a quinta fase tem o intuito de concluir o trabalho e propor trabalhos futuros. Esta fase é discutida com mais detalhes no capítulo 5.

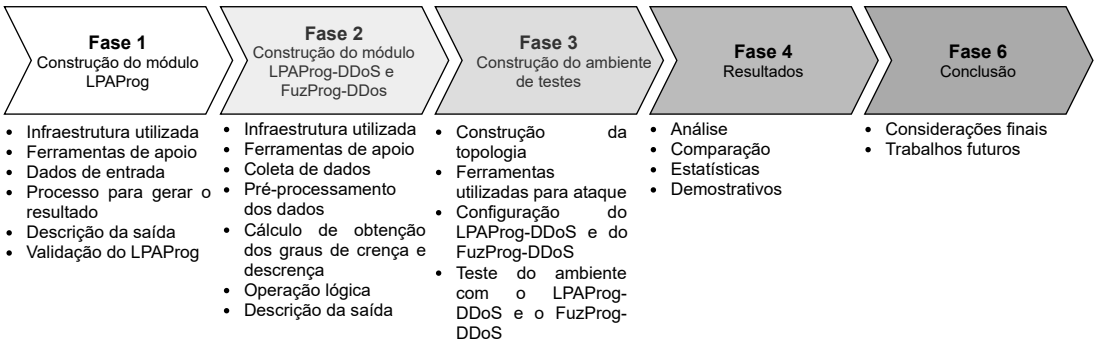

Figura 16: Fases do trabalho.
Fonte: Autoria própria (2020)

3.1 PROPOSTA DO TRABALHO

As redes de computadores, principalmente a Internet, está em franca expansão e desenvolvimento, de modo que é crescente o número de dispositivos conectados (CISCO, 2020). Além disso, a oferta por serviços *On-lines* mostram-se necessários à medida que os negócios virtuais e os recursos virtuais são cada vez mais utilizados. Ao se tratar do ponto de vista da área de segurança computacional, que é uma preocupação cada vez maior em função da utilização da Internet, destaca-se o DDoS, como o ataque ativo mais utilizado em 2019 no Brasil (CERT.BR, 2020), na qual aproveita-se de um grande número de dispositivos para impedir que o serviço seja entregue ao usuário. No âmbito mundial, o ataque DDoS do tipo SYN *flood*, obteve a primeira posição dentre os ataques DDoS registrados em 2019 (KUPREEV; BADOVSKAYA; GUTNIKOV, 2019). Desta forma, este trabalho apresenta uma proposta inovadora para auxiliar na detecção do ataque DDoS do tipo SYN *flood* através da utilização da LPA. Com isso, o trabalho proposto visa introduzir a interdisciplinaridade que alia redes, segurança e a área da lógica não clássica na busca de novas técnicas de detecção de ataques e auxílio na tomada de decisão.

3.2 LPAPROG

A construção do módulo LPAProg ocorreu em um ambiente com arquitetura Linux com o *kernel* na versão 5.8.16. Já a linguagem de programação utilizada

foi o *Python* na versão 3.8.3. Para a implementação foram necessária a instalação das bibliotecas: *NumPy*, *Pandas*, *Argparse* e *Matplotlib*, nas versões 1.19.0, 1.0.5, 1.4.0 e 3.2.2 respectivamente. Para atender este trabalho, o módulo foi dividido em cinco componentes, no qual dois deles foram construídos utilizando o paradigma de programação Orientado a Objeto (OO) e os três restantes foram desenvolvido através do paradigma estruturado. A figura 17 apresenta os componentes junto com os seus atributos e métodos.

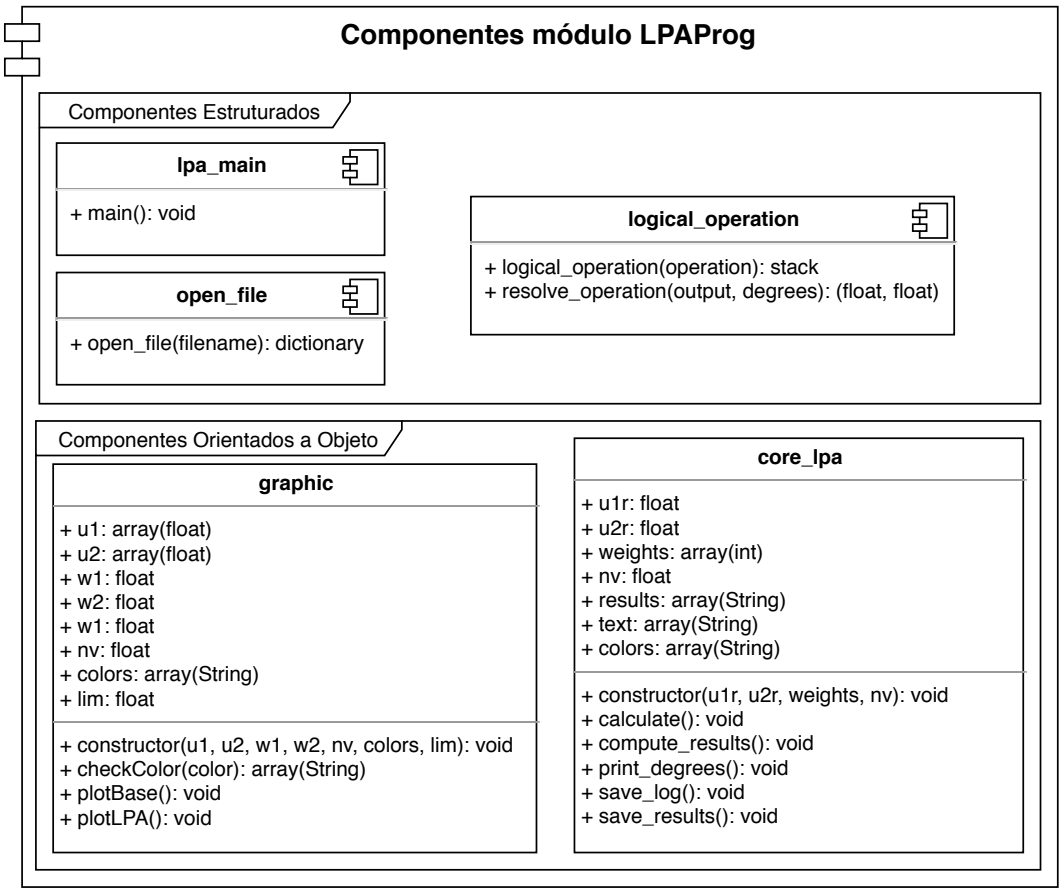

Figura 17: Componentes do módulo LPAProg.
Fonte: Autoria própria (2020)

O componente *open file* (abrir arquivo), como o próprio nome diz, é responsável por realizar a abertura do arquivo de entrada e fazer a validação para obter os dados necessários para a execução da LPA. Para isso, a função *open file* recebe como parâmetro o nome do arquivo (*filename*) a ser aberto. Portanto, a função abre o arquivo, remove os espaços e linhas em branco. Em seguida, é validado se o arquivo contém todos os dados necessários: o cabeçalho (*@header*), os dados de

graus de crenças e descrenças (*@data*), os pesos de cada fator (*@weight*), a operação lógica a ser atribuída sobre os graus de crenças e descrenças para se obter os graus resultantes (*@operation*) e os seus respectivos valores. Posteriormente, é validado se os operandos na operação lógica estão corretos, ou seja, se condiz com os valores passados no cabeçalho. Por fim, os valores são armazenados em um dicionário e retornado pela função. A complexidade de tempo deste componente é $O(n)$, em que n é a quantidade de linhas do arquivo de entrada. O código 3.1 mostra um exemplo do arquivo de entrada padrão do LPAProg.

Código 3.1: Exemplo do arquivo de entrada da função *open file*.

```
# Nome dos "Especialista"
@header
ua, ub, uc, ud

# Dados dos graus dos especialistas
# Seguindo da direita para esquerda
# Graus de crencas e descren as do especialista 1 (ua)
# Graus de crencas e descren as do especialista 2 (ub)
# Graus de crencas e descren as do especialista N
# O numero de colunas tem que ser o dobro do numero de headers
@data
0.3, 1.0, 0.3, 1.0, 0.2, 0.8, 0.2, 1.0
0.9, 0.2, 1.0, 0.2, 0.9, 0.1, 0.8, 0.1
0.6, 0.4, 0.4, 0.4, 0.6, 0.5, 0.5, 0.5
0.1, 1.0, 0.2, 1.0, 0.2, 1.0, 0.0, 0.9
0.9, 0.9, 1.0, 0.8, 1.0, 0.1, 0.2, 0.9
0.6, 0.5, 0.7, 0.3, 0.7, 0.4, 0.6, 0.4
0.1, 1.0, 0.3, 0.9, 0.3, 0.7, 0.0, 0.9
1.0, 0.2, 0.9, 0.2, 0.9, 0.1, 0.8, 0.2

# Pesos dos fatores de entradas
# A quantidade de pesos tem que ser a mesma de linhas dos dados
# Caso nao for setado os pesos sera considerado como 1 para todos
@weight
1, 1, 1, 3, 1, 1, 2, 2

# Operacao logica a ser aplicada entre os especialista
```

```
# Para obter os graus de cren a e descren a resultante
# Os operandos devem ser os mesmo contidos no header
# As operacoes devem conter espacos como
@operation
ua and ub and (uc or ud)
```

O componente *logical operation* (operação lógica) é responsável por realizar a leitura da operação lógica, verificar se ela está válida e desmembrar os elementos para realizar a operação de forma correta sobre as suas precedências. Tal componente é dividido em duas funções: *logical operation* e *resolve operation*. A primeira função é responsável por validar a operação e analisar a expressão lógica, de modo a transformá-la em uma notação polonesa inversa (do inglês, *Reverse Polish Notation* – RPN). Enquanto, a segunda função é encarregado de resolver a operação lógica a partir da RPN.

Para realizar a transformação da operação em uma RPN, é necessário utilizar o algoritmo *shunting-yard*, inventado por Edsger Dijkstra (CHÁVEZ-BOSQUEZ; PARRA; MCAREAVEY, 2016). Este algoritmo, é um método utilizado na ciência da computação para analisar expressões matemáticas especificas que estão em uma notação lógica ou aritmética comum (notação infixa). No qual, o algoritmo utiliza de pilhas para que um texto (*string*) seja transcrito em ordem pós-fixada (RPN), de forma a auxiliar a resolver a operação. Além disso, com este algoritmo é possível realizar a análise sintática (*parsing*) da operação lógica, de modo que é possível garantir que a operação recebida esteja correta.

O código 3.2 apresenta a solução proposta com a utilização do algoritmo *shunting-yard*. No qual, a função *logical operation* recebe como parâmetro um texto (*string*) e o transforma em um vetor (*array*). Posteriormente é validado se a quantidade de parenteses aberto é a mesma que fechado, caso não seja o algoritmo exibe uma mensagem de error e finaliza. Após a análise é inicializado algoritmo *shunting-yard*, que verifica cada elemento do *array* da operação lógica, também o grau de precedência dos operadores, para então construir o *array* de saída que contém a operação em RPN. A complexidade de tempo deste função é $O(n)$, em que n é a quantidade de

caracteres existentes na expressão lógica de entrada (texto). A figura 18 apresenta um exemplo de como é a entrada inicial, o *array* construído a partir da entrada e o *array* de saída.

Código 3.2: Função *logical operation*.

```
def logical_operation(operation):
    exp = operation.replace('(',' ( ').replace(')', ' ) ').split()
    if exp.count('(') != exp.count(')'):
        print("Logical operation invalid!!!")
        exit(1)

    # Shunting-yard algorithm
    operators = {'not':3, 'and':2, 'or':1}
    output = []
    temp = []
    for x in exp:
        if x in operators.keys():
            if len(temp) > 0 and temp[-1] != '(':
                if operators[x] < operators[temp[-1]]:
                    output.append(temp.pop())
            temp.append(x)
        elif x == '(':
            temp.append(x)
        elif x == ')':
            a = temp.pop()
            while a != '(':
                output.append(a)
                a = temp.pop()
        else:
            output.append(x)
    while len(temp) > 0:
        output.append(temp.pop())
```

A RPN reduz o número de passos lógicos para se resolver operações binárias. Além de minimizar os erros de computação e maximizar a velocidade operacional na solução de problemas. Uma operação lógica em uma RPN, pode ser resolvida com pilhas, sem se preocupar com a precedência dos operadores, pois a notação

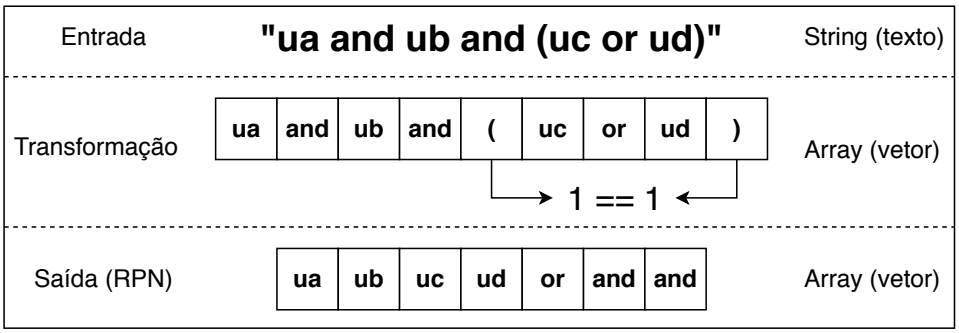

Figura 18: Exemplo da transformação da operação lógica em RPN.
Fonte: Autoria própria (2020)

garante que a operação está na ordem correta e sem os parênteses.

Para a resolução da operação lógica em RPN, a função *resolve operation* recebe como parâmetros a RPN e os graus de crenças e descrenças. No qual, estes graus são os principais fatores da LPA, pois a partir deles que se obtém o resultado final. Deste modo, esta função é responsável em obter os graus de crença e descrença resultante a partir das operações da LPA, conforme apresentado na seção 2.4. A complexidade de tempo deste função é $O(n)$, em que n é a quantidade de de operandos e operadores da expressão lógica. Desta forma, o algoritmo 1 apresenta como é realizada a operação lógica a partir do RPN.

Através da operação lógica resolvida são obtidos os graus de crenças e descrenças resultantes, no qual é possível obter a resposta final da LPA. Entretanto, é necessário passar pelo processo de validação da LPA, através dos cálculos do grau de certeza e contradição (vide seção 2.4). Com estes resultados, é gerado a saída com base no nível de exigência definido, ou seja, o estado lógico do problema analisado. Desta forma, a tabela 1 (tabela de regiões), apresentada na seção 2.4, mostra os possíveis resultados lógicos, baseado na região condicional e resultante. Como é possível realizar a análise de vários graus de crenças e descrenças resultantes, é realizado também o cálculo do baricentro. No qual, é realizado uma média ponderada entre os graus para gerar um estado lógico médio. Este estado médio pode indicar a veracidade do problema como um todo.

O conjunto de métodos e atributos para calcular e validar a LPA estão contidos na classe *core lpa* (componente OO). Além dos cálculos e comparações dos resultados,

Algoritmo 1 *resolve_operation*(rpn)

```
1:  stack = ∅
2:  while rpn ≠ ∅ do
3:      x = extractFirst(rpn)
4:      if x == "or" then
5:          v2 = pop(stack)
6:          v1 = pop(stack)
7:          μR = max(v1, v2)
8:          push(stack, μR)
9:      else if x == "and" then
10:         v2 = pop(stack)
11:         v1 = pop(stack)
12:         μR = min(v1, v2)
13:         push(stack, μR)
14:     else if x == "not" then
15:         v1 = pop(stack)
16:         μR = reverse(v1, v2)
17:         push(stack, μR)
18:     else
19:         push(stack, x)
20:     end if
21: end while
22: μR = pop(stack)
23: return μR
```

esta classe contém os métodos para exibir os resultados e salvá-los em arquivos para consultas posteriores. A maior complexidade de tempo desta classe é $O(n)$ (método *compute result*), em que n é a quantidade de dados analisados mais o baricentro. A figura 19 apresenta o exemplo de um arquivo de *log*, no formato *AAmmdd-hhmmss.log*, gerado após a análise do LPA. Neste exemplo, o arquivo contém os graus de crenças e descrenças resultantes, assim como os graus de crença e descrença do baricentro, os graus de certeza, contradição e os estados lógicos finais para cada fator, bem como para o baricentro. Além deste arquivo de log, o método LPAProg gera um arquivo no formato CSV, com o nome *Output_ AAmmdd-hhmmss.csv* (Figura 20), que contém os graus de crenças e descrenças iniciais, e os resultados lógicos para eles.

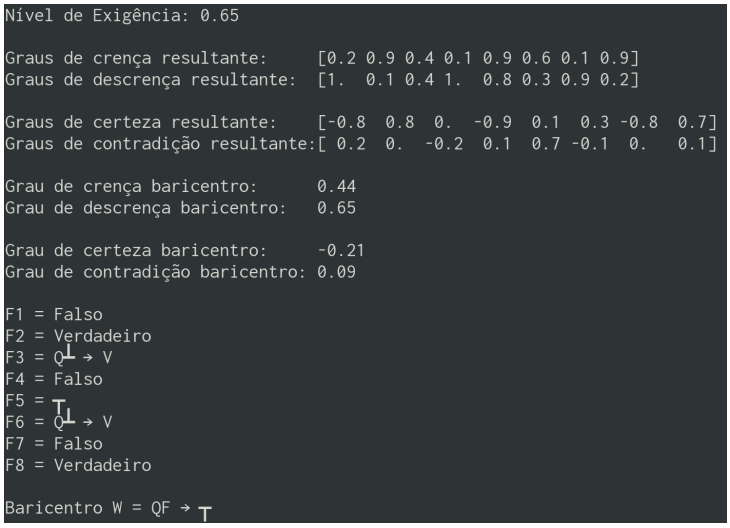

Figura 19: Exemplo do *log* gerado pelo módulo LPAProg.
Fonte: Autoria própria (2020)

ua1	ua2	ub1	ub2	uc1	uc2	ud1	ud2	Result
0.3	1.0	0.3	1.0	0.2	0.8	0.2	1.0	Falso
0.9	0.2	1.0	0.2	0.9	0.1	0.8	0.1	Verdadeiro
0.6	0.4	0.4	0.4	0.6	0.5	0.5	0.5	Q⊥ → V
0.1	1.0	0.2	1.0	0.2	1.0	0.0	0.9	Falso
0.9	0.9	1.0	0.8	1.0	0.1	0.2	0.9	⊤
0.6	0.5	0.7	0.3	0.7	0.4	0.6	0.4	Q⊥ → V
0.1	1.0	0.3	0.9	0.3	0.7	0.0	0.9	Falso
1.0	0.2	0.9	0.2	0.9	0.1	0.8	0.2	Verdadeiro
							Baricentro	QF → ⊤

Figura 20: Exemplo do arquivo de saída gerado pelo módulo LPAProg.
Fonte: Autoria própria (2020)

Além da análise textual, obtém-se a análise gráfica a partir do QUPC (apresentado na seção 2.4), através da classe *graphic* do módulo LPAProg. Para a construção do objeto a partir desta classe, é necessário apresentação dos parâmetros utilizados pelo método construtor, são eles: graus de crenças e descrenças iniciais, graus de crença e descrença do baricentro, nível de exigência, cores dos pontos a serem plotados e limite (opcional), que define o tamanho do gráfico. A partir destes atributos, os métodos preenchem o gráfico, salva em um arquivo (Graphic_LPA_AAmmdd-hhmmss.png) e exibe-o ao usuário. A maior complexidade de tempo desta classe é $O(n)$ (método *plot LPA*), em que n é a quantidade de dados a serem plotados. A figura 21 apresenta o exemplo do gráfico gerado ao executar o módulo LPAProg com o arquivo 3.1 como entrada.

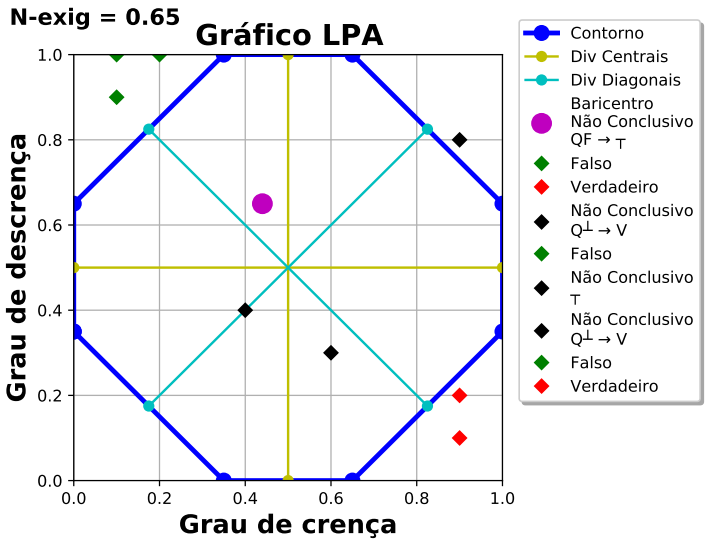

Figura 21: Exemplo do QUPC gerado pelo módulo LPAProg.
Fonte: Autoria própria (2020)

Por fim, o módulo LPAProg utiliza-se do componente estruturado *lpa main* para fazer todo o processo de ligação entre os demais componentes. Além disso, este componente trata os argumentos passados pelo usuário, no qual é responsável em receber o arquivo dos dados e o nível de exigência (opcional). A figura 22 apresenta o diagrama de sequência do módulo LPAProg, no qual exibe o controle do componente *lpa main* sobre os demais.

Para validar o módulo LPAProg, utilizou-se dos dados (Figura 23) do trabalho de estudo de tomada de decisão baseado em LPA, no qual determina a viabilidade

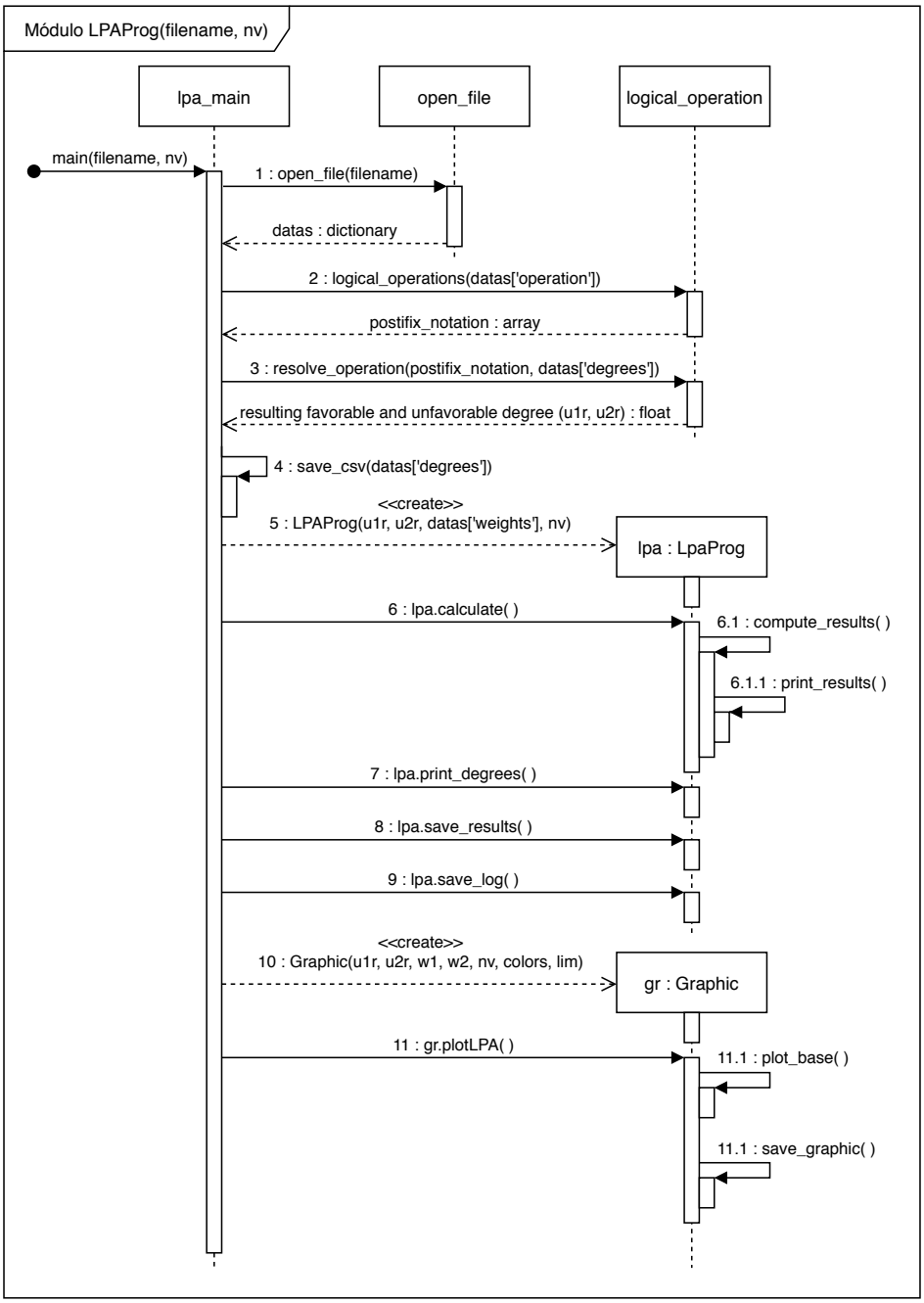

Figura 22: Diagrama de sequência do módulo LPAProg.
Fonte: Autoria própria (2020)

de um projeto de uma fábrica (CARVALHO; BRUNSTEIN; ABE, 2003). Os dados da figura 23 podem ser comparados com os das figuras 20 e 19, no qual a primeira figura corresponde aos dados apresentados por CARVALHO, Brunstein e Abe (2003) e as demais são correspondentes a saída do LPAProg. Observa-se a equivalência dos resultados dos graus ao comparar as figuras. O que difere entre as figuras é a decisão. No qual a figura 23 utiliza-se dos termos viável, inviável e não conclusivo. Enquanto

a figura 20 e 19 utilizam-se dos termos verdadeiro, falso e outros. Neste caso, ao se tratar de um resultado verdadeiro, considera-se como viável, o resultado falso como inviável e os demais resultados como não conclusivo. A seção 2.4 aborda sobre as região de tomadas de decisões e as regiões não conclusiva.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Fator	**Peso**	**Seção**	**Grupo A**		**Grupo B**		**Grupo C**				**Grupo C**		**A AND B AND C**		**Nível de Exig.**		**0,65**
			Espec 1		**Espec 2**		**Espec 3**		**Espec 4**		**E3 OR E4**				**Conclusões**		
			μ_{11}	μ_{12}	μ_{21}	μ_{22}	μ_{31}	μ_{32}	μ_{41}	μ_{42}	μ_{1C}	μ_{2C}	μ_{1R}	μ_{2R}	**Hcert**	**Gcontr**	**Decisão**
F1	1	S3	0,3	1,0	0,3	1,0	0,2	0,8	0,2	1,0	0,2	1,0	0,2	1,0	-0,80	0,20	INVIÁVEL
F2	1	S1	0,9	0,2	1,0	0,2	0,9	0,1	0,8	0,1	0,9	0,1	0,9	0,1	0,80	0,00	VIÁVEL
F3	1	S2	0,6	0,4	0,4	0,4	0,6	0,5	0,5	0,5	0,6	0,5	0,4	0,4	0,00	-0,20	NÃO CONCLUSIVO
F4	3	S3	0,1	1,0	0,2	1,0	0,2	1,0	0,0	0,9	0,2	1,0	0,1	1,0	-0,90	0,10	INVIÁVEL
F5	1	S1	0,9	0,9	1,0	0,8	1,0	0,1	0,2	0,9	1,0	0,9	0,9	0,8	0,10	0,70	NÃO CONCLUSIVO
F6	1	S2	0,6	0,5	0,7	0,3	0,7	0,4	0,6	0,4	0,7	0,4	0,6	0,3	0,30	-0,10	NÃO CONCLUSIVO
F7	2	S3	0,1	1,0	0,3	0,9	0,3	0,7	0,0	0,9	0,3	0,9	0,1	0,9	-0,80	0,00	INVIÁVEL
F8	2	S1	1,0	0,2	0,9	0,2	0,9	0,1	0,8	0,2	0,9	0,2	0,9	0,2	0,70	0,10	VIÁVEL
Baricentro W: média ponderada dos graus resultantes													**0,44**	**0,65**	**-0,21**	**0,09**	NÃO CONCLUSIVO

Figura 23: Dados utilizados para comparação.
Fonte: (CARVALHO; BRUNSTEIN; ABE, 2003)

Além dos dados, foi obtido do trabalho de (CARVALHO; BRUNSTEIN; ABE, 2003) o QUPC (Figura 24) para realizar a comparação. Observa-se o comportamento idêntico entre os pontos plotados no QUPC do trabalho citado (Figura 24) com o QUPC deste trabalho (Figura 21).

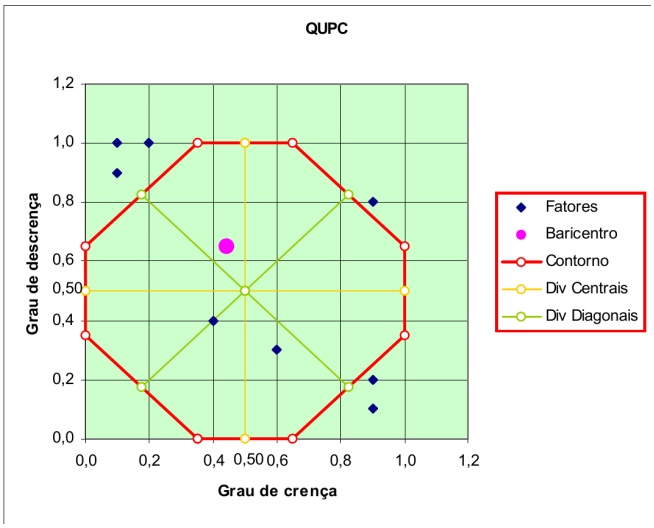

Figura 24: QUPC utilizado para comparação.
Fonte: (CARVALHO; BRUNSTEIN; ABE, 2003)

Através da validação dos resultados do LPAProg *versus* o trabalho apresentado por CARVALHO, Brunstein e Abe (2003), foi possível prosseguir para a próxima

fase, no qual abrange a construção do módulo LPAProg-DDoS. Esta fase utiliza-se do módulo apresentado LPAProg para a implementação do detector de ataques DDoS. No qual, recorre da saída do estado lógico da LPA para indicar se há ataque na rede vistoriada ou mesmo em um arquivo de captura de rede. Este módulo validado é de suma importância para prosseguir para a próxima fase, de tal modo que o mesmo será o núcleo de decisão do módulo LPAProg-DDoS.

3.3 LPAPROG-DDOS

Para a construção do módulo (sistema) de detecção de ataque DDoS com a LPA (LPAProg-DDoS), foi utilizado um ambiente com arquitetura *Linux* com o *kernel* 5.8.16. Inicialmente, para a programação, foi realizada a mesclagem da linguagem *Python* com o *Shell Script*. Atualmente, com o auxílio das bibliotecas, o sistema LPAProg-DDoS utiliza-se apenas da linguagem *Python*. A tabela 2 mostra as dependências com suas respectivas versões utilizadas para a construção e execução deste módulo. Em comparação ao módulo LPAProg, este utiliza apenas uma biblioteca diferente, a *Scapy*. Com esta biblioteca foi possível realizar toda a integração do sistema na linguagem Python. Na qual, oferece uma gama de ferramentas para se trabalhar com o tráfego da rede, como abrir arquivo de captura de rede (pcap), capturar pacotes de rede em tempo real e percorrer as camadas dos pacotes.

Nome	Versão
Python	3.8.3
Argparse	1.1
Matplotlib	3.3.1
NumPy	1.18.5
Pandas	1.1.0
Scapy	2.4.3

Tabela 2: Tabela de dependências do LPAProg-DDoS
Fonte: Autoria própria (2020)

Neste sistema é incluído três componentes do módulo LPAProg: *logical operation*, *core lpa* e *graphic*. Além deles, também há os componentes *calculate degrees*, *sniff opcap*, *online* e *offline*. Dentre os novos módulos citados, dois são orientados a objetos e dois são estruturados. Os componentes estruturados (*online* e *offline*)

são os responsáveis por inicializar o sistema. Enquanto, os orientados a objetos são responsável em realizar todo o processo de capturar e processar os dados. A figura 25 apresenta a estrutura dos componentes do sistema LPAProg-DDoS, no qual utiliza de outro módulo para seu funcionamento. Além disso, a figura 25 exibe as classes *calculate degrees* e *sniff opcap* com seus respectivos atributos e métodos.

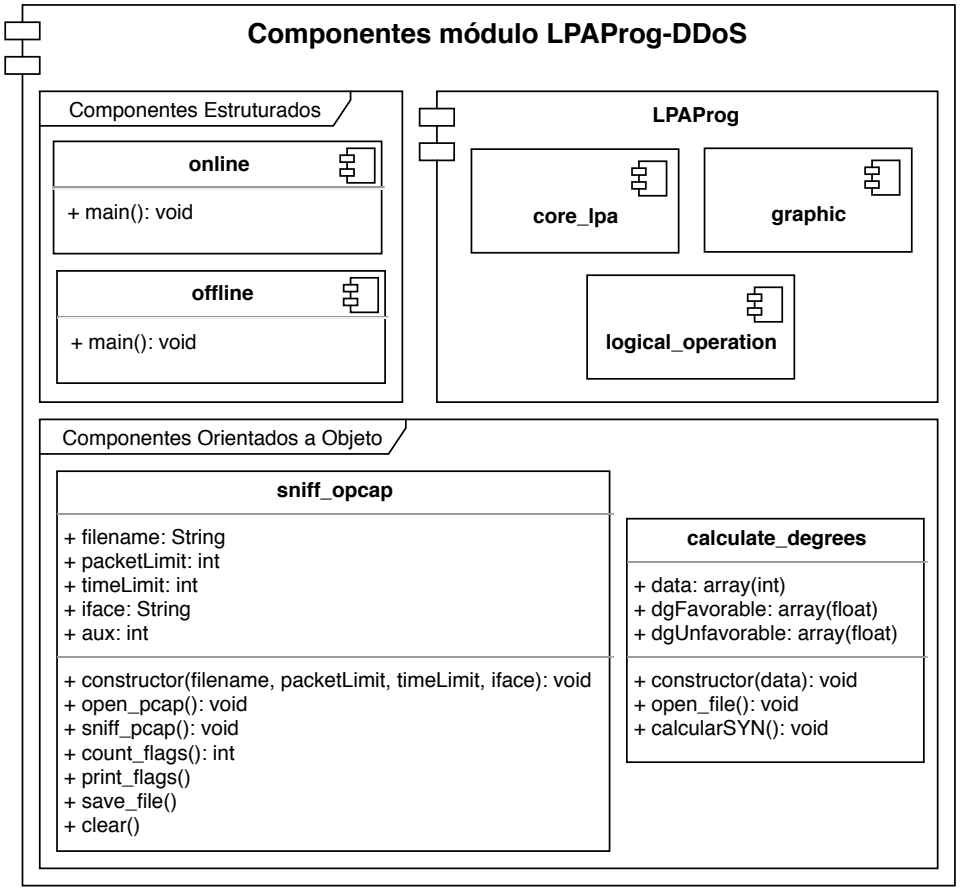

Figura 25: Componentes do módulo LPAProg-DDoS.
Fonte: Autoria própria (2020)

Os processos realizados pelo LPAProg-DDoS foram dividido para melhor explicação. O primeiro processo é o da coleta dos dados, no qual há duas formas de obter os dados a serem analisados: capturar os dados da rede ou abrir um aquivo de captura de rede. Com os dados carregados, o sistema realiza um pré-processamento, de forma que os dados possam ser processados pela terceira fase. A terceira fase, é responsável em realizar os cálculos para se obter os graus de crenças e descrenças. Com os graus obtidos, a quarta fase computa-os com o LPAProg. O resultados então são exibidos para o administrador. A figura 26 apresenta a arquitetura do sistema

LPAProg-DDoS com as divisões citadas. O interior de cada etapa do processo, junto como os componentes do sistema, serão detalhados nas próximas subseções.

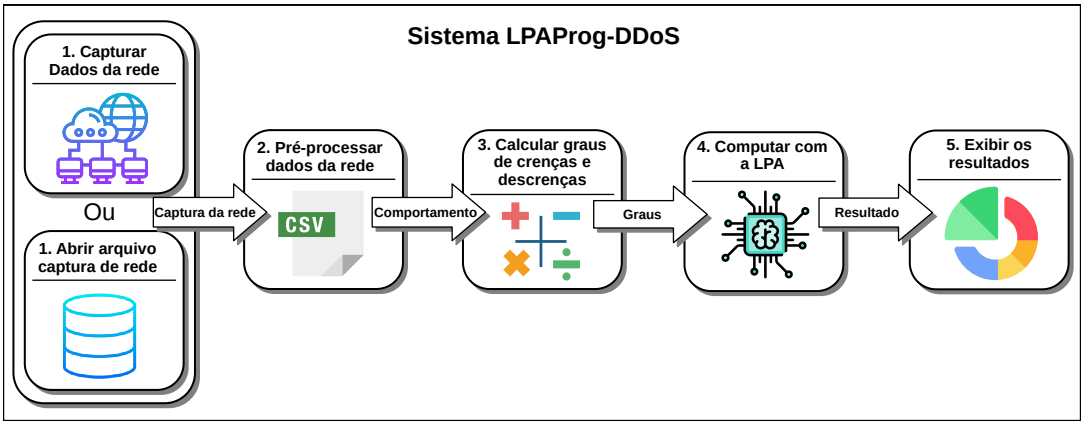

Figura 26: Arquitetura do sistema LPAProg-DDoS.
Fonte: Autoria própria (2020)

3.3.1 Coleta e pré-processamento dos dados

Uma das premissas para que o sistema funcione é a necessidade da análise dos dados trafegados na rede. Para isso, há no sistema duas formas de coletar os dados. A primeira é a partir da abertura de um arquivo de captura de rede (*pcap*), gerado a partir de uma ferramenta de *sniff*, como por exemplo o *tcpdump*, *scapy* ou *wireshark*. Estas ferramentas têm a capacidade de capturar os pacotes que estão em tráfego no dispositivo corrente, ou seja, ela é capaz de coletar as informações que estão em tráfego na rede. Além disso, é possível salvar em um arquivo *pcap* os pacotes coletados para uma análise posterior. A segunda opção, é realizar a coleta dos dados em tempo real. Esta opção é semelhante ao processo anterior, o que difere que não é preciso salvar o arquivo de captura (*pcap*). De tal modo, é possível realizar a análise dos pacotes no presente momento.

Com as informações da captura disponíveis, é preciso realizar o pré-processamento de modo a filtrar o conjunto de dados para extrair o necessário. O pré-processamento analisa cada pacote para coletar os dados em seu interior e gerar um novo conjunto de dados consistente e pronto para ser processado. Nesta etapa, é preciso deslocar-se pelas camadas do pacote de rede. Neste trabalho, serão analisados apenas pacotes TCP/IP e os dados capturados serão da camada de transporte.

Para a realização da coleta e pré processamento dos dados, o LPAProg-DDoS possui o componente *sniff opcap*. Este componente, é uma classe que contém um conjunto de atributos e métodos. Os atributos encontrados são: filename, packetLimit, timeLimit, iface, aux, packages. No qual são responsáveis em armazenar os seguintes valores respectivamente: nome do arquivo de captura, quantidade de pacotes máximo que pode conter a janela, quantidade de tempo (em segundos) máximo que pode conter a janela, a interface de rede a ser capturado os pacotes, um auxiliar que contém o endereço do último pacote analisado e o conjunto de pacotes (tráfego da rede). Neste caso, o termo janela é considerado como uma parte da captura de pacotes que atingiu o limite de quantidade de pacotes ou de tempo.

Além dos atributos, a classe *sniff opcap* contém os métodos que são responsáveis em coletar e pré-processar os dados. Os métodos são: *constructor*, *open pcap*, *sniff pcap*, *count flags*, *print flags*, *save file* e *clear*. Estes métodos realizam respectivamente as funções: atribuir os valores passados aos atributos ao instanciar o objeto, efetuar a abertura de um arquivo de captura, realizar a captura dos pacotes em tempo real, efetivar a contagem de *flags* em cada janela, exibir a quantidade de pacotes total e TCP/IP e a contagem de *flags* ao usuário, salvar a contagem em um arquivo CSV (*comma-separated value*) e limpar os pacotes. A complexidade de tempo deste componente é $O(n)$, em que n é a quantidade de pacotes analisados.

Os métodos *open pcap* e *sniff pcap* utilizam da biblioteca *scapy* para obter sucesso e, abrir um arquivo de captura de rede ou capturar os pacotes em tempo real. Ambos os métodos armazenam os dados (conjunto de pacotes) no atributo *packages*. De tal forma, que o pré-processamento é realizado sobre um arquivo ou sobre a captura atual da rede. O método *sniff pcap* utiliza dos atributos de limite para parar a captura e gerar uma janela. No qual, as janelas podem ser formadas pelo limite de tempo em segundos, ou pela quantidade de pacotes, ou por ambos os limites.

Com os dados armazenados no atributo *packages*, é possível realizar o pré-processamento. Neste trabalho, o pré-processamento realiza a contagem de pacotes total, pacotes que utiliza o protocolo TCP, pacotes somente com a *flag* SYN ativa, pacotes com a *flag* FIN ativa, pacotes com a *flag* SYN e ACK ativa, pacotes com a

flag ACK ativa, pacotes com a *flag* RST ativa e pacotes com a *flag* PSH ativa.

Como apresentado na seção 2.2, as *flags* pertencem aos pacotes do tipo TCP, por isso a importância de obter pacotes TCP. Para isso é preciso caminhar pelas camadas do pacote, no qual inicia-se na camada de interface de rede (*Ethernet*). Está camada informa qual protocolo utilizado na camada internet, no qual espera-se que o protocolo seja o IP. Desta forma, pode-se esperar que o protocolo utilizado na camada de transporte seja do tipo TCP. Para realizar a validação, o protocolo IP tem em seu cabeçalho a informação de qual é o protocolo da camada de transporte. Ao validar que o protocolo da camada de transporte é TCP, é possível obter as *flags* a partir de seu cabeçalho, como apresentado na figura 3. A figura 27 exibe um exemplo da representação de um pacote que utiliza do protocolo TCP, as marcações na figura representa os campos citados para a validação e também o campo que contém as *flags*.

```
<Ether  dst=00:00:00:aa:00:01 src=00:00:00:aa:00:00 type=IPv4 |<IP  version=4
 ihl=5 tos=0x0 len=60 id=53667 flags=DF frag=0 ttl=64 proto=tcp chksum=0x54fb
 src=10.0.0.20 dst=10.0.0.10 |<TCP  sport=38082 dport=ssh seq=1031783807 ack=
0 dataofs=10 reserved=0 flags=S window=64240 chksum=0x4bbb urgptr=0 options=[
('MSS', 1460), ('SAckOK', b''), ('Timestamp', (281232540, 0)), ('NOP', None),
 ('WScale', 7)] |>>>
```

Figura 27: Exemplo de representação de pacote no *Scapy*.
Fonte: Autoria própria (2020)

Após a realização da contagem, é possível ver os resultados e salvá-los em um arquivo CSV com os métodos *print flags* e *save file* respectivamente. O método *save file* mantém no máximo dez linhas de informações das contagens no arquivo, ou seja, o arquivo contém no máximo dez janelas. Este processo é realizado desta forma, pelo fato de se optar em realizar a análise das janelas no decorrer do tempo, no qual ao atingir o limite de dez janelas, a análise é reinicializada, recomeçando com uma janela. Este processo na captura em tempo real envolve um ciclo infinito, do qual é interrompido somente quando o usuário solicitar. A figura 28 apresenta um exemplo do arquivo de saída que contém dez janelas.

Com o arquivo da contagem gerado, é possível prosseguir nas etapas de processamento do LPAProg-DDoS. Este arquivo é utilizado pela terceira etapa da arquitetura

Data	Total	TCP	SYN	SYN_ACK	FIN	ACK	RST	PSH
2020-06-19 14:48:25	2307	2254	46	54	71	1452	258	373
2020-06-19 14:48:35	3760	3759	37	37	63	3093	210	319
2020-06-19 14:48:45	1621	1602	48	45	62	795	249	403
2020-06-19 14:48:55	1114	1098	37	36	51	531	191	252
2020-06-19 14:49:05	1413	1405	41	41	71	680	250	322
2020-06-19 14:49:15	1332	1325	41	41	61	655	229	298
2020-06-19 14:49:25	50000	49998	26128	23638	10	117	37	68
2020-06-19 14:49:26	50000	50000	26285	23694	2	12	0	7
2020-06-19 14:49:27	50000	49999	26294	23691	0	7	0	7
2020-06-19 14:49:27	50000	50000	26232	23686	2	50	17	13

Figura 28: Exemplo do arquivo CSV com a contagem.
Fonte: Autoria própria (2020)

do sistema, no qual realiza os cálculos para gerar os graus de crenças e descrenças.

3.3.2 Calcular os graus de crenças e descrenças

A partir dos dados pré-processado, o sistema precisa realizar os cálculos (Equações 3.2 e 3.1) para se obter os graus de crenças e descrenças. Estes graus são de suma importância para se obter o resultado final, pois eles são os fatores de entrada para a LPA. A partir deles, é possível alcançar o estado lógico se há ou não um ataque DDoS eminente nos dados analisados da rede.

Para se obter os graus, o sistema dispõe da classe *calculate degrees*. Esta classe possui três atributos e três métodos. Os atributos são: *data*, *dgFavorable* e *dgUnfavorable*, no qual representam respectivamente os dados de entrada com a contagem de *flags* e a os dados de saídas, que são os graus de crenças e descrenças. Os métodos são: *constructor*, *open file* e *calculate SYN*, que representam respectivamente o método construtor (utilizado ao instanciar um objeto), o método responsável em abrir o arquivo que contém a contagem e armazenar os dados no atributo *data*, e o método que realiza os cálculos e armazena os resultados nos atributos *dgFavorable* e *dgUnfavorable*. O método construtor tem um parâmetro opcional, no qual pode-se atribuir os dados da contagem diretamente sem a realização da abertura do arquivo. A maior complexidade de tempo desta classe é $O(n)$ (método *calculate SYN*), em que n é a quantidade de linha contida no arquivo de entrada ou a quantidade valores passado.

Como mencionado, o método *calculate SYN* é responsável por realizar os cálculos de obtenção dos graus de crenças e descrenças. Para isso, é utilizado um conjunto de quatro equações propostas para obter-se os graus de crenças. A partir dos graus de crenças, é realizado o cálculo de $1 - \mu 1$ para adquirir-se os graus de descrenças.

A equação 3.2 apresenta a validação da fórmula para que não se tenha uma divisão por zero. Caso a base da divisão não seja zero, o resultado é dado a partir da relação da quantidade de pacotes que tenha a *flag* FIN ativa com pacotes que tenha as *flags* SYN ou SYN e ACK ativas. Está relação demonstra que uma conexão aberta será fechada, como mostrado na seção 2.2. Portanto, a equação 3.2 representa o resultado final do grau de crença, no qual ela garante que o resultado se mantenha no intervalo [0, 1].

$$y_a = \begin{cases} 0 & \text{if } S + SA = 0 \\ 1 - \frac{F}{S+SA} & \text{if } S + SA \neq 0 \end{cases} \tag{3.1}$$

$$\mu_{1a} = \begin{cases} 0 & \text{if } y_a < 0 \\ y_a & \text{if } 0 \leq y_a \leq 1 \end{cases} \tag{3.2}$$

A equação 3.3 exibe a validação para a segunda fórmula, na qual verifica se a quantidade de pacotes somente com *flag* SYN ativa é diferente de zero. Caso seja, y_b recebe a relação da quantidade de pacotes que tenham a *flag* SYN e ACK ativas com pacotes que tenha apenas a *SYN* ativa. Tal relação valida se houve uma quantidade equivalente entre as solicitações de abertura de conexão do cliente com as respostas de confirmação do servidor. Logo, a equação 3.4 é equivalente a 3.2, o que difere é que a 3.4 verifica o intervalo do resultado da equação 3.3.

$$y_b = \begin{cases} 0 & \text{if } S = 0 \\ 1 - \frac{SA}{S} & \text{if } S \neq 0 \end{cases} \tag{3.3}$$

$$\mu_{1b} = \begin{cases} 0 & \text{if } y_b < 0 \\ y_b & \text{if } 0 \leq y_b \leq 1 \end{cases} \tag{3.4}$$

Para se obter o valor do terceiro grau de crença, a fórmula foi dividida em quatro equações. A equação 3.5 representa a quantidade de dígitos do valor da quantidade de pacotes TCP. Com o valor obtido, é verificado na equação 3.6 se o mesmo é menor que dois. No qual, cria-se um limite inferior para o resultado de d, neste caso o valor é limitado em dois. Este limite é atribuído pelo fato da equação 3.7 utilizar o fator $d-2$ em sua composição. Para que o valor da subtração não seja negativo, é utilizado a equação 3.6. Entretanto, a equação 3.7 valida se o valor em SYN é diferente de zero, para evitar uma divisão por zero. Caso SYN seja diferente de zero, faz-se uma relação com a quantidade de pacotes que tenham a *flag* ACK ativa com pacotes que tenha apenas a SYN ativa. Está relação é atribuída sobre os fatore de solicitações de abertura de conexão com as troca de mensagens em conexões já estabelecidas. Para que o valor se comporte próximo do intervalo [0, 1], é realizado uma divisão da relação por 10^{d-2}. Por fim, a equação 3.8 se comporta como a equação 3.2, a diferença é que a variável analisada.

$$x = \text{Quantidade de dígitos do valor } TCP \tag{3.5}$$

$$d = \begin{cases} 2 & \text{if } x < 2 \\ x & \text{if } x \geq 2 \end{cases} \tag{3.6}$$

$$y_c = \begin{cases} 0 & \text{if } S = 0 \\ 1 - \frac{\frac{A}{S}}{10^{d-2}} & \text{if } S \neq 0 \end{cases} \tag{3.7}$$

$$\mu_{1c} = \begin{cases} 0 & \text{if } y_c < 0 \\ y_c & \text{if } 0 \leq y_c \leq 1 \end{cases} \tag{3.8}$$

A última equação para se obter o grau de crença é apresentada pela equação 3.9. Está equação, como as outras, verifica se o divisor (TCP) é diferente de zero, para que não haja erro na divisão. Caso o valor do divisor seja zero, é atribuído diretamente o resultado como zero. Caso contrário, a equação realiza a operação matemática, que relaciona a quantidade de pacotes que tem a *flag* ACK, ou FIN, ou RST, ou PSH ativa com a quantidade de pacotes TCP. Tal relação verifica se houve um pequeno tráfego de pacotes que tem a *flag* SYN ativa com relação a quantidade total de pacotes TCP.

$$\mu_{1d} = \begin{cases} 0 & \text{if } TCP = 0 \\ 1 - \frac{A+F+R+P}{TCP} & \text{if } TCP \neq 0 \end{cases} \tag{3.9}$$

A tabela 3 apresenta as descrições das principais variáveis utilizadas nas equações apresentadas.

Variável	Descrição
S	Quantidade de pacotes que tem somente a *flag* SYN ativa
SA	Quantidade de pacotes que tem somente as *flag* SYN e a *flag* ACK ativas
F	Quantidade de pacotes que tem a *flag* FIN ativa
A	Quantidade de pacotes que tem a *flag* ACK ativa
R	Quantidade de pacotes que tem a *flag* FIN ativa
P	Quantidade de pacotes que tem a *flag* PSH ativa
TCP	Quantidade de pacotes do que são do protocolo TCP
d	Quantidade de dígitos do valor TCP

Tabela 3: Descrição das variáveis do cálculo de obtenção dos graus.
Fonte: Autoria própria (2020)

As equações apresentadas são referentes a obtenção dos graus de crença. Neste caso, as equações representa qual é a crença de haver um ataque DDoS do tipo SYN *flood* nos dados de rede analisados. Tais equações são provenientes das investigações realizadas sobre o funcionamento normal de uma rede de computadores e sobre o comportamento de um ataque DDoS SYN *flood*. Estas e entre outras informações são apresentadas nas seções 2.2 e 2.3 respectivamente.

Os graus de descrenças são obtidos ao realizar a subtração de um pelo graus de crenças. As equações 3.10, 3.11, 3.12, 3.13 apresentam como é fórmula para se realizar o cálculo.

$$\mu_{2a} = 1 - \mu_{1a} \tag{3.10}$$

$$\mu_{2b} = 1 - \mu_{1b} \tag{3.11}$$

$$\mu_{2c} = 1 - \mu_{1c} \tag{3.12}$$

$$\mu_{2d} = 1 - \mu_{1d} \tag{3.13}$$

Esta seção apresentou como é realizado os cálculos para se obter os graus de crenças e descrenças. Os resultados obtidos por essa etapa são utilizados pela etapa 4, no qual serão validados com a LPA e se obter um resultado lógico sobre eles.

3.3.3 Computar com a LPA e exibir os resultados

Esta seção aborda as duas últimas etapas da arquitetura definida do sistema LPAProg-DDoS. Na qual, a etapa 4 aborda o processo de computar os graus de crença e descrença providos da etapa anterior e a etapa 5 de exibir os resultados aos administradores.

Para computar os graus, é utilizado o módulo LPAProg, apresentado na seção 3.2. O que difere, que os dados de entrada não são provenientes de um arquivo e sim passados diretamente para o componente responsável em realizar a operação.

Com os graus de crenças e descrenças obtidos, é preciso realizar a operação lógica para se obter os graus de crença e descrença resultante. Para isso, foi proposta uma operação lógica, que utiliza dos quatro graus obtidos pela etapa 3 para se obter apenas um grau. A operação lógica para se obter os graus resultantes é apresentada pela equação 3.14.

$$u_R = (u_a \text{ or } u_b) \text{ and } (u_c \text{ or } u_d) \tag{3.14}$$

O componente *logical operation* é responsável em resolver a operação lógica e retornar os graus de crença e descrença resultante. Para que posteriormente, o componente *core lpa* receba os graus resultante e realiza o processo de computá-los com a LPA e disponibilizar o resultado (estado lógico).

Os resultados são apresentados como mostrados na seção 3.2, na qual é gerado um arquivo de *log* (Figura 19) e um arquivo CSV (Figura 20). Dentre os possíveis resultados (Tabela 1), destacam-se o verdadeiro e o falso. No qual, o verdadeiro representa que a rede sofreu um ataque DDoS do tipo SYN *flood* naquela janela e o falso representa que o tráfego da janela é normal. Além disso, o baricentro pode indicar que o conjunto de janelas representam ou não um ataque. De tal modo, que o baricentro pode ser considerado como o resultado decisivo ao se tratar de um conjunto de janelas analisadas.

Como apresentado na seção 3.2, também é gerado um gráfico (QUPC) com os resultados (Figura 21), no qual os pontos vermelhos indicam ataque, os pontos verdes tráfego normal e os pontos pretos não conclusivo. Entre os pontos encontra-se um ponto maior que os demais da cor magenta, no qual representa o baricentro. Os demais pontos representados no gráfico indicam cada janela analisada.

Para a realização de testes do sistema LPAProg-DDoS foi preciso construir um ambiente de rede controlado, no qual fosse possível simular o tráfego normal e de ataque DDoS. Deste modo, o trabalho atinge a sua terceira fase, que será abordada com mais detalhes na próxima seção 3.5.

3.4 FUZPROG-DDOS

Para a construção do sistema de detecção de ataque DDoS com a Lógia *Fuzzy* (FuzProg-DDoS), foi utilizado um ambiente com arquitetura *Linux* com o a versão 5.9.16 do *kenel*. O sistema FuzProg-DDoS foi implementado utilizando a versão

3.8.3 da linguagem *Python*. Para a construção e computação da Lógica Fuzzy foi utilizado a biblioteca *SciKit Fuzzy*. Alguns componentes foram importados dos módulos LPAProg-DDoS. A tabela 4 mostra as dependências com suas respectivas versões utilizadas para a construção e execução deste módulo.

Nome	Versão
Python	3.8.3
Argparse	1.1
Matplotlib	3.3.1
NumPy	1.18.5
Pandas	1.1.0
Scapy	2.4.3
SciKit Fuzzy	0.4.2

Tabela 4: Tabela de dependências do FuzProg-DDoS
Fonte: Autoria própria (2020)

Neste sistema é incluído dois componentes do módulo LPAProg-DDoS: *sniff opcap* e *calculate degrees*. Além deles, também há os componentes *core fuzzy*, *fuzzy online* e *fuzzy offline*. Dentre os novos módulos citados, um é orientado a objeto e dois são estruturados. Os componentes estruturados (*fuzzy online* e *fuzzy offline*) são os responsáveis por inicializar o sistema. Enquanto, o orientado a objeto é responsável em realizar processar os dados. A figura 29 apresenta a estrutura dos componentes do sistema FuzProg-DDoS, no qual utiliza de outro módulo para seu funcionamento. Além disso, a figura 29 exibe a classe *core fuzzy* com seus respectivos atributos e métodos.

Os processos realizados pelo FuzProg-DDoS são parecidos com os do LPAProg-DDoS apresentados na seção 3.3. O que difere é a quarta fase que computa os graus de crenças com a Lógica *Fuzzy*. Como já mencionado anteriormente, a primeira fase é a coleta dos dados, de modo que a coleta pode ser realiza a partir de um arquivo de captura (*offline*), ou por meio da captura em tempo real (*online*). A segunda fase é a realização do pre processamento dos dados, na qual é realizada a contagem de *flags* da janela a ser analisada. A primeira e segunda fase são explicadas com mais detalhes subseção 3.3.1 do LPAProg-DDoS, uma vez que o componente utilizado para a coleta dos dados é importado deste módulo (*sniff opcap*).

Com os dados carregados e pre processados, é utilizado o componente *calculete*

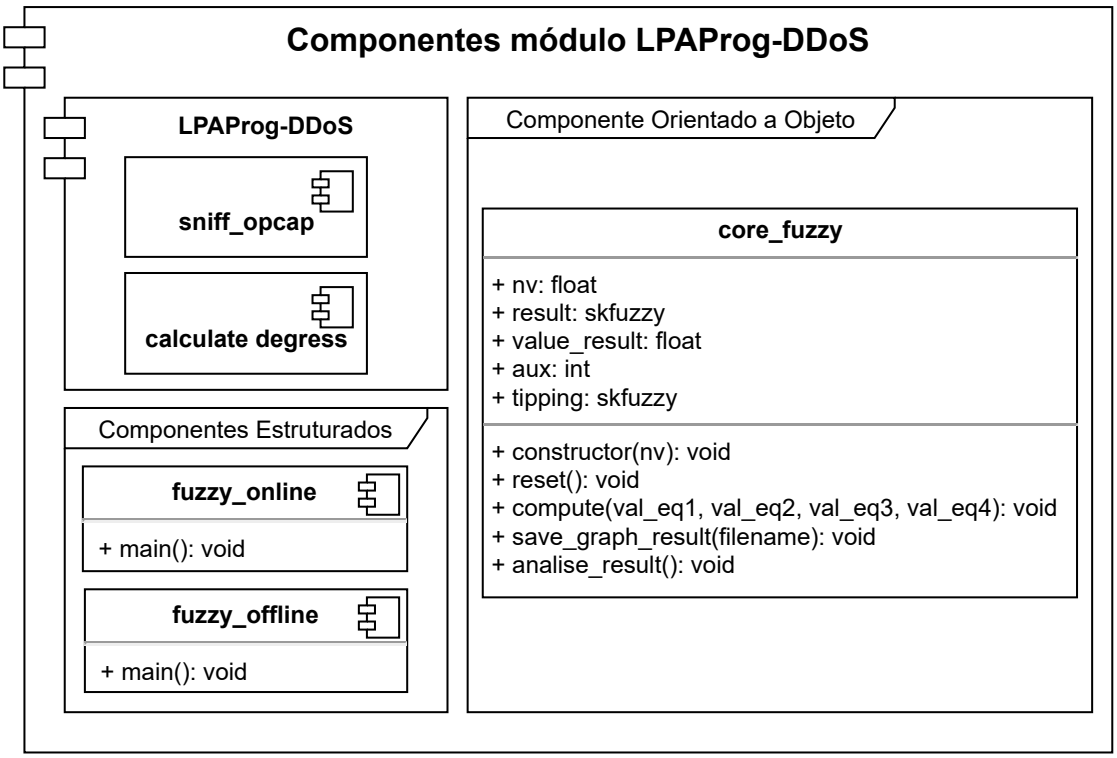

Figura 29: Componentes do módulo FuzProg-DDoS.
Fonte: Autoria própria (2021)

degrees, também importado do LPAProg-DDoS, para realizar os cálculos para se obter os graus de crenças e descrenças. O módulo FuzProg-DDoS utiliza apenas os graus de crenças, dado que a Lógica *Fuzzy* trata apenas a crença. Os detalhes como são obtidos os graus de crenças podem ser visualizados na subseção 3.3.2, na qual são explicadas as equações 3.2, 3.4, 3.8 e 3.9 que os respectivos resultados são utilizados como entrada para computar a Lógica *Fuzzy*.

A quarta fase é responsável em computar os graus de crenças a partir da Lógica *Fuzzy*. Esta fase utiliza-se do componente *core fuzzy* para realizar os cálculos. O método construtor deste componente, realiza a construção de quatro conjuntos lógicos de entrada e um conjunto lógico de saída. Além disso, é inicializada a variável de nível de exigência (N_{exig}), na qual a mesma é utilizada no interior da construção dos conjuntos. Por fim, é criado as regras a partir dos conjuntos e o sistema de controle da Lógica *Fuzzy* a partir das regras.

Para a elaboração do sistema *Fuzzy* foi utilizado quatro conjuntos de entrada, na qual são responsáveis em analisar os dados da equação 3.2 (eq1), 3.4 (eq2), 3.8 (eq3)

e 3.9 (eq4). Os conjuntos de entradas foram divididos em três termos: baixo, médio e alto. As funções de pertinência utilizadas no interior da cada conjunto foram a trapezoidal e a triangular. Em que os termos baixo e alto utilizam a representação trapezoidal e o termo médio a triangular. Os pontos no plano cartesiano da função de pertinência baixo são: $A = (0, 0)$, $B = (0, 1)$, $C = (1 - N_{exig}, 1)$ e $D = (0.5, 0)$. Os pontos da função médio são: $A = (1 - N_{exig}, 0)$, $B = (0.5, 1)$ e $C = (N_{exig}, 0)$. Por fim, os pontos da função alto são: $A = (0.5, 0)$, $B = (N_{exig}, 1)$, $C = (1, 1)$ e $D = (1, 0)$. Os pontos são interligados por uma linha reta em cada fator na ordem apresentada. A figura 30 apresenta o exemplo o conjunto eq1 com o nível de exigência igual a 0,65, os conjuntos eq2, eq3 e eq4 são iguais visualmente ao eq1.

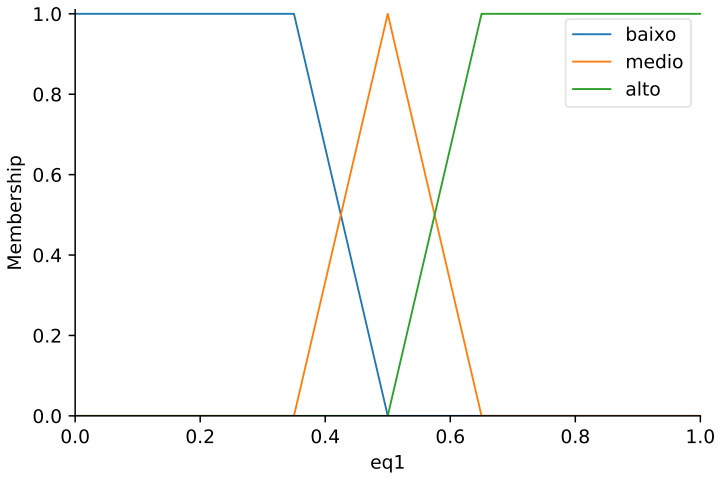

Figura 30: Exemplo do conjunto de entrada com o nível de exigência igual a 0,65.
Fonte: Autoria própria (2021)

Além dos conjuntos de entrada, o sistema *Fuzzy* contém um conjunto de saída, na qual corresponde a dois fatores: ataque e normal. O fator ataque representa que a rede pode estar sobre um ataque na janela analisada. Por outro lado o fator normal representa a normalidade da rede. Ambos os fatores são representados pelo modelo trapezoidal. Na qual, os pontos no plano cartesiano da função de pertinência normal são: $A = (0, 0)$, $B = (0, 1)$, $C = ((1 - N_{exig}) \cdot 100, 1)$ e $D = (N_{exig} \cdot 100, 0)$. Em contrapartida, os pontos da função ataque são: $A = ((1 - N_{exig}) \cdot 100, 0)$, $B = (N_{exig} \cdot 100, 1)$, $C = (100, 1)$ e $D = (100, 0)$. A escala do eixo x neste caso é de 0 a 100. Os pontos são interligados por uma linha reta em cada fator na ordem apresentada. A figura 31 exibe a representação gráfica do conjunto de saída (*result*)

com o valor do nível de exigência em 0,65.

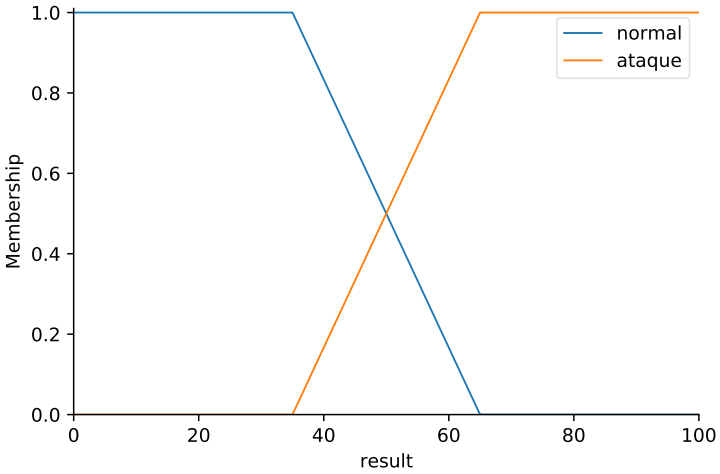

Figura 31: Exemplo do conjunto de saída com o nível de exigência igual a 0,65.
Fonte: Autoria própria (2021)

Como mencionado, o método construtor também instância as regras que são realizadas para se obter o resultado final. Foram elaboradas quatro regras, na qual duas são designadas ao tráfego normal (Equações 3.17 e 3.18) e duas a um ataque (Equações 3.15 e 3.16). As regras foram obtidas de modo empírico a partir de testes e ajustes. Vale ressaltar que as regras não garantem o resultado final, na qual elas são utilizadas pelo processo de inferência na etapa de *fuzzificação*. Com o resultado desta etapa é possível realizar a *deffuzificação* para interpretar as informações obtidas nesta etapa e obter uma saída precisa.

$$\begin{aligned} &\text{se } ((eq1 \text{ é } alto)\ OR\ (eq2 \text{ é } alto))\ AND \\ &((eq3 \text{ é } alto)\ OR\ (eq4 \text{ é } alto)) \\ &\text{então } (result \text{ é } ataque) \end{aligned} \tag{3.15}$$

$$\begin{aligned} &\text{se } ((eq1 \text{ é } alto)\ OR\ (eq2 \text{ é } alto))\ AND \\ &((eq3 \text{ é } baixo)\ OR\ (eq4 \text{ é } baixo)) \\ &\text{então } (result \text{ é } ataque) \end{aligned} \tag{3.16}$$

$$\begin{aligned} &\text{se } ((eq1 \text{ é } medio)\ OR\ (eq2 \text{ é } medio))\ AND \\ &((eq3 \text{ é } medio)\ OR\ (eq4 \text{ é } medio)) \\ &\text{então } (result \text{ é } normal) \end{aligned} \tag{3.17}$$

$$\begin{aligned} &\text{se } ((eq1 \text{ é } baixo)\ OR\ (eq2 \text{ é } baixo))\ AND \\ &((eq3 \text{ é } baixo)\ OR\ (eq4 \text{ é } baixo)) \\ &\text{então } (result \text{ é } normal) \end{aligned} \tag{3.18}$$

O método *compute* é responsável em computar as entradas e gerar o resultado numérico a partir da *deffuzificação*. Este método então recebe os resultados das quatro equações e adiciona ao sistema de controle criado no método construtor. Com isso, é chamado o método *compute* do sistema de controle (biblioteca *skfuzzy*), que realiza as etapas do método Mamdani. O modelo de *deffuzificação* utilizado por este método é o centro de gravidade (centroide). Com isso, é armazenado o resultado numérico em uma variável.

Após computar a Lógica *Fuzzy* é preciso gerar o resultado final, na qual é analisado o resultado obtido na *deffuzificação* e retornado a resposta se é ou não um ataque, ou se é não conclusivo. O método *analise result* é o responsável em realizar esta análise. Este método então verifica qual intervalo o resultado se encontra, para então retornar o resultado final. A equação 3.19 apresenta os intervalos e os resultados possíveis, na qual o *value result* representa o valor obtido pela *defuzzificação*. O método *analise result* também gera um arquivo *log* contendo a data e hora do início da janela e o resultado textual e numérico (0 para normal, 1 para ataque e 2 para não conclusivo).

$$Result = \begin{cases} \text{Normal} & \text{if } value\_result \leq (1 - N_{exig}) \cdot 100 \\ \text{Não Conclusivo} & \text{if } (1 - N_{exig}) \cdot 100 < value\_result < N_{exig} \cdot 100 \\ \text{Ataque} & \text{if } value\_result \geq N_{exig} \cdot 100 \end{cases} \tag{3.19}$$

No *core lpa* também encontra-se o método **save graph result**, no qual é responsável em salvar o gráfico com a porcentagem das regras de saída e também a linha do centroide. As figuras 32 e 33 apresentam um exemplo do gráfico de saída após a *defuzzificação*, no qual a figura 32 representa uma analise voltada para o tráfego normal e a figura 33 para ataque. Para que fosse possível salvar os gráficos, foi preciso adicionar o método *save* no interior da biblioteca *skfuzzy*.

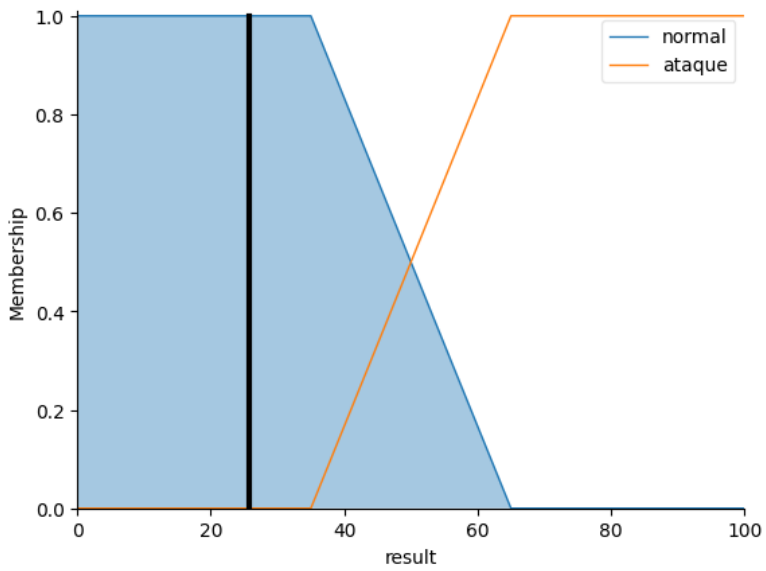

Figura 32: Exemplo da saída *deffuzificada* com o nível de exigência igual a 0,65 e o resultado tende a ser tráfego normal.
Fonte: Autoria própria (2021)

Para a realização dos testes do sistema FuzProg-DDoS, foram utilizados os mesmos ambientes do sistema LPAProg-DDoS, com intuito de realizar a comparação dos resultados e tempo de processamento de ambos os sistemas. Nos testes prévios, o sistema se mostrou promissor para realizar a detecção de ataque DDoS do tipo SYN *flood*.

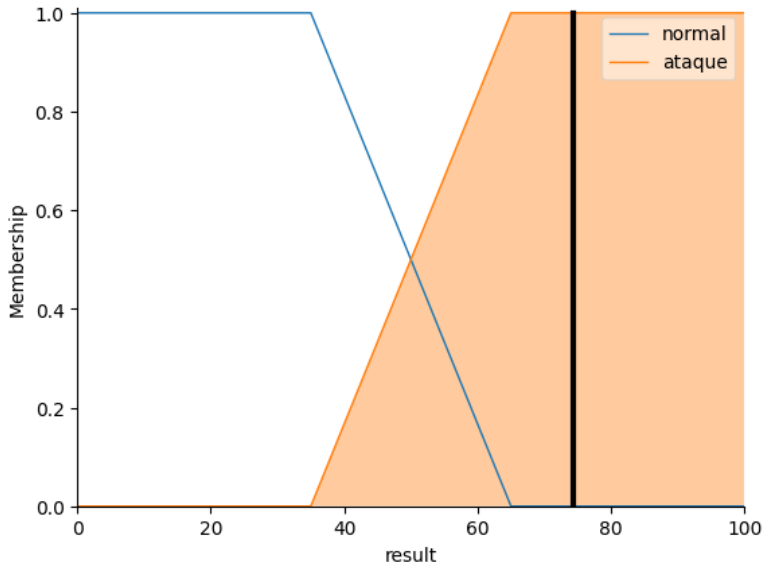

Figura 33: Exemplo da saída *deffuzificada* com o nível de exigência igual a 0,65 e o resultado tende a ser ataque.

Fonte: Autoria própria (2021)

3.5 AMBIENTES DE TESTES

Esta seção abrange os ambientes utilizados e construídos para realizar os testes e validar o sistema LPAProg-DDoS. Além de expor as bases de dados, topologias utilizadas e as configurações dos equipamentos, esta seção informa sobre as ferramentas utilizadas para a construção de um cenário virtual e também as ferramentas empregadas para realizar o ataque DDoS e simular o tráfego normal de uma rede. Além disso, é exibida as configurações aplicadas na execução do sistema LPAProg-DDoS. Por fim, é apresentado os testes do sistema realizados nos ambientes e cenários descritos no decorrer desta seção.

3.5.1 Ferramentas

Para realizar a simulação de um ataque DDoS e do tráfego normal da rede, optou-se por recorrer as ferramentas já implementadas que estão a disposição para testes. Está subseção apresenta as ferramentas utilizadas neste trabalho com a finalidade realizar a simulação.

3.5.1.1 Wget

O wget (WGET, 2020) é uma ferramenta livre que possibilita que usuários realizam o *download* de dados da Internet. Está ferramenta suporta os protocolos HTTP, HTTPS e FTP. Com ela é possível simular a solicitação e o acesso do usuário a um servidor *web* (site). No qual, quando solicitado o acesso a uma página *web*, o wget realiza o *download* do arquivo HTML, de forma similar a requisição de acesso realizada pelo navegador *web*. De tal modo, o acesso pode ser redirecionado para a porta desejada.

3.5.1.2 Iperf

O iperf (IPERF, 2020) é uma ferramenta utilizada para testar largura de banda, de tal modo que pode-se escolher qual protocolo utilizar nos pacotes, TCP ou UDP. Por se tratar de uma ferramenta para medir o desempenho de rede de computadores, pose-se utilizá-la para realizar a simulação de um alto fluxo de dados. Pelo fato do DDoS *flood* enviar um grande volume de dados, com esta ferramenta pode-se analisar um tráfego normal intenso.

3.5.1.3 T50

A T50 (T50, 2020) é uma ferramenta de *pentest* utilizada para gerar ataques DoS ou DDoS. Esta ferramenta conta com diversos recursos para realizar a inundação de um dispositivo com uma extensa quantidade de requisições de pacotes. De tal forma, é possível congestionar a vítima, de modo que fique lenta a entrega do serviço, ou mesmo, impeça que usuários legítimos acessem o servidor. Além disso, esta ferramenta realiza ataque DoS do tipo SYN *flood*.

3.5.1.4 Hping3

A ferramenta hping (HPING, 2020) é parecida com a T50, e tem o mesmo objetivo de gerar ataque DoS. Esta ferramenta é utilizada para a realização de

auditoria de segurança e teste de *firewalls* e redes. Além disso, entre os tipos de ataques DoS realizado por essa ferramenta, encontra-se o ataque do tipo SYN *flood*. A versão utilizada neste trabalho é o hping3.

3.5.2 Base de dados Bot-IoT

A base de dados *Bot IoT* (MOUSTAFA, 2019) foi criada a partir de um ambiente de rede realista. Ela foi construída no *Cyber Range Lab* do *The center of UNSW Canberra Cyber*. Além disso, a base de dados incorpora tráfego de rede IoT legítimo e simulado, junto com vários tipos de ataques. Também, esta base passou por avaliações estatísticas e de aprendizagem de máquina para fins forenses em comparação com os conjuntos de dados de referência. A figura 34 apresenta o ambiente construído para a criação da base de dados Bot-IoT.

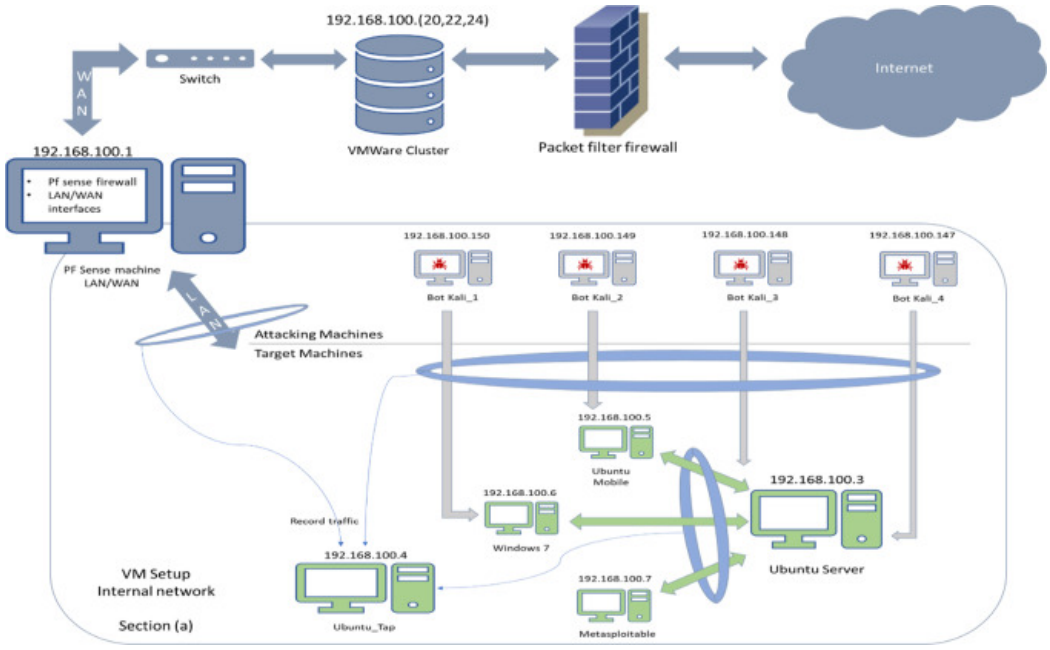

Figura 34: Ambiente de teste da base de dados Bot-IoT.
Fonte: (MOUSTAFA, 2019)

Os arquivos de origem da base de dados são fornecidos em diferentes formatos, incluindo os arquivos *pcap* originais, os arquivos *argus* gerados e os arquivos CSV. Além disso, os arquivos foram separados com base na categoria e subcategoria de ataque, para melhor auxiliar no processo de rotulagem. Entre as categorias encontra-se o ataque DDoS, no qual há a subcategoria SYN *flood*.

Esta base de dados, a *Bot-IoT*, adéqua-se ao escopo do trabalho, pois nela encontra-se o ataque DDoS do tipo SYN *flood*. Com isso, é possível utilizá-la para realizar testes no sistema LPAProg-DDoS e validar os seus resultados.

3.5.3 Cenário Real

Este cenário foi construído no ambiente da Universidade Federal Tecnológica Federal do Paraná campus Santa Helena (UTFPR-SH). Os equipamentos utilizados foram disponibilizado para o Grupo de Estudo em Redes e Segurança (GENETSEC), no qual acarretou o desenvolvimento deste trabalho.

Para a construção do cenário, foi utilizado dois servidores que contêm o processador Intel Xeon com 16 CPUs de 2.40 GHz e a arquitetura de 64 bits. Além disso, os servidores são equipados com 32 GB de memória RAM e utilizam o sistema operacional Proxmox 6.1, no qual é responsável em gerenciar um ambiente de virtualização de servidor. Um servidor contém seis máquinas virtuais, enquanto o outro servidor contém cinco *containers*, totalizando onze máquinas virtuais que são utilizadas.

Além dos servidores, o cenário contém dois computadores e um *notebook*. No qual, os dois computadores contêm o processador Intel Core i5-6400 com 4 CPUs de 2.7 0GHz com a arquitetura de 64 bits, 16 GB de memória ram e o sistema operacional Ubuntu 18.04 LTS (*kernel* 5.3.0). O *notebook* contém um processador Intel Core i3-5051U com 4 CPUs de 2.10 GHz com a arquitetura de 64 bits, 8 GB de memória ram e o sistema operacional Manjaro 21.0.4 (*kernel* 5.10.34).

Um dos computadores ficou responsável em realizar a função de servidor de hospedagem *web*, e o outro computador é responsável em realizar a captura dos pacotes e analisá-los. Para que o computador recebesse todo o tráfego da rede, o *switch* gerenciável foi configurado para espelhar os tráfegos de todas as portas para a interface que é ligada ao computador. Dois servidores ficaram responsável em realizar os acessos normais e ataques, com as suas máquinas virtuais, ao servidor *web*. O *notebook* teve a responsabilidade de acessar uma máquina virtual a partir de

um IP externo, para então acessar os demais dispositivos e gerenciar todo o processo de teste. A figura 35 apresenta o cenário com as suas respectivas ligações.

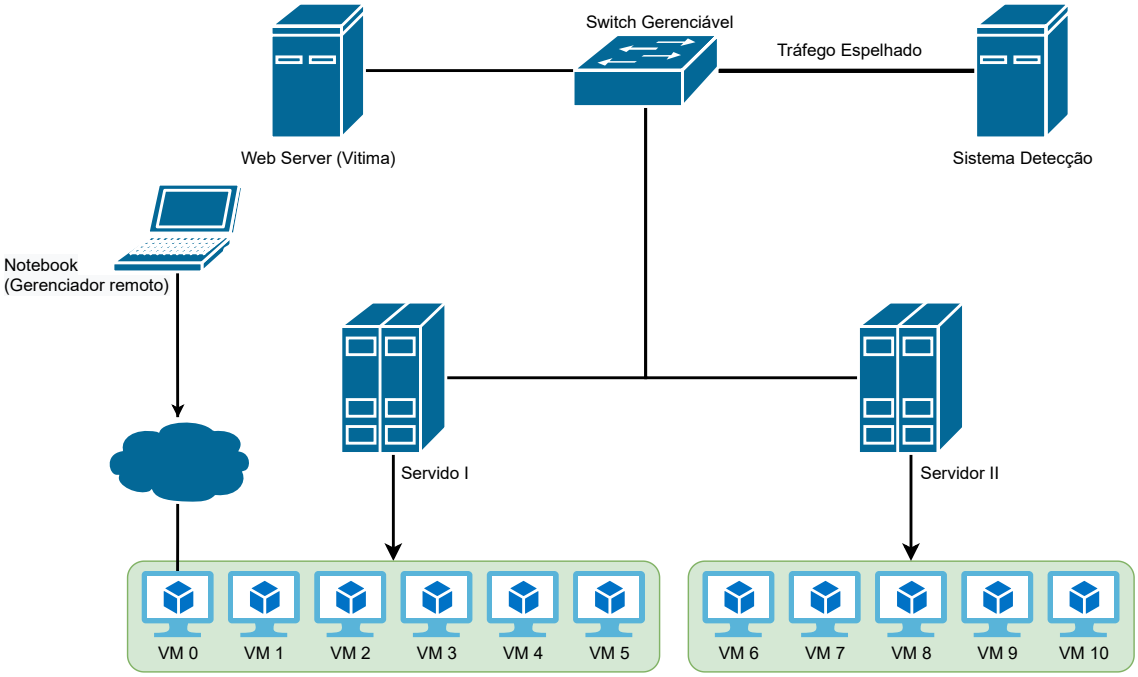

Figura 35: Ambiente de teste do cenário real.
Fonte: (MOUSTAFA, 2019)

Para a realização de testes posteriores, os dados foram gravados em um arquivo *pcap*, de modo que uma base de dados. Esta base de dados contém um intervalo de dez minutos de captura, no qual foi divido os processos de acesso normal e ataque por intervalos de tempo. A tabela 5 apresenta a descrição dos intervalos de tempo realizados pelo processo de criação da base de dados a partir do cenário real. O processo inicia-se no tempo zero, no qual nos dois minutos iniciais é realizado apenas o tráfego normal, com o acesso simultâneo de dez dispositivos. Posteriormente, é iniciado o ataque, no qual é inserido um novo atacante a cada trinta segundos, totalizando dez dispositivos atacantes. Aos oito minutos e trinta segundos de captura é dado inicio a pausa dos ataques, de forma que a cada trinta segundos é parado um ataque. Aos treze minutos não há mais ataques na rede. Entretanto aos quatorze minutos é aumentado o fluxo de acesso normal do iperf, intensificando o uso da rede. Logo aos quinze minutos é inicializado os dez ataques simultâneos, com duração de quatro minutos. Por fim, aos vinte minutos a captura de pacotes é encerrada.

Os ataques realizados pelas ferramentas hping3 e T50 foram do tipo SYN *flood*,

Tempo	Descrição
00:00	Início: Cinco acessos normais com o wget Cinco acessos normais com o iperf
02:00	Início de um ataque com T50
02:30	Um novo ataque com hping3
03:00	Um novo ataque com T50
03:30	Um novo ataque com hping3
04:00	Um novo ataque com T50
04:30	Um novo ataque com hping3
05:00	Um novo ataque com T50
05:30	Um novo ataque com hping3
06:00	Um novo ataque com hping3
06:30	Um novo ataque com hping3
08:30	Para um ataque
09:00	Para mais um ataque
09:30	Para mais um ataque
10:00	Para mais um ataque
10:30	Para mais um ataque
11:00	Para mais um ataque
11:30	Para mais um ataque
12:00	Para mais um ataque
12:30	Para mais um ataque
13:00	Para mais um ataque Todos os ataques pararam
14:00	Aumenta o fluxo de acesso do iperf
15:00	Inicia todos os dez ataques
19:00	Para todos os dez ataques
20:00	Fim da captura

Tabela 5: Descrição dos intervalos de tempo da base de dados do cenário real.
Fonte: Autoria própria (2020)

de modo que os dados gerados possam ser testados no sistema LPAProg-DDoS. Além disso, este cenário tem a vantagem de ser realizado com equipamentos físicos reais. Apesar de haver máquinas virtuais, os servidores têm uma configuração que suporta uma grande carga de processamento, junto com um sistema operacional específico para se trabalhar com virtualização de servidor. Inclusive, o *switch* físico suporta uma grande carga de tráfego de rede, junto com o computador que representa o servidor *web*. Com isso, o ataque não satura totalmente a rede, desta forma a base de dados, mesmo em momento de ataque, há um grande volume de tráfego normal.

3.5.4 Aplicação dos Testes

Nesta sessão, foram realizados os testes de aplicação dos algoritmos, programas e ambientes, o qual seguiu a seguinte ordem:

- Testes com a ferramenta wget: foi verificado se a comunicação para HTTP funcionou corretamente, o qual reportou OK;
- Teste com a ferramenta iperf: foi verificado se a troca de mensagens dos protocolos UDP ou TCP funcionaram corretamente, o qual reportou OK;
- Testes com a ferramenta T50: foi verificado se a o ataque provocado pelo programa foi obtido com sucesso. Houve apenas um problema de funcionamento em ambiente de *container*, entretanto foi optado por fazer em máquinas virtuais sem a utilização do *container* para tal, qual reportou OK;
- Testes com a ferramenta hping3: Os testes aplicados com o hping3 foram baseados em *SYN flood*, o qual reportou OK;
- Base de dados Bot-IoT: Esta base foi testada com apenas alguns arquivos menores de *pcap*, o qual os sistemas LPAProg-DDoS e FuzProg-DDoS obtiveram resultados satisfatórios e relevantes para iniciar os experimentos reais;
- Cenário real: Foi verificado se os sistemas LPAProg-DDoS e FuzProgDDoS capturava os pacotes, analisavam e geravam os resultados, o qual reportou OK.

Estes testes foram necessários para que possamos ter a real dimensão do que é possível ou não nas aplicações dos algoritmos LPA e Lógica *Fuzzy*.

3.6 MELHORIAS NOS SISTEMAS

A partir dos testes realizados na base de dados Bot-IoT (3.5.2) observou um tempo relevante para a execução do pré processamento dos pacotes. Com isso, optou-se com a melhoria dos sistemas de modo a realizar os testes no cenário real

(3.5.3). A melhoria realizada foi a paralelização do sistema com o auxilio de *threads*. Na qual, torna-se a captura uma *thread* independente, e as demais etapas do sistema (pré processar, calcular os graus, computar e exibir os resultados) uma outra *thread*.

Para isso, foi realizado modificações nos componentes tanto para o sistema LPAProg-DDoS, quanto para o FuzProg-DDoS. A imagem 37 apresenta a nova distribuição dos componentes para o LPAProg-DDoS. De modo que foi adicionado o componente sniff_online, que é responsável em fazer a captura de pacotes de modo *online*, ou seja, captura os pacotes que estão em tráfegos em uma determinada interface. Quando é finalizado, a captura é adicionado em uma fila de janelas, na qual quando ao realizar o pré processamento é consumido o primeiro elemento da fila. Quando um novo elemento é adicionado a fila, a etapa de pré processamento será realizada e passa os dados para as próximas etapas.

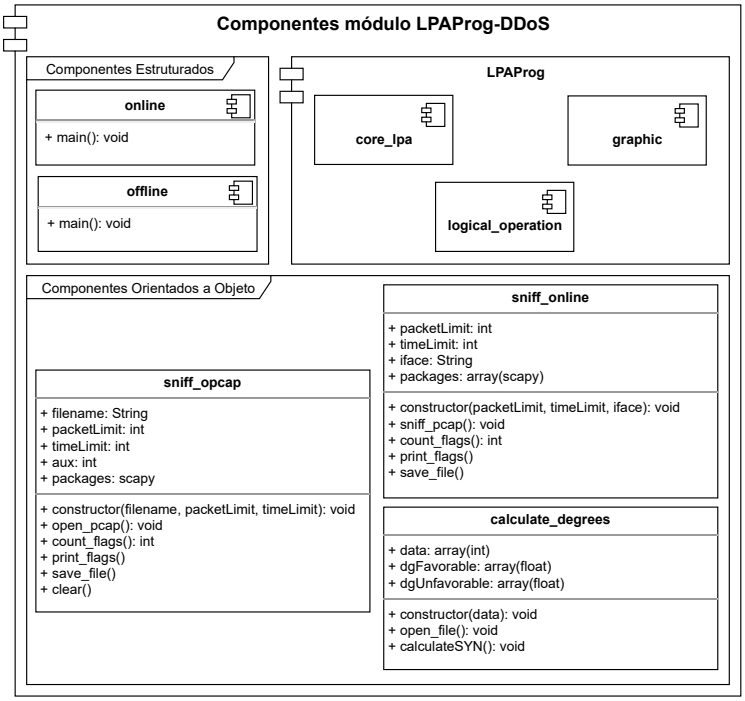

Figura 36: Nova distribuição dos componentes do módulo LPAProg-DDoS.
Fonte: Autoria própria (2020)

As melhorias realizadas permitiram uma melhor eficiência dos sistemas. Na qual ao realizar a captura no modo online, o sistema perderá menos pacotes, pois enquanto estiver realizado o pré processamento e as etapas seguintes, o sistema já estará coletando novamente os pacotes em paralelo.

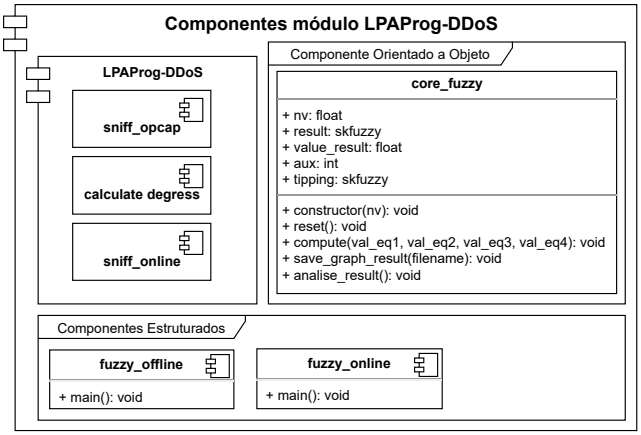

Figura 37: Nova distribuição dos componentes do módulo LPAProg-DDoS.
Fonte: Autoria própria (2020)

4 ANÁLISE DE RESULTADOS

Neste capítulo foram analisadas as eficiências de tempo e de acerto do dos sistemas LPAProg-DDoS e o FUZProg-DDoS. Para a análise de tempo, foi considerado o tempo de demora do algoritmo desde o seu início até sua finalização em cada rodada. De tal modo que, foram analisados os tempos de abertura do arquivo (para testes *offline*), o tempo para pré-processar os dados, o tempo para realizar os cálculos de obtenção dos graus de crença e descrença, o tempo para computar e por fim o tempo para gerar os resultados. Para a avaliação dos algoritmos em relação ao acerto, foram aplicadas as seguintes métricas: acurácia, precisão, *recall* e *F1-Score*.

Para o cálculo das métricas foi utilizada a matriz de confusão. Na qual, esta matriz é uma tabela que indica os acertos e erros do sistema. De modo que é comparado os resultados obtidos com os resultados esperados (*labels*). A figura 38 apresenta um exemplo de uma matriz de confusão binária. Os valores verdadeiros positivos (VP) indicam que a classificação dada pelo sistema como positiva está correta. Os falsos negativos (FN), considerados como erro do tipo II, indicam que o sistema classificou como negativo o dado que é positivo. Já os falsos positivos (FP), erro do tipo I, apresentam que o dado classificado errado como positivo é negativo. Por fim, os verdadeiros negativos (VN), indicam que o dado classificado como negativos foram classificado corretamente (POWERS, 2020).

		Predito	
		Positivo	**Negativo**
Real	**Positivo**	Verdadeiro Positivo (VP)	Falso Negativo (FN)
	Negativo	Falso Positivo (FP)	Verdadeiro Negativo (VN)

Figura 38: Exemplo da matriz de confusão.
Fonte: Adaptado de (POWERS, 2020)

Com base na matriz de confusão é possível obter os resultados da acurácia, da precisão, do *recall* e do *F1-Score*. A acurácia é uma métrica que indica a performance

geral do sistema, ou seja, dentre todas as classificações, quantas foram classificadas corretamente (POWERS, 2020). A equação 4.1 exibe a fórmula matemática para se obter a acurácia. Já a precisão, analisa a quantidade de valores positivos que o sistema classificou corretamente. A equação 4.2 apresenta a fórmula matemática para se obter a precisão (JUNIOR, 2008). Por outro lado, o *recall* analisa inversamente comparado a precisão, na qual verifica quantos positivos são esperados e quantos o sistema classificou corretamente (JUNIOR, 2008). A equação 4.3 exibe a fórmula matemática para se obter o *recall* (JUNIOR, 2008). Por fim, o *F1-Score* calcula a média harmônica entre precisão e *recall*, como mosra a equação 4.4 (POWERS, 2020).

$$Acuracia = \frac{VP + VN}{VP + VN + FP + FN} \tag{4.1}$$

$$Precisao = \frac{VP}{VP + FP} \tag{4.2}$$

$$Recall = \frac{VP}{VP + FN} \tag{4.3}$$

$$F1Score = 2 \cdot \frac{Precisao \cdot Recall}{Precisao + Recall} \tag{4.4}$$

Os testes foram realizados em um computador com o sistema operacional Linux Ubuntu 18.04.4 LTS com a versão 5.3.0 do *kernel*. Na qual, contém um processador Intel(R) Core(TM) i5-6400 de 2,70GHz com 4 núcleos. Além disso, o computador tem um total de 16GB de memória RAM e 1TB de disco rígido (do inglês *Hard Disc* – HD).

Este capítulo é dividido em duas seções: Base de dados Bot-IoT e Cenário real. Na qual a primeira seção representa os testes, resultados e análises provenientes na base de dados Bot-IoT apresentado na subseção 3.5.2. A segunda seção trata-se dos testes, resultados e analises provenientes do cenário real apresentado na subseção

3.5.3. No interior de cada seção também são apresentados os resultados separadamente entre os sistemas LPAProg-DDoS e FuzProg-DDoS. Além disso, é apresentado as configurações atribuídas a ambos os sistemas.

4.1 BASE DE DADOS BOT-IOT

A base de dados Bot-IoT é apresentada na subseção 3.5.2, no qual em seu interior encontram-se trechos de ataque DDoS do tipo SYN *flood*. Para a realização dos testes foram realizados os *downloads* dos arquivos no formato PCAP que encontram-se no diretório PCAPs/DDoS/DDoS_TCP[1], onde ao total foram baixados 72 arquivos, em que a média de tamanho de cada arquivo é 215 MB, totalizando 15,1 GB. Deste modo, os arquivos foram analisados separadamente, de tal forma que fosse possível realizar a sua abertura na memória RAM. O tempo total da captura é de 2376 segundos, ou seja, 39 minutos e 36 segundos.

As configurações dos algoritmos de análise foram atribuídas de modo a gerar janelas[2] com 10 segundos de tráfegos e o nível de exigência igual a 0,65. Por se tratar de analisar cada arquivo separadamente, houve janelas que tiveram um tempo menor que 10 segundos. No qual, são janelas construídas a partir do final do arquivo analisado. No total, foram geradas e analisadas 278 janelas. A figura 39 apresenta o gráfico de tempo das 277 primeiras janelas em segundos, na qual os pontos que estão zerados indicam um valor menor que 1 segundo e os pontos acima de 10 segundos representam que o valor foi arrendondado para um valor superior. A figura 40 exibe o *boxplot* da variação do tempo das janelas.

As etapas realizadas pelos algoritmos são: 1) abertura do arquivo; 2) pré processamento; 3) cálculos de grau de crença e descrença; 4) computar e 5) exibir os resultados. As três primeiras etapas são realizadas igualmente por ambos os algoritmos, LPAProg-DDoS e FuzProg-DDoS. Com isso, apresenta-se apenas um gráfico de tempo para estas etapas.

[1]Link da base de dados: <https://cloudstor.aarnet.edu.au/plus/s/umT99TnxvbpkkoE>. Acesso em: 5 abr. 2021.

[2]Termo de gerenciamento de rede que designa a parte da captura de pacotes que atingiu o limite de tempo ou encerrou o arquivo analisado

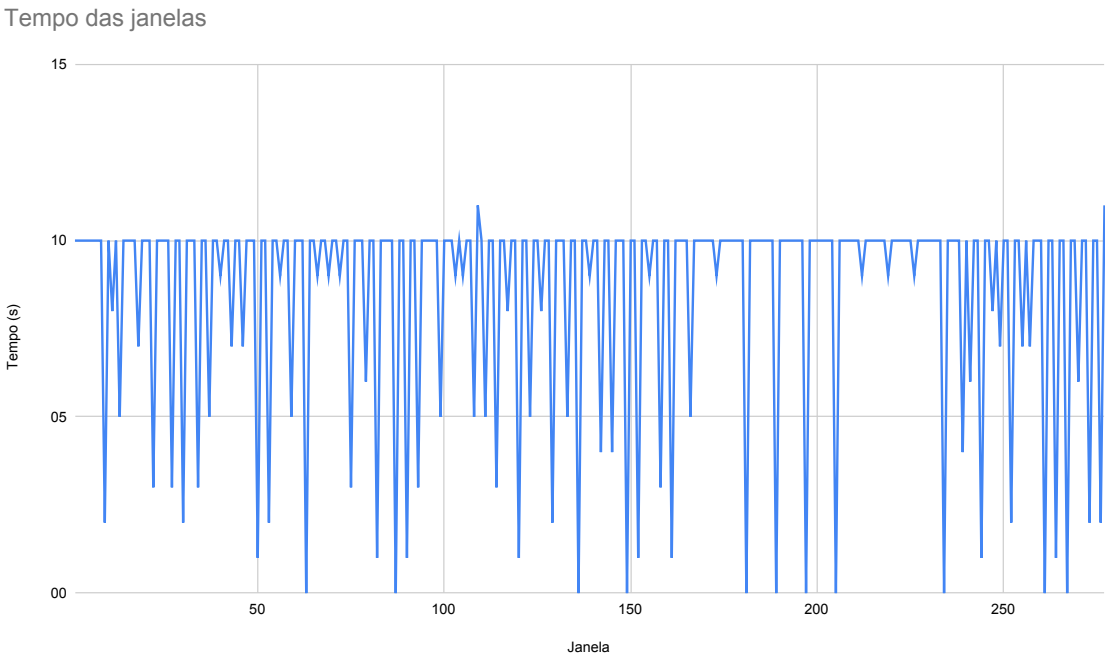

Figura 39: Gráfico de tempo das janelas geradas pelo sistema.
Fonte: Autoria própria (2021)

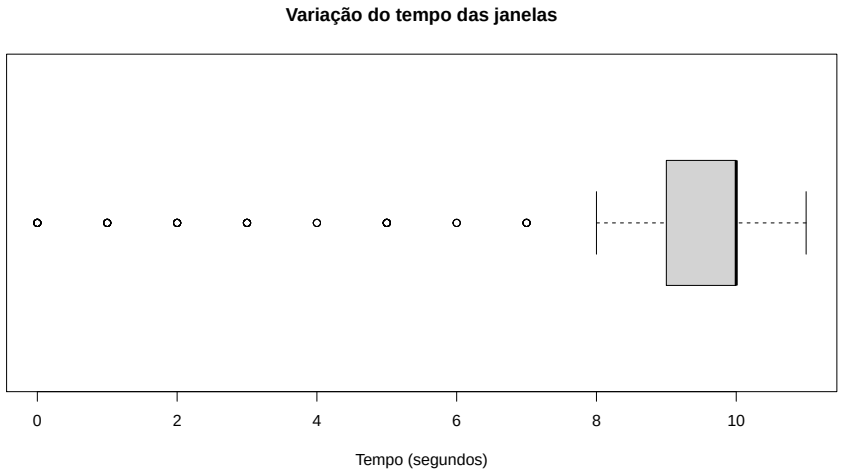

Figura 40: *Boxplot* da variação do tempo das janelas.
Fonte: Autoria própria (2021)

A etapa de abertura do arquivo, realiza a leitura do arquivo e armazena os dados na memória RAM. A variação de tempo desta etapa diverge de acordo com o tamanho do arquivo a ser aberto e também pela quantidade de pacotes existentes no arquivo. A média de tempo gasta para os 72 arquivos da base de dados Bot-IoT foi de aproximadamente 397 segundos. Na qual, esta é a etapa mais demorada pelos

sistemas. A figura 41 apresenta a evolução do tempo gasto para a abertura de cada arquivo. Entretanto, a figura 42 apresenta o *boxplot* da variação de tempo com base em cálculos estatísticos.

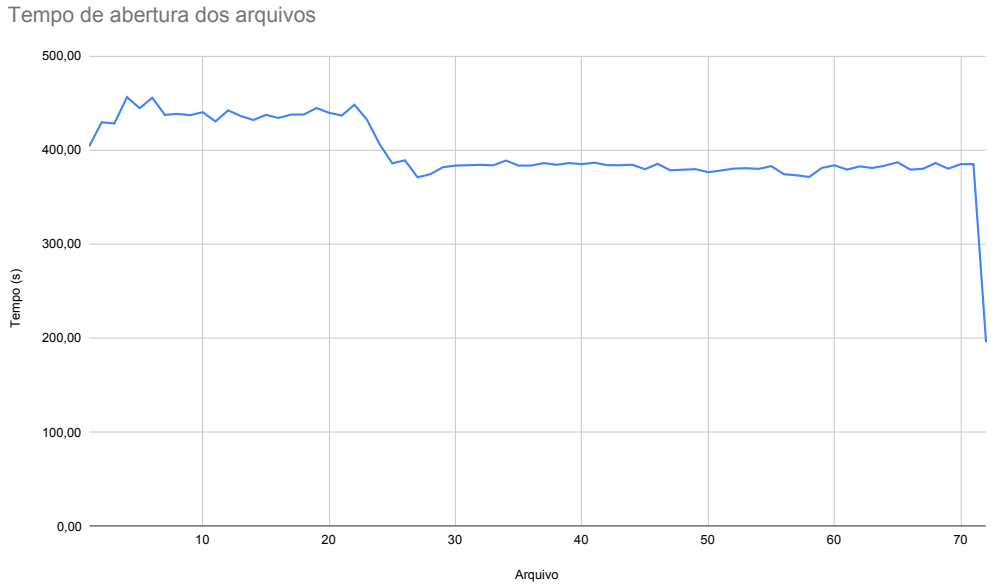

Figura 41: Gráfico de tempo gasto para abertura dos arquivos.
Fonte: Autoria própria (2021)

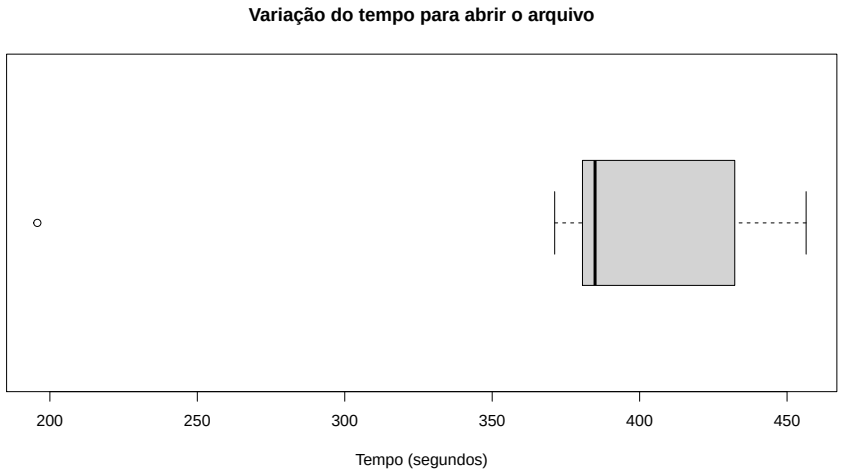

Figura 42: *Boxplot* da variação do tempo para abrir os arquivos.
Fonte: Autoria própria (2021)

A próxima etapa realizada é o pré processamento. Nesta etapa é realizada a quebra do arquivo aberto em janelas, que neste caso foi atribuído apenas limitação por tempo (10 segundos). Além disso, o pré processamento realiza a contagem de

pacotes para cada uma das *flags* e de pacotes TCP. A média de tempo gasto para pré processar as 278 janelas foi de 7,28 segundos. A figura 43 apresenta a evolução do tempo gasto para realizar o pré processamento de cada janela. No entanto, a figura 44 apresenta o *boxplot* com a variação de tempo com base em cálculos estatísticos.

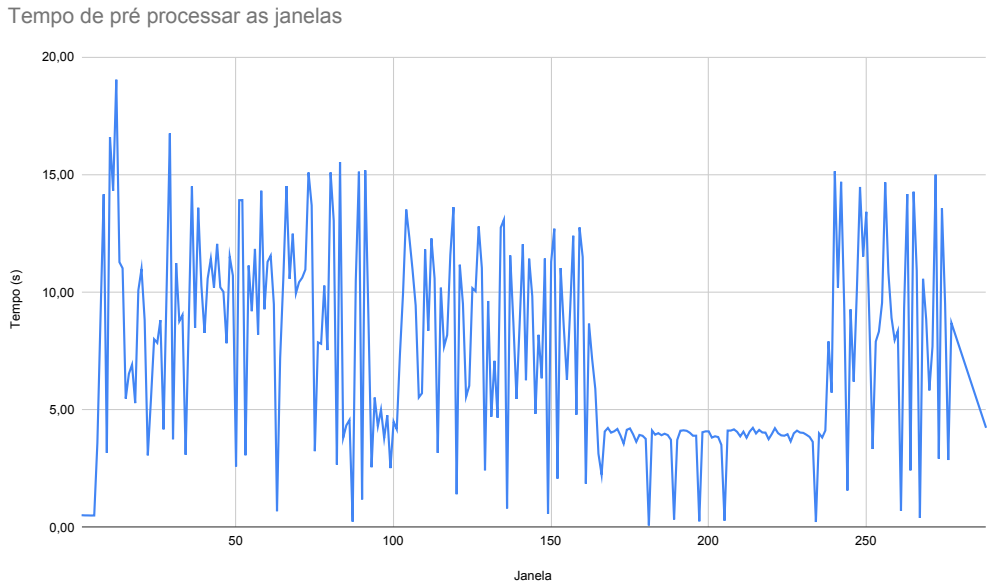

Figura 43: Gráfico de tempo gasto para pré processar as janelas.
Fonte: Autoria própria (2021)

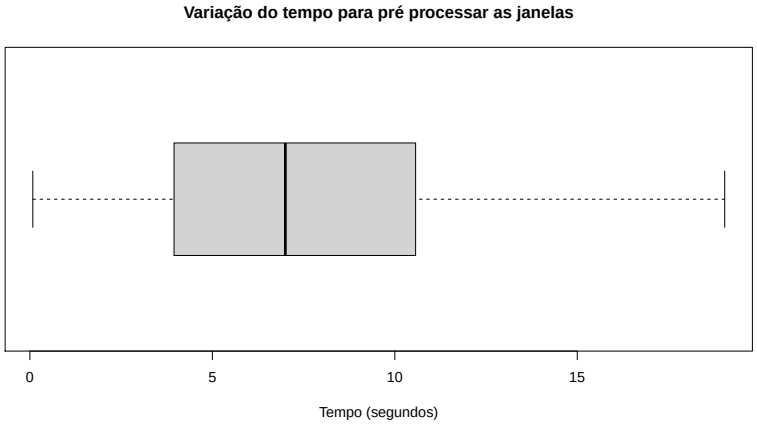

Figura 44: *Boxplot* da variação do tempo para pré processar as janelas.
Fonte: Autoria própria (2021)

A última etapa em comum entre os sistemas LPAProg-DDoS e FuzProg-DDoS é o cálculo dos graus de crenças e descrenças. Logo, esta etapa utiliza-se dos dados pré processados e realiza quatro equações matemáticas, obtendo os graus. Esta etapa

exige pouco de processamento, com isso obteve um tempo de execução baixo, na qual a média do tempo gasto da execução das 278 janelas foi de 0,0119 segundos. O desvio padrão desta etapa foi de apenas 0,0005 segundos, logo uma baixa variação no tempo de execução. A figura 45 apresenta a evolução do tempo gasto para realizar o cálculo dos graus de cada janela. Observa-se que apenas quatro janelas tiverem o tempo acima de 0,0130 segundos e as demais se mantiveram em um intervalo acima de 0,0110 segundos até 0,0130 segundos. A figura 46 apresenta o *boxplot* com a variação de tempo com base em cálculos estatísticos.

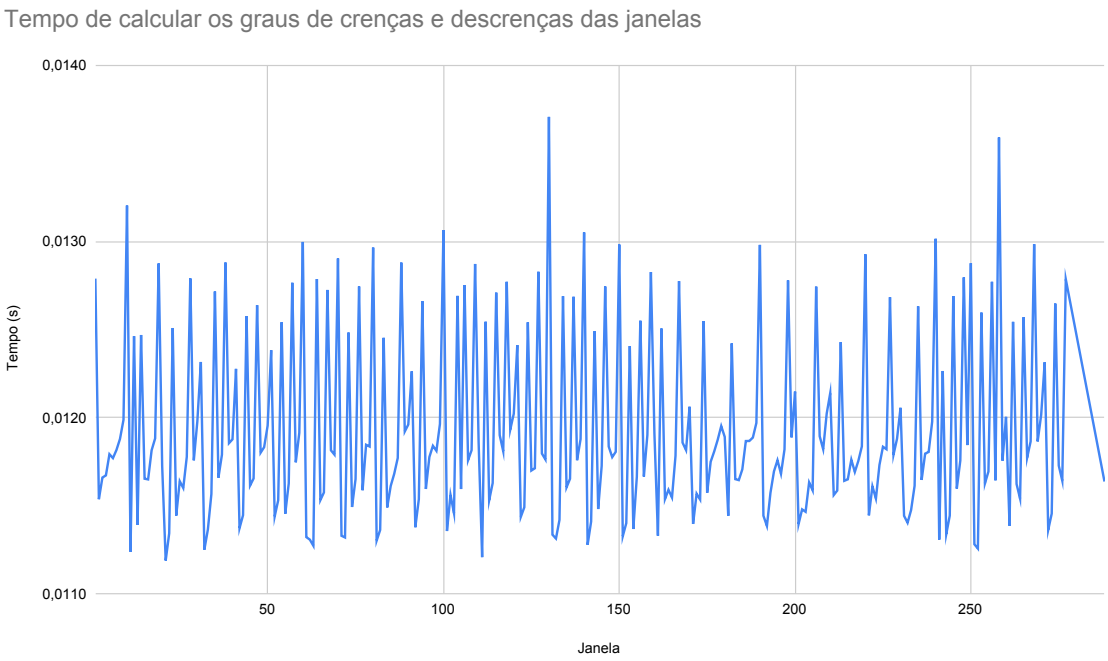

Figura 45: Gráfico de tempo gasto para calcular os graus das janelas.
Fonte: Autoria própria (2021)

Por fim, a etapa computar e exibir os resultados são processadas de forma independentes para cada sistema, na qual o LPAProg-DDoS utiliza a Lógica Paraconsistente Anotada para computar e o FuzProg-DDoS a Lógica *Fuzzy*. Até mesmo a forma de exibir os dados são distintos entre os sistemas, mas ambos geram um gráfico de saída. A média do tempo desta etapa foram próximas entre os dois sistemas, com uma variação de apenas 0,0244 segundos. Na qual, a média de tempo do LPAProg-DDoS foi de 0,6581 segundos e do FuzProg-DDoS de 0,6826 segundos. Vale ressaltar que o LPAProg-DDoS realiza um processo que a cada rodada acumula janelas até

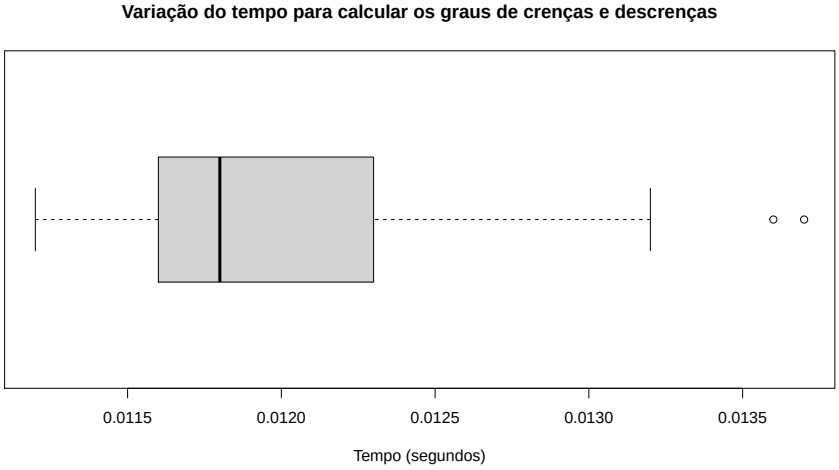

Figura 46: *Boxplot* da variação do tempo para calcular os graus das janelas.
Fonte: Autoria própria (2021)

atingir o limite de 10 janelas, então retorna a processar apenas uma janela e reinicia o acúmulo. Esse processo do LPAProg-DDoS é realizado para obter-se o baricentro em um conjunto de até 10 janelas, de modo que seja possível analisar o ponto médio central deste conjunto, além de observar o movimento do baricentro quando passa de um momento de tráfego normal para ataque, ou em ordem invertida. A figura 47 apresenta o gráfico comparativo entre o tempo gasto para computar e exibir os resultados do LPAProg-DDoS e do FuzProg-DDoS. Observa-se os momentos em que o LPAProg-DDoS aumenta o tempo de execução, até atingir o limite de 10 janelas, logo após o tempo decai novamente quando inicia-se um novo ciclo. A figura 48 exibe o *boxplot* com alguns resultados estatísticos do conjunto de dados gerados a partir dos testes.

Como mencionado, o LPAProg-DDoS e o FuzProg-DDoS geram gráficos de saída contendo o resultado gerado a partir de um conjunto de janelas (no caso do LPAProg-DDoS) ou apenas de uma única janela. A figura 49 exibe quatro gráficos gerados ao executar o LPAProg-DDoS na base de dados *Bot IoT*. No qual o primeiro gráfico representa o inicio da base de dados, contendo apenas tráfego normal. O segundo gráfico exibe o momento em que se encontra o primeiro ataque. Enquanto o terceiro gráfico mostra a continuidade do ataque até atingir 10 janelas, no qual é possível observar a movimentação do baricentro. Por fim, o quarto gráfico apresenta o mo-

Figura 47: Gráfico de tempo gasto para computar e exibir os resultados das janelas.
Fonte: Autoria própria (2021)

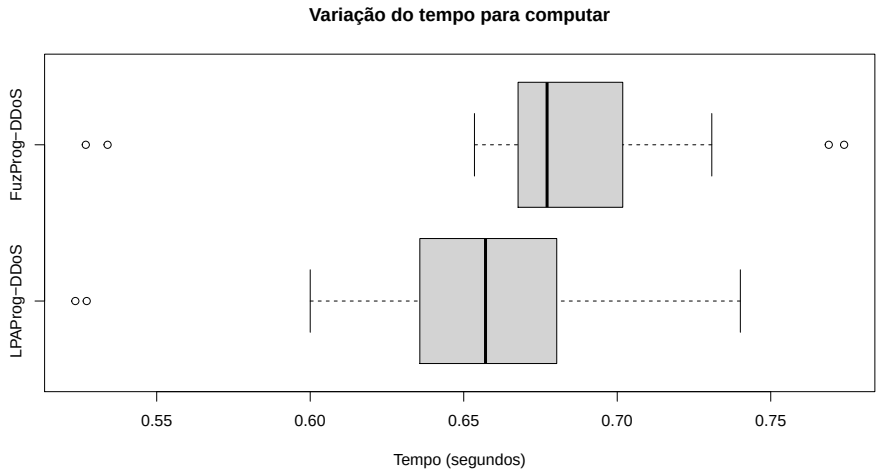

Figura 48: *Boxplot* da variação do tempo para computar as janelas e exibir os resultados.
Fonte: Autoria própria (2021)

mento em que 10 janelas foram classificadas como ataque. O FuzProg-DDoS realiza a analise de janela por janela, na qual gera apenas o gráfico da única janela analisada no momento. A figura 50 apresenta dois gráficos gerados ao executar o FuzProg-DDoS na base de dados *Bot IoT*. No qual, o primeiro gráfico exibe um momento de

tráfego normal e o segundo um momento de ataque.

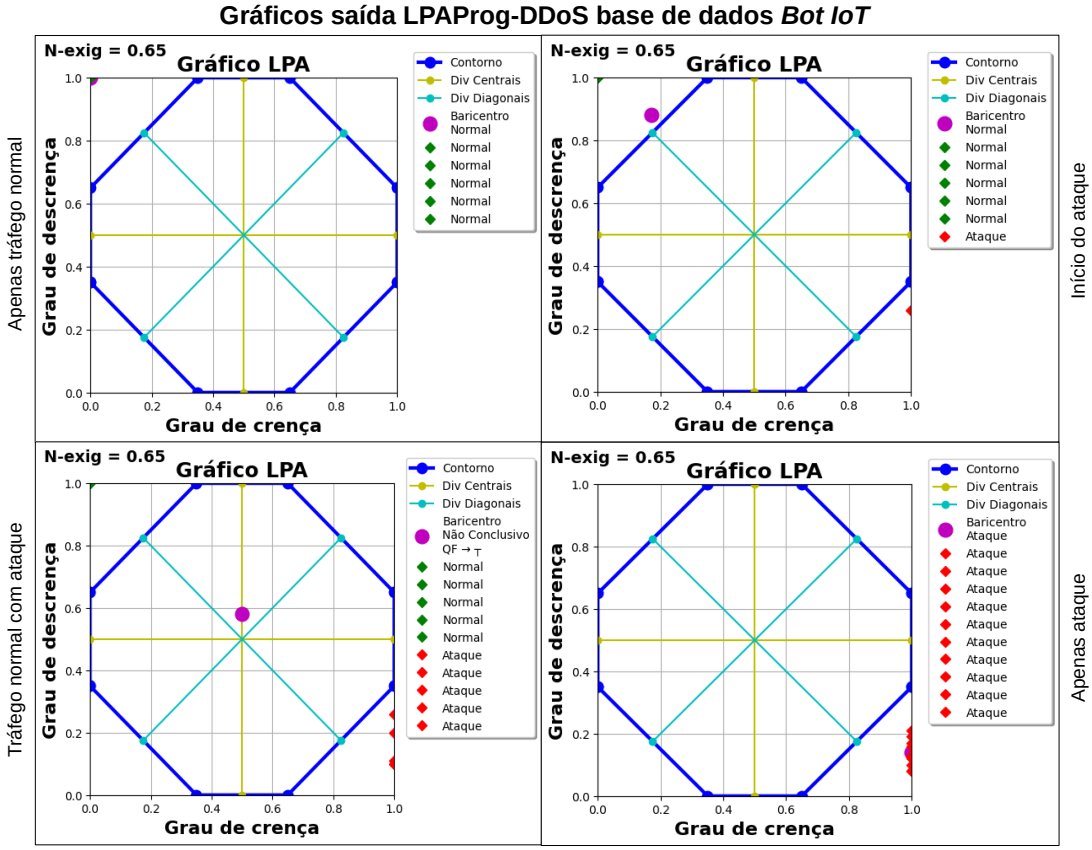

Figura 49: Alguns dos gráficos gerados pelo LPAProg-DDoS ao ser executado na base de dados *Bot IoT*

Fonte: Autoria própria (2021)

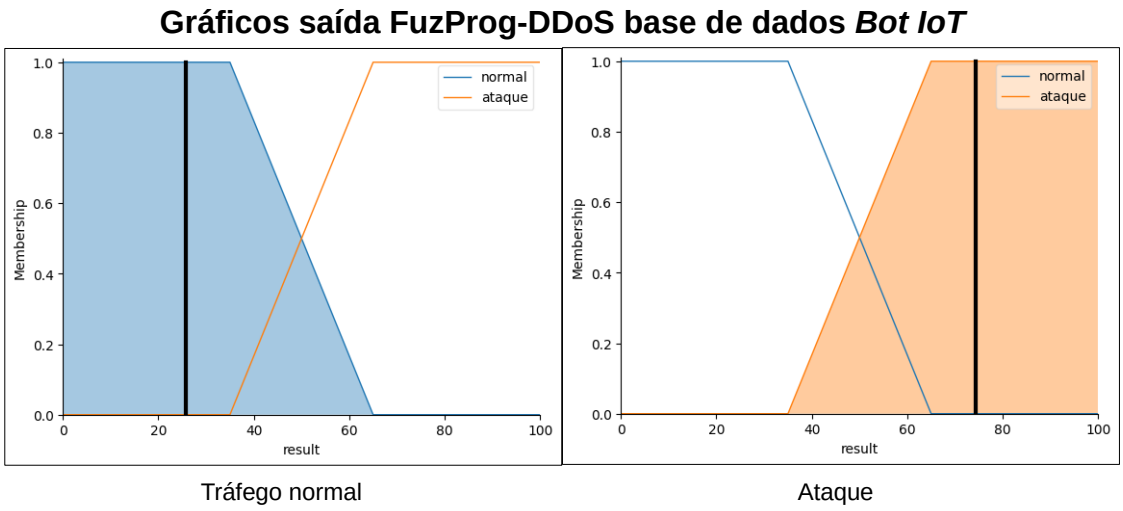

Figura 50: Alguns dos gráficos gerados pelo FuzProg-DDoS ao ser executado na base de dados *Bot IoT*

Fonte: Autoria própria (2021)

A base de dados *Bot-IoT* disponibiliza de um arquivo no formato CSV contendo os rótulos (*labels*) de cada pacote tráfego. Deste modo, identificou se no momento de tráfego de cada janela, há ou não um ataque eminente. Logo, gerou-se os rótulos para cada janela que foram criadas e analisadas pelos sistemas. Das 278 janelas, 273 referem-se a momentos de ataques e 5 são de momentos de tráfego normal. As 5 janelas referentes ao tráfego normal estão localizadas no início da base de dados, ou seja, as 5 primeiras janelas são exclusivamente de tráfego normal.

Os resultados gerados pelos sistemas foram comparados pelos rótulos adquiridos da descrição da base de dados. Com isso, foi possível gerar uma matriz de confusão para então calcular as métricas de avaliações (acurácia, precisão, *recall* e *F1-score*). Os sistemas LPAProg-DDoS e o FuzProg-DDoS obtiveram os mesmo resultados, por isso é apresentada apenas uma matriz de confusão, na qual equivale para ambos os sistemas. A tabela 6 apresenta a matriz de confusão. Ambos os sistemas tiveram 100% de acurácia geral, na qual todos os respostas dadas foram equivalentes aos rótulos obtidos da base de dados. A tabela 7 apresenta os resultados da acurácia, precisão, *recall* e *F1-score*. Observa-se que os sistemas aceraram em todos os casos, mesmo para os tráfegos normais e de ataques.

	Ataque	Normal	Não Conclusivo
Ataque	273	0	0
Normal	0	5	0
Não Conclusivo	0	0	0

Tabela 6: Matriz de confusão da saída dos sistemas LPAProg-DDoS e FuzProg-DDoS

Fonte: Autoria própria (2021)

Classe	n (verdade)	n (classificado)	Acurácia	Precisão	*Recall*	*F1-score*
Ataque	273	273	100%	100%	100%	100%
Normal	5	5	100%	100%	100%	100%
Não Conclusivo	0	0	100%	–	–	–

Tabela 7: Resultados dos sistemas LPAProg-DDoS e FuzProg-DDoS

Fonte: Autoria própria (2021)

Os sistemas LPAProg-DDoS e FuzProg-DDoS obtiveram os resultados satisfatórios na base de dados *Bot-IoT* para todas as métricas de avaliação. Os testes foram realizados em distintas janelas com distintos intervalos de tempo. Observa-se que

mesmo em janelas com o tamanho inferior a 1 segundo, ambos os sistemas conseguiram detectar se há ou não ataque no momento analisado. O tempo de abertura dos arquivos tiveram uma média de aproximadamente 397 segundos, na qual este tempo em uma análise *online*, ou seja, captura em tempo real, será definido pelo método de formação de janela (tempo ou quantidade de pacotes). A etapa que realiza o pré processamento é a que tem maior impacto de tempo na analise de uma janela. Na qual, nos testes realizados na base de dados *Bot-IoT*, observou uma máxima de aproximadamente 19 segundos. Sendo que este é um tempo de espera para se obter o resultado sobre se há ou não um ataque. Além disso, as etapas seguintes têm o seu maior tempo de execução inferior a 1 segundo.

4.2 CENÁRIO REAL

O cenário real é apresentado na subseção 3.5.3, na qual foram gerados trechos de ataque DDoS do tipo SYN *flood*. Os testes foram realizados de modo que a captura de pacotes acontecesse *online*, ou seja, no decorrer do processo de envio de pacotes, o sistema captura-os, gera-se a janela, para então realizar as etapas de pré processamento e processamento, de modo que a captura é inciada novamente em paralelo. Na qual, o tempo de tráfego de dados foi de 20 minutos. As configurações dos algoritmos de análise foram atribuídas de modo a gerar janelas com 10 segundos de tráfegos e o nível de exigência igual a 0,65. Foram realizadas duas simulações no cenário real, de modo que na primeira foi executado o sistema LPAProg-DDoS e na segunda o sistema FuzProg-DDoS.

A analise de tempo foi dividia em quatro etapas: 1) captura de pacotes; 2) pré processamento dos pacotes; 3) cálculo dos graus de crenças e descrenças; e 4) computar e exibir os resultados. As etapas 1, 2 e 3 são análogas entre os sistemas LPAProg-DDoS e FuzProg-DDoS.

Por se tratar de um cenário real e o modo ser *online*, em momentos que o ataque estava intenso houve um aumento no tempo da captura de pacotes, na qual pode ter ocorrido perdas de pacotes. Mas as análises não foram prejudicadas por este motivo.

No total, foram geradas e analisadas 62 janelas com o LPAProg-DDoS e 65 janelas com FuzProg-DDoS. A figura 51 apresenta o tempo gasto para capturar os pacotes e gerar as janelas com os testes realizados a partir do sistema LPAProg-DDoS e FuzProg-DDoS. Além disso, a figura 52 apresenta o *boxplot* da variação de tempo com base em cálculos estatísticos.

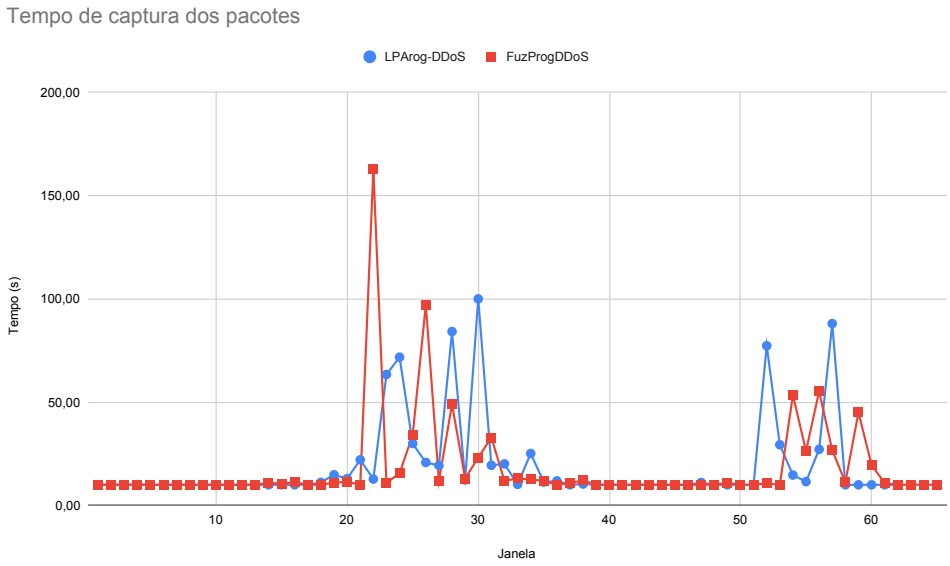

Figura 51: Gráfico de tempo gasto para capturar os pacotes e gerar as janelas.
Fonte: Autoria própria (2021)

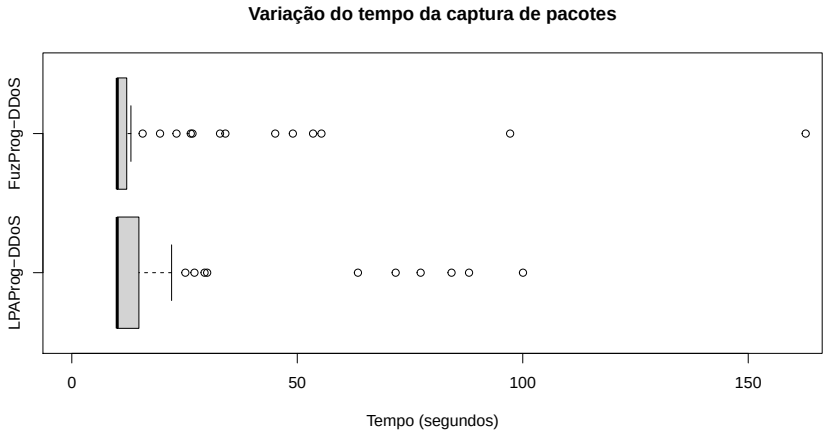

Figura 52: Gráfico de tempo das janelas geradas pelo sistema.
Fonte: Autoria própria (2021)

A etapa de pré processamento realiza a contagem de pacotes para cada uma das *flags* e de pacotes TCP. Esta etapa é realizada em paralelo com a etapa de capturar

caso haja alguma janela a ser pré processada. A média de tempo gasto para pré processar as 62 janelas nos testes com o LPAProg-DDoS foi de 0,57 segundos. Já a média de tempo gasto nos testes com o FuzProg-DDoS foi de 0,62 segundos. A figura 53 apresenta a evolução do tempo gasto para realizar o pré processamento de cada janela com o teste de ambos os sistemas. Entretanto, afigura 54 apresenta o *boxplot* com a variação de tempo com base em cálculos estatísticos.

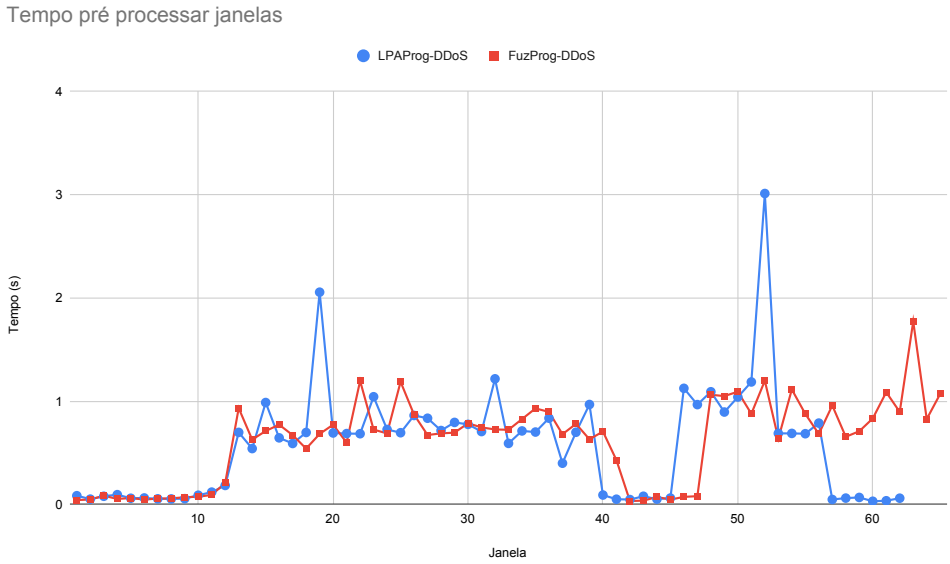

Figura 53: Gráfico de tempo gasto para pré processar as janelas.
Fonte: Autoria própria (2021)

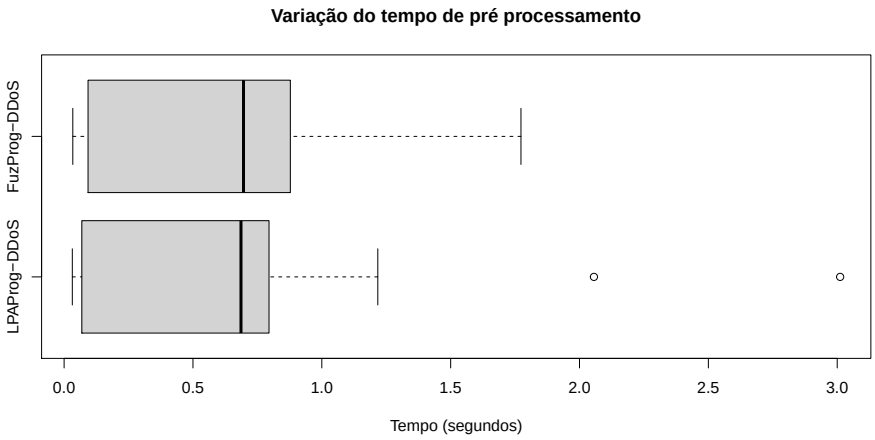

Figura 54: Gráfico de tempo das janelas geradas pelo sistema.
Fonte: Autoria própria (2021)

A próxima etapa, calcular os graus de crenças e descrenças, obtém os graus a

partir dos dados pré processados. Esta etapa no primeiro teste com o LPAProg-DDoS obteve uma média de tempo de 0,014 segundos. Já para o teste com o FuzProg-DDoS a média do tempo foi de 0,015 segundos. A figura 55 apresenta a evolução do tempo gasto para realizar os cálculos dos graus de cada janela com o teste de ambos os sistemas. Observa-se que em ambos os testes houve uma máxima no tempo próximo a 0,08 segundos. A figura 56 apresenta o *boxplot* com a variação de tempo com base em cálculos estatísticos.

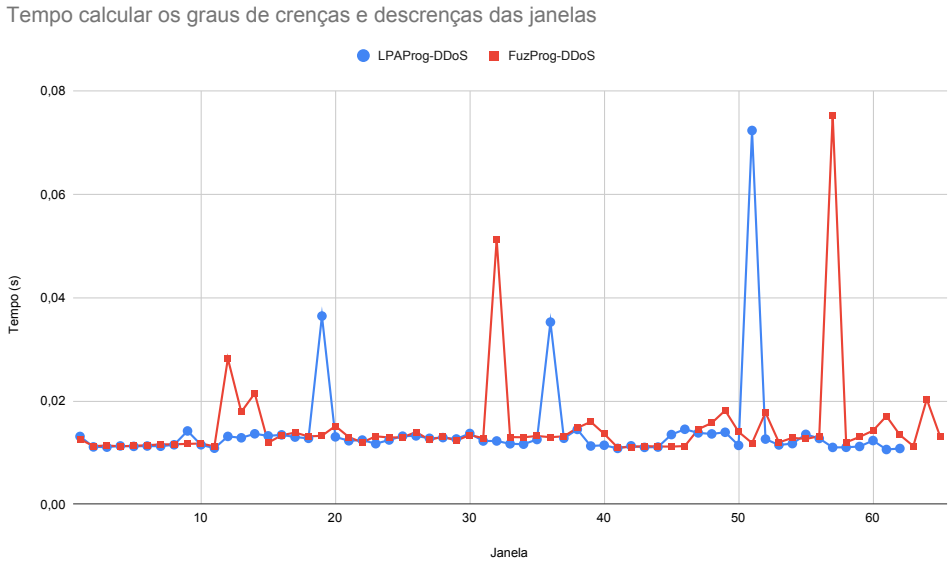

Figura 55: Gráfico de tempo gasto para calcular os graus das janelas.
Fonte: Autoria própria (2021)

A última etapa que foi analisada o tempo é a de computar e exibir os resultados. As médias de tempo desta etapa foram próximas entre os teste com o LPAProg-DDoS e o FuzProg-DDoS. Na qual, as médias foram 1,25 segundos e 1,22 segundos respectivamente. Vale ressaltar que o LPAProg-DDoS realiza um processo que a cada rodada acumula janelas até atingir o limite de 10 janelas, então retorna a processar apenas uma janela e reinicia o acúmulo. Esse processo do LPAProg-DDoS é realizado para obter-se o baricentro em um conjunto de até 10 janelas, de modo que seja possível analisar o ponto médio central deste conjunto, além de observar o movimento do baricentro quando passa de um momento de tráfego normal para ataque, ou em ordem invertida. A figura 57 apresenta o gráfico de tempo gasto para computar e exibir os resultados de ambos os testes. Entretanto, a figura 58 exibe

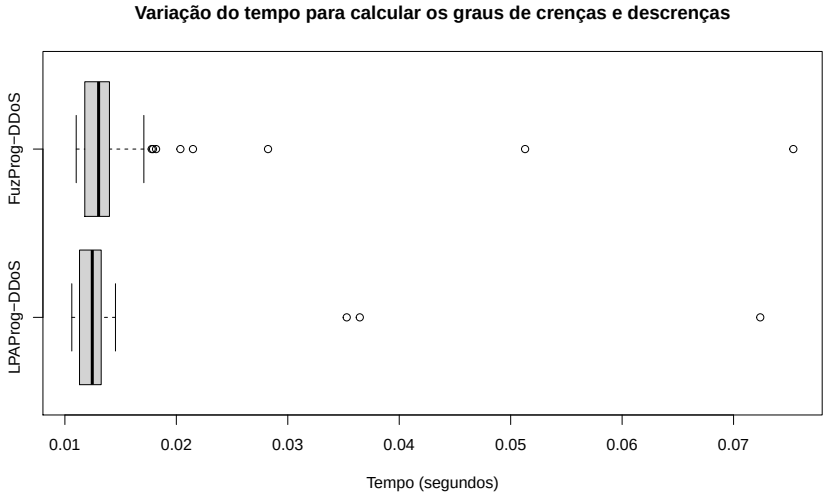

Figura 56: *Boxplot* da variação do tempo para calcular os graus das janelas.
Fonte: Autoria própria (2021)

o *boxplot* com alguns resultados estatísticos do conjunto de dados gerados a partir dos testes.

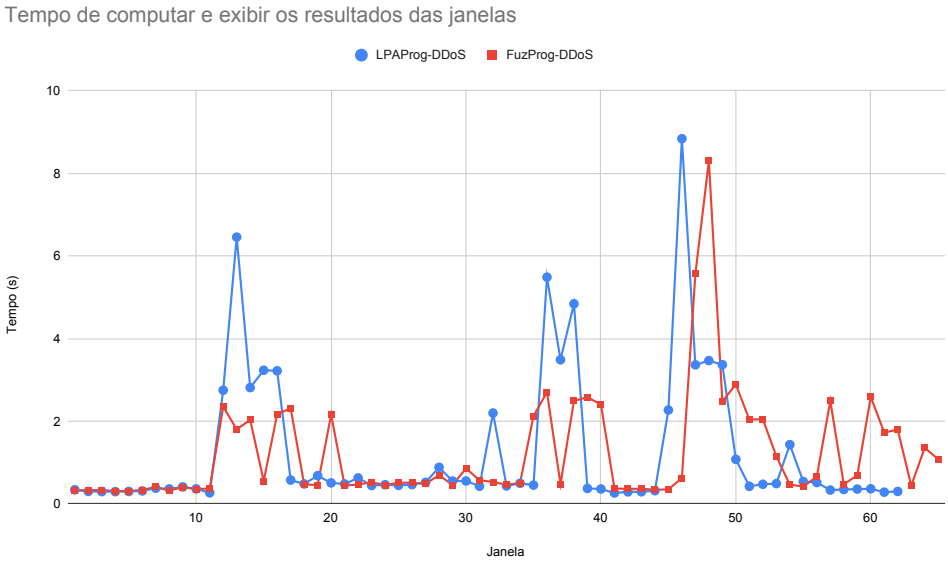

Figura 57: Gráfico de tempo das janelas geradas pelo sistema.
Fonte: Autoria própria (2021)

Como mencionado, o LPAProg-DDoS e o FuzProg-DDoS geram gráficos de saída contendo o resultado gerado a partir de um conjunto de janelas (no caso do LPAProg-DDoS) ou apenas de uma única janela. A figura 59 exibe quatro gráficos gerados ao executar o LPAProg-DDoS no cenário real. No qual o primeiro gráfico representa

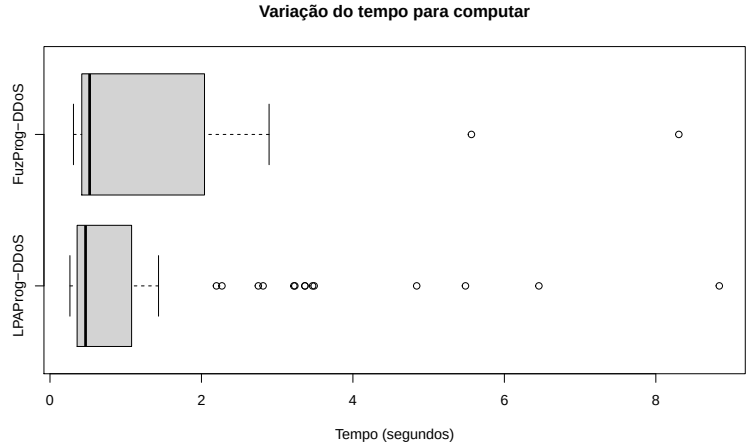

Figura 58: Gráfico de tempo das janelas geradas pelo sistema.
Fonte: Autoria própria (2021)

o inicio da simulação, contendo apenas tráfego normal. O segundo gráfico exibe o momento em que é realizado o primeiro ataque. Enquanto o terceiro gráfico mostra os momentos que há apenas ataque. Por fim, o quarto gráfico apresenta o momento em que houve a paralisação dos ataques. O FuzProg-DDoS realiza a analise de janela por janela, na qual gera apenas o gráfico da única janela analisada no momento. A figura 60 apresenta quatro gráficos gerados ao executar o FuzProg-DDoS no cenário real. No qual, o primeiro gráfico exibe um momento de tráfego normal, o segundo um momento de ataque e o terceiro e quarto gráficos apresentam momentos que os resultados foram classificados como não conclusivos.

As métricas de avaliação foram distintas entre os testes com o sistema LPAProg-DDoS e com o sistema FuzProg-DDoS. De modo geral, a acurácia do LPAProg-DDoS foi de 80,65% e do FuzProg-DDoS foi de 90,77%. Tratando-se da precisão, ambos os testes obtiveram uma precisão de 100% para momentos de ataque e tráfego normal, na qual mostra que quando os sistemas classificaram como sendo ataque ou tráfego normal, os mesmos acertaram sempre. A tabela 8 apresenta a matriz de confusão do teste com o LPAProg-DDoS. Entretanto, a tabela 9 apresenta os resultados da acurácia, precisão, *recall* e *F1-Score* para o teste com o LPAProg-DDoS. A matriz de confusão do teste com o FuzProg-DDoS é apresentada na tabela 10. Enquanto, os resultados das métricas de avaliação do teste com o FuzProg-DDoS é apresentado na tabela 11.

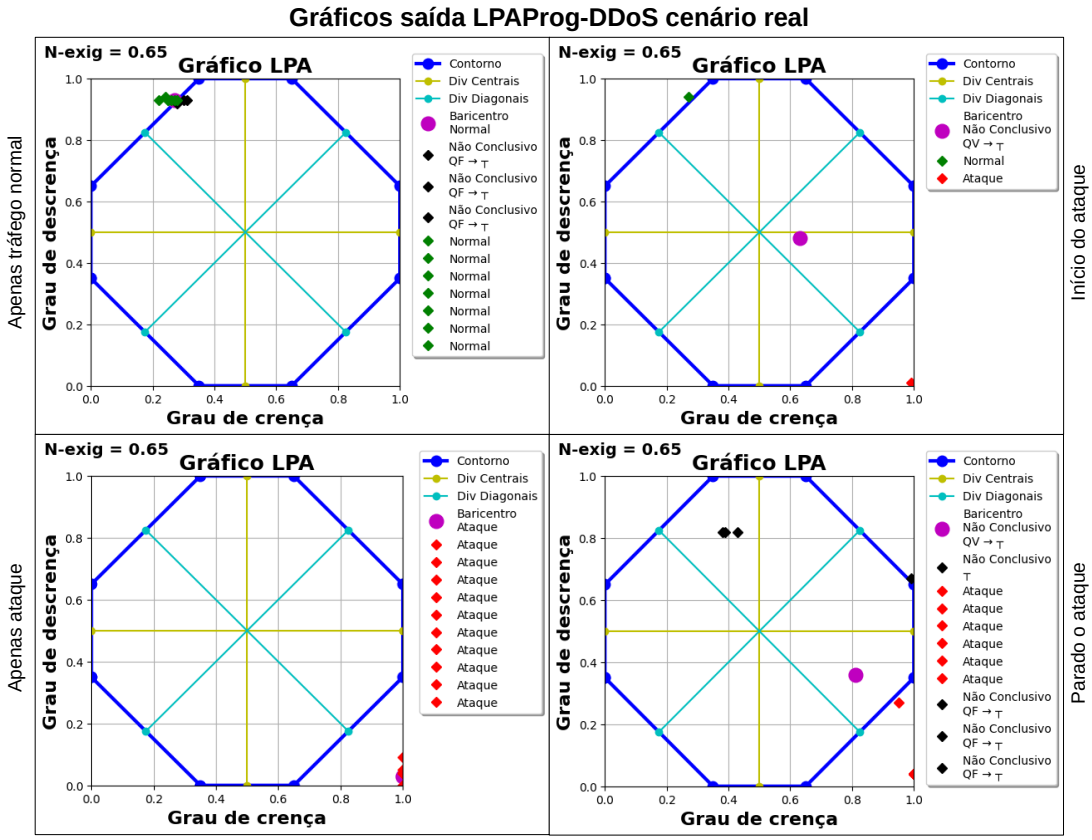

Figura 59: Alguns dos gráficos gerados pelo LPAProg-DDoS ao ser executado no cenário real

Fonte: Autoria própria (2021)

	Ataque	Normal	Não Conclusivo
Ataque	33	0	0
Normal	0	17	0
Não Conclusivo	2	10	0

Tabela 8: Matriz de confusão da saída do teste com o sistema LPAProg-DDoS
Fonte: Autoria própria (2021)

Classe	n (verdade)	n (classificado)	Acurácia	Precisão	*Recall*	*F1-score*
Ataque	35	33	96,77%	100%	94%	97%
Normal	27	17	83,87%	100%	63%	77%
Não Conclusivo	0	12	80,65%	–	–	–

Tabela 9: Resultados do teste com o sistema LPAProg-DDoS
Fonte: Autoria própria (2021)

Os testes provenientes no cenário real são não determinísticos, sendo que a cada simulação pode haver alterações no resultados. Neste caso observa-se a variação na quantidade de janelas entre os testes no sistema LPAProg-DDoS e FuzProg-DDoS, na qual a quantidade de janela foi 62 e 65 respectivamente. Além disso,

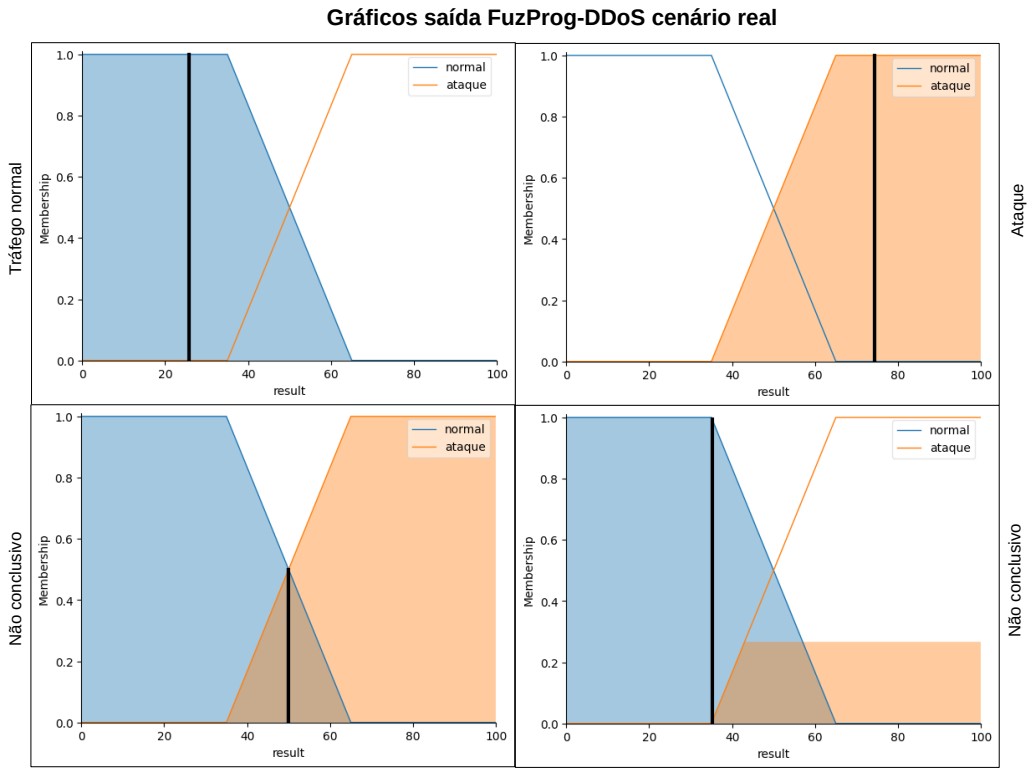

Figura 60: Alguns dos gráficos gerados pelo FuzProg-DDoS ao ser executado no cenário real

Fonte: Autoria própria (2021)

	Ataque	Normal	Não Conclusivo
Ataque	37	0	0
Normal	0	22	0
Não Conclusivo	1	5	0

Tabela 10: Matriz de confusão da saída do teste com o sistema FuzProg-DDoS
Fonte: Autoria própria (2021)

Classe	n (verdade)	n (classificado)	Acurácia	Precisão	*Recall*	*F1-score*
Ataque	38	37	98,46%	100%	97%	99%
Normal	27	22	92,31%	100%	81%	90%
Não Conclusivo	0	6	90,77%	–	–	–

Tabela 11: Resultados do teste com o sistema FuzProg-DDoS
Fonte: Autoria própria (2021)

em momentos de ataque intenso, gera um congestionamento na captura, na qual ultrapassa o limite de 10 segundos que foi configurado. Houve momento que a captura demorou mais de 150 segundos para ser finalizada.

Os sistemas LPAProg-DDoS e FuzProg-DDoS obtiveram a acurácia geral acima

de 80%, na qual a maior parte do erro foi sobre o tráfego normal, sendo que em momentos o resultado dado foi não conclusivo. O *recall* exibe o erro maior sobre o tráfego normal, na qual para o teste do sistema LPAProg-Prog o *recall* para o ataque foi de 94% e para o tráfego normal foi de 63%. Ao se tratar do teste do sistema FuzProg-DDoS o *recall* ficou em 97% e 81% para ataque e tráfego normal respectivamente. Ambos os sistemas conseguem detectar os ataques DDoS do tipo SYN *flood*, mas nos testes realizados o FuzProg-DDoS foi melhor na assertividade.

5 CONSIDERAÇÕES FINAIS

Este trabalho apresentou os conceitos relativos a redes de computadores, segurança de redes, segurança computacional, lógica paraconsistente e lógica *fuzzy*. Deste modo, foi possível a interdisciplinaridade entre os conteúdos estudados. Além disso, este trabalho apresenta um sistema inovador que utiliza-se da LPA para auxiliar administradores na segurança da rede, pois trata-se de um sistema desenvolvido para detectar ataques DDoS do tipo SYN *flood*.

Através dos estudos e das aplicações efetuadas em ambientes de redes, como a base de dados *Bot IoT* e também o cenário real construído, obtivemos dois objetivos específicos. O primeiro, se dá na validação do algoritmo LPAProg-DDoS desenvolvido tanto em uma base de dados estática (*Bot IoT*), como uma base física em tempo real, construída nas dependências da UTFPR. Com isto, foi possível responder o objetivo geral que versa sobre a viabilidade do uso da Lógica Paraconsistente na segurança de redes aplicado na detecção de ataques. A partir desta conclusão, se fez necessária uma validação do algoritmo LPA frente a outros algoritmos que já são muito explorados na comunidade científica, na detecção de ataques baseado em Inteligência Artificial (IA). Desta forma, optou-se pelo algoritmo denominado Lógica *Fuzzy*, em função das suas características de aplicações, serem um tanto quanto semelhantes aos *modus operandis* da LPA. Desta forma, de acordo com os resultados obtidos, em um estágio considerado ainda preliminar, foi possível constatar que a LPA tem potencial de aplicação prática em segurança de redes.

Em análises dos resultados, foi possível constatar que neste primeiro momento, observando o cenário considerado *offline* (*Bot IoT*), ambos os sistemas (LPAProg-DDoS e FuzProg-DDoS) tiveram a assertividade de 100%. Além disso, os resultados gerados a partir do *Bot IoT*, observou que mesmo em janelas inferiores a 1 segundo, foi possível realizar a detecção do ataque. Outro ponto importante, foi a analise do tempo gasto pelo pré processador, na qual atingiu uma máxima de 19 segundos, com isso foi possível validar a importância do paralelismo do sistema, de modo a impedir que houvesse perdas de pacotes em um intervalo de tempo nos testes *online*. No cenário *online*, ambos os sistemas obtiveram a acurácia geral acima de 80%, na

qual o LPAProg-DDoS teve 80,65% de acurácia e o FuzProg-DDoS 90,77%. Vale ressaltar que ao se tratar da precisão do ataque e tráfego normal, ambos os sistemas obtiveram 100%. Na qual mostra que quando a janela é classificada como tráfego normal ou ataque, os sistemas acertaram.

Este trabalho tem como diferencial a analise do tráfego da rede de forma não supervisionada, ou seja, não é necessário de dados antigos para a realização de treinamento. Além disso, a analise realizada ocorre no presente momento da janela gerada, deste modo não é necessário de tráfegos passados para se obter o resultado se é ou não um ataque recorrente na janela, ou mesmo se a analise foi não conclusiva.

Recomenda-se para trabalhos futuros a exploração dos sistemas em outras bases de dados e em outros cenários reais, de modo a obter novos resultados com a finalidade de analisar novos valores de métricas de validação e comparar entre ambos os sistemas. Além disso, recomenda-se os estudos de outros tipos de ataque DDoS (UDP *flood*, HTTP *flood* e *ping* da morte e entre outros) para incorporar ao sistema LPAProg-DDoS, de modo a agregar valores.

As contribuições que este trabalho de conclusão de curso propôs, foram as mais diversas possíveis, pois foi possível o entendimento aprofundado no gerenciamento de redes, no gerenciamento de segurança de rede, bem como, das aplicações de IA para detecção de ataques, além do aprimoramento de diversas técnicas avançadas de desenvolvimento de programas que se fizeram necessárias para a obtenção dos resultados o mais promissor possível. Diante destes aprendizados, uma das maiores contribuições, se deu na produção e publicação de um artigo científico num dos principais eventos de segurança do Brasil, que foi o SBSEG 2019 e a possível publicação de um segundo artigo para congressos e eventos de segurança de redes. Desta forma, pode-se concluir com assertividade que todos os objetivos propostos para este trabalho, foram atendidos por completo e além disso, suas contribuições foram atendidas e novas oportunidades de continuidade foram abertas para estudos futuros.

Referências

MARIO, M. C.; ABE, J. M.; ORTEGA, N. R. S.; SANTO, M. D. Jr. Análise cefalométrica para auxílio ao diagnóstico ortodôntico utilizando as Redes Neurais Artificiais Paraconsistentes. *In*: ABE, J. M. **Aspectos de Computação Inteligente Paraconsistente**. São Paulo: Instituto de Estudos Avançados da Universidade de São Paulo, 2013. p. 88-107.

ALSIRHANI, A.; SAMPALLI, S.; BODORIK, P. DDoS Detection System: Using a Set of Classification Algorithms Controlled by Fuzzy Logic System in Apache Spark. **IEEE Transactions on Network and Service Management**, v. 16, n. 3, p. 936–949, set. 2019, DOI 10.1109/TNSM.2019.2929425.

BHAYA, W.; EBADYMANAA, M. DDoS attack detection approach using an efficient cluster analysis in large data scale. *In*: **2017 Annual Conference on New Trends in Information & Communications Technology Applications (NTICT)**. IEEE, 2017. p. 168-173. DOI 10.1109/NTICT.2017.7976110.

BROWN, L.; STALLINGS, W. **Segurança de computadores: princípios e práticas**. Rio de Janeiro: Elsevier Brasil, 2017. v. 2. 1535 p.

CARVALHO, F. R. de; BRUNSTEIN, I.; ABE, J. M. Um estudo de tomada de decisão baseado em lógica paraconsistente anotada: avaliação do projeto de uma fábrica. **Revista Pesquisa e Desenvolvimento Engenharia de Produção**, n. 1, p. 47-62, 2003.

CARVALHO, R. N. **DoSSEC**: proposta de detecção e mitigação de ataques SYN Flood em redes SDN. 2020. 97 f. Dissertação. (Mestrado em Informática) - Universidade de Brasília, Distrito Federal, Brasília.

CERT.BR - Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil. **Incidentes Reportados ao CERT.br – Janeiro a Dezembro de 2019.** 2020. Disponível em: <https://www.cert.br/stats/incidentes/2019-jan-dec/tipos-ataque.html>. Acesso em: 29 jul. 2020.

CHÁVEZ-BOSQUEZ, O.; PARRA, P. P.; MCAREAVEY, K. On the developmentof a logic calculator: a novel tool to perform logical operations. **LANMR**, p. 9–16, 2016.

CISCO, V. N. I. **Cisco Annual Internet Report (2018–2023).** 2020. Disponível em: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf>. Acesso em: 21 jul. 2020.

COMER, D. E. **Redes de Computadores e Internet**. Tradução: José V. de Lima; Valter Roesler. 6. ed. Porto Alegre: Bookman Editora, 2016. 557 p.

DA COSTA, N. C.; KRAUSE, D.; BUENO, O. Paraconsistent Logics and Paraconsistency. *In*: JACQUETTE, D. (Ed.). **Philosophy of Logic**. Amsterdam: North-Holland, 2007. (Handbook of the Philosophy of Science). p. 791–911. DOI 10.1016/B978-044451541-4/50023-3.

DANTAS, M. **Redes de Comunicação e Computadores: Abordagem Quantitativa**. Florianópolis: Visual Books, 2010. 443 p.

DONG, S.; ABBAS, K.; JAIN, R. A survey on distributed denial of service (DDoS) attacks in SDN and cloud computing environments. **IEEE Access**, v. 7, p. 80813–80828, 2019.

DOSHI, R.; APTHORPE, N.; FEAMSTER, N. Machine Learning DDoS Detection for Consumer Internet of Things Devices. *In*: **2018 IEEE Security and PrivacyWorkshops (SPW)**. IEEE, 2018. p. 29–35.

FALCÃO, A. V. d. S. **Fuzdetect**: sistema de detecção e classificação de ataques de negação de serviço. 2019. 100 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal da Paraíba, Paraíba, João Pessoa.

FERNANDES, M. B. et al. Análise de segurança cibernética em veículos autônomos utilizando lógica paraconsistente. **Iberoamerican Journal of Project Management**, v. 10, n. 1, p. 30–48, 2019.

FERREIRA, A. E.; NOGUEIRA, M. Identificando botnets geradoras de ataques ddos volumétricos por processamento de sinais em grafos. *In*: **SBC. ANAIS do XXIII Workshop de Gerência e Operação de Redes e Serviços**. SBC, 2018.

FOROUZAN, B. A. **Comunicação de dados e redes de computadores**. Tradução: Ariovaldo Griesi. 4. ed. Porto Alegre: AMGH Editora, 2009. 1134 p.

FOROUZAN, B. A.; MOSHARRAF, F. **Redes de computadores**: uma abordagem top-down. Porto Alegre: AMGH Editora, 2013. 896 p.

FOX, E. F. B.-L. **Detecção de ataques syn-flooding em redes definidas por software**. 2019. 63 f. Dissertação (Mestrado em Informática) – Universidade Federal da Paraíba, Paraíba, João Pessoa.

GUTTMAN, B.; ROBACK, E. A. **Sp 800-12. an introduction to computersecurity**: the NIST handbook. Washigton: National Institute of Standards & Technology, 1995. 296 p.

HPING. **Hping**. Disponível em: <http://www.hping.org/authors.php>. Acessoem: 23 nov. 2020.

HUSSAIN, K. et al. SYN Flood Attack Detection based on Bayes Estimator (SFADBE) For MANET. *In*: **2019 International Conference on Computer andInformation Sciences (ICCIS)**. IEEE, 2019. p. 1–4.

IPERF. **Iperf**. Disponível em: <https://iperf.fr/>. Acesso em: 23 nov. 2020.

JUNIOR, E. P. F.; TAVARES, A. C. J.; NOGUEIRA, M. Lógica Paraconsistente Anotada Aplicada na Detecção de Ataques DDoS em Redes SDN. *In*: S**BC. ANAIS do**

XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais. SBC, 2019. p. 439-444.

JUNIOR, E. P. F. **Estudo comparativo entre algoritmos de regras de associação de forma normal e incremental de dados**. 2008. 131 f. Dissertação (Mestre em Informática) – Pontifícia Universidade Católica do Paraná, Paraná, Curitiba.

KUPREEV, O.; BADOVSKAYA, E.; GUTNIKOV, A. **DDoS attacks in Q22019**. Kaspersky, 2019. Disponível em: <https://securelist.com/ddos-report-q2-2019/91934/>. Acesso em: 30 jul. 2020.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a Internet: uma abordagem top-down**. 6. ed. São Paulo: Pearson Education do Brasil, 2013.

CHENG, J.; LIU, Y.; TANG, X. *et al*. DDoS Attack Detection via Multi-Scale Convolutional Neural Network. Computers. *In*: **Materials & Continua**, v. 62, n. 3, p. 1317–1333, 2020.

MAHJABIN, T. *et al*. A survey of distributed denial-of-service attack, prevention,and mitigation techniques. **International Journal of Distributed Sensor Networks.** SAGE Publications, 2019. v. 13, n. 12. DOI 10.1177/1550147717741463.

MALLIKARJUNAN, K. N.; MUTHUPRIYA, K.; SHALINIE, S. M. A survey ofdistributed denial of service attack. *In*: **IEEE. 2016 10th International Conferenceon Intelligent Systems and Control (ISCO)**. IEEE, 2016. p. 1–6.

MARRO, A. A. *et al.* Lógica fuzzy: conceitos e aplicações. *In*: **Universidade Federal do Rio Grande do Norte (UFRN)**, 2010.

MOUSTAFA, N. The Bot-IoT dataset. *In*: **IEEE Dataport**. IEEE, 2019. DOI 10.21227/r7v2-x988.

NETO, M. C. L.; VENSON, N. Lógica paraconsistente. *In*: **Universidade Federal de Santa Catarina**, 2002.

NOVAES, M. P. *et al.* Long Short-Term Memory and Fuzzy Logic for AnomalyDetection and Mitigation in Software-Defined Network Environment.

In: **IEEE Access**. IEEE, 2020. v. 8, p. 83765–83781. DOI 10.1109/ACCESS.2020.2992044.

OO, N. H.; HTEIN MAW, A. Effective Detection and Mitigation of SYN Flooding Attack in SDN. *In*: **2019 19th International Symposium on Communications and Information Technologies (ISCIT)**. IEEE, 2019. p. 300–305.

PONTES SARAIVA, G. J. de. Lógica Fuzzy. **Revista da Escola Superior de Guerra**, 2006. v. 21, n. 45. DOI 10.47240/revistadaesg.v21i45.335.

POWERS, D. M. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. **International Journal of Machine Learning Technology**. p. 37-63, 2011.

RIGHI, M. A. Detecção de DDoS através da análise da quantificação da recorrência baseada na extração de características dinâmicas e clusterização adaptativa. 2017. 86 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Santa Maria, Santa Maria, Rio Grande do Sul.

SHIREY, R. **RFC2828**: Internet security glossary. RFC Editor, 2000.

SILVA FILHO, J. I. da. Lógica ParaQuântica LPQ (parte I): introdução aos conceitos fundamentais. **Seleção Documental: Inteligência Artificial e novas Tecnologias - Universidade Santa Cecilia**, Santa Cecilia, n. 18, p. 17–26, 2010.

SIMÕES, M. G.; SHAW, I. S. **Controle e modelagem fuzzy**. 2. ed. São Paulo: Editora Blucher, 2007. 200 p. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788521215479/>. Acesso em: 7 mar. 2021.

STALLINGS, W. **Criptografia e segurança de redes**: princípios e práticas. 6. ed. São Paulo: Pearson Education do Brasil, 2015. 560 p.

T50. **T50**. Disponível em: <https://gitlab.com/fredericopissarra/t50>. Acesso em: 23 nov. 2020.

TANENBAUM, A. S. **Redes de Computadores**. 5. ed. São Paulo: Pearson Prentice Hall, 2011. 600 p.

TANSCHEIT, Ricardo. Sistemas fuzzy. **Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro**, Rio de Janeiro, p. 338-353, 2004.

WGET. **GNU Wget**. Disponível em: <https://www.gnu.org/software/wget/>. Acesso em: 23 nov. 2020.