

**UNIVERSIDADE FEDERAL DA PARAÍBA**

CENTRO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

**FUZDETECT: SISTEMA DE DETECÇÃO E  
CLASSIFICAÇÃO DE ATAQUES DE NEGAÇÃO  
DE SERVIÇO**

**ARIANE VENTURA DE SOUSA FALCÃO**

**ORIENTADOR: PROF. DR. VIVEK NIGAM**

**CO-ORIENTADOR: PROF. DR. IGUATEMI EDUARDO DA FONSECA**

João Pessoa – PB

Fevereiro/2019

**UNIVERSIDADE FEDERAL DA PARAÍBA**

CENTRO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

**FUZDETECT: SISTEMA DE DETECÇÃO E  
CLASSIFICAÇÃO DE ATAQUES DE NEGAÇÃO  
DE SERVIÇO**

**ARIANE VENTURA DE SOUSA FALCÃO**

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.  
Orientador: Prof. Dr. Vivek Nigam.

João Pessoa – PB

Fevereiro/2019

**Catálogo na publicação**  
**Seção de Catalogação e Classificação**

F178f Falcão, Ariane Ventura de Sousa.

FUZDETECT: SISTEMA DE DETECÇÃO E CLASSIFICAÇÃO DE  
ATAQUES DE NEGAÇÃO DE SERVIÇO / Ariane Ventura de Sousa  
Falcão. - João Pessoa, 2019.

98 f. : il.

Orientação: Nigam Vivek.

Coorientação: Iguatemi Eduardo da Fonseca.

Dissertação (Mestrado) - UFPB/CI.

1. SDN. 2. Segurança. 3. DDoS. 4. Lógica Fuzzy. 5. PSO.  
I. Vivek, Nigam. II. Fonseca, Iguatemi Eduardo da. III.  
Título.

UFPB/BC



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



Ata da Sessão Pública de Defesa de Dissertação de Mestrado de Ariane Ventura de Sousa Falcão, candidata ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 25 de fevereiro de 2019.

1 Aos vinte e cinco dias do mês de fevereiro, do ano de dois mil e dezenove, às onze horas, no  
2 Centro de Informática da Universidade Federal da Paraíba, em Mangabeira, reuniram-se os  
3 membros da Banca Examinadora constituída para julgar o Trabalho Final da Sr<sup>a</sup>. Ariane  
4 Ventura de Sousa Falcão, vinculada a esta Universidade sob a matrícula nº 20171003849,  
5 candidata ao grau de Mestre em Informática, na área de "Sistemas de Computação", na linha  
6 de pesquisa "Computação Distribuída", do Programa de Pós-Graduação em Informática, da  
7 Universidade Federal da Paraíba. A comissão examinadora foi composta pelos professores:  
8 Vivek Nigam (PPGI-UFPB) Orientador e Presidente da Banca, Iguatemi Eduardo da Fonseca  
9 (UFPB), Examinador Interno, Anand Subramanian (UFPB), Examinador Interno, Fernando  
10 Menezes Matos (UFPB), Examinador Interno, Magnos Martinello (UFES), Examinador  
11 Externo à Instituição. Dando início aos trabalhos, o Presidente da Banca, cumprimentou os  
12 presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra à candidata  
13 para que a mesma fizesse a exposição oral do trabalho de dissertação intitulado:  
14 "FUZDETECT: sistema de detecção e classificação de ataques de negação de serviço".  
15 Concluída a exposição, a candidata foi arguida pela Banca Examinadora que emitiu o  
16 seguinte parecer: "*aprovada*". Do ocorrido, eu, Clauriton de Albuquerque Siebra,  
17 Coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai  
18 assinada por mim e pelos membros da banca examinadora. João Pessoa, 25 de fevereiro de  
19 2019.

  
Prof. Dr. Clauriton de Albuquerque Siebra





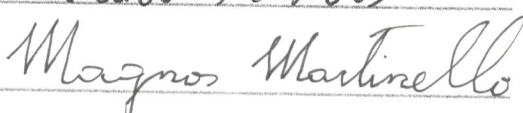
Prof. Dr. Vivek Nigam  
Orientador (PPGI-UFPB)

Prof. Dr. Anand Subramanian  
Examinador Interno (UFPB)

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

Prof. Dr. Fernando Menezes Matos  
Examinador Interno (UFPB)

Prof. Dr. Magnos Martinello  
Examinador Externo ao Programa (UFES)

## AGRADECIMENTOS

Diante de tantos desafios lançados a mim pela própria vida e de outros aos quais eu me lancei, nunca ousei imaginar um dia chegar até aqui. Embora medo e hesitação fossem sentimentos constantes em mim, a vontade de me desafiar mais uma vez e saber até onde a minha sede por conhecimento iria me levar prevaleceu, como vem prevalecendo sempre.

No entanto, sem sombra de dúvidas, sei que a caminhada até aqui não teria êxito se ela fosse realizada por mim sozinha. A cada medo confrontado, a cada insegurança dissipada, sinto Deus em mim, no bater do meu coração e no ar que respiro. E essa certeza de que Ele está em mim é a fonte de toda a minha força.

Esse mesmo Deus também me privilegiou com um anjo de amor que muitas vezes parece até esquecer que, antes de ser mãe, é também mulher, de tão bem desempenhado que ela faz esse papel. Eu te amo, mãe, e nem vivendo mil vidas eu conseguiria agradecer tudo o que você fez e faz por mim!

Agradeço a ti também, Deus, por ter colocado em minha vida um companheiro paciente e amável que, apesar da minha ausência em tantos momentos, sempre me admira e se esforça em me compreender. Eu te amo, Victor!

Gratidão pelos tantos amigos amados, os quais, cada um da sua forma e com seu amor, ajudaram-me para eu chegar até aqui. Seja com um sorriso, um abraço bem forte ou uma palavra de apoio: "Mas é o que você ama, siga seu coração". A lista de nomes seria extensa, mas cada um que ler essa passagem saberá, em seu coração, que é a ele(a) a quem me refiro.

Dentre esses amigos, agradeço especialmente a Leandro Espíndola, que foi além do apoio fraterno e me ajudou bastante, compartilhando boa parte do seu conhecimento comigo, sempre com muita paciência e boa vontade.

Destaco aqui também minha enorme gratidão pelo Professor Vivek Nigam, o qual, mesmo sem nunca ter ouvido falar de mim na área acadêmica, deu-me um voto de confiança e continua fazendo-o até hoje. Destaco ainda a minha admiração por ele como professor, cujo conheci-

mento e humildade em saber ouvir os discentes são notáveis.

Agradeço também ao Professor Iguatemi Fonseca, sempre muito solícito e disponível em ajudar, além de, a todo momento, lembrar-se de mim em meio a oportunidades na área acadêmica. Aqui vai o meu muito obrigada!

Por último, mas não menos importante, agradeço a todo apoio dado pelo Professor Anand Subramanian, o qual, com seu vasto conhecimento, contribuiu com muita presteza e me ajudou bastante com ideias e novas perspectivas acerca do trabalho aqui proposto.

E para finalizar, espero que a vida não deixe de nos desafiar nunca. Afinal, o que é viver, senão a vontade de se descobrir sempre a cada dia?

*"Um barco, no ancoradouro, está seguro. Mas não é para isso que os barcos são feitos."*

William Shedd

## RESUMO

Com o aumento da quantidade de dispositivos conectados à Internet, a rapidez, assim como o dinamismo na troca de informações, foi promovida, junto com uma maior interatividade entre as partes da comunicação. Embora tal aumento tenha trazido convergência imediata na troca de informações, um dos grandes efeitos colaterais acarretados por essa nova visão do modelo da Internet, conhecido como Internet das Coisas, é a facilidade na perpetração do acesso desses dispositivos, uma vez que eles, em sua grande maioria, não apresentam aparatos de segurança essenciais em sua implementação, ocasionando, assim, um crescimento discrepante na magnitude de ataques já existentes, como os Ataques de Negação de Serviço Distribuídos. Ataques de Negação de Serviço variam de acordo com a sua natureza, entretanto, possuem em comum o objetivo de impedir o alvo de atender a novas solicitações e similaridade com o tráfego legítimo. Diante desses desafios, o presente trabalho propõe a criação da solução FuzDetect, o qual não só alerta quando um ataque ocorre, mas também o classifica. O FuzDetect se inicia por meio da coleta de dados em uma Rede Definida por *software*, de forma leve, sem a necessidade do método de inspeção de pacotes tradicional. Em seguida, tais dados são repassados para uma base de dados, a fim de torná-los persistentes. Na etapa final, um sistema de classificação, implementado com Lógica *Fuzzy*, analisa os últimos metadados de fluxos coletados e, então, classifica-os em subtipos de ataque ou em tráfego legítimo. O sistema é capaz de se adaptar ao tráfego da rede de forma dinâmica, com o apoio da Otimização por Enxame de Partículas. Os resultados gerados foram extraídos a partir de testes em redes com topologias e configurações de tráfego diferentes. Esses resultados também foram concebidos com e sem o uso da otimização adaptativa via Otimização por Enxame de Partículas, demonstrando, assim, a eficiência da proposta, na qual, nas situações em que a otimização foi usada, os tráfegos, posteriormente classificados, apresentaram a saída aproximada correta.

**Palavras-chave:** SDN. Segurança. DDoS. Lógica Fuzzy. PSO.



## ABSTRACT

With the increase in the number of devices connected to the Internet, the speed and dynamism in the exchange of information became more relevant, along with greater interactivity between the communication parties. Although such an increase has brought immediate convergence in the exchange of information, one of the biggest side effects of this new Internet model, known as the Internet of Things, is the increasing facility for perpetration in these devices, since in most cases they don't have essential safety appliances in their implementation, this leads to a discrepant growth in the magnitude of existing attacks, such as Distributed Denial of Service Attacks. Denial of Service attacks may vary upon its nature, however they all have a final goal: to prevent the target to receive new requests and at the same time, be similar to a legitimate traffic. Faced with these challenges, this work proposes the creation of FuzDetect solution. A system that not only alerts when an attack is happening, but also classifies it. The FuzDetect system collect data from a Software Defined Networking, through a lightweight method, without inspect packages like traditional methods. Then, all metadata collected will be saved on a database in order to make them persistent. In the final step, a classification system, implemented with fuzzy logic, will analyze the last metadata collected and then classify each one into attack subtypes or legitimate traffic. The system is able to adapt itself to any network dynamically, with the support of Particle Swarm Optimization. The results were extracted from tests in networks with different topologies and traffic configurations. They were also generated with and without the use of Particle Swarm Optimization, thus demonstrating the efficiency of the proposal. When using Particle Swarm Optimization, the traffics, later classified, presented the correct approximate output, in all cases.

**Keywords:** SDN. Security. DDoS. Fuzzy Logic. PSO.

## LISTA DE FIGURAS

2.1	Modelo de funcionamento: Rede Tradicional vs. Rede SDN . . . . .	20
2.2	Ataques DDoS no segundo trimestre de 2017 . . . . .	27
2.3	Conjuntos Clássicos . . . . .	29
2.4	Conjunto <i>Fuzzy</i> : Meia idade . . . . .	30
2.5	Função triangular como função de relacionamento . . . . .	32
2.6	Fuzzificação . . . . .	35
2.7	Conjuntos de saída da variável Z e suas respectivas pertinências de acordo com as regras . . . . .	36
2.8	Conjuntos de saída agregados da variável de saída Z . . . . .	37
2.9	Partículas antes e após convergência em um espaço de busca . . . . .	39
2.10	Arquiteturas baseadas na vizinhança em PSO . . . . .	42
3.1	Utilização de um estimador <i>Fuzzy</i> . . . . .	46
3.2	Sistema de funcionamento - Detecção de ataques DRDoS . . . . .	55
3.3	Esquema de detecção e bloqueio de requisições maliciosas pelo SLICOTS . . . . .	57
4.1	FuzDetect: Arquitetura fluxo de dados . . . . .	59
4.2	FuzDetect: Arquitetura - Otimizador e Classificador . . . . .	61
4.3	Exemplo de saída <i>defuzzificada</i> . . . . .	64
4.4	Conjuntos <i>Fuzzy</i> antes da execução do PSO . . . . .	67
4.5	Conjuntos <i>Fuzzy</i> ajustados pelo PSO . . . . .	68
5.1	FuzDetect: Fluxo de funcionamento entre as máquinas virtuais . . . . .	71
5.2	Topologia I . . . . .	72

5.3	Topologia II . . . . .	73
5.4	Topologia III . . . . .	74
5.5	Topologia IV . . . . .	75
6.1	Tempo em processamento e consumo da memória do classificador em topologias menores . . . . .	85
6.2	Tempo em processamento e consumo da memória do classificador em topologias maiores . . . . .	86
6.3	Tempo em processamento e consumo da memória do PSO em topologias menores	87
6.4	Tempo em processamento e consumo da memória do PSO em topologias maiores	88
6.5	Tempo em processamento e consumo da memória do FuzDetect durante a coleta de tráfego em topologias menores . . . . .	88
6.6	Tempo em processamento e consumo da memória do FuzDetect durante a coleta de tráfego em topologias maiores . . . . .	89
6.7	Tempo em processamento e consumo da memória do FuzDetect durante extração e armazenamento dos dados de tráfego em topologias menores . . . . .	89
6.8	Tempo em processamento e consumo da memória do FuzDetect durante extração e armazenamento dos dados de tráfego em topologias maiores . . . . .	90

## LISTA DE TABELAS

2.1	Campos verificados do cabeçalho do pacote em fluxos . . . . .	21
2.2	Contadores verificados em fluxos SDN . . . . .	22
2.3	Regras <i>Fuzzy</i> . . . . .	36
3.1	Resultado do algoritmo de detecção de ataques DDoS com base em gráficos de controle . . . . .	49
3.2	Campos avaliados em <i>datasets</i> . . . . .	51
3.3	Tráfegos de ataque durante a fase de treinamento e testes utilizando SOM . . .	52
3.4	Resultados da classificação por meio de redes SOM - com 4 e 6 tuplas . . . . .	53
4.1	Estrutura da Tabela na Base de Dados . . . . .	59
4.2	Variáveis de entrada e de saída e seus respectivos conjuntos . . . . .	62
4.3	Regras <i>Fuzzy</i> utilizadas . . . . .	63
6.1	Resultados em tempos de execução (em minutos) com diferentes instâncias do PSO . . . . .	77
6.2	Resultados da função <i>fitness</i> em diferentes instâncias do PSO . . . . .	77
6.3	Resultados - Topologia I - Sem PSO - Experimentos I, II, III, IV . . . . .	78
6.4	Resultados - Topologia II - Sem PSO - Experimentos I, II, III, IV . . . . .	79
6.5	Resultados - Topologia I - Com PSO - Experimentos I, II, III, IV . . . . .	79
6.6	Resultados - Topologia II - Com PSO - Experimentos I, II, III, IV . . . . .	80
6.7	Resultados - Topologia III - Sem PSO - Experimentos I, II, III, IV, V e VI . . .	81
6.8	Resultados - Topologia IV - Sem PSO - Experimentos I, II, III, IV, V e VI . . .	82
6.9	Resultados - Topologia III - Com PSO - Experimentos I, II, III, IV, V e VI . . .	83

6.10	Resultados - Topologia IV - Com PSO - Experimentos I, II, III, IV, V e VI . . .	84
6.11	Resultados em tempos de execução do otimizador por porte da topologia . . . .	84
6.12	Resultados em tempos de execução da coleta de tráfego por porte da topologia .	85
6.13	Resultados em tempos de execução da extração e armazenamento dos dados por porte da topologia . . . . .	86
6.14	Resultados em tempos de execução da classificação por porte da topologia . . .	87
6.15	<i>Precision e Recall</i> por topologia . . . . .	87

## LISTA DE SIGLAS

---

---

**DBA** – *DDoS Blocking Application*

**DDoS** – *Distributed Denial of Service*

**DNS** – *Domain Name System*

**DRDoS** – *Distributed Reflection Denial of Service*

**DWT** – *Discret Wavelet Transform*

**DoS** – *Denial of Service*

**HTTP** – *Hypertext Transfer Protocol*

**IP** – *Internet Protocol*

**ISP** – *Internet Service Provider*

**IoT** – *Internet of Things*

**PSO** – *Particle Swarm Optimization*

**RTT** – *Round Trip Time*

**SDN** – *Software Defined Network*

**SIC** – *Schwarz Information Criterion*

**SOM** – *Self Organizing Maps*

**TCP** – *Transport Control Protocol*

**TTL** – *Time To Live*

**UDP** – *User Datagram Protocol*

# SUMÁRIO

<b>CAPÍTULO 1 – INTRODUÇÃO</b>	<b>15</b>
1.1 Objetivos . . . . .	18
1.2 Estrutura da Dissertação . . . . .	18
<b>CAPÍTULO 2 – FUNDAMENTAÇÃO TEÓRICA</b>	<b>19</b>
2.1 Software Defined Networks - SDN . . . . .	19
2.2 A Segurança da Informação . . . . .	22
2.3 Ataques de Negação de Serviço . . . . .	24
2.3.1 Tipos de ataques DDoS . . . . .	25
2.3.2 Ataques DDoS em 2017 . . . . .	26
2.4 Lógica <i>Fuzzy</i> . . . . .	28
2.4.1 Conjuntos Clássicos vs. Conjuntos <i>Fuzzy</i> . . . . .	29
2.4.2 Variáveis Linguísticas . . . . .	31
2.4.3 Conjuntos <i>Fuzzy</i> e Funções de Pertinência . . . . .	31
2.4.4 Inferência e Operadores <i>Fuzzy</i> . . . . .	33
2.4.5 Relacionamentos <i>Fuzzy</i> . . . . .	34
2.4.6 <i>Defuzzificação</i> por meio da Inferência de Mamdani . . . . .	36
2.5 Otimização por Enxame de Partículas - PSO . . . . .	38
2.5.1 Funcionamento do PSO . . . . .	38
2.5.2 Modelos de Vizinhança em PSO . . . . .	41

<b>CAPÍTULO 3 – TRABALHOS RELACIONADOS</b>	<b>43</b>
3.1 Lógica <i>Fuzzy</i> . . . . .	43
3.2 Aprendizado de Máquina . . . . .	50
3.3 Outras abordagens . . . . .	53
<b>CAPÍTULO 4 – FUZDETECT - COMPONENTES E FUNCIONAMENTO</b>	<b>58</b>
4.1 Coleta de Fluxos . . . . .	58
4.2 Extração de Dados . . . . .	58
4.3 Base de Dados . . . . .	59
4.4 Estabelecimento das Variáveis de Entrada . . . . .	60
4.5 Sistema de Classificação - Lógica <i>Fuzzy</i> . . . . .	61
4.5.1 Variáveis de Entrada e Saída <i>Fuzzy</i> estabelecidas . . . . .	62
4.5.2 Regras <i>Fuzzy</i> utilizadas . . . . .	62
4.5.3 Defuzzificação . . . . .	63
4.6 Execução do PSO . . . . .	66
<b>CAPÍTULO 5 – TESTBED</b>	<b>70</b>
5.1 Ambiente . . . . .	70
5.2 Topologias . . . . .	71
5.3 Geração de Tráfego DDoS e Legítimo . . . . .	72
<b>CAPÍTULO 6 – RESULTADOS E DESEMPENHO</b>	<b>76</b>
6.1 Otimização . . . . .	76
6.2 Classificação . . . . .	77
6.3 Resultados . . . . .	78
6.4 Performance FuzDetect . . . . .	81
<b>CAPÍTULO 7 – CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS</b>	<b>91</b>





# Capítulo 1

## INTRODUÇÃO

---

---

Com o avanço da Internet, assim como da *Web*, boa parte do que se conhece como informação passou a ser obtida por meio virtualizado, pela rede mundial de computadores. Apesar do termo ainda ser bastante utilizado na literatura, a rede mundial não é representada unicamente por computadores, mas vai desde geladeiras a automóveis, os quais possuem identificação na rede por meio do Protocolo de Internet (*Internet Protocol* - IP). A essa inclusão de diversos novos tipos de dispositivos dá-se o nome de Internet das Coisas (*Internet of Things* - IoT). Conforme definido pela *Hewlett Packard Enterprise Development* [1], o termo IoT descreve o grande, e cada vez maior, conjunto de dispositivos digitais — agora na casa dos bilhões — que operam entre redes de escala potencialmente global.

De acordo com Gubbi et al. [2], IoT é mais que uma rede de objetos interconectados, mas também usa os padrões existentes da Internet para fornecer serviços para transferência de informações, análises, aplicativos e comunicações.

Embora essa abrangência de novos dispositivos, bem como o barateamento no custo de *links* de Internet, traga inclusão, dinamismo e fluidez na entrega e no envio das informações, principalmente quando tais equipamentos estão na Internet, alguns problemas emergiram desse novo cenário tecnológico. Novos tipos de ataques a serviços específicos estão aparecendo de forma gradativa e cada vez mais refinados, assim como outros ataques, que já eram conhecidos, ganharam magnitude. Quanto mais dispositivos são introduzidos na rede mundial, maiores são as chances de se tornarem mão de obra para determinados tipos de ataques, como afirma a Arbor Networks [3].

Nesse sentido, para o fim proposto aqui neste trabalho, o foco será o Ataque de Negação de Serviço (*Denial of Service* - DoS), também conhecido pela sua versão distribuída, Ataque de Negação de Serviço Distribuído (*Distributed Denial of Service* - DDoS), em que, nesse

último, vários atacantes executam o mesmo ataque. O principal objetivo de ambos é afetar a disponibilidade da rede ou do servidor alvo.

Assim, de acordo com a Global Dots [4]):

Um ataque DDoS tem o objetivo de tornar um site ou uma rede indisponível para os usuários, de modo que um bem-sucedido atingirá uma base de dados de usuários inteira. Em um ataque DoS, um perpetrador pode usar uma única conexão à Internet para explorar uma vulnerabilidade de *software* ou inundar o alvo com solicitações falsas e, finalmente, fazer com que o site fique indisponível, o impedindo de responder aos pedidos dos usuários legítimos.

Logo, como também afirmam Mirkovic e Reither [5], o objetivo de um ataque DDoS é impedir que usuários legítimos acessem o serviço alvo. Ainda, em 2017, foi registrado pela Arbor Networks [3] um pico de 622Gbps (*Gigabits* por segundo) de tráfego DDoS. De acordo com a Global Dots [4], a Akamai acredita que, até 2020, a média de ataques DDoS chegará a 1,5Tbps (*Terabits* por segundo).

No CertBR [6], é definido que os ataques DDoS podem ser subdivididos em tipos, de acordo com sua natureza. São alguns deles: ataques à camada de aplicação, ataques de exaustão de recursos de *hardware* e ataques volumétricos. Esses ataques podem ser realizados de forma isolada ou em conjunto. Um atacante pode, por exemplo, realizar um ataque apenas à camada de aplicação ou combiná-lo com um ataque volumétrico. Ataques combinados costumam ter mais impacto, pois requerem o uso conjunto de diferentes formas de preparação, detecção, análise e mitigação. Esses ataques também podem ser classificados de acordo com a camada da pilha TCP/IP alvo, logo, definidos como Ataques à camada de aplicação e Ataques à camada de rede, como afirma a Global Dots [4].

Mirkovic e Reither [5], por sua vez, alegam que:

Os ataques DDoS representam uma ameaça à Internet e muitos mecanismos de defesa foram propostos para combater o problema. Os atacantes estão modificando suas ferramentas de forma constante a fim de burlar mecanismos de segurança utilizados pelas instituições.

Pode-se afirmar, assim, que uma das grandes dificuldades em se identificar ataques de negação de serviço é sua semelhança com o tráfego convencional, desde os cabeçalhos dos pacotes de rede à sua carga útil. Atualmente, sistemas de detecção no mercado, que tanto executam a identificação do tráfego de ataque como o classificam também, são proprietários e de alto custo financeiro e computacional, considerando que precisam analisar a carga de dados pacote a pacote, até a camada mais alta, conforme asseveram Braga, Mota e Passito [7].

Dessa forma, diante dos desafios apresentados neste trabalho, propõe-se a criação do FuzDetect, cujas principais contribuições são:

- **Identificação e classificação do tráfego globalmente:** a solução é capaz de identificar ataques DDoS em qualquer ponto de enlace da rede e não exclusivamente no *backbone*. O FuzDetect também é apto a categorizar ataques (tais categorias serão apresentadas posteriormente) por meio da Lógica *Fuzzy* [8], a qual reflete a maneira como as pessoas pensam, tentando modelar o seu senso de palavras, tomada de decisão ou senso comum. Como consequência, a introdução da Lógica *Fuzzy* tem conduzido as pesquisas para sistemas inteligentes mais humanos e mais adequados à realidade, como atestam Costa et al. [9].
- **Autoconfiguração em função da rede alvo:** diante das singularidades que cada rede possui quanto às características do seu tráfego, torna-se necessário algum meio de adaptar a configuração do seu classificador à rede analisada. Essa propriedade adaptativa é obtida por meio da Otimização por Enxame de Partículas (*Particle Swarm Optimization* - PSO) [10].
- **Mecanismo leve de coleta de dados:** o FuzDetect coleta e extrai dados da rede de forma leve, sem inspeção de pacotes. Para esse fim, ele usufrui de um dos principais benefícios de uma Rede Definida por Programação (*Software Defined Networking* - SDN) [11], que é a visão global da rede alvo, por meio do protocolo *OpenFlow* [12], o que torna mais ágil o processamento dos dados.

Assim sendo, em face do comportamento complexo em uma rede de dados e de padrões nem sempre facilmente identificáveis, a Lógica *Fuzzy* foi implementada no FuzDetect com o objetivo de classificar o tráfego dessa rede. De acordo com Ross (2009, cap. 1, p. 7) [8]:

O benefício primário da teoria dos sistemas difusos é compreender os sistemas que são desprovidos de análises formuladas: sistemas complexos. Uma solução aproximada e que possa economizar custos computacionais; em que os insumos para um problema são vagos, ambíguos ou desconhecidos [...].

Com a finalidade de tornar o FuzDetect adaptado ao tráfego da rede analisada de forma dinâmica, uma vez que cada rede possui suas particularidades de volume e demanda de tráfego, introduziu-se o PSO como mecanismo de adaptação do classificador à rede, durante a configuração dos conjuntos *fuzzy*. Logo, conforme mencionado por Xiahoui [13]:

Comparado com os Algoritmos Genéticos, o mecanismo de compartilhamento de informações no PSO é significativamente diferente. Em Algoritmos Genéticos, os cromossomos compartilham informações uns com os outros, assim toda a população se move como um grupo para uma área ideal. No PSO, apenas o gBest (ou lBest) fornece a informação a outros indivíduos. É um mecanismo de compartilhamento de informações de uma via. A evolução apenas procura a melhor solução. Em comparação com Algoritmos Genéticos, todas as partículas tendem a convergir rapidamente para a melhor solução, mesmo na versão local, na maioria dos casos.

## 1.1 Objetivos

Este trabalho tem o objetivo geral de classificar, de forma aproximada, o tráfego de uma determinada rede alvo, por meio da Lógica *Fuzzy*. Para atingir esse objetivo, seguem os objetivos específicos:

- coletar dados estratégicos através de uma rede SDN, por meio de informações *built-ins* do protocolo *OpenFlow* [12];
- simular tráfego de diversos tipos (legítimo e de ataque) e com configurações diferentes quanto ao tamanho da carga de dados, intervalo de tempo entre o envio dos pacotes e outros;
- tornar o classificador adaptado à massa de dados da rede analisada, por meio do PSO;
- validar o classificador por meio de diversos experimentos com tráfegos diferentes.

## 1.2 Estrutura da Dissertação

Após a Introdução, apresenta-se a estrutura desta dissertação. Assim, no Capítulo 2, será dado o embasamento teórico acerca dos temas anteriormente mencionados: SDN, Segurança da Informação, Ataques DDoS, Lógica *Fuzzy* e PSO. No Capítulo 3, em sequência, serão mencionados os trabalhos relacionados a esta pesquisa. Já no Capítulo 4, serão descritas todas as etapas executadas pelo FuzDetect, desde a coleta de fluxos à classificação do tráfego. No Capítulo 5, por sua vez, será apresentada a configuração dos experimentos em detalhes, desde a configuração dos ambientes a ferramentas e parâmetros utilizados. No Capítulo 6 serão exibidos detalhes acerca da otimização e da classificação do FuzDetect, os resultados obtidos com o FuzDetect, aliados a uma discussão sobre eles e à análise de desempenho do sistema. Por fim, no Capítulo 7, a proposta será finalizada com algumas ideias acerca dos benefícios da solução e do trabalho como um todo, além de possíveis aperfeiçoamentos para o futuro.

# Capítulo 2

## FUNDAMENTAÇÃO TEÓRICA

---

---

Neste Capítulo, serão abordados os principais tópicos citados: SDN, Segurança da Informação, Ataques DDoS, Lógica *Fuzzy* e PSO, fornecendo-se, assim, maiores detalhes conceituais sobre eles.

### 2.1 Software Defined Networks - SDN

Diferentemente das redes tradicionais, em que a definição de como e para onde enviar determinado pacote é feita pelo próprio ativo de rede (roteador, por exemplo), o *Software Defined Networks* (SDN) propõe um novo paradigma de funcionamento de rede, de modo que o plano de controle é desacoplado do plano de dados. Logo, as políticas de envio de tráfego, interfaces de saída selecionadas, dentre outras informações inerentes ao plano de controle, são definidas por uma única entidade, normalmente chamada de controlador, enquanto que, nos ativos de rede, o plano de dados continua existindo, executando a transmissão dos dados de acordo com as regras ditadas pelo controlador envolvido.

Assim, de acordo com Nadeau e Gray (2013, cap. 1, p. 7) [11]:

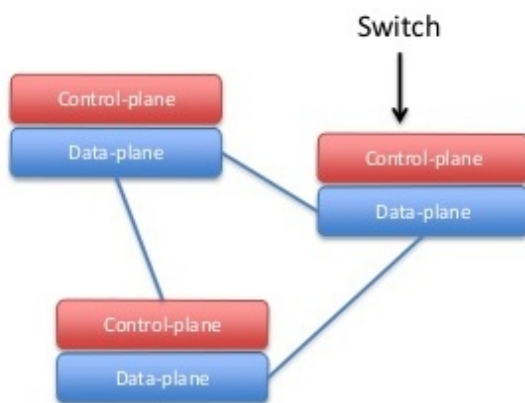
SDN é uma abordagem arquitetônica que otimiza e simplifica as operações de rede ao associar de forma mais vinculada a interação (ou seja, provisionamento, mensagens e alarmes) entre aplicativos, serviços e dispositivos de rede, sejam reais ou virtualizados. Isto é obtido empregando um ponto de controle lógico de rede centralizado - o que muitas vezes é realizado por meio de um controlador SDN - que então orquestra, media e facilita a comunicação entre aplicações que desejam interagir com elementos de rede e elementos de rede que desejam transmitir informações para essas aplicações. O controlador então expõe e abstrai funções de rede e operações, através de interfaces programáticas modernas, amigáveis e bidirecionais para aplicações.

Em Sezer et al. [14], afirma-se que, na arquitetura SDN, o plano de dados e o de controle

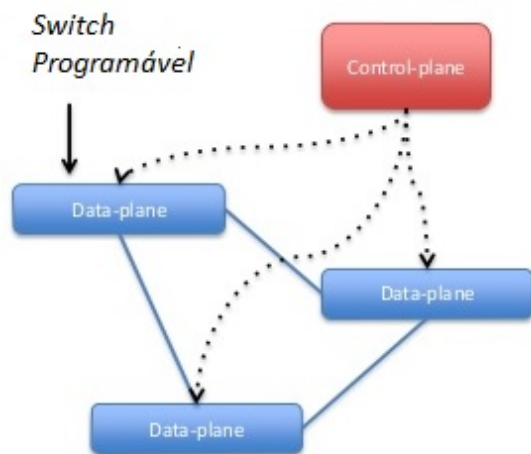
estão desvinculados e a inteligência da rede é logicamente centralizada, assim como a infraestrutura de rede subjacente é abstraída das aplicações. Na Figura 2.1, é apresentado o modelo tradicional, com o plano de dados e o de controle no mesmo equipamento e o modelo trazido pelas redes SDN, no qual o plano de controle, representado pelo controlador, mantém-se como uma entidade à parte.

## SDN - O NOVO PARADIGMA

### Redes Tradicionais



### Redes SDN



**Figura 2.1: Modelo de funcionamento: Rede Tradicional vs. Rede SDN**

Fonte: Disponível em:

<<https://www.slideshare.net/commtechpolimi/tutorial-on-sdn-data-plane-evolution>>. Acesso em: 5 jul. 2018.

De acordo com Shin, Nam e Kim [15], algumas das principais motivações quanto à implementação do SDN são:

- o SDN traria facilidade e eficiência no controle e no gerenciamento da rede;
- capacidade de orquestrar, de forma global, milhões de ativos de rede de forma centralizada e inteligente;
- implementação de qualidade de serviço por cliente, de maneira dinâmica e seguramente isolada do tráfego de outros clientes;

- redução na complexidade de configuração, pois todos os ativos compreenderiam o mesmo padrão de comunicação.

A programação de uma rede SDN é feita por meio do padrão *OpenFlow*. Um *switch OpenFlow*, por sua vez, consiste em uma ou mais tabelas de fluxo e uma tabela de grupo que executam inspeção de cabeçalho de pacotes e seu encaminhamento, além de um canal *OpenFlow* para um controlador externo. Usando esse protocolo, o controlador pode adicionar, atualizar e excluir os fluxos nas tabelas de fluxos dos ativos controlados por ele. Cada tabela de fluxo no *switch* contém um conjunto de entradas de fluxo e cada entrada de fluxo compreende campos chaves, contadores e um conjunto de instruções para aplicar nos pacotes correspondentes, conforme McKeown et al. [12].

Nesse sentido, uma regra na tabela de fluxos de um *switch* se baseia em um conjunto de campos chaves que indicam quando uma regra deve ser ativada. A cada novo pacote que alcança o *switch*, este checa se alguma de suas entradas na tabela de fluxo satisfaz o pacote. Alguns campos checados são: IP de origem, IP de destino, portas de camada de transporte, protocolo de transporte e outros. Caso exista uma regra correspondente, o *switch* a aplica; caso contrário, o *switch* questiona o controlador, o qual poderá instalar uma nova regra na tabela de fluxos. A Tabela 2.1 a seguir apresenta os principais campos checados durante a verificação de cabeçalhos de pacotes que acabam de chegar em uma determinada interface de um *OpenFlow Switch*.

**Tabela 2.1: Campos verificados do cabeçalho do pacote em fluxos**

<b>Cabeçalhos - Fluxos SDN</b>
Interface de Entrada
Metadados
Endereço físico - Interface de Entrada
Endereço físico - Interface de Saída
Protocolo de Camada de Enlace
VLAN id
VLAN prioritária
MPLS rótulo
MPLS classe de tráfego
IPV4 origem
IPV4 destino
IPV4 ToS
IPV4 protocolo
TCP/UDP porta de origem
TCP/UDP porta de destino

Além desses campos que identificam um fluxo de dados, os *switches* também contêm contadores com mais detalhes sobre o fluxo, como o número de vezes que uma regra da tabela



de fluxo foi ativada, ou o número de pacotes/*bytes* por fluxo. A Tabela 2.2 possui alguns dos principais contadores em fluxos.

**Tabela 2.2: Contadores verificados em fluxos SDN**

<b>Contadores - Fluxos SDN</b>	
Contadores	Bits
Por Tabela	
Entradas Ativas	32
Pacotes <i>Lookups</i>	64
Pacotes <i>Matching</i>	64
Por Fluxos	
Pacotes Recebidos	64
Bytes Recebidos	64
Duração (segundos)	32
Duração (nanossegundos)	32
Por Porta	
Pacotes Recebidos	64
Pacotes Enviados	64
Bytes Recebidos	64
Bytes Transmitidos	64
Drops Recebidos	64
Drops Transmitidos	64
Erros Recebidos	64
Erros Transmitidos	64
Quadros Recebidos por Erros de Alinhamento	64
Erros excedentes recebidos	64
CRC Erros	64
Colisões	64
Por Fila	
Pacotes transmitidos	64
Bytes transmitidos	64
Erros excedentes recebidos	64
Por Grupo	
Quantidade de Fluxos	32
Quantidade de Pacotes	64
Quantidades de Bytes	64
Por Bucket	
Quantidade de Bytes	64
Quantidade de Pacotes	64

Diante disso, é possível definir características relevantes que podem ser obtidas por meio desses campos chaves e de seus contadores, em toda a rede e não só no tráfego que passa pelo *backbone*. Tais características serão estratégicas para o fim aqui proposto.

## 2.2 A Segurança da Informação

Conforme mencionado na Introdução, embora as últimas décadas tenham trazido rapidez e eficiência na forma de se comunicar por meio da alta tecnologia de dispositivos IPs, tal avanço

também acarretou novas preocupações acerca da segurança da informação.

Assim, de acordo com Oliveira e Moura [16], a segurança precisa ser obtida dentro de um nível hierárquico da informação, que pode ser compreendido por: o dado, a informação e o conhecimento. Entretanto, é necessário que se tenham noções de segurança da informação, pois a informação poderá sofrer ruídos ou ser danificada até que se chegue ao destino final. Os conceitos desses três níveis hierárquicos (dado, informação e conhecimento) sofrem variações, mas existem explicações dentro desse contexto, ou seja, um conjunto de dados não produz necessariamente uma informação, nem um conjunto de informações representa necessariamente um conhecimento.

Em síntese, um dado pode ser entendido como registros ou fatos em sua forma primária, não necessariamente física. E quando esses fatos e registros são organizados ou têm uma combinação significativa, eles se transformam em uma informação. Da mesma forma que a informação é produzida a partir de dados, o conhecimento também tem como origem a informação, quando ela é agregada com outros elementos. Isto é, a informação está diretamente ligada à representatividade ou significância do dado ou dados em conjunto. Quanto à importância da informação, afirma Gouveia [17]:

A informação é utilizada pelas organizações independentemente de serem lucrativas ou não. A informação satisfaz as necessidades de informação, de modo a permitir o reconhecimento das características de um determinado contexto e suportar a decisão, com vista a garantir uma ação de acordo com os objetivos pretendidos e com a realidade envolvida.

Logo, manter de forma eficiente todo esse banco de informação é indispensável para qualquer negócio, independentemente de sua natureza ou ramo. Tendencialmente, a segurança da informação não deve mais ser vista como área meio, pois, ainda que a informação em foco não esteja diretamente vinculada ao serviço ofertado, obtendo uma visão global do negócio, é possível verificar como todas as informações são partes indispensáveis para o funcionamento do todo. Portanto, a apresentação correta desses aspectos da informação garante a credibilidade e a confiança nas organizações e nos negócios e pode ser alcançada com base na aplicação de controles. E para atingir esse objetivo, é necessário preservar três aspectos críticos da informação: a confidencialidade, a integridade e a disponibilidade, conforme Gouveia [17], os quais são descritos a seguir:

- **confidencialidade:** de acordo com Wetherall e Tanenbaum (2011, cap. 8, p. 479) [18], a confidencialidade ou sigilo está relacionado ao fato de manter as informações longe de usuários não autorizados. Vinculado a esse pilar, existem outros dois pontos: a autenticação, que determina quem está tentando obter a informação, e o não repúdio, ou ação de

comprovar que você é quem você diz ser. Esses aspectos são satisfeitos unicamente na camada de aplicação;

- integridade: o princípio da integridade é respeitado quando a informação acessada está completa, sem alterações, e, portanto, confiável. Ou seja, quando a informação é alterada ou chega ao destino final de forma incorreta, isso faz com que a integridade se quebre Campos [19];
- disponibilidade: segundo Gouveia [17], a disponibilidade significa que a informação está acessível a sistemas e utilizadores autorizados, sem qualquer interferência ou obstrução e de modo a ser devidamente percebida, isto é, no formato requerido.

Está fora do propósito deste trabalho adentrar no mundo das normas de segurança da informação, no entanto, para a finalidade proposta e dentro do escopo deste estudo, a atenção será voltada para o pilar da disponibilidade, o qual é o mais afetado em um cenário de DDoS.

## 2.3 Ataques de Negação de Serviço

Como já dito antes e citado pela Global Dots [4], com a crescente popularidade de sites e aplicativos baseados na *web*, veio um *boom* correspondente nos números, tipos, magnitudes e custos de ataques que visam especificamente às vulnerabilidades desses sistemas. Dentre esse emaranhado de modalidades de ataques, o que mais se destaca para o fim aqui descrito são os ataques DDoS. Ainda de acordo com a Global Dots [4], tais ataques representam mais de 55% de todos os crimes cibernéticos anuais e são os que causam mais prejuízos às instituições alvos. Segundo o CertBR [6], um ataque de Negação de Serviço pode ser definido como:

Negação de serviço é uma técnica pela qual um atacante utiliza um equipamento conectado à rede, para tirar de operação um serviço, um computador ou uma rede conectada à Internet. Quando usada de forma coordenada e distribuída, ou seja, quando um conjunto de equipamentos é utilizado no ataque, recebe o nome de Ataque Distribuído de Negação de Serviço (DDoS). Um ataque DDoS não tem o objetivo direto de invadir e nem de coletar informações, mas sim de exaurir recursos e causar indisponibilidade ao alvo [...].

O objetivo principal dessa vertente de ataque é afetar diretamente o pilar da disponibilidade da segurança da informação (descrito em 2.2). Como bem exposto por Mirkovic e Reither [5], em seu trabalho sobre a taxonomia de ataques DDoS, esses estão constantemente sendo atualizados e ficando cada vez mais sofisticados, a fim de se tornarem mais invisíveis aos olhos

do administrador da rede ou sistema. O campo dos ataques DDoS, portanto, está cada vez mais complexo, trazendo estruturas de ataques variadas e difíceis de explorar e endereçar.

No próximo tópico, será feita uma descrição acerca da natureza dos ataques DDoS e, em seguida, serão enfatizados casos reais ocorridos no ano de 2017, descrevendo-se desde a tendência dos últimos ataques às *botnets* utilizadas com mais destaque até outras informações mais recentes.

### 2.3.1 Tipos de ataques DDoS

Ataques DDoS possuem diversas variantes, mas basicamente seus objetivos se resumem a três: saturar a largura de banda da rede alvo; exaurir recursos computacionais de *hardware*, seja carga ou memória; ou, ainda, em mais alto nível, esgotar a capacidade de atendimento de novas requisições por parte da aplicação, por meio de estouro de *threads* ou conexões, em um servidor HTTP, por exemplo. Ainda nesse contexto, os ataques se dividem de acordo com a camada da pilha TCP/IP que se deseja atacar, podendo ser Ataques à camada de Rede ou Transporte (também chamados de Ataques à Infraestrutura) e Ataques à camada de Aplicação, como segue:

- **Ataques à Infraestrutura:** esses ataques são aqueles vistos com frequência na mídia, os quais são atribuídos à interrupção em muitos sites. Inundações SYN, Inundações ACK ou ataques de amplificação baseados em UDP podem ser classificados como ataques desse tipo. São normalmente medidos em Gbps (*gigabits* por segundo), pela quantidade de largura de banda que podem consumir por segundo [4]. A Arbor Networks relatou alguns em 2015 e descreveu que 17% de todos os ataques que eles manipularam como maiores que 1GBps tinham tamanho médio de 804 Mbps (*megabits* por segundo) e os grandes atingiram o pico de 335 Gbps [3]. Esses ataques também tentam consumir a capacidade dos equipamentos e exaurir seus recursos, por exemplo, em roteadores, tentam consumir recursos como carga e memória, além da capacidade de encaminhamento de pacotes por segundo [6];
- **Ataques à Camada de Aplicação:** esses podem ser menos notáveis e silenciosos em comparação com os ataques à infraestrutura, mas são perturbadores e, de fato, mais complexos de serem manipulados. Tal gama de ataques foge do escopo deste trabalho.

Assim sendo, a proposta aqui apresentada busca a detecção de dois subtipos de ataques de negação de serviço à infraestrutura, a saber: a) **Ataques de Exaustão:** buscam saturar recursos

lógicos como físicos, por exemplo, esgotamento de sessões *web* e TCP em geral, *threads* HTTP ou *threads* do sistema operacional, degradação de arquivos descritores, entre outros; e b) **Ataques Volumétricos**: são direcionados à capacidade da largura de banda do alvo e focados em saturação de tráfego da placa de rede e do enlace envolvido.

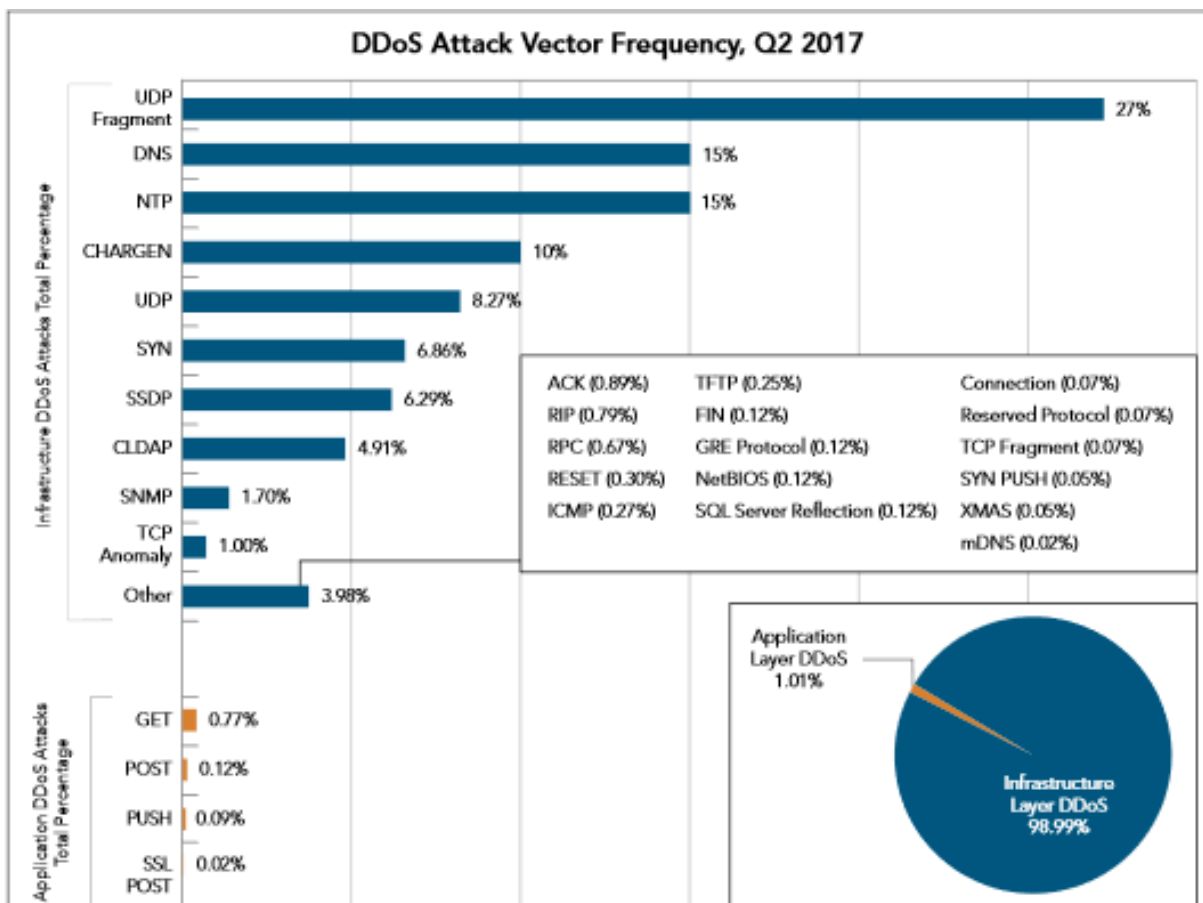
### 2.3.2 Ataques DDoS em 2017

De acordo com os relatórios da Akamai, por Lewis [20], em relação ao segundo trimestre de 2017, ataques relacionados à infraestrutura representaram 99% do tráfego DDoS, como tem sido típico em trimestres recentes. Os ataques de infraestrutura dominam porque os atacantes acham esses muito mais fáceis do que lançar um ataque à camada de aplicação. Os ataques DDoS da camada de aplicação representaram 1% dos ataques DDoS vistos pela Akamai.

Na Figura 2.2 a seguir, referente ao relatório citado, as barras de progressão em azul se referem a alguns ataques DDoS voltados à infraestrutura, já as em laranja se referem aos voltados à camada de aplicação. Outros ataques cujo alvo foi a camada de transporte ou ataques voltados à infraestrutura que ocorreram com menos frequência não estão incluídos nas barras de progressão, mas podem ser vistos também na mesma figura.

De acordo com a Arbor Networks, a causa desse crescimento nos ataques DDoS se deve ao fato de que os ataques DDoS foram democratizados por serviços de ataque de baixo custo (também conhecido como *booter* / *stressers*), que lançam ataques com grande facilidade. Existem também centenas de ferramentas que permitem a qualquer pessoa com uma conexão à Internet iniciar um ataque. Além disso, os dispositivos incorporados IoT são altamente vulneráveis, geralmente sempre ligados, e as redes onde residem oferecem conexões de alta velocidade, o que permite uma quantidade relativamente alta de volume de tráfego de ataque DDoS por dispositivo comprometido [3].

É possível verificar também, por meio dos dados registrados pela Akamai, por Lewis [20], que o foco desses ataques passou a ser a indústria de jogos, nos períodos do segundo e do terceiro trimestre de 2017. Khalomonenko, Kupreev e Ilganaev [21] descrevem que em meados de agosto, a *Blizzard Entertainment* reportou uma inundação de tráfego que causou problemas para jogadores de *Overwatch* e *World of Warcraft*. No início de setembro, o site de *Poker online* da *Americas Cardroom* começou a ter dificuldades. O ataque seguiu o padrão notório "demonstrar força, exigir um resgate". A administração do site se recusou a pagar, mas foi forçada a cancelar - ou mais precisamente, a adiar - um campeonato de *poker* que já estava em andamento. No final do trimestre, em 30 de setembro, o site da Loteria Nacional do Reino Unido foi seriamente afetado: por 90 minutos, os jogadores não conseguiram colocar suas



**Figura 2.2: Ataques DDoS no segundo trimestre de 2017**

Fonte: Lewis (AKAMAI), 2017.

apostas *online* ou por meio de aplicativos, o que causou prejuízos graves no serviço.

Ainda de acordo com Khalomonenko, Kupreev e Ilganaev [21], o ano de 2017 também trouxe mais sofisticções nas ferramentas de ataque, como o *botnet WireX*: centenas de milhares de dispositivos em mais de uma centena de países. O *botnet* (rede de *bots*) estava secretamente trabalhando em dispositivos Android e sendo proliferado por meio de aplicativos legítimos do *Google Play*. As ações conjuntas do Google, Samsung e vários outros grandes fornecedores de segurança de Tecnologia da Informação (TI) conseguiram mitigar o *botnet*. Dado o deplorável estado de segurança em IoT, essas descobertas agora provavelmente ocorrerão de forma bastante regular. Em meados de agosto, a Imperva descreveu a tecnologia *Pulse Wave*, capaz de aumentar o poder de um ataque DDoS graças a uma vulnerabilidade em tecnologias híbridas e em nuvem. Os analistas da Imperva acreditam que a maioria dos ataques de DDoS seguirão em breve um padrão similar: ataques de "pontamentos" súbitos breves, mas poderosos e que durarão várias horas ou vários dias.

## 2.4 Lógica *Fuzzy*

A Lógica *Fuzzy* nasceu da motivação de que nem sempre cada elemento de um universo de amostras pode pertencer totalmente a um grupo ou categoria. Os graus de pertinência dos elementos possuem variações suaves ou granulares no intervalo real  $[0,1]$ , representando, assim, o conhecimento humano. Logo, conforme Zadeh [22]:

Lógica *Fuzzy* é uma metodologia em computação voltada para o uso de palavras. Tradicionalmente, computação envolve o uso de números e símbolos, entretanto, o homem emprega mais palavras durante a própria computação e o raciocínio, dando assim às expressões uma percepção mental.

Já em Ross (2009, cap. 1, p. 7) [8], é afirmado que:

O principal benefício da teoria dos sistemas difusos é aproximar-se ao comportamento de sistemas onde as funções analíticas ou as relações numéricas não existem. Por isso, os sistemas *fuzzy* têm alto potencial para compreender os sistemas que são desprovidos de análises formuladas: sistemas complexos. Sistemas complexos podem ser novos sistemas que não foram testados, podem ser sistemas envolvidos com a condição humana, como biológico ou sistemas médicos; ou podem ser sistemas sociais, econômicos ou políticos, onde o vasto vetor de entradas e saídas não poderiam ser capturados analiticamente ou controlados em qualquer sentido convencional. Além disso, a relação entre as causas e os efeitos nesses sistemas geralmente não são entendidos, mas muitas vezes podem ser observados. Alternativamente, a teoria dos sistemas difusos pode ter utilidade na avaliação de alguns de nossos sistemas mais convencionais e menos complexos. Por exemplo, para alguns problemas, as soluções exatas são nem sempre necessárias. Uma solução aproximada, mas rápida, pode ser útil para fazer uma decisão preliminar ou como uma estimativa inicial em uma técnica numérica mais precisa para economizar custos computacionais; ou na miríade de situações em que os insumos para um problema são vagos, ambíguos ou desconhecidos.

De acordo com Mamdani [23], o fato da matemática como um todo ter sido tomada como sinônimo de precisão vem ganhando preocupação entre cientistas e filósofos ao lidar com problemas do mundo real. Existe uma espécie de *gap* entre o que se aplica na teoria e a interpretação dos resultados em um mundo inexato. Muitos pensadores vêm contribuindo com as discussões acerca da imprecisão, conseqüentemente, focando-se na subjetividade humana.

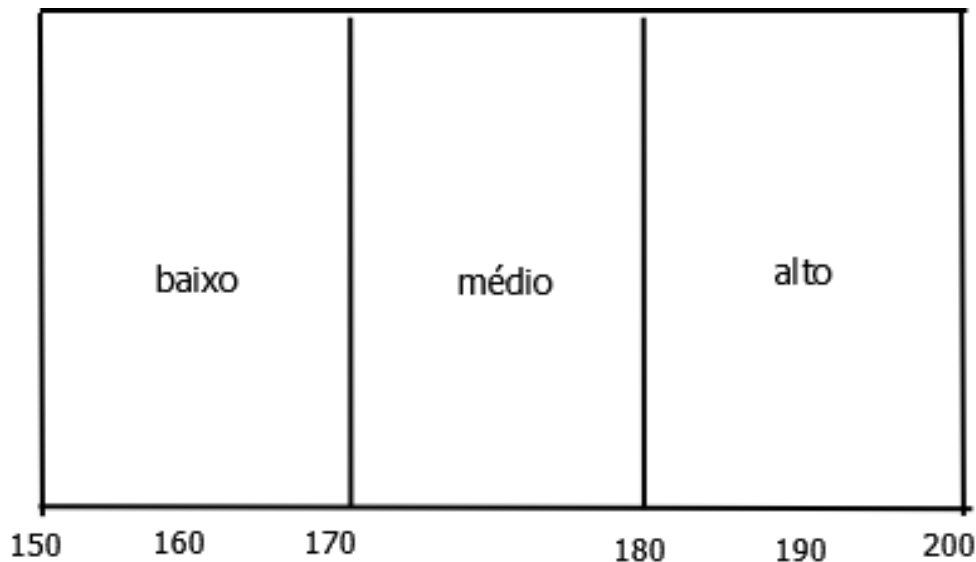
Lee [24], por sua vez, reitera que Lógica *Fuzzy* é muito mais próxima do pensamento humano que as lógicas tradicionais, fornecendo assim um meio de capturar a inexatidão do mundo real. Em essência, um sistema *fuzzy* é capaz de construir, de forma automatizada, um sistema de controle por meio do conhecimento humano no contexto inserido.

### 2.4.1 Conjuntos Clássicos vs. Conjuntos *Fuzzy*

Na teoria dos conjuntos clássicos, um subconjunto  $A$  de um conjunto  $X$  pode ser definido por suas características, em função de  $\chi_A$ , como um mapeamento dos elementos de  $X$  para os elementos do conjunto  $\{0, 1\}$ , conforme Equação 2.1.

$$\chi_A : X \rightarrow \{0, 1\} \quad (2.1)$$

Esse mapeamento pode ser representado como um conjunto de pares ordenados, com exatamente um par para cada elemento de  $X$ . O primeiro elemento do par é um elemento do conjunto  $X$  e o segundo elemento é um elemento do conjunto  $\{0, 1\}$ . O valor zero é usado para representar não pertencimento ao conjunto e o valor um é usado para representar a associação, como explica Fuller [25]. Na Figura 2.3, é apresentado um exemplo de conjuntos clássicos quanto à altura de um indivíduo, em que esta é dividida em três conjuntos: baixo, médio e alto, os quais possuem limitadores bem específicos de modo que o indivíduo só será exclusivamente baixo, médio ou alto. A função de pertinência de cada elemento do universo de discurso para um subconjunto clássico é apresentada na Equação 2.1 supracitada.



**Figura 2.3: Conjuntos Clássicos**

Fonte: Elaborada pela autora.

Desse modo, um conjunto *fuzzy*  $A$ , a partir de um universo  $X$ , pode ser definido como um conjunto de pares, no qual, em cada um, o primeiro elemento é um elemento de  $X$  e o segundo elemento pertence ao intervalo  $[0, 1]$ , com exatamente um par ordenado presente para cada elemento de  $X$ . Isso define um mapeamento,  $\mu_A$ , entre elementos do conjunto  $X$  e os valores

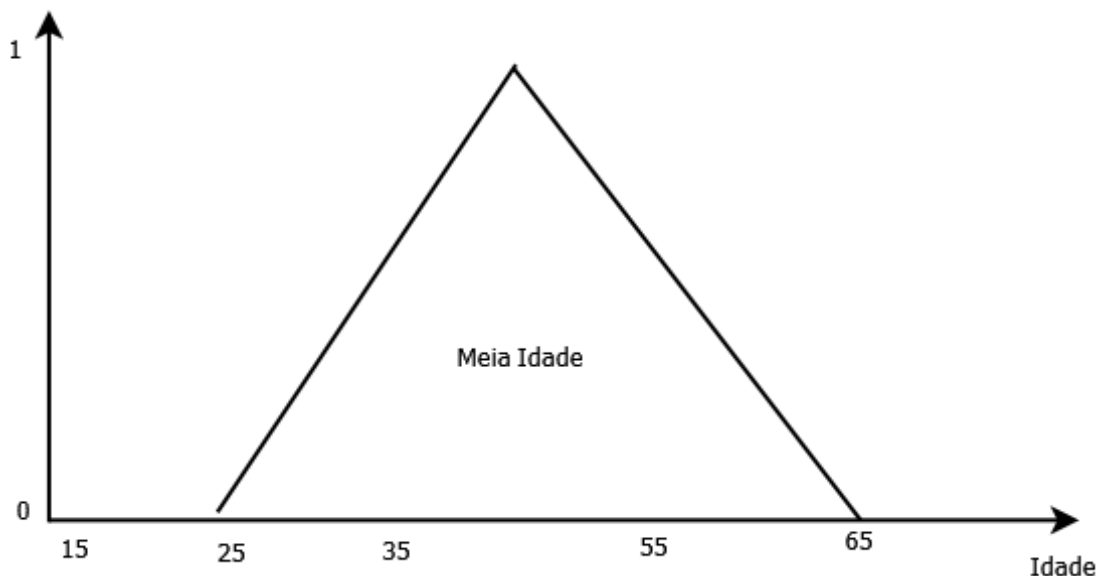


no intervalo  $[0, 1]$ . O valor zero é usado para representar o total não relacionamento e o valor um é usado para representar a adesão completa, como dito anteriormente. Porém, em conjuntos *fuzzy*, os valores entre o intervalo  $[0,1]$  são usados para representar graus intermediários de associação. O conjunto  $X$ , portanto, é referido como o universo de discurso para o subconjunto difuso  $A$ . Frequentemente, o mapeamento  $\mu_A$  é descrito como uma função de pertinência de  $A$ . O grau ou nível de verdade da afirmação é o segundo elemento do par ordenado.

Nota-se, assim, que a função de associação de termos e o subconjunto difuso se utilizam de forma intercambiável [25]. A função de pertinência de cada elemento do universo de discurso para um subconjunto *fuzzy* é apresentada na Equação 2.2. Dessa forma, um valor de pertinência pode representar uma meia verdade, de acordo com Chenci, Rignel e Lucas [26]. Logo, a altura de um determinado indivíduo pode colocá-lo com 0.5 de pertinência no grupo médio e 0.3 no grupo alto.

$$\mu_A : X \rightarrow \{0, 1\} \quad (2.2)$$

Na Figura 2.4 adiante, é apresentado um conjunto *fuzzy*, Meia idade. Percebe-se que a função utilizada permite que elementos pertencentes ao conjunto (indivíduos com suas respectivas idades) possuam níveis de pertencimento, variando de 0 a 1.



**Figura 2.4: Conjunto Fuzzy: Meia idade**

Fonte: Elaborada pela autora.

A partir do que foi mencionado, é possível determinar  $A$  por meio da Equação 2.3, em que  $x$  representa um elemento pertencente ao universo de discurso  $\chi$  e seu nível de pertencimento em  $A$  é estabelecido pela função de relação de pertinência,  $\mu_A(x)$ .

$$A = \{(x, \mu_A(x)) | x \in \mathcal{X}\} \quad (2.3)$$

Nas próximas subseções, será dado embasamento sobre termos e passos necessários a fim de se estabelecer um sistema *fuzzy*, o qual corresponde ao componente principal do FuzDetect.

### 2.4.2 Variáveis Linguísticas

Uma variável linguística admite apenas valores definidos na linguagem *fuzzy* que se utiliza dela. Em computação com palavras, existe o conceito de granularidade, a qual pode ser compreendida como um conjunto de pontos em um conjunto *fuzzy* que se aglomeram por similaridades entre si. Nesse sentido, uma palavra  $w$  pode ser considerada um rótulo de uma granularidade, assim como a granularidade  $g$  denota uma palavra. Uma palavra pode ser atômica, como "jovem", por exemplo, ou composta, como "muito velho". A palavra representa, assim, uma restrição *fuzzy* por meio da variável [22].

De acordo com Costa et al. [9]:

Pode-se considerar uma variável linguística (ou *fuzzy*) como uma entidade utilizada para representar de modo impreciso – e, portanto, linguístico – um conceito ou uma variável de um dado problema. Uma variável linguística, diferentemente de uma variável numérica, admite apenas valores definidos na linguagem *fuzzy* que está utilizando-se dela.

Dessa forma, como dito anteriormente, quando os valores passam a ser compostos, a esses advérbios que indicam intensidade para os conjuntos das variáveis, dá-se o nome de modificadores. Para ficar mais claro, segue definição dada por Chenci, Rignel e Lucas [26]:

Uma variável linguística é uma variável cujos possíveis valores para ela são conjuntos *fuzzy*. Sua principal função é fornecer de maneira sistemática uma aproximação para sistemas complexos e mal definidos.

No exemplo *João é muito alto*, "João" representa uma variável linguística, "alto" é um dos valores permitidos a esta, aí representando também um conjunto *fuzzy*, e "muito" um modificador para tal conjunto "alto".

### 2.4.3 Conjuntos Fuzzy e Funções de Pertinência

Conjuntos clássicos permitem pertencimento total ou nulo de um elemento do universo de discurso, exclusivamente, diferente de conjuntos *fuzzy*, que atribuem níveis de pertencimento

de um elemento ao conjunto.

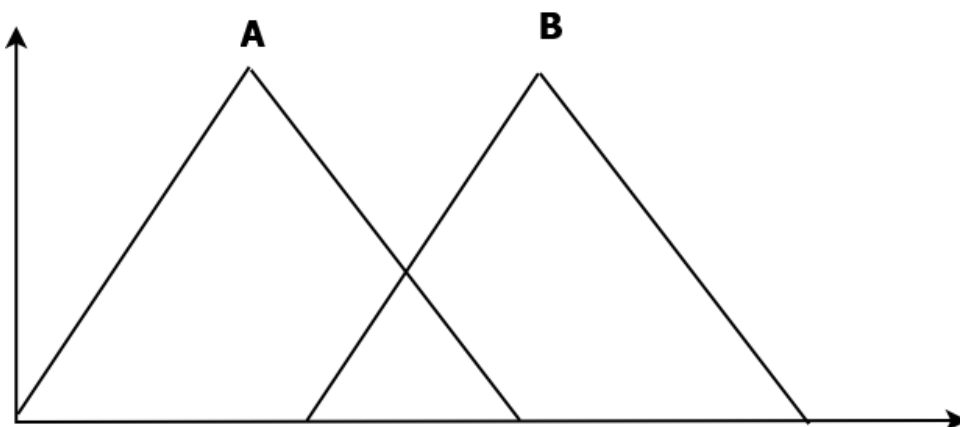
Assim, conforme afirmado por Ross (2009, cap.2, p. 34) [8]:

Em conjuntos clássicos, a definição da transição de um elemento no universo entre a associação e o não pertencimento a um determinado conjunto é abrupta e bem definida (diz ser "*crisp*"). Para um elemento em um universo que contém conjuntos *fuzzy*, esta transição pode ser gradual. A transição possui vários graus de associação e pode ser considerada de acordo com os limites dos conjuntos *fuzzy*, que são vagos e ambíguos. Assim, a adesão de um elemento do universo nesse conjunto é medido por uma função que tenta descrever a imprecisão e a ambiguidade, a função de pertinência  $\mu(x)$ . [...] Os elementos de um conjunto *fuzzy* são mapeados para um universo de valores de associação usando uma forma de função-teoria.

De acordo com Lee [24], as funções de pertinência se dividem em duas naturezas, dependendo se o universo de discurso é discreto ou contínuo, sendo estes numérico ou funcional, respectivamente.

- Numérico: nesse caso, a função de relacionamento de um elemento em um conjunto *fuzzy* é definido por um vetor de números cuja dimensão depende do nível de discretização;
- Funcional: as funções de relacionamento são estabelecidas por meio de funções *bell-shaped*, funções triangulares ou trapezoidais. Tais funções utilizam aritmética a fim de estabelecer o nível de associação e podem prontamente se adaptar a mudanças na normalização do universo de discurso, uma vez que são específicas para lidarem com dados contínuos.

Na Figura 2.5 a seguir, são apresentados dois conjuntos *fuzzy* estabelecidos por meio de funções triangulares.



**Figura 2.5: Função triangular como função de relacionamento**

Fonte: Elaborada pela autora.

### 2.4.4 Inferência e Operadores *Fuzzy*

Um operador *fuzzy* é capaz de converter um determinado elemento de um universo de discurso em uma representação quanto ao nível de pertinência em conjuntos *fuzzy*, de acordo com Lee [24]. Assim, definidos dois conjuntos difusos A e B no universo X, para um dado elemento x do universo, tem-se as seguintes operações teóricas de função: União e Interseção são definidos para A e B:

União:

$$\mu_{AOR_B}(x) = \mu_A(x) \cup \mu_B(x). \quad (2.4)$$

Interseção:

$$\mu_{AND_B}(x) = \mu_A(x) \cap \mu_B(x). \quad (2.5)$$

Como na teoria clássica, os conjuntos *fuzzy* obedecem a certas propriedades e podem ser operados de diversas maneiras. As operações entre conjuntos são extremamente importantes para os sistemas que se utilizam dessa lógica, pois os cálculos proposicionais, por exemplo, são baseados nessas operações, de acordo com Costa et al. [9].

Nas Equações 2.4 e 2.5, respectivamente, a operação de união extrai o maior valor entre as pertinências enquanto que a interseção, o menor valor entre os níveis comparados. Os conjuntos *fuzzy* seguem as mesmas propriedades que os conjuntos clássicos. Por isso e também porque os valores de associação de um conjunto clássico são um subconjunto do intervalo [0,1], conjuntos clássicos podem ser pensados como um caso especial de conjuntos *fuzzy*, conforme afirma Ross [8].

As definições anteriormente expressadas são originadas do trabalho de Zadeh e respeitam algumas propriedades ou normas, como bem descritas em [25] [9]:

- Comutatividade:  $S(a,b) = S(b,a)$ ;
- Associatividade:  $S(a, S(b,c)) = S(S(a,b), c)$ ;
- Monotonicidade:  $a < b$  e  $c < d$ , logo:  $S(a,c) < S(b,d)$ ;
- Coerência nos Contornos:

Em operações OR:  $S(a,0) = a$  e  $S(a,1) = 1$ ;

Em operações AND:  $S(a,0) = 0$  e  $S(a,1) = a$ .

Exclusivamente, as propriedades desses dois operadores serão apresentadas, uma vez que os demais não fazem parte do escopo deste trabalho.

### 2.4.5 Relacionamentos *Fuzzy*

Em um sistema *fuzzy*, o comportamento dinâmico é caracterizado por um conjunto de regras que descrevem contextos linguísticos baseados no conhecimento de quem as implementou, como define Lee [24]. Esse conhecimento é geralmente estabelecido na forma:

**If** (um conjunto de condições aqui) **then** (um conjunto de consequentes podem ser inferidos).

A seguir, uma definição em Zadeh [22] para esse processo:

Dado um conjunto de proposições expressadas em uma linguagem natural, que constituem um conjunto de dados de entrada, a partir destes dados é desejado que se expresse uma resposta por meio de uma linguagem natural, produzindo assim o conjunto de dados de saída. Logo, o problema estabelecido é que o conjunto de dados de saída é derivado do conjunto de dados de entrada.

Mamdani [23] exemplifica que:

Dado  $A + B$ , estabelece-se que  $A$  e  $B$  são variáveis linguísticas de universos de discurso diferentes, supondo-se  $X$  e  $Y$ , respectivamente. Nota-se que a implicação ou inferência é na variável individualmente. Assim, a relação  $R$  entre  $A$  e  $B$  é um subconjunto *fuzzy*, entre os universos  $X$  e  $Y$ . O *cross-product* entre  $X$  e  $Y$  é  $\mu_R : X \text{ and } Y \rightarrow [0, 1]$ .

As regras ou relações *fuzzy* são, portanto, regras normais utilizadas para operar, da maneira correta, conjuntos *fuzzy* com o intuito de obter consequentes. Para criar tais regras, é preciso de um raciocínio coerente com o que se deseja manusear e obter. Para isso, esse raciocínio deve ser dividido em duas etapas: (1) avaliar o antecedente da regra e (2) aplicar o resultado no consequente, de acordo com Costa et al. [9].

As definições supracitadas serão ilustradas adiante por meio de exemplos com um (*single*) e vários antecedentes.

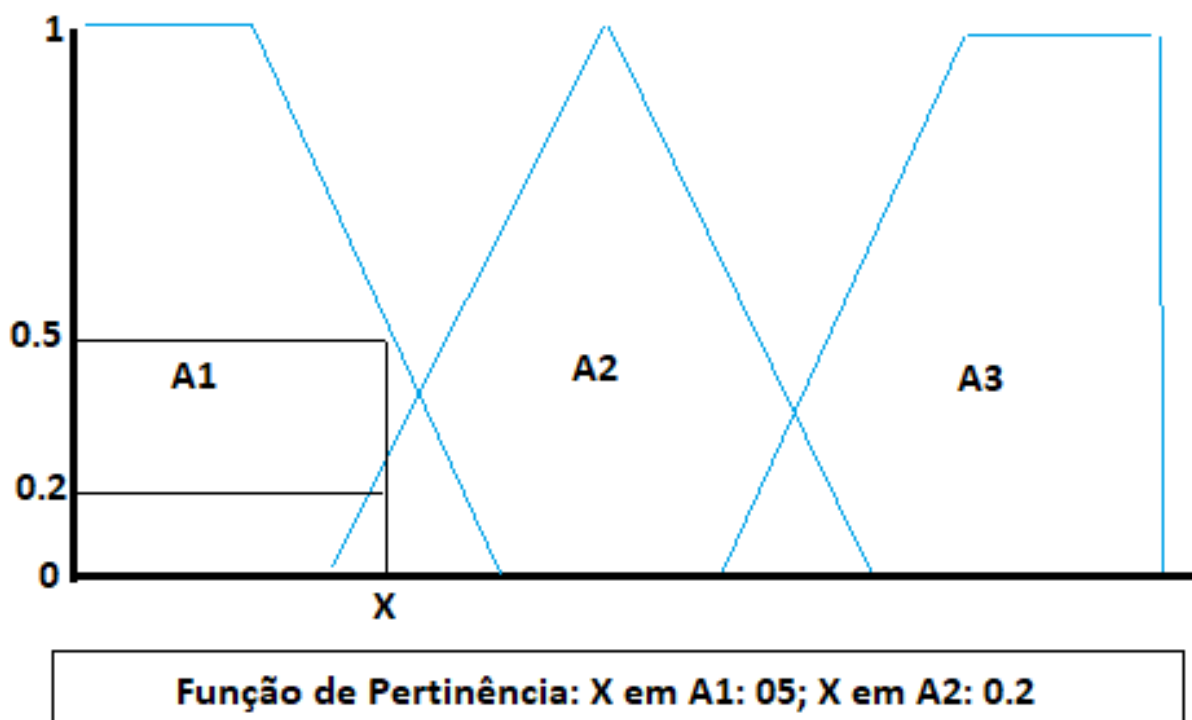
Exemplo 1 - Obtendo-se o nível do peso pesado por meio da altura de João.

Se João é alto ( $x$ ) então seu peso é pesado ( $y$ ).

Tem-se que, por exemplo, para  $x = 1,90\text{m}$ , deve-se primeiramente verificar o grau de pertinência da entrada para o conjunto ao qual está descrito na afirmação: alto, que é para este

caso,  $\mu(x) = 0.5$  (por exemplo). Como o grau de pertinência da entrada  $x$  é tal, então se deve passar esse valor de pertinência para  $y$ , que extrairá o valor numérico de saída (esse processo será definido mais à frente).

Na Figura 2.6, é possível identificar que a amostra  $x$  possui valor de pertinência de saída de 0.5 no conjunto A1 e de 0.2 no conjunto A2.



**Figura 2.6: Fuzzificação**

Fonte: Elaborada pela autora.

Exemplo 2 - Obtendo-se o nível do tamanho da cama grande a partir do peso e da altura de Maria.

Se Maria é alta ( $x$ ) e seu peso é grande ( $y$ ), o tamanho da sua cama deverá ser grande ( $z$ ).

Assim como no outro exemplo, o valor  $x$  deverá ser *fuzzificado*, bem como o de  $y$ . Ambos retornarão os valores de pertinência entre seus respectivos conjuntos. Em seguida, a operação definida na regra para se estabelecer a pertinência de saída deverá ser executada entre os conjuntos de entrada.

Salienta-se que, conforme apresentado anteriormente, na Subseção 2.4.4, com o operador *or* obtém-se o maior entre os graus dos conjuntos comparados; já nas operações *and*, o menor valor entre os graus de pertinência dos conjuntos comparados. Tal valor de saída será remetido para o conjunto  $z$ , da variável de saída, e este retornará um determinado valor numérico de saída (esse processo será definido mais à frente).

Na Tabela 2.3 são apresentados alguns relacionamentos estabelecidos por meio de regras. São elencadas três regras e é apresentado o valor de pertinência de  $x$  de acordo com cada conjunto de entrada e o nível de pertencimento de saída gerado, baseado nos operadores envolvidos.

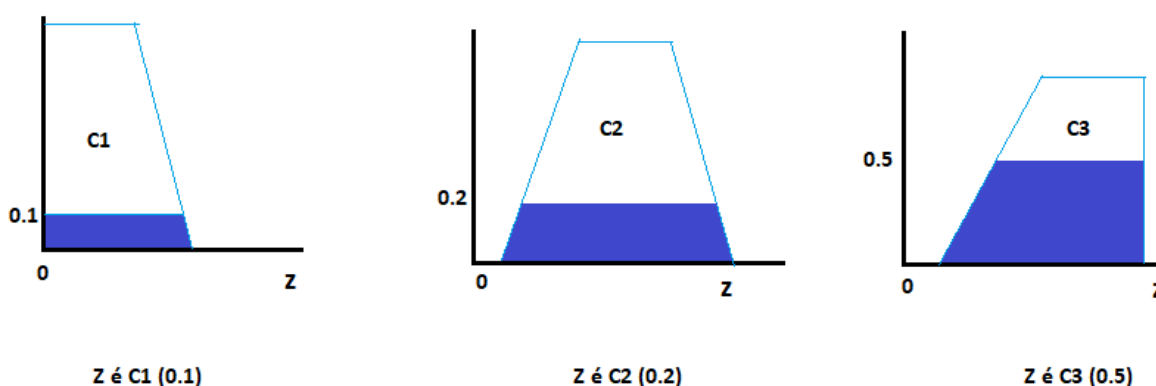
**Tabela 2.3: Regras Fuzzy**

If (x is A3(0) or y is B1(0.1)) then (z is C1(0.1))
If (x is A2(0.2) and y is B2(0.7)) then (z is C2(0.2))
If (x is A1(0.5)) then (z is C3(0.5))

### 2.4.6 Defuzzificação por meio da Inferência de Mamdani

O estilo de inferência Mamdani foi criado pelo professor Ebrahim Mamdani, da Universidade de Londres, em 1975, no contexto do desenvolvimento de sistemas *fuzzy*, com base em regras de conjuntos *fuzzy* e no intuito de representar experiências da vida real [9]. Boa parte dos passos seguidos por esse modelo já foi explicada anteriormente, então, a partir daqui, será dado início ao processo de *defuzzificação*, por meio da agregação das regras.

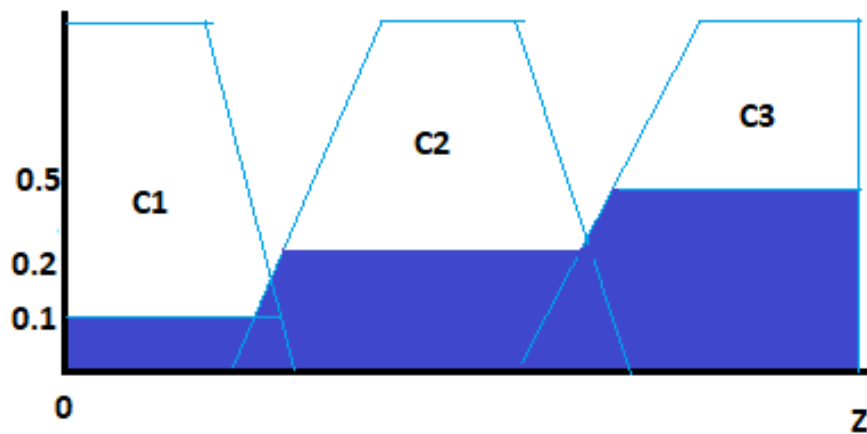
Torna-se necessário, primeiramente, aglomerar todos os conjuntos de saída e suas respectivas pertinências, de acordo com as regras (conforme descritas na Subseção 2.4.5), em um único conjunto *fuzzy*. Na Figura 2.7, é apresentada a variável de saída  $Z$  e seus conjuntos, assim como as pertinências de saída de acordo com as regras estabelecidas na Tabela 2.3. E na Figura 2.8, é apresentada a saída agregada em um único conjunto *fuzzy*.



**Figura 2.7: Conjuntos de saída da variável  $Z$  e suas respectivas pertinências de acordo com as regras**

Fonte: Elaborada pela autora.

*Defuzzificar* consiste em converter o valor de pertinência gerado a partir da consolidação das regras (descrito anteriormente) em uma saída numérica. A saída do processo de inferência até agora é um conjunto *fuzzy*, especificando uma possibilidade de distribuição de ação de



**Figura 2.8:** Conjuntos de saída agregados da variável de saída Z  
Fonte: Elaborada pela autora.

controle. No controle, uma não-fuzzy (*crisp*) ação de controle geralmente é necessária. Consequentemente, é preciso *defuzzificar* a ação de controle *fuzzy* (saída) inferida do algoritmo de controle *fuzzy* [25]:

$$z_0 = \text{defuzzifier}(C) \quad (2.6)$$

Em que, na Equação 2.6,  $z_0$  representa a saída numérica esperada e *defuzzifier* o operador de *defuzzificação* recebendo as saídas geradas do controlador *fuzzy* (C).

Na etapa de *defuzzificação*, existe uma infinidade de funções que podem ser utilizadas. São algumas delas: *Center-of-Area/Gravity*, *Center-of-Sums*, *Center-of-Largest-Area*, *First-of-Maxima*, *Middle-of-Maxima*, *Max-Criterion*, *Height Defuzzification* [25]. Para o propósito deste trabalho, usou-se a função Centro de Área da Gravidade, o método de *defuzzificação* mais comum, que obtém o ponto no qual uma linha vertical divide ao meio um conjunto agregado [9].

O método de centroide favorece a regra com a saída de maior área. A seguir, apresenta-se a fórmula do centroide na Equação 2.7, em que  $x$  representa os valores do escopo do conjunto de saída,  $\mu(x)$  representa o valor de pertinência gerado de acordo com as regras, por conjunto de saída, e o somatório varre todas as regras. Ao final, um valor numérico será gerado correspondente à saída *defuzzificada*, de acordo com as regras e com os operadores definidos.

$$\frac{\sum_{k=1}^N x \cdot \mu(x)}{\sum_{k=1}^N \mu(x)} \quad (2.7)$$



Durante a descrição do funcionamento do FuzDetect no Capítulo 4, serão introduzidos detalhes acerca das variáveis de entrada e de saída *fuzzy* utilizadas na proposta do presente estudo, funções de relacionamento em conjuntos e regras, entre outros.

## 2.5 Otimização por Enxame de Partículas - PSO

Otimização por Enxame de Partículas (PSO) é um método de busca inspirado na biologia e técnica de otimização computacional desenvolvida em 1995 por Eberhart e Kennedy, com base nos comportamentos sociais dos pássaros e peixes durante a busca por alimentos, como descreve Rini, Shamsuddin e Yuhaniz [27].

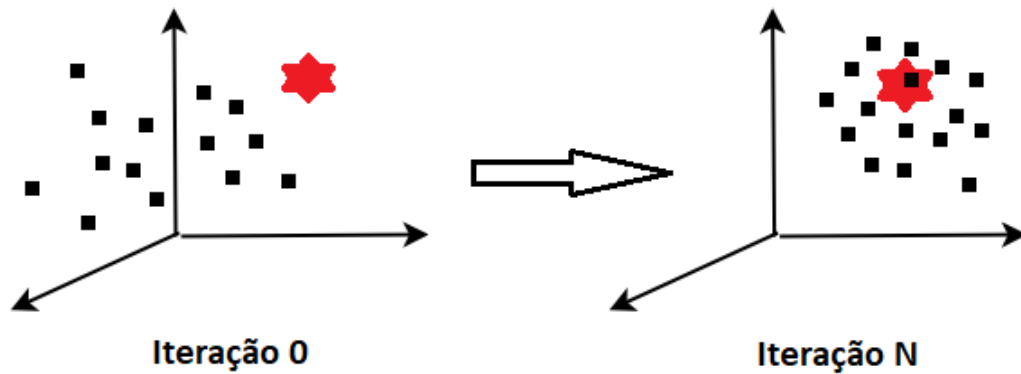
Assim, em vez de utilizar operadores evolucionários a fim de provocar alterações nos indivíduos, como em outros algoritmos evolucionários (algoritmos genéticos, por exemplo), cada indivíduo em PSO, conhecido como partícula, voa em um espaço de busca de acordo com o ajuste de sua velocidade e em função da experiência das demais partículas do espaço de busca e da sua própria experiência de voo Shi e Eberhart [10].

De acordo com Chia-Nan e Chia-Ju [28], o PSO sobressai outros algoritmos de otimização em muitos critérios, como rapidez na convergência em comparação com outros algoritmos de ordem estocástica. Além disso, ele tem sido usado com sucesso em outras aplicações cujos problemas são de alta complexidade e apresentam não linearidade.

### 2.5.1 Funcionamento do PSO

Na Figura 2.9 a seguir, é apresentada a posição das partículas no espaço de busca, em que cada dimensão representa uma característica da partícula. Durante a inicialização do algoritmo, à esquerda, percebe-se que as partículas se encontram dispersas ainda e até então nenhuma inteligência está associada à movimentação delas. Já ao lado direito da figura, durante a última iteração do algoritmo, é possível verificar que as partículas se posicionam quase como um "cluster", em torno da melhor global (a partícula em formato de estrela). Consequentemente, as características das partículas no espaço de busca, a cada iteração, vão ficando mais similares à da melhor partícula global.

Conforme descrito por Shi e Eberhart [10], Rini, Shamsuddin e Yuhaniz [27] e Xiaohui [13], a execução do PSO consiste em um espaço de busca onde existem as possíveis soluções, conhecidas como partículas. Estas, por sua vez, são formadas por posições de dimensão D cada uma e as posições são alteradas a cada iteração do algoritmo (o vetor posição compõe as caracte-



**Figura 2.9:** Partículas antes e após convergência em um espaço de busca

Fonte: Elaborada pela autora.

terísticas da solução/partícula que se altera por instância do problema). As posições mudam de acordo com a variação do vetor velocidade da partícula, o qual também possui dimensão  $D$ , e tal alteração acontece a cada iteração do algoritmo.

Ainda, cada partícula possui duas variáveis que persistem, a cada iteração, com duas informações, respectivamente: a sua melhor posição até aquela iteração e a partícula de melhor posição em todo o espaço de busca naquela iteração, conhecida como melhor global. E estas só serão alteradas caso a nova posição ajustada naquela iteração (em função da nova velocidade) possa ocasionar tanto uma melhor posição dentre todas as outras da própria partícula, como uma nova melhor global. Esse processo de ajuste de velocidade e posição (nessa ordem) acontece até que se alcance um valor ótimo esperado ou que se alcance o limite máximo de iterações.

Dessa forma, tem-se:

$$V_{i_d} = V_{i_d} + c1 \cdot rand() \cdot (P_{i_d} - X_{i_d}) + c2 \cdot Rand() \cdot (Pg_d - X_{i_d}) \quad (2.8)$$

$$X_{i_d} = X_{i_d} + V_{i_d} \quad (2.9)$$

Na Equação 2.8, é apresentada a função de ajuste de velocidade da partícula, em que:  $V_{i_d}$  representa o vetor velocidade da partícula  $i$ , na dimensão  $d$ ;  $c1$ ,  $c2$  são constantes que correspondem a taxas de aprendizado da partícula cognitiva e social, respectivamente [27];  $rand()$ ,  $Rand()$  são funções randômicas que variam no range  $[0,1]$  [10];  $P_{i_d}$  é a variável que mantém a melhor posição da partícula  $i$  individualmente, até o tempo corrente e na dimensão

$d$ ;  $Pg_d$  é a variável que mantém a posição da melhor partícula, em todo o espaço de busca, no tempo corrente e na dimensão  $d$ .

Já na Equação 2.9, é apresentada a função de ajuste de posição da partícula, logo, tem-se que  $X_{i_d}$  representa a posição da partícula  $i$  na dimensão  $d$ .

Para se obter o nível de qualidade da solução ou partícula, é preciso uma função que possa avaliá-la, chamada de função de custo ou função *fitness*, a qual é implementada por instância do problema (por exemplo, maximização ou minimização).

No Pseudocódigo 1, é demonstrado o funcionamento do algoritmo de maneira sucinta. Primeiramente, as partículas são iniciadas, geralmente de forma aleatória, linha 1 a 3. Para cada partícula, seu nível de qualidade é mensurado (por meio da função *fitness*, mencionada anteriormente), linha 6. Caso o valor retornado seja melhor que os demais mantidos no histórico da partícula, ele será atualizado (*pBest*), linha 8. Em seguida, é verificado se existe um novo *gBest*, ou seja, uma nova melhor global dentre todas as partículas do espaço de busca, e, em caso positivo, todas deverão salvar a referência a essa última, linha 11. No próximo *loop*, calcula-se então a nova velocidade e depois a nova posição (em função da nova velocidade) de cada partícula, linhas 13 e 14, respectivamente. Todos os passos são repetidos até que se alcance o melhor esperado ou uma quantidade máxima de iterações, previamente estabelecida, conforme *loop* na linha 4.

---

**Algoritmo 1:** Pseudocódigo do PSO

---

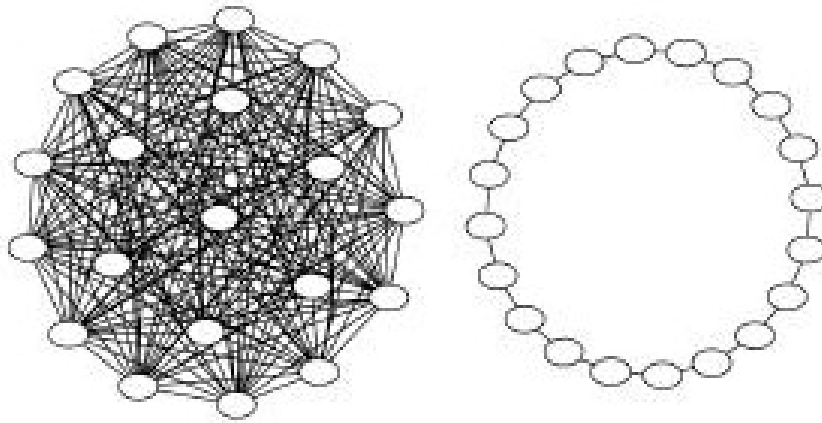
```
1 for Para cada Partícula do
2   | Inicialize A Partícula ();
3 end
4 while Quantidade máxima de iterações não foi alcançada Ou Ótimo Esperado do
5   | for Para cada Partícula do
6     | Calcule Fitness Partícula();
7     | if O valor fitness atual é melhor que o até então melhor fitness do histórico da
8       | partícula then
9         | Configure novo pBest
10      | end
11   | end
12   | Escolha Partícula Com Melhor pBest Dentre Todas As Partículas (gBest);
13   | for Para cada Partícula do
14     | Calcule Velocidade Da Partícula (De acordo com Equação 2.8);
15     | Atualize Posição Da Partícula (De acordo com Equação 2.9);
16   | end
17 end
```

---

### 2.5.2 Modelos de Vizinhança em PSO

No PSO original, dois tipos diferentes de modelos de vizinhança foram definidos. No modo global, todas as partículas são vizinhas umas das outras, portanto, a posição da melhor partícula global no enxame é usada no termo social da equação de atualização de velocidade. Supõe-se que assim o enxame possa convergir rapidamente, pois todas as partículas são atraídas simultaneamente para a melhor parte do espaço de busca. No entanto, se a partícula não estiver, de fato, perto do ótimo global, pode ser impossível ao enxame explorar outras áreas, e isso significa que o enxame pode ficar preso em um ótimo local. No modo local, apenas um número específico de partículas (adjacentes) pode afetar a velocidade de uma dada partícula. O enxame converge de forma lenta, mas pode localizar a melhor posição com uma maior chance [29].

Na Figura 2.10, é apresentada a formação da vizinhança em modo global, à esquerda, em que todas as partículas se interligam entre si. Já à direita, em modo local, as partículas buscam pela melhor partícula na vizinhança, por meio das partículas adjacentes. Outras arquiteturas também são citadas na literatura, mas fogem ao propósito deste trabalho.



**Figura 2.10: Arquiteturas baseadas na vizinhança em PSO**

Fonte: Disponível em: <<http://www.computacaointeligente.com.br/algorithmos/otimizacao-por-enxame-de-particulas-pso/>>. Acesso em: 13 set. 2018.

Na solução proposta, o PSO será usado de forma auxiliar ao FuzDetect, configurando os limites dos conjuntos *fuzzy* que serão estabelecidos de forma dinâmica, adaptada à rede analisada, de modo que as partículas equivaleriam a possíveis valores para os pontos dos conjuntos. Mais à frente, será apresentada a configuração utilizada do PSO.

# Capítulo 3

## TRABALHOS RELACIONADOS

---

---

A seguir serão apresentados trabalhos relacionados ao tema proposto, separados por área de atuação.

### 3.1 Lógica *Fuzzy*

Dickerson e Dickerson [30] propõem um sistema de detecção de ataques DDoS utilizando *Data Mining* e Lógica *Fuzzy*. Conforme apresentado, em um primeiro momento, é feita uma coleta de tráfego da rede de forma bruta por meio de um *sniffer*. Na etapa seguinte, um mecanismo de *data mining* analisa os dados e os sumariza de modo a categorizá-los por meio de características mais significativas, ou seja, características que mais se diferem entre si desde a última coleta.

Por intermédio do *Data Mining* são estabelecidas conexões pares (requisição/resposta), quantidade de pacotes vistos por cada uma dessas conexões, quantas vezes elas foram registradas durante um determinado intervalo de tempo, quantidade de conexões recentes, quantidade de conexões antigas, quantidade de portas em *listen* ou conectadas por intervalo, variância da quantidade de pacotes por intervalo, quantidade de conexões estabelecidas fora da rede local e total de conexões TCP. A partir das variáveis citadas, outras são estabelecidas como entrada da Lógica *Fuzzy*, a saber: Unicidade, Quantidade e Variância dos dados.

Desse modo, constata-se que o trabalho apresentado utiliza inspeção de pacotes, o que é mais caro computacionalmente. Embora conste nele que é possível detectar subgrupos de ataque, exclusivamente regras para identificação de *scanners* na rede foram usadas, de fato. Além disso, quanto aos resultados, ainda que o sistema processe dados de uma rede em produção, não é apresentado de que forma se pôde validar se a detecção como ataque ou não estava correta (a

ordem falsos positivos e negativos). Tampouco foi apresentado algum mecanismo que torne os conjuntos do classificador *fuzzy* adaptado à rede analisada.

No trabalho de Saini e Suryawanshi [31], é apresentado um sistema de detecção de intrusão em que a Lógica *Fuzzy* e o *data mining* trabalham juntos na classificação do tráfego. O classificador é binário, sendo o tráfego, simplesmente, ataque ou não. Além disso, apesar de utilizar *Multivariate Correlation Analysis* no tráfego da rede, a fim de designar as variações no tráfego, não é explicado como a configuração do classificador se adaptaria a outras redes.

Em Mondal et al. [32], a Lógica *Fuzzy* é utilizada em um ambiente em nuvem como mecanismo de detecção de ataque. Nesse trabalho, a quantidade de regras é alta, o que leva mais tempo de processamento. A quantidade de parâmetros analisada é baixa, facilitando falsos negativos e positivos, e, além do mais, não é descrita a fonte do tráfego utilizada. Assim como as variáveis de entrada não são justificadas quanto à motivação do seu uso, também não é apresentado nenhum mecanismo para que o sistema *Fuzzy* pudesse se adaptar à massividade do tráfego, caso esse fosse utilizado em outro ambiente ou houvesse mudanças no desenho da rede.

No trabalho apresentado por Tuncer e Tatar [33], é descrito um sistema para detecção de ataques *synflood*, de modo que a variável de entrada *fuzzy* é unicamente a quantidade de pacotes *syn*. Os limitadores dos conjuntos *fuzzy* não foram especificados e, ainda, foram configurados *thresholds* estáticos como forma de classificar o tráfego. Além disso, a quantidade de regras usadas é bem maior que a proposta aqui apresentada, o que é mais custoso computacionalmente.

Dang-Van e Truong-Thu [34] apresentam um sistema de detecção de ataques DDoS de forma generalista, em que parâmetros estáticos são coletados a partir da rede SDN (utilizando o tráfego de um Provedor de Serviços de *Internet (Internet Service Provider - ISP)* do Vietnã). Tais parâmetros são monitorados periodicamente e, em seguida, repassados ao controlador da rede. Esses parâmetros compreendem o intervalo de tempo entre os pacotes, a quantidade de pacotes por fluxo e a quantidade total de fluxos. Após a coleta, um algoritmo que é executado no controlador analisa os parâmetros e, por meio de regras, utilizando *thresholds* estáticos, define se o tráfego passante é de ataque ou não. Caso o ataque seja detectado, a Lógica *Fuzzy* é utilizada como meio de estabelecer a taxa de descarte dos pacotes dos fluxos identificados como de ataque.

De acordo com o algoritmo citado, segue o Pseudocódigo 2, em que A1, A2 e A3 representam as variáveis previamente citadas. É possível verificar também os limites estáticos fornecidos para cada uma durante a tomada de decisão como tráfego de ataque ou não, nas linhas 1 e 4. Na última condicional, linha 7, a Lógica *Fuzzy* é invocada com o objetivo de estabelecer a taxa de descarte de pacotes.

**Algoritmo 2:** Pseudocódigo *Attack Prevention System*


---

**Data:** A1: Taxa de Pacotes com IAT [34] acima de  $0.2 \times 10^3$   
**Data:** A2: Taxa de Fluxos com um pacote por fluxo  
**Data:** A3: Número de Fluxos por Servidor

```

1 if A1 [0.99,1] OR A2 [0.9,1] OR A3 [4128] then
2   | Ação = Descarte 100% dos Pacotes;
3 end
4 if A1 [0,0.5] AND A2 [0,0.1] AND A3 [0,150] then
5   | Ação = Encaminhar Todos os Fluxos;
6 else
7   | Ação = FuzzyLogic()
8 end

```

---

Uma vez definido o tráfego como de ataque, a Lógica *Fuzzy* (utilizando o modelo Takagi-Sugeno [35]) estabelece qual a percentagem de pacotes a serem descartados. O modelo conta com as variáveis de entrada: Intervalo de tempo entre pacotes e Quantidade de fluxos com um pacote, para estabelecer essa saída.

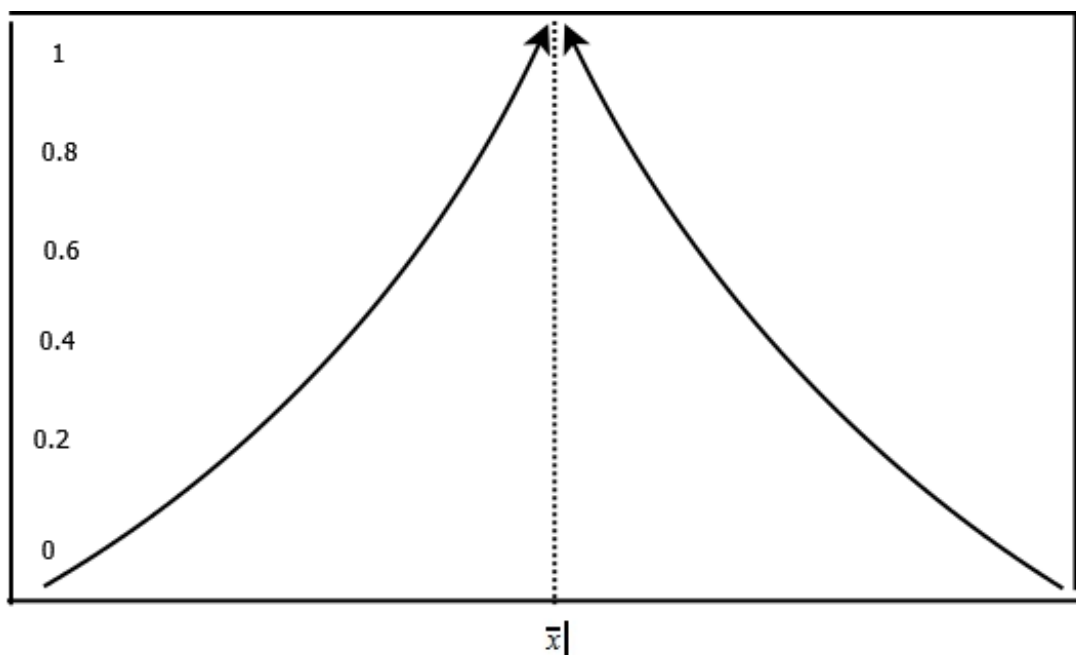
Apesar de o tráfego analisado ser de uma rede real, fica clara a dependência do sistema como um todo com o tráfego específico. Além disso, o estabelecimento dos parâmetros durante a validação do tráfego como de ataque ou não é feito de forma estática (*thresholds* fixos), mais uma vez dependente da amostra de tráfego testada. E por realizar detecção de ataques de forma generalista, ele não conta com outras variantes de ataque cuja quantidade de pacotes por fluxo, por exemplo, possa ser significativa.

Em Shiaeles et al. [36], considera-se primeiramente a quantidade de conexões estabelecidas, sabendo-se de antemão que essa quantidade vai variar de acordo com o tempo. Em seguida, esse período de tempo monitorado é dividido em intervalos de tempo pequenos o bastante para detectar um ataque DDoS e, ao mesmo tempo, preencher um modelo de Poisson. Para cada modelo, calcula-se a média do intervalo de tempo entre pacotes. Uma vez estabelecida a média, é verificado se as novas médias coletadas são menores, entretanto, a comparação estática não seria tão eficiente.

Nesse trabalho, a Lógica *Fuzzy* é utilizada como forma de suavizar o modelo assumido, como é possível visualizar na Figura 3.1, a qual demonstra que, quanto mais disperso (à esquerda) da média previamente estabelecida for o intervalo de tempo entre os pacotes das amostras geradas, maior é a chance de um ataque estar acontecendo.

As amostras utilizadas são coletas de tráfego do campus de uma Universidade (por meio do





**Figura 3.1:** Utilização de um estimador *Fuzzy*

Fonte: Elaborada pela autora.

espelhamento de portas) e do DARPA *data set*. Quanto ao tráfego da Universidade, a partir de 5.000 pacotes já foi possível identificar o ataque, mas a precisão da classificação só foi maior a partir de 20.000 pacotes. Quanto ao tráfego do DARPA *data set*, os resultados não foram tão significativos, uma vez que o modelo não lida com outros protocolos além do Protocolo de Transferência de HiperTexto (*Hypertext Transfer Protocol* - HTTP), que constava no *dataset* da DARPA.

A Lógica *Fuzzy*, por sua vez, conta com apenas um único fator de entrada: o intervalo de tempo entre os pacotes. Caso o atacante passe a gerar pacotes com intervalo de tempo maiores, ou com grandes variações de tempo, a classificação perderia sua eficácia, assim como um aumento na quantidade de requisições legítimas poderia gerar falsos positivos. Conforme mencionado anteriormente, o modelo definido lida restritamente com o protocolo HTTP. Tráfego de ataque do Protocolo de Datagrama de Usuário (*User Datagram Protocol* - UDP) ou outros protocolos de aplicação, ainda que se use Protocolo de Controle de Transporte (*Transport Control Protocol* - TCP), não são identificados pela ferramenta apresentada.

Além disso, a detecção é generalista, não são dados detalhes sobre o tipo do ataque. Um outro ponto acerca deste trabalho é que o autor define unicamente como ataque as amostras cuja média é menor que a média pico; para outros tipos de ataques, talvez a dispersão da média à direita do pico representasse um forte indício para eles.

Assim como em [36] [34], o trabalho apresentado por Santos e Silva [37] apresenta um sis-

tema que propõe a detecção de ataques DDoS por meio do intervalo de tempo entre os pacotes. A Lógica *Fuzzy* é utilizada como mecanismo para inferir o tamanho da janela de tempo em que os pacotes serão acumulados e em seguida analisados.

Desse modo, foram utilizados *traces* por meio de *tcpdump* com *Datasets* da DARPA, de um dos Laboratórios do *Massachusetts Institute of Technology* (MIT) e da Força Aérea Norte-Americana. Realizaram-se simulações dos anos de 1998, 1999 e 2000 e, de cada ano, cinco semanas foram geradas, de modo que as três primeiras semanas foram de treinamento e as outras duas para testes. Ainda, em cada, a primeira e a terceira semana eram de tráfego legítimo e a segunda de ataque, a as demais semanas possuíam tráfego legítimo e de ataque.

Gráficos de controle de Duclos [38] [39], são inseridos (de acordo com o autor, uma boa escolha, uma vez que os dados não são normalmente distribuídos), de modo que os estimadores de localização deles são estabelecidos pela média do intervalo entre os pacotes, enquanto que os estimadores de dispersão são estabelecidos por meio do desvio padrão do intervalo de tempo entre os pacotes. Em seguida, é introduzido o conceito de janela de observação, que definirá o período de tempo em que será realizado o cálculo dos estimadores (por meio do intervalo de tempo entre os pacotes). Sendo assim, os valores dos estimadores dessas janelas serão comparados (a janela  $i$  será comparada com a janela  $i + 1$ ). Segundo o autor, o tamanho da janela de observação está diretamente ligado à qualidade dos resultados e é nessa etapa que a Lógica *Fuzzy* é inserida, como meio de estabelecer esse período a cada nova janela.

Durante essa etapa, as variáveis de entrada *fuzzy* são os estimadores de localização e dispersão; já a variável de saída é o tamanho da próxima janela de observação. Quanto à configuração das regras, definiu-se basicamente que, quanto menor forem os valores dos estimadores, maior deverá ser a janela, visto ser um forte indício de que nada está acontecendo; e, em caso contrário, estimadores cada vez maiores que zero são uma indicação significativa de comportamento fora do padrão e, assim, a janela é reduzida proporcionalmente.

Nesse sentido, o funcionamento do sistema se mantém da seguinte forma, conforme algoritmo apresentado no Pseudocódigo 3: na primeira Janela (J0), calculam-se os estimadores (linha 1); nessa primeira passagem, não são feitas comparações (linha 2) nem verificação de mudança de comportamento (linha 3), uma vez que não existem janelas anteriores. Na sequência, a janela seguinte é estabelecida (J1) (linha 3 a 10), repetindo esse processo até o final do *trace*, conforme linha 13. E, a partir daqui, o algoritmo executa todas as ações pertinentes e os valores dos estimadores de dispersão e localização são calculados, armazenados e comparados com os valores desses mesmos estimadores na janela anterior. Se os valores estiverem fora do limite esperado, considera-se que esteja acontecendo uma mudança de comportamento ou

um suposto ataque (os limites são calculados a partir dos valores dos estimadores; no caso do  $i$ -ésimo valor da janela de observação, esses limites são dados por uma vizinhança do centro do estimador de localização e raio igual ao triplo do desvio padrão do mesmo intervalo) [38].

---

**Algoritmo 3:** Pseudocódigo do algoritmo de detecção de ataques DDoS com base em gráficos de controle

---

```

Data:  $J_0 =$  Criar Primeira Janela ()
Data:  $[\mu, \theta] =$  Ductos ( $J_{i+1}$ )
1 Armazene Estimadores ();
2 Realize A Comparação Entre  $i$  e  $i+1$  ();
3 if Tem Mudanças then
4   | Notifique Usuário ();
5 else
6   | if Se for o fim das análises then
7     | Fuzzificação ();
8     | Máquina de Inferência ();
9     | Faça Busca Na Base De Conhecimento ();
10    | Defuzzificação ()
11   | end
12 end
13 if Se não for o Final do Trace then
14   | Comece Novamente
15 else
16   | Fim
17 end

```

---

Com a apresentação dos resultados na Tabela 3.1, é possível verificar que, embora a detecção do tráfego de fato de ataque tenha sido relevante, a quantidade de falsos negativos foi bem significativa, mesmo para as ferramentas de ataque alvos de estudo. Nos casos de ferramentas não diretamente relacionadas ao algoritmo, essa taxa foi ainda maior, chegando a ultrapassar mais de 50% em alguns casos, o que seria inviável em uma rede real. Uma outra característica é que a detecção do ataque não é refinada, ou seja, não informa mais detalhes sobre a natureza dele durante a classificação.

Em Xia et al. [40], o *Fuzzy* é introduzido como mensurador da intensidade do tráfego de ataque, unicamente, por meio de variações no parâmetro Hurst. De acordo com o autor, diante de várias métricas estabelecidas a partir de diversos tipos de rede, cabo, fibra óptica, *wireless*, todas demonstram possuir uma autossimilaridade mesmo em altas escalas de tempo. Autos-

**Tabela 3.1: Resultado do algoritmo de detecção de ataques DDoS com base em gráficos de controle**

Ataques	Identificação		Falso Negativo	
	Quantidade	%	Quantidade	%
Apache2	2	66	1	33
Mailbomb	2	33	4	66
Smurf	3	60	2	40
UdpStorm	0	0	1	100
back	3	50	3	50
pod	3	50	3	50
land	2	50	2	50
crashiis	1	9	10	91
synflood	1	17	5	83
processtable	1	25	3	75
arppoisson	2	40	3	60
dosnuke	0	0	4	80
syslogd	3	75	1	25
selfping	2	100	0	0
tcpreset	2	66	1	33
teardrop	0	0	2	100
warezmaster	4	100	0	0

similaridade é a propriedade associada a um objeto, de modo que estruturas deste não sofrem mudanças, mesmo em diferentes escalas, e esse nível de ruído pode ser estabelecido pelo parâmetro Hurst.

O autor prossegue afirmando que vários estudos comprovam que, durante ataques DDoS, é apresentada uma notável autossimilaridade no tráfego da rede e esse pode ser obtido por meio do parâmetro Hurst, como mecanismo de identificação de nível de variação. Aqui são introduzidos dois métodos utilizados para o estabelecimento do parâmetro Hurst, a Transformada Discreta em Pequenas Ondas (*Discret Wavelet Transform - DWT*) e Critério de Informação de Schwarz (*Schwarz Information Criterion - SIC*) (tais técnicas não serão detalhadas com mais profundidade, pois são complexas e conseqüentemente fugiriam do escopo deste trabalho. No entanto, para maiores informações, consultar [40]).

Assim, a cada iteração, o parâmetro Hurst é checado pelas funções anteriormente mencionadas, baseado nas novas amostras de tráfego (*bytes*), e, caso um ponto de mudança da autossimilaridade ocorra em escalas suficientes, ele será sinalizado em seguida. Uma vez sinalizado o ataque, o tráfego é dividido em segmentos, a fim de descobrir onde o ataque inicia. Durante a mensuração da intensidade do ataque, duas variáveis de entrada são usadas: o próprio parâme-

tro Hurst e a taxa de variação de sua mudança ao longo do processamento das amostras, em que o parâmetro Hurst é calculado novamente em cada um dos segmentos de tráfego supracitados.

Além disso, o autor alega utilizar um simulador de tráfego, de modo que o tráfego gerado como normal, anormal e malicioso de fato é criado com base também no parâmetro Hurst. Não são fornecidas maiores informações, como, por exemplo, se ferramentas consolidadas como de ataque DDoS foram utilizadas. A intensidade do tráfego de ataque é verificada ainda pela variação do parâmetro Hurst em escala. Embora os resultados sejam satisfatórios de acordo com as medidas utilizadas pelo autor para classificar o tráfego, ele também não fornece uma classificação mais refinada acerca de subtipos de ataques DDoS *high rates*.

No trabalho apresentado por Rodriguez, Briones e Nolazco [41], a única forma de considerar um determinado tráfego como de ataque ou não é com base no tempo de vida do pacote, por meio do campo Tempo de Vida (*Time to Live - TTL*) do cabeçalho IP. Por meio desse campo, uma tabela é construída, chamada de IP2HC (do termo *hop count* ou contador de salto), na qual consta uma tripla: IP de origem, a quantidade de saltos calculados (a partir do TTL decrementado) e a frequência de vezes em que foi constatado esse IP de origem. Todas as vezes que essa lista é incrementada, sugere-se uma ação de IP *Spoofing*, em que requisições ilegítimas são executadas, alterando o campo IP de origem por um outro IP qualquer. Qualquer mudança na contagem de saltos no momento em que a lista fosse incrementada apontaria para uma requisição maliciosa. Assim, nessa abordagem, a Lógica *Fuzzy* foi introduzida como forma de definir a taxa de pacotes que teriam sua rota modificada uma vez detectado o ataque.

Diante do exposto, é possível destacar que posteriores alterações nesse campo TTL, inclusive tornando-o mais próximo possível da quantidade de saltos entre sua origem legítima e o alvo, tornariam o mecanismo ineficiente. Além disso, conforme informado pelo próprio autor, houve quantidades significativas de pacotes legítimos tomados como de ataque durante o processo de mitigação (quando as rotas foram alteradas).

## 3.2 Aprendizado de Máquina

Em Khajuria e Srivastava [42], é apresentado um sistema híbrido, o *Neuro-Fuzzy*, utilizado para detecção de ataques DDoS. Conforme definido pela PUC Rio [43], sistemas híbridos são formados pela sinergia obtida pela combinação de duas ou mais técnicas. Dependendo da forma em que são combinadas, uma técnica pode contribuir com a outra em menor ou maior grau, com o objetivo de corrigir alguma deficiência.

Assim, um sistema *Neuro-Fuzzy* é um tipo de sistema híbrido incorporado, conhecido pela

junção de duas técnicas de modelagem: a Lógica *Fuzzy* e as Redes Neurais Artificiais. Khajuria e Srivastava [42] descrevem que, para fins de treinamento e teste, foram utilizados *datasets* com tipos de tráfego previamente conhecidos e outros gerados por meio de *Data Mining*. Tal tráfego, quando malicioso, era subdividido em dois tipos: ataques de exaustão e ataques voltados para saturação de banda. As bases de dados utilizadas possuíam tanto tráfego anômalo como tráfego legítimo. Na Tabela 3.2, são apresentadas algumas das informações extraídas a partir do tráfego analisado.

**Tabela 3.2: Campos avaliados em *datasets***

Parâmetro	Descrição	Tipo
<b>Duração</b>	Tempo em Segundos da Conexão	Contínuo
<b>Protocolo</b>	Tipo do Protocolo TCP/IP	Discreto
<b>Serviço</b>	Protocolo em Alto Nível (http, telnet ...)	Discreto
<b>Src Bytes</b>	Número de <i>bytes</i> da origem ao destino	Contínuo
<b>Dst Bytes</b>	Número de <i>bytes</i> do destino à origem	Contínuo
<b>flag</b>	Status de erro ou normal da conexão	Discreto
<b>land</b>	Conexões em que a origem e o destino são do mesmo <i>host</i>	Discreto
<b>Wrong fragment</b>	Número de fragmentos "quebrados"	Contínuo
<b>urgent</b>	Número de <i>urgent</i> pacotes	Contínuo

De acordo com Fuller [25], sistemas *neuro-fuzzy* funcionam como uma "caixa preta", podendo também ocasionar lentidão durante o aprendizado. Por conta disso, eles devem trabalhar com um número baixo de entradas devido ao crescimento discrepante das regras em função da quantidade delas e de seus conjuntos [43]. Fuller [25] também acrescenta que é praticamente impossível integrar uma informação nova sobre o problema em uma rede neural, uma vez executado o processo de aprendizado.

Os sistemas *fuzzy* são mais favoráveis quanto a isso com base em regras difusas e, portanto, seu desempenho pode ser ajustado mais facilmente. Enquanto as redes neurais são boas ao reconhecer padrões, elas não são boas explicando como eles alcançam suas decisões, daí o maior nível de complexidade durante posteriores ajustes. Apesar de tipos de ataques diferentes terem sido utilizados, o sistema não promove detecção de ataque em um modo mais refinado, não fornecendo, assim, mais detalhes sobre a sua natureza.

No trabalho apresentado por Braga, Mota e Passito [7], foi desenvolvido um sistema de detecção de ataques DDoS baseado na coleta leve (diferente das tradicionais) por meio das redes SDN. Sendo assim, a visão do tráfego global da rede era obtida por meio do protocolo *Open-Flow*. Uma vez que esses dados são coletados, algumas variáveis estratégicas são criadas: **APF**: Mediana de Pacotes por Fluxo; **ABF**: Mediana de *Bytes* por Fluxo; **ADF**: Mediana de Duração dos Fluxos ou Tempo Mediano dos Fluxos no Ativo antes de sofrerem *timeout*; **PPF**: Percenta-

gem de Fluxos pareados ou Quantidade de Fluxos cujo par requisição/resposta é encontrado em fluxos por coleta; **GSF**: Percentagem de Fluxos por coleta não pareados; **GDP**: Percentagem do crescimento da quantidade de portas diferentes por fluxo.

Assim, após executar a coleta e extrair essas informações, elas são repassadas para uma rede neural de Mapas Auto-organizáveis (*Self Organizing Maps* - SOM), a qual é capaz de transformar uma entrada com um padrão n-dimensional em um padrão bidimensional. Essa *clusterização* dimensional acontece de modo que as amostras ou entradas, quanto mais similares entre si, mais próximas ficam uma da outra no mapa. Esse modelo de rede neural artificial não é, por sua vez, supervisionado, pois a rede vai identificando os padrões por si só, durante a fase de treinamento, assim como ajustando seus pesos.

Os tráfegos analisados durante os experimentos foram obtidos a partir de um *link* entre o Japão e os Estados Unidos ao longo de sete anos e o tráfego de ataque foi gerado a partir de uma ferramenta à parte. Na Tabela 3.3, observam-se os variados tipos de ataques concebidos nas fases de treinamento e teste. Os valores entre parênteses indicam o tamanho máximo dos pacotes gerados ao longo das iterações e os números nas colunas de teste e treinamento representam o número estimado de fluxos por ataque e por fase.

**Tabela 3.3: Tráfegos de ataque durante a fase de treinamento e testes utilizando SOM**

<b>Tipos de Ataque</b>	<b>Treinamento</b>	<b>Testes</b>
TCP/SYN flood	6080	137612
UDP(60) flood	2967	52733
UDP(200) flood	-	31858
UDP(400) flood	3598	40019
UDP(800) flood	-	103165
ICMP(60) flood	-	57973
ICMP(1024) flood	5113	55344

Durante a fase de treinamento, 106.000 fluxos diferentes foram gerados; destes, 43.000 durante a fase de ataque e 63.000 durante o tráfego não nocivo. Na segunda fase, fase de testes, durante a validação do sistema, o tamanho dos pacotes foi variado, conforme Tabela 3.3 (detalhada anteriormente). 478.000 fluxos foram gerados e, destes, 253.000 foram de ataques DDoS e os outros 225.000 de tráfego legítimo.

Na Tabela 3.4 a seguir, são apresentados os resultados pela taxa de medição de acertos (DR) e pelos alarmes falsos (FA). Foi verificado também o descarte de duas variáveis: GSF e PPF, em que essas possuíam valores iguais (tal justificativa pode ser vista com detalhes no próprio artigo).

**Tabela 3.4: Resultados da classificação por meio de redes SOM - com 4 e 6 tuplas**

	6 tuplas		4 tuplas	
	DR(%)	FA(%)	DR(%)	FA(%)
<b>OF Switch 1</b>	98.61%	0.59%	98.57%	0.48%
<b>OF Switch 2</b>	99.11%	0.46%	98.73%	0.62%
<b>OF Switch 3</b>	3.52%	0.12%	3.27%	0.13%

Apesar dos resultados satisfatórios, a classificação dos ataques é generalista e o uso de Mapas Auto-organizáveis, como rede neural, exige significativo poder computacional [34]. Além disso, conforme descreve Fuller [25], é praticamente impossível integrar uma informação nova sobre o problema em uma rede neural, uma vez treinada a rede. Segundo o próprio autor, quando a rede está ociosa e o tráfego legítimo passa a trafegar, falsos positivos podem ser vistos, e isso ocorre pois as variáveis analisadas (detalhadas anteriormente) durante ataques DDoS acabam com um intervalo de valores semelhantes aos apresentados pelo tráfego legítimo.

### 3.3 Outras abordagens

No trabalho apresentado por Lim et al. [44], a detecção dos ataques DDoS, sendo esses específicos - ataques à camada de aplicação -, é feita pela própria aplicação, por meio de *thresholds* a nível de carga e quantidade máxima de sessões HTTP concorrentes.

Assim, uma vez detectado o ataque, o servidor de aplicação envia um alerta à Aplicação de Bloqueio de DDoS (*DDoS Blocking Application - DBA*). Após isso, o DBA informa ao servidor para que estabeleça o *socket* do serviço em uma outra interface virtual ou física dele (os autores afirmam que se poderia acionar uma ação de réplica do serviço também). Em seguida, uma mensagem de redirecionamento é enviada para aqueles clientes que desejarem estabelecer novas conexões, assim como aqueles que estão utilizando suas conexões previamente estabelecidas.

Esse redirecionamento, por sua vez, carrega um *captcha* de modo a validar que o cliente não é um robô (os autores contam com o fato de que os *bots*, até então, não possuem mecanismos inteligentes de seguir redirecionamentos ou preenchimento de *captchas*). Dessa forma, assim que clientes não legítimos passarem a tentar se reconectar por uma quantidade de vezes máxima (previamente estabelecida), uma regra de *drop* ou descarte será inserida para ele no *switch* perimetral.

Diante do exposto, constata-se que a identificação dos ataques é feita pelo próprio servidor de aplicação, logo, quando o ataque é identificado, provavelmente o servidor já está com boa parte de seus recursos ocupados por ele. Também é declarado pelos autores que a solução não



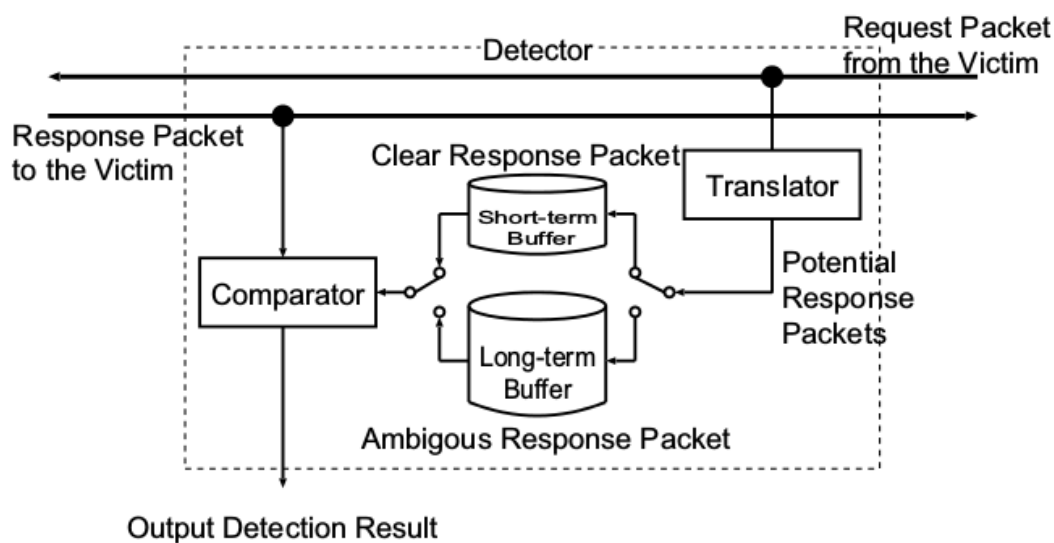
lida com IPs forjados, o que é bastante utilizado em ataques DDoS. Além disso, uma questão ficou pendente, o possível *overhead* que poderá ser ocasionado à memória dos *switches Open-Flow*, visto a quantidade de *drops* poder ficar alta, o que é bem possível, já que o mecanismo não leva em conta o IP *Spoofing*.

Em Tsunoda et al. [45], é apresentado um sistema de detecção de ataques de Negação de Serviço Distribuídos por Reflexão (*Distributed Reflection Denial of Service - DRDoS*). Ataques DDoS dessa modalidade se utilizam do mecanismo de IP *Spoofing* (por meio do qual requisições são geradas com o IP de origem falsificado) para gerar requisições ilegítimas de modo a "bombardear" um *host* alvo. A fim de atingir esse objetivo, os ataques usam um servidor de reflexão (geralmente executando algum serviço cujas respostas são frequentemente  $n$  vezes maiores que as requisições), o qual recebe e processa a requisição e acaba por enviar a resposta legítima ao *host* alvo. Para que o ataque obtenha sucesso, milhões de requisições são enviadas de vários *hosts* atacantes com destino ao servidor de reflexão, que, em seguida, transmitirá as respostas para o alvo original.

Dessa forma, Tsunoda et al. propõem um sistema de detecção desse tipo de ataque no lado da vítima, o qual não utiliza inspeção de estado, conforme outras soluções de inspeção *stateful*. Para esse fim, o mecanismo funciona gerando, de forma prévia, respostas esperadas, de acordo com as requisições criadas. Desse modo, para cada *request* executado, uma *response* esperada é mantida. Os autores também acrescentam que existem dois tipos de requisições: a **um pra um**, em que uma única resposta atenderá unicamente a uma determinada requisição; e a  **$n$  para um**, em que várias requisições simultâneas esperam pela mesma resposta. Nesse último caso, mais complexo, uma solução alternativa é utilizada de modo a criar uma *request* genérica para a resposta esperada.

Assim, a fim de conservar essa estrutura de funcionamento, são inseridos dois *buffers*, os quais irão manter as respostas esperadas: um para respostas simples (um para um) e outro para respostas ambíguas, em que várias requisições são para a mesma resposta. Para o *buffer* mais simples (um para um), seu tamanho é estabelecido proporcionalmente ao Tempo de Ida e Volta (*Round Trip Time - RTT*) do par requisição/resposta, o qual geralmente não passava de um segundo. Já os *buffers* mais complexos são proporcionais à quantidade de respostas esperadas de requisições ambíguas pelo RTT e ao longo da conexão. A estrutura descrita anteriormente é apresentada na Figura 3.2 adiante.

Conforme exposto pelos autores, uma limitação nesse mecanismo seria quando a rede da vítima aumentasse, de forma significativa, sua demanda de requisições e principalmente quando essas solicitassem serviços que esperam respostas ambíguas. Logo, no caso em que o tempo



**Figura 3.2: Sistema de funcionamento - Detecção de ataques DRDoS**

Fonte: Tsunoda et al., 2008.

de conexão é fator crítico para o tamanho do *buffer*, se o mesmo não for escolhido de forma minuciosa, o sistema poderá gerar falsos positivos (uma vez que a resposta recebida, embora legítima, não está mais em *buffer*, pelo tempo expirado). Esse fato foi apresentado pelos autores [45] durante os experimentos: "Para ambos os métodos, uma conexão de longa duração pode ser a causa de falsos positivos. [...]. Estes pacotes RST chegam 10 minutos após o último pacote chegar e o pacote de resposta potencial para confirmar esses pacotes já foram excluídos do *buffer*". Além disso, o sistema não é voltado para ataques quando o alvo é o serviço em si.

Na solução apresentada por Mohammadi, Javidan e Conti [46], um determinado sistema é capaz de identificar ataques *SYN Flooding*, assim como mitigá-los. Sendo assim, em um ataque desse tipo, milhares de tentativas de conexões TCP, que iniciam com a *flag SYN* ativada, são direcionadas a um servidor alvo. Isso ocorre de modo que, normalmente, quando uma tentativa de conexão alcança um servidor, ele reserva uma pilha TCP (que envolve recursos de *hardware* como processamento, memória, além de recursos do próprio sistema operacional, como arquivos descritores, *threads* por conexão, entre outros). Uma vez alocados tais recursos, o servidor responde ao requisitante com um *SYN-ACK*, aguardando, assim, que ele em seguida o retorne com uma mensagem *ACK*, estabelecendo, dessa forma, o *three-way handshake*, conforme Wetherall e Tanenbaum (2011, cap.6, p. 323) [18]. Aproveitando-se dessa funcionalidade do TCP, atacantes enviam inúmeras conexões *half-open* de forma simultânea, de modo a saturar os recursos computacionais do alvo, não estabelecendo de fato as conexões requisitadas.

Nessa proposta, portanto, a cada tentativa de conexão que ingressa na rede (com a *flag SYN* ativada), esta, ao alcançar o *switch*, é redirecionada, em seguida, ao controlador, a fim de que ele

informe ao *switch* o que fazer com esse novo tráfego. Ao chegar ao controlador, este adiciona a tentativa de conexão em uma lista, convenientemente chamada de *pending list*. Nessa lista, os dados de IP, porta de origem e destino são salvos e, ao final do registro, o campo *status*, com a última *flag* recebida dessa conexão (nesse caso, SYN). Uma vez salva a conexão na *pending list*, o controlador agora responderá ao *switch* para que este adicione o fluxo entre requisitante e servidor, mas de forma temporária, por três segundos. Quando o servidor responder à requisição, o mesmo processo ocorrerá, sendo que agora o *status* da conexão mudará para *SYN-ACK* e um novo fluxo, também temporário, será adicionado (entre servidor e requisitante). Caso o requisitante responda com a mensagem ACK, a entrada será deletada da *pending list* (visto ter sido constatado que o *three-way handshake* foi de fato concluído; logo, requisitante não é um atacante) e o fluxo passa a durar por mais de três segundos.

Em casos de ataque, quando a *pending list* alcançar um limite estático  $K$  de entradas, com status *SYN* ou *SYN-ACK*, o controlador passará a bloquear determinadas conexões ao servidor alvo. Esse último caso é descrito na Figura 3.3 a seguir, por meio da qual é possível verificar todos os *hops* envolvidos, desde o momento em que a requisição sai do *host* atacante, passando pelo *switch* que questiona o controlador (juntamente com o módulo do Slicots) e o servidor alvo.

Mohammadi et al. descreve, de forma clara, os resultados envolvendo quatro atributos checados: quantidade de fluxos criados ao longo dos cenários de ataque, tempo de resposta HTTP (também nesses cenários), tempo tomado por detecção e utilização de processamento ao longo dos experimentos. A solução, apesar de ter apresentado resultados satisfatórios, não foi capaz de lidar com o mascaramento de IP, tanto que, conforme enfatizado pelo autor, foi usado *MAC Spoofing* (salientando que *IP Spoofing* é geralmente utilizado em ataques *Syn Flooding*, executados a partir da Internet). Além disso, a ferramenta lida especificamente com uma única linha de ataques DDoS, baseada em flags *SYN*. Embora o autor tenha mostrado os resultados com o valor de  $K$  variando, ele não especifica o intervalo de tempo para se acumular o valor de  $K$ . Por exemplo, para um valor de  $K$  muito baixo, em um intervalo de tempo pequeno (intervalo em que  $K$  passa a ser contado), falsos positivos poderiam ser gerados, de modo que o *status SYN* poderia ser advindo de uma conexão prestes a se estabelecer, ainda.

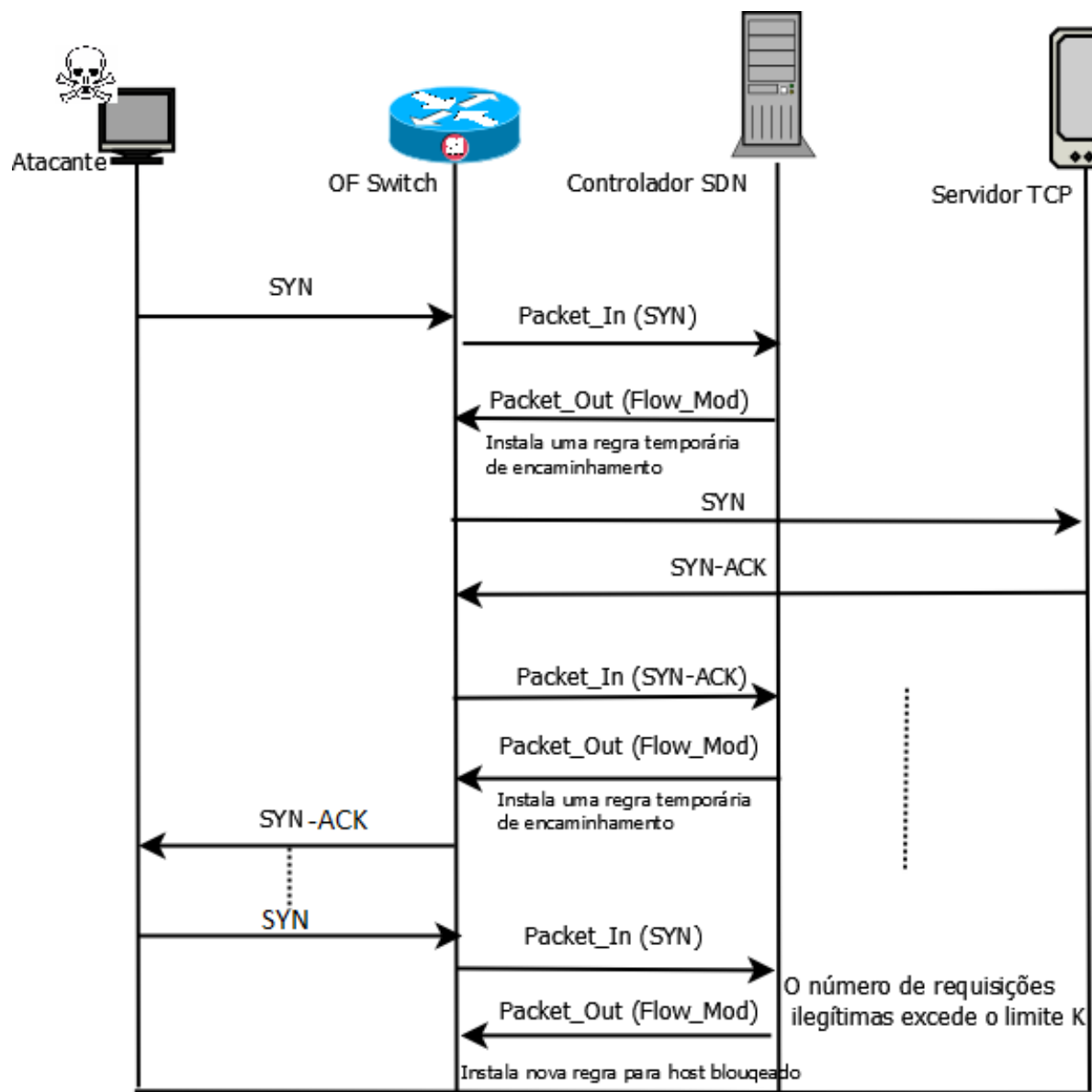


Figura 3.3: Esquema de detecção e bloqueio de requisições maliciosas pelo SLICOTS

Fonte: Elaborada pela autora.

# Capítulo 4

## FUZDETECT - COMPONENTES E FUNCIONAMENTO

---

---

Na sequência, será descrito todo o funcionamento da arquitetura do FuzDetect, partindo da coleta do tráfego via protocolo OpenFlow [12], logo em seguida, o módulo de extração que analisará tal coleta e estabelecerá a partir da mesma variáveis específicas que serão mantidas numa base de dados. Após a etapa anterior, novas variáveis serão parametrizadas por meio dos dados mantidos na base. As últimas variáveis mencionadas servirão de entrada tanto para execução do otimizador, como também do classificador. Tais etapas serão descritas em detalhes a seguir.

### 4.1 Coleta de Fluxos

Em um primeiro momento, conforme indicado na Figura 4.1 adiante, um módulo coletor de fluxos funciona de forma remota, via um canal que utiliza o protocolo *OpenFlow* exclusivo com cada *switch*, capturando os fluxos de tráfego deles naquele momento a cada dois minutos (tempo escolhido a fim de não sobrecarregar os *switches* e nem a rede com as consultas e por ser um intervalo de tempo suficiente para detecção de desvio no comportamento do tráfego). Metadados apresentados na Seção 2.1, como IPs, portas TCP e UDP, assim como alguns contadores, são coletados e servirão como amostras.

### 4.2 Extração de Dados

No segundo passo, de acordo com a mesma Figura, é verificado se a quantidade de coletas necessárias foi gerada para cada ativo da rede. Se sim, o módulo extrator irá analisar as coletas

(que estão em formato bruto de dados) e, a partir delas, irá gerar os seguintes dados, com base nos contadores de fluxos e outros campos: **Quantidade de portas TCP e UDP conectadas (GDP)**: portas TCP e UDP por coleta. Da mesma forma que a falsificação de IP é gerada por ataques DDoS, as portas também podem ser geradas aleatoriamente por ataques [7]. **Quantidade de bytes (QNT\_BYTES)**: quantidade massiva ou bruta de *bytes* por coleta. **Quantidade de pacotes (QNT\_PKTS)**: quantidade massiva ou bruta de pacotes por coleta.

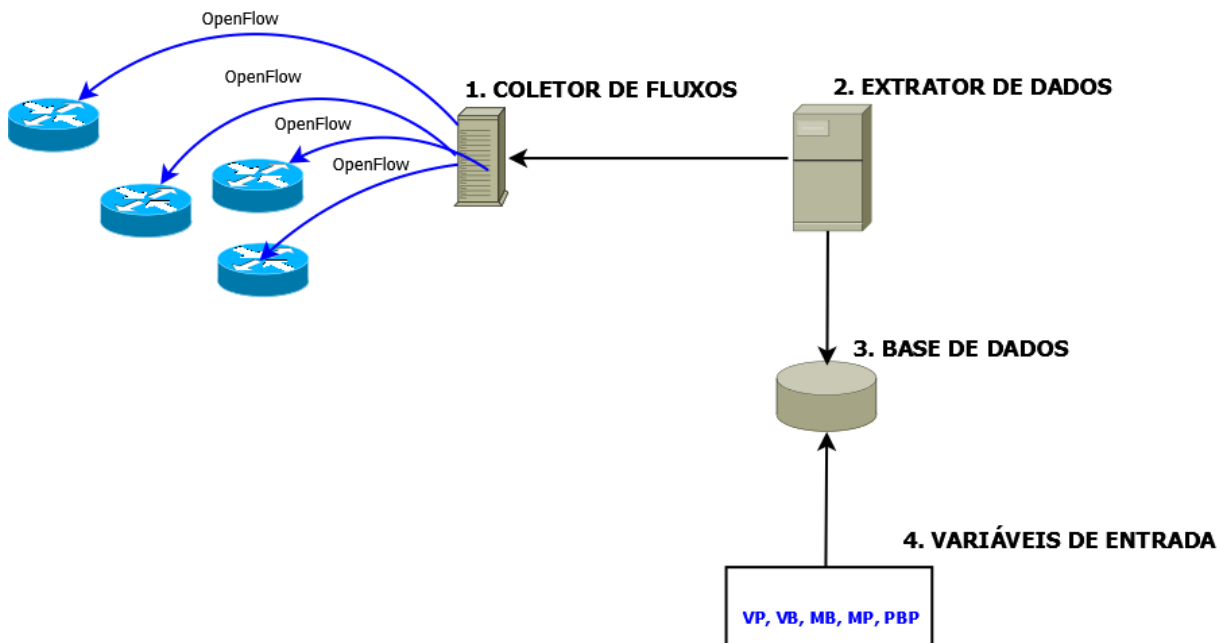


Figura 4.1: FuzDetect: Arquitetura fluxo de dados  
 Fonte: Elaborada pela autora.

### 4.3 Base de Dados

Uma vez configuradas tais variáveis por coleta, estas serão salvas em uma base de dados, conforme passo 3 da Figura 4.1 anterior. Na base de dados, esses valores são persistidos e separados por *switch*, de modo que cada tabela representa um *switch* e será composta por colunas, e estas, por sua vez, representarão, cada uma, as variáveis anteriormente citadas, de acordo com a Tabela 4.1.

Tabela 4.1: Estrutura da Tabela na Base de Dados

<b>Campos</b>
GDP
QNT_BYTES
QNT_PKTS

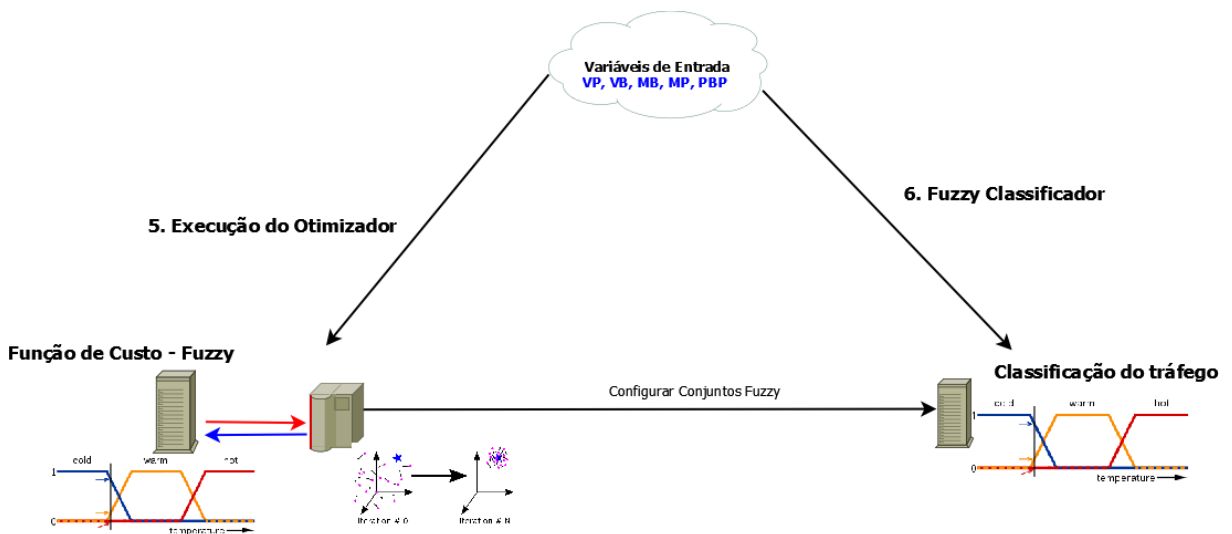
## 4.4 Estabelecimento das Variáveis de Entrada

Na quarta etapa, um módulo de análise consulta a base procurando pelos dados gerados a partir das últimas coletas (dados mais recentes), conforme Figura 4.1. Por meio desses dados, outras variáveis serão geradas e servirão como dados de entrada para o PSO e para o Sistema *Fuzzy*, conforme passo 5 e 6 da Figura 4.2, respectivamente. A seguir, a descrição das variáveis:

- **Variância de portas (VP):** a variância de portas é estabelecida por meio da quantidade de portas TCP ou UDP das últimas coletas. De acordo com o tipo do ataque, o crescimento da quantidade de portas TCP e UDP conectadas apresenta pouca estabilidade da média ao longo do tempo;
- **Variância de bytes (VB):** a variância de *bytes* é estabelecida a partir da quantidade de *bytes* das últimas coletas salvas. Representa o mesmo que a variância anterior, exceto que aqui lida-se com a média de *bytes* por coleta;
- **Mediana da quantidade de bytes (MB):** outra peculiaridade dos ataques DDoS é o tamanho da sua carga útil, que muitas vezes é muito pequeno, a fim de aumentar a eficácia desse tipo de ataque [7]. Na Equação 4.1, é apresentada essa mediana, em que  $QB$  corresponde à quantidade de *bytes* por coleta e  $QF$  à quantidade de fluxos por coleta;
- **Mediana da quantidade de pacotes (MP):** um efeito colateral de ataques DDoS é a geração de fluxos com um pequeno número de pacotes, isto é, cerca de 3 pacotes por fluxo. Dado que o tráfego normal geralmente envolve um maior número de pacotes, calcula-se o valor mediano deste [7]. Essa mediana é apresentada na Equação 4.2, na qual  $QP$  corresponde à quantidade de pacotes por coleta e  $QF$  é a quantidade de fluxos por coleta;
- **Porcentagem de bytes em portas (PBP):** durante os experimentos, foi identificado que existe uma relação, de acordo com o ataque, entre média de portas e média de *bytes*. Essa porcentagem representa o valor que a média de *bytes* corresponde na média de portas. Se o valor é alto, caracteriza desproporcionalidade entre a quantidade de *bytes* em relação à quantidade de portas TCP ou UDP conectadas, e se é baixo o valor, indica desproporção também, mas em relação à quantidade de portas TCP ou UDP conectadas. A média de portas é estabelecida a partir da quantidade de portas TCP ou UDP das últimas coletas e a média de *bytes* é estabelecida a partir da quantidade de *bytes* das últimas coletas.

$$mediana(QB) = \begin{cases} QB((QF + 1)/2), & \text{se } QF \text{ é ímpar;} \\ \frac{QB(QF/2) + QB((QF+1)/2)}{2} & \text{se } QF \text{ é par.} \end{cases} \quad (4.1)$$

$$mediana(QP) = \begin{cases} QP((QF + 1)/2), & \text{se } QF \text{ é ímpar;} \\ \frac{QP(QF/2) + QP((QF+1)/2)}{2} & \text{se } QF \text{ é par.} \end{cases} \quad (4.2)$$



**Figura 4.2: FuzDetect: Arquitetura - Otimizador e Classificador**

Fonte: Elaborada pela autora.

## 4.5 Sistema de Classificação - Lógica Fuzzy

Todas as etapas descritas nesta Seção correspondem à execução da Função de Custo do Otimizador, durante o passo 5, e ao passo 6, em que ocorre a classificação propriamente dita. Ambos os passos podem ser vistos na Figura 4.2. Quanto ao passo 5, vale ressaltar que a função *fitness* ou de custo utilizada pelo PSO a fim de validar a qualidade das soluções encontradas é implementada por meio também da Lógica Fuzzy, entretanto, durante o passo 5, o classificador, a cada saída gerada, irá comparar tal saída com o tráfego esperado (uma vez que nesta etapa ele saberá previamente qual o tipo de tráfego gerado), calculando, dessa forma, a taxa de acertos e, assim, tornando o classificador o mais adaptado possível à rede analisada. Esse processo, referente ao passo 5, será detalhado posteriormente, na Seção 4.6.

Diferentemente, no passo 6, o classificador *fuzzy*, já em produção, não saberá com que tipo de tráfego estará lidando previamente, portanto, a classificação ocorrerá de acordo com as



informações contidas nas regras e com a configuração dos conjuntos, que é dinâmica e acontece no passo 5.

É importante destacar que o processo de classificação não acontece em tempo real, é *offline*. A coleta é feita, em seguida refinada pelo módulo extrator até estabelecer as variáveis de entrada em alto nível.

#### 4.5.1 Variáveis de Entrada e Saída *Fuzzy* estabelecidas

As variáveis de entrada e de saída utilizadas e seus conjuntos se encontram na Tabela 4.2 a seguir:

**Tabela 4.2: Variáveis de entrada e de saída e seus respectivos conjuntos**

Variáveis de Entrada				Variáveis de Saída			
VP	VB	MB	MP	PBP	AV	AE	TNM
B	B	B	B	B	B	B	B
M	M	-	-	-	M	M	M
A	A	A	A	A	A	A	A

Nesta tabela, as variáveis de entrada são: VP: Variância de Portas, VB: Variância de *Bytes*, MB: Mediana de *Bytes*, MP: Mediana de Pacotes e PBP: Percentagem de *Bytes* em Portas; e as variáveis de saída são: AV: Ataques Volumétricos, AE: Ataques de Exaustão e TNM: Tráfego Não Malicioso. Quanto aos conjuntos, tem-se: A: Alto, B: Baixo e M: Médio. Destaca-se ainda que as variáveis MB, MP e PBP não possuem o conjunto Médio, mas unicamente Baixo e Alto.

A motivação no uso dessas variáveis de entrada é apresentada na Seção 4.4. Quanto às definições das variáveis de saída, estas podem ser vistas novamente na Subseção 2.3.1. Vale ressaltar também que os limitadores dos conjuntos das variáveis de entrada são dinâmicos, via a otimização descrita anteriormente.

#### 4.5.2 Regras *Fuzzy* utilizadas

As regras usadas no sistema *fuzzy* proposto constam na Tabela 4.3 adiante. A conversão das siglas é a mesma da Tabela 4.2 (seu funcionamento e dos operadores está detalhado em 2.4). Salienta-se que o arranjo dessas regras, bem como o uso das operações, foi feito de forma

empírica, a partir de várias observações acerca do comportamento desses dados de entrada de acordo com o tráfego e utilizando como referência o trabalho feito por Braga, Mota e Passito [7].

**Tabela 4.3: Regras Fuzzy utilizadas**

Regras	Tráfego
$((MP - B \text{ AND } VP - A) \text{ OR } (MB - A \text{ AND } PBP - A))$	$(AV - A) (AE - B) (TNM - M)$
$((VP - A \text{ AND } VB - B) \text{ OR } (MP - B \text{ AND } PBP - B))$	$(AV - M) (AE - A) (TNM - B)$
$((VP - B \text{ OR } VP - M) \text{ AND } (MP - A)) \text{ OR } (MB - A \text{ AND } MP - A)$	$(AV - B) (AE - M) (TNM - A)$

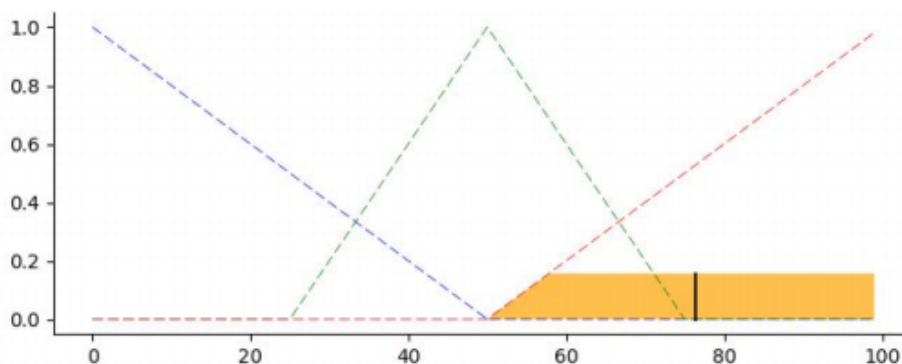
De acordo com a Tabela 4.3 é definido que:

- Na primeira linha, se **Mediana de Pacotes é Baixa e Variância de Portas é Alta ou Mediana de Bytes é Alta e Percentagem de Bytes em Portas é Alta**, então: **Ataques Volumétricos** têm alta chance de acontecer, **Ataques de Exaustão** têm chance **Baixa** e **Tráfego não Malicioso** tem **Média** chance;
- Na segunda linha, por meio de expressões numéricas, é definido que: se **Variância de Portas é Alta e Variância de Bytes é Baixa ou Mediana de Pacotes é Baixa e Percentagem de Bytes em Portas é Baixa**, então: **Ataques Volumétricos** têm **Média** chance de acontecer, **Ataques de Exaustão** têm chance **Alta** e **Tráfego não Malicioso** tem **Baixa** chance;
- Na terceira linha, se: **Variância de Portas é Baixa ou Variância de Portas é Média e Mediana de Pacotes é Alta ou Mediana de Bytes é Alta e Mediana de Pacotes é Alta**, então: **Ataques Volumétricos** têm **Baixa** chance de acontecer, **Ataque de Exaustão** tem chance **Média** e **Tráfego não Malicioso** tem **Alta** chance.

### 4.5.3 Defuzzificação

No processo de *defuzzificação*, a saída gerada após a execução das regras é processada, por meio da função centroide (vide Subseção 2.4.6), e uma saída numérica é obtida. Conforme mencionado anteriormente, no modelo implementado da Lógica Fuzzy, foram utilizadas três variáveis de saída (por tipo de tráfego, cada), dessa forma, a função de *defuzzificação* usada irá agregar os conjuntos das variáveis, por variável de saída, e em seguida irá "varrer" todas as regras por variável de saída, como detalhado em 2.4.6.

Assim, cada saída numérica gerada, por variável de tráfego, exibirá (em uma escala de 0 a 100%, uma vez que o universo de discurso - eixo  $x$ , das três variáveis de saída, foi estabelecido dessa forma) as chances de um determinado ataque estar ocorrendo. Neste sentido, na Figura 4.3 adiante, é apresentada uma variável de saída *defuzzificada*. É possível identificar que, para o conjunto Alto dessa variável, as chances do tráfego que ela discrimina estar ocorrendo é em torno de 79%.



**Figura 4.3:** Exemplo de saída *defuzzificada*

Fonte: Elaborada pela autora.

Nos Pseudocódigos 4, 5 e 6, são apresentados, respectivamente: o processo de configuração das variáveis de entrada e estabelecimento de pertinência por conjuntos das variáveis de entrada; o processo de extração das pertinências por conjunto das variáveis de saída, de acordo com as regras; e a saída final *defuzzificada*, após a agregação dos conjuntos *fuzzy* das variáveis de saída.

No Pseudocódigo 4, as variáveis são extraídas por amostra e por *switch*, conforme linha 4, por meio de dois *loops*, um por *switch* linha 1 e outro por amostra, linha 2. Logo em seguida, as variáveis são *fuzzificadas*, conforme linha 5 a 9 e suas respectivas pertinências por conjunto das variáveis de entrada são geradas. Após, no Pseudocódigo 5, esses valores de pertinência são verificados por regra estabelecida, conforme Tabela 4.3, de acordo com variável e conjunto, conforme linha 2 a 10. E no Pseudocódigo 6, esses valores de saída gerados são agregados por variável e conjuntos de saída, linhas 2, 3 e 4, respectivamente. Logo depois, a *defuzzificação* ocorre, por variável de saída agregada, linhas 6,7 e 8, respectivamente, gerando, dessa forma, um resultado numérico que indica, de forma aproximada, as chances de determinado tipo de tráfego estar em evidência, podendo ser: tráfego não malicioso, ataques por exaustão e ataques volumétricos. No último passo, linha 10, esses valores são devidamente reportados.

**Algoritmo 4:** Estabelecimento das variáveis de entrada *fuzzy* - FuzDetect

---

```

1 for i in Switches do
2   for j in Amostras Tráfego do
3     Variáveis conforme Tabela 4.2.
4     VP, VB, MP, MP, PBP := Configurar Variáveis Entrada (i,j);
5     Pertinência VP: = Extrair Pertinência Fuzzy VP (VP);
6     Pertinência VB: = Extrair Pertinência Fuzzy VB (VB);
7     Pertinência MP: = Extrair Pertinência Fuzzy MP (MP);
8     Pertinência MB: = Extrair Pertinência Fuzzy MB (MB);
9     Pertinência PBP: = Extrair Pertinência Fuzzy PBP (PBP);
10  end
11 end

```

---

**Algoritmo 5:** Extração de Pertinências *Fuzzy* - FuzDetect

---

```

1 Variáveis de entrada e saída e regras implementadas, conforme Tabela 4.2 e 4.3,
  respectivamente.
2 AV Alto: = Pertinência Saída Regra 1 (Pertinência VP, Pertinência VB, Pertinência MP,
  Pertinência MB, Pertinência PBP);
3 TNM Médio: = Pertinência Saída Regra 1 (Pertinência VP, Pertinência VB, Pertinência
  MP, Pertinência MB, Pertinência PBP);
4 AE Baixo: = Pertinência Saída Regra 1 (Pertinência VP, Pertinência VB, Pertinência
  MP, Pertinência MB, Pertinência PBP);
5 AE Alto: = Pertinência Saída Regra 2 (Pertinência VP, Pertinência VB, Pertinência MP,
  Pertinência MB, Pertinência PBP);
6 AV Médio: = Pertinência Saída Regra 2 (Pertinência VP, Pertinência VB, Pertinência
  MP, Pertinência MB, Pertinência PBP);
7 TNM Baixo: = Pertinência Saída Regra 2 (Pertinência VP, Pertinência VB, Pertinência
  MP, Pertinência MB, Pertinência PBP);
8 TNM Alto: = Pertinência Saída Regra 3 (Pertinência VP, Pertinência VB, Pertinência
  MP, Pertinência MB, Pertinência PBP);
9 AE Médio: = Pertinência Saída Regra 3 (Pertinência VP, Pertinência VB, Pertinência
  MP, Pertinência MB, Pertinência PBP);
10 AV Baixo: = Pertinência Saída Regra 3 (Pertinência VP, Pertinência VB, Pertinência
  MP, Pertinência MB, Pertinência PBP);

```

---

---

**Algoritmo 6:** Agregação e Defuzzificação *Fuzzy* - FuzDetect

---

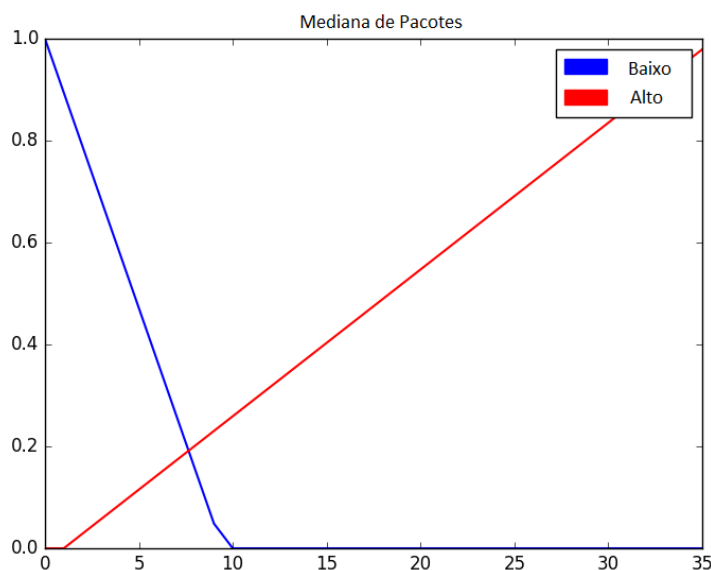
- 1 São agregados todos os conjuntos de saída por variável de saída:
  - 2 **Agregação AV:** = (AV Alto, AV Médio, AV Baixo);
  - 3 **Agregação AE:** = (AE Alto, AE Médio, AE Baixo);
  - 4 **Agregação TNM:** = (TNM Alto, TNM Médio, TNM Baixo);
  - 5 Processo de *defuzzificação* definido pela Equação 2.7.
  - 6 **AV Saída Defuzzificada:** = Defuzzificação (Agregação AV);
  - 7 **AE Saída Defuzzificada:** = Defuzzificação (Agregação AE);
  - 8 **TNM Saída Defuzzificada:** = Defuzzificação (Agregação TNM);
  - 9 Informar Saída Gerada (estimação de tráfego) - Etapa Final
  - 10 **Informar Saída Por Variável**(AV Saída Defuzzificada, AE Saída Defuzzificada, TNM Saída Defuzzificada);
- 

Todos os Pseudocódigos relacionados à Lógica *Fuzzy* apresentados foram implementados por meio da biblioteca Skfuzzy, escrita em Python. Para maiores detalhes acerca dela, consultar Skfuzzy [47].

## 4.6 Execução do PSO

O FuzDetect não poderia funcionar com sua configuração de forma estática, visto que as características dos dados variam de acordo com o ambiente analisado. Desse modo, é ideal que, antes de classificar o tráfego, o FuzDetect possa se adaptar à rede. Sendo assim, uma vez geradas as amostras a partir da rede SDN, em que nesta etapa tráfegos de vários tipos são criados intencionalmente e conhecendo-se, de forma prévia, o tipo de tráfego gerado, o algoritmo de otimização é executado e cada possível solução para os conjuntos *fuzzy* é validada por meio da função *fitness*. Na Figura 4.1, é possível ver a função de custo em conjunto com o PSO. Como mencionado na Seção 4.5, tal função também é implementada com Lógica *Fuzzy*, exceto que aqui o tráfego classificado é previamente rotulado com seu tipo, quanto mais acertos no tipo de tráfego a partícula/solução alcançar, mais chances ela terá de ser usada na configuração dos conjuntos, durante a etapa de classificação do FuzDetect.

Na Figura 4.4, são apresentados os conjuntos baixo e alto da variável Mediana de Pacotes e todos esses são limitados por funções triangulares (a motivação para o uso dessas funções se baseia no que é mencionado por Lee [24] acerca da utilização delas para lidar com dados contínuos). Nesta figura é apresentada também a variável com seus conjuntos antes da execução do PSO. Já na Figura 4.5, após a execução do PSO, é possível ver que os conjuntos sofreram



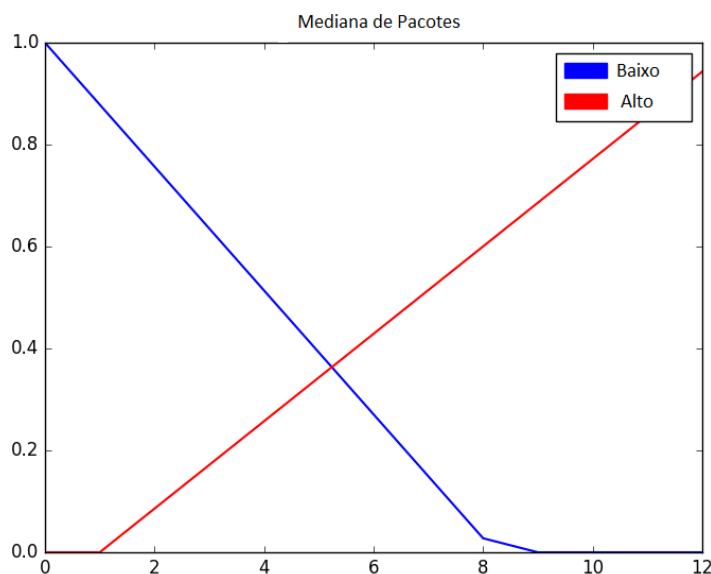
**Figura 4.4:** Conjuntos *Fuzzy* antes da execução do PSO

Fonte: Elaborada pela autora.

ajustes nos eixos, alterando, assim, o escopo de cada um.

No Pseudocódigo 7, é apresentada a execução do PSO, em que cada partícula ou solução equivale a um conjunto de valores que servirão como limitadores para os conjuntos *fuzzy* das variáveis de entrada (*loop* da linha 1), e a qualidade dessa solução é caracterizada de acordo com a taxa de acerto no tipo de tráfego classificado. Logo, reforçando-se o que foi dito anteriormente, a função de custo ou de checagem da qualidade da partícula equivale às funções apresentadas nos Pseudocódigos 4, 5 e 6, na Subseção 4.5.3, exceto que nesta etapa o tráfego é previamente conhecido e os conjuntos *fuzzy* das variáveis de entrada recebem possíveis valores (partículas) para seus limites, sendo que, quanto mais acertos tais configurações de limites proverem, de acordo com a identificação do tráfego, maior a chance de essa configuração dos conjuntos das variáveis de entrada (representada pela partícula) ser usada nos conjuntos *fuzzy* em produção.

Na linha 4, ainda do Pseudocódigo 7, as variáveis de entrada são configuradas. Em seguida, linha 5, os conjuntos *fuzzy* das variáveis de entrada *fuzzy* são configurados em função da partícula  $z$ . No próximo passo, da linha 7 a 11, a pertinência de cada variável de entrada é extraída. Na linha 12, as regras são processadas em função dessas pertinências geradas e os conjuntos das variáveis de saída são agregados. As saídas numéricas *defuzzificadas* por variável de saída é gerada, conforme linhas 13, 14 e 15. E por último, três condicionais são estabelecidas, de modo que, cada uma acumula acertos por partícula  $z$  de acordo com cada possibilidade de tráfego esperado. As três condicionais são declaradas nas linhas 17, 20 e 23, cada uma.



**Figura 4.5: Conjuntos Fuzzy ajustados pelo PSO**

Fonte: Elaborada pela autora.

Todos os pseudocódigos mencionados na Seção 4.5 e 4.6 foram implementados em linguagem Python. De acordo com Borges (2014, cap. 1, p. 13) [48], algumas vantagens em se utilizar essa linguagem são:

A linguagem inclui diversas estruturas de alto nível (listas, dicionários, data, hora, complexos e outras) e uma vasta coleção de módulos prontos para uso, além de *frameworks* de terceiros que podem ser adicionados. Também possui recursos encontrados em outras linguagens modernas, tais como: geradores, introspecção, persistência, metaclasses e unidades de teste. Multiparadigma, a linguagem suporta programação modular e funcional, além da orientação a objetos. [...]

Mesmo os tipos básicos no Python são objetos. A linguagem é interpretada através de *bytecode* pela máquina virtual Python, tornando o código portátil. Com isso é possível compilar aplicações em uma plataforma e rodar em outros sistemas ou executar direto do código fonte. [...]. Python é um software de código aberto (com licença compatível com a General Public License (GPL), porém menos restritiva, permitindo que o Python seja inclusive incorporado em produtos proprietários). A especificação da linguagem é mantida pela Python Software Foundation<sup>2</sup> (PSF). [...] É possível integrar o Python a outras linguagens, como a Linguagem C e Fortran.

**Algoritmo 7:** Pseudocódigo função *fitness* PSO - FuzDetect

```

1 for z in Partícula do
2   for i in Switches do
3     for j in Amostras Tráfego Rotulada do
4       VP, VB, MP, MB, PBP: = Configurar Variáveis Entrada (i,j);
5       C: = Configurar Conjuntos Variáveis Entrada (z);
6       Siglas das variáveis de acordo com a Tabela 4.2
7       Pertinência VP: = Extrair Pertinência Fuzzy VP (C, VP);
8       Pertinência VB: = Extrair Pertinência Fuzzy VB (C, VB);
9       Pertinência MP: = Extrair Pertinência Fuzzy MP (C, MP);
10      Pertinência MB: = Extrair Pertinência Fuzzy MB (C, MB);
11      Pertinência PBP: = Extrair Pertinência Fuzzy PBP (C, PBP);
12      Processamento das Regras e Agregação das Variáveis de Saída, conforme
13      Pseudocódigos 5 e 6, da Subseção 4.5.3
14      AV Saída Defuzzificada: = Defuzzificação (Agregação AV);
15      AE Saída Defuzzificada: = Defuzzificação (Agregação AE);
16      TNM Saída Defuzzificada: = Defuzzificação (Agregação TNM);
17      Resposta Esperada: = Checar Tráfego Rotulado (i,j);
18      if AV Saída Defuzzificada > AE Saída Defuzzificada AND AV Saída
19      Defuzzificada > TNM Saída Defuzzificada AND Resposta Esperada == AV
20      then
21        Acertos Partícula++;
22      end
23      if AE Saída Defuzzificada > AV Saída Defuzzificada AND AE Saída
24      Defuzzificada > TNM Saída Defuzzificada AND Resposta Esperada == AE
25      then
26        Acertos Partícula++;
27      end
28      if TNM Saída Defuzzificada > AE Saída Defuzzificada AND TNM Saída
29      Defuzzificada > AV Saída Defuzzificada AND Resposta Esperada == TNM
30      then
31        Acertos Partícula++;
32      end
33    end
34  end
35 end

```



# Capítulo 5

## TESTBED

---

---

Na etapa aqui descrita, a intenção é montar um ambiente virtual o mais próximo possível de ambientes físicos em produção (desde quantidade e disposição de *switches* até a demanda de tráfego da rede). Nesse ambiente, tipos de tráfego variados são gerados (ataques volumétricos, ataques de exaustão, tráfego legítimo).

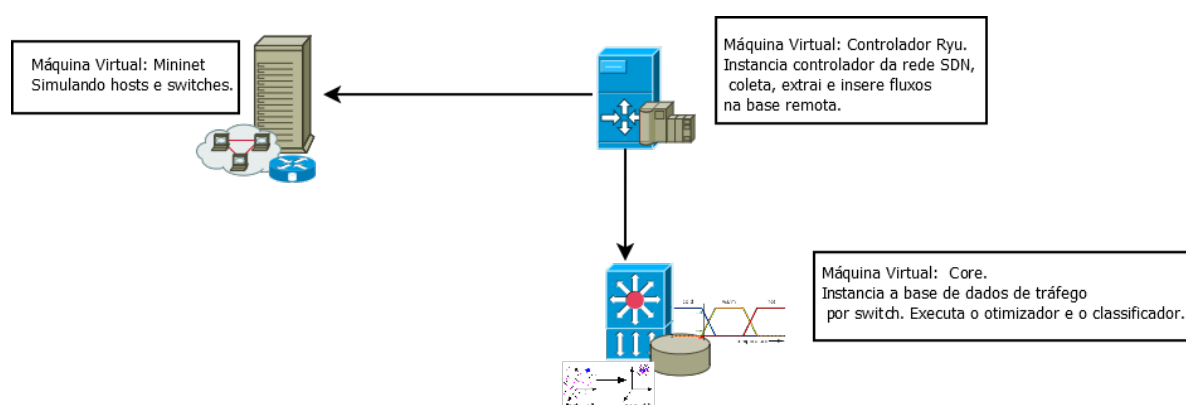
### 5.1 Ambiente

O ambiente físico onde todo o cenário foi implementado possui a seguinte configuração de *hardware*: Intel(R) Core(TM) i7-6500 CPU @2,5GHz, 16G de RAM. Nesse ambiente, três máquinas virtuais distintas foram utilizadas. A configuração de cada uma e de seus respectivos propósitos serão apresentados a seguir:

- **Mininet**: Sistema Operacional: Ubuntu 16.0 LTS, 64 bits, 5G RAM, 2 cores de processamento. O Mininet [49] foi utilizado com a finalidade de simular uma rede SDN. Os *hosts* virtuais que simulam clientes legítimos e atacantes eram executados no Mininet, no mesmo ambiente virtualizado dos *switches*;
- **Controlador Ryu**: Sistema Operacional: Ubuntu 16.0 LTS, 64 bits, 5G RAM, 4 cores de processamento. O controlador da rede localizava-se em uma máquina virtual remota e foi implementado utilizando-se o *framework Ryu* [50]. Nesse mesmo servidor, eram executados também o módulo de coleta, o módulo de extração e o módulo de inserção de dados de fluxos na base, pelo FuzDetect;
- **Core**: Sistema Operacional: Ubuntu 16.0 LTS, 64 bits, 6G RAM, 4 cores de processamento. Nesse servidor é mantida a base dos dados de fluxos da rede. Aqui também

funcionam o PSO (durante a fase de configuração dos conjuntos *fuzzy*) e o classificador de tráfego *Fuzzy* propriamente dito.

Na Figura 5.1 adiante, é apresentada a separação supracitada e uma breve definição das funções das respectivas máquinas.



**Figura 5.1: FuzDetect: Fluxo de funcionamento entre as máquinas virtuais**

Fonte: Elaborada pela autora.

## 5.2 Topologias

Os *hosts* virtuais que simulam clientes legítimos e atacantes também eram executados no Mininet, no mesmo ambiente virtualizado dos *switches*.

Com o intuito de gerar resultados mais confiáveis, optou-se por usar topologias significativamente diferentes entre si, sendo duas mais simples, com pouca ramificação do enlace, a fim de tornar a classificação menos árdua e outras duas mais complexas, com mais ramificações, com atacante e alvo mais distantes e *switches* mais dispersos. Em cada uma, realizaram-se alguns experimentos: ataques volumétricos, ataques de exaustão e tráfego não malicioso. Em todos os experimentos, houve fluxos de tráfego legítimo e de ataque simultaneamente.

Durante os testes, foram usadas quatro topologias diferentes, apresentadas nas Figuras 5.2 - Topologia I, 5.3 - Topologia II, as de menor porte e mais simples e 5.4 - Topologia III e 5.5 - Topologia IV, as mais complexas.

As topologias III e IV, Figuras 5.4 e 5.5, respectivamente, foram instanciadas utilizando-se duas máquinas virtuais para os *switches* e *hosts*, de modo que a comunicação entre elas é feita por meio de um túnel virtual, do próprio Mininet.

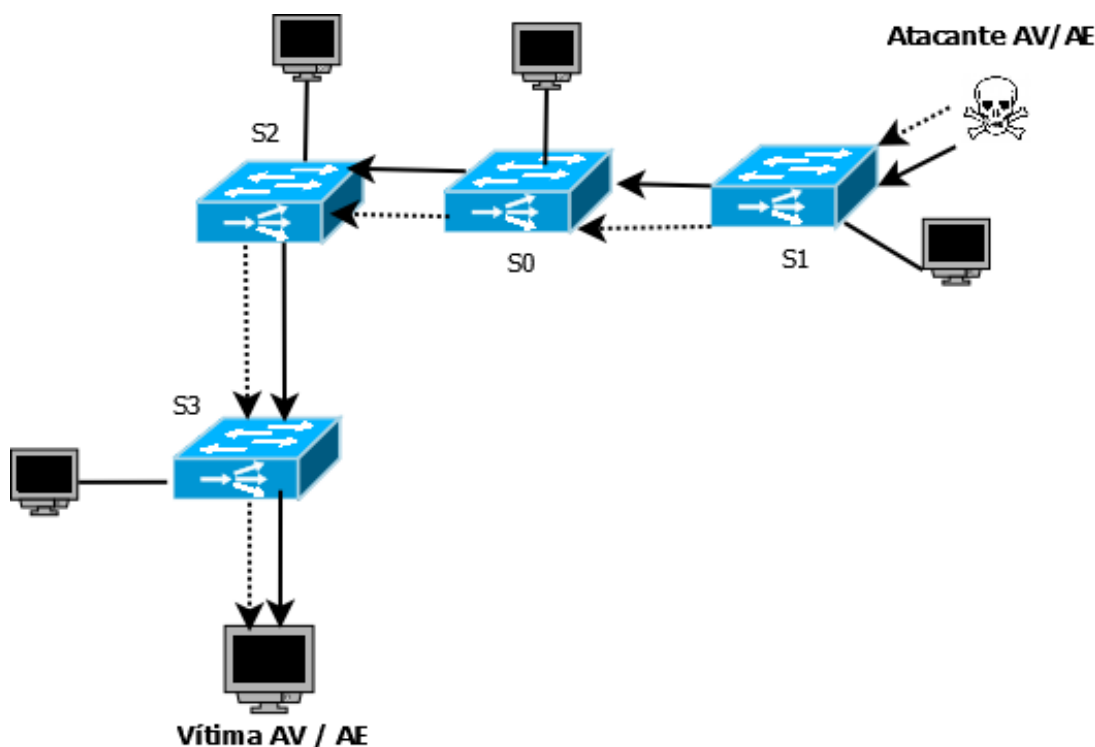


Figura 5.2: Topologia I

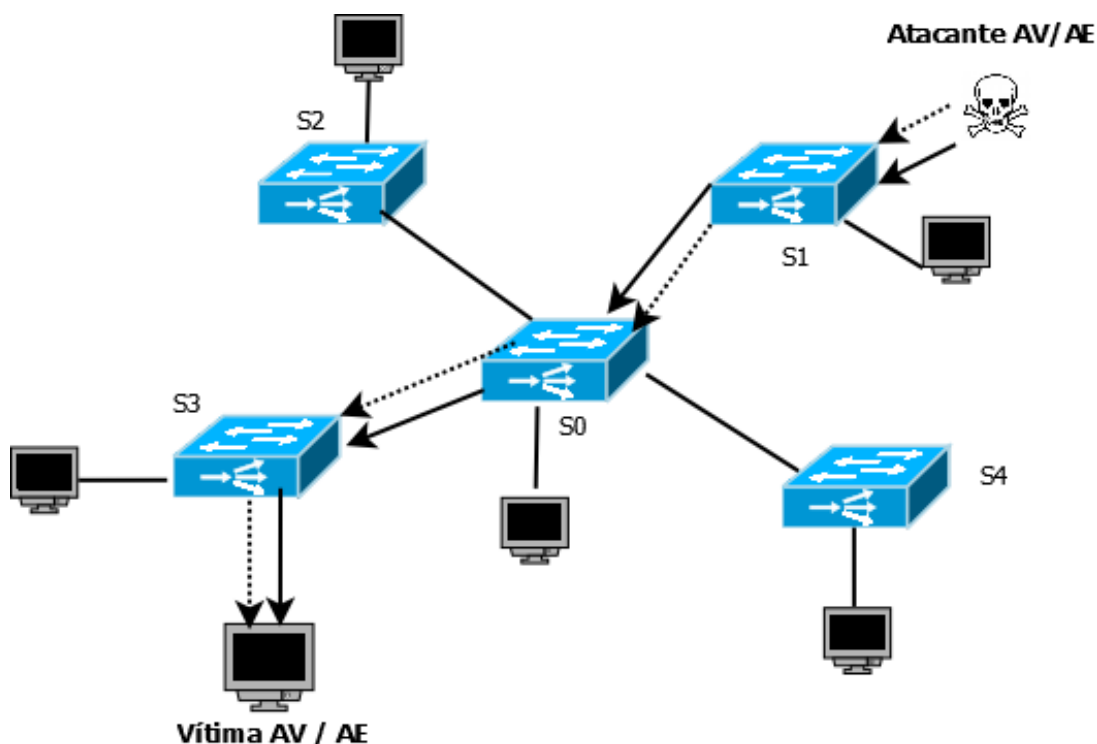
Fonte: Elaborada pela autora.

### 5.3 Geração de Tráfego DDoS e Legítimo

Para a simulação de ataques volumétricos, utilizou-se o *dnsdrdos* [51] e o *Hping3* [52], que geram requisições DNS amplificadas e pacotes UDP de tamanhos variados, respectivamente. O *dnsdrdos* redireciona respostas do tipo *any* do protocolo Sistema de Nomes de Domínios (*Domain Name System* - DNS) consideravelmente grandes, de um servidor DNS para um alvo cujo IP foi forjado (é utilizado o IP da vítima). Já para a execução de ataques de exaustão do serviço, o *Hping3* foi utilizado para gerar ataques *SynFlood*. Por último, para simular conexões de clientes legítimos, foi empregado o *Siege* [53]. Além do *Siege*, um *script* foi criado usando a biblioteca *Urllib3* [54], de modo a simular, por meio de várias *threads*, requisições do tipo *GET*, do protocolo HTTP, a várias páginas de sites *web* não específicos. A configuração detalhada das ferramentas será descrita no próximo tópico.

A fim de tornar os testes mais confiáveis, parâmetros das ferramentas de geração de tráfego de ataque foram alterados ao longo das execuções dos experimentos. A seguir, algumas configurações utilizadas por essas ferramentas, tanto na fase em que o PSO é executado como na fase de classificação:

- *hping3*, durante ataques volumétricos: 300 instâncias gerando 6000 pacotes de 3000 *bytes*



**Figura 5.3: Topologia II**

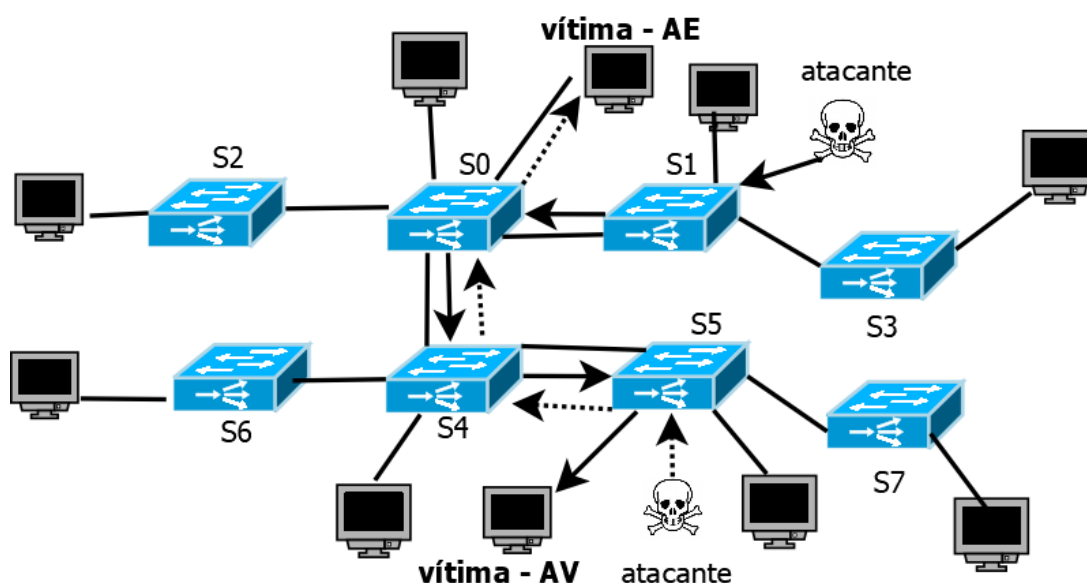
Fonte: Elaborada pela autora.

(forçando fragmentação), no intervalo de 1 segundo; 100 instâncias gerando 2500 pacotes de 2000 *bytes* (forçando fragmentação), no intervalo de 1 segundo; 400 instâncias gerando 400 pacotes de 4000 *bytes* (forçando fragmentação), no intervalo de 1 segundo. Todos os casos utilizaram IPs forjados e tráfego UDP.

- *dnsdrdos*, durante ataques volumétricos: 1000 requisições por iteração, com máximo de 2000 mil iterações. As respostas em retorno às consultas do tipo *any* chegavam a alcançar cerca de 50x o fator de amplificação. Todos os casos utilizaram IPs forjados.
- *hping3*, durante ataques de exaustão: envio especificamente da *flag syn* com 10 instâncias gerando 10000 pacotes de 120 *bytes*, em intervalos de milissegundos; envio especificamente da *flag syn* com 4 instâncias gerando 5000 pacotes de 200 *bytes*, em intervalos de milissegundos; envio especificamente da *flag syn* com 1 instância gerando 6000 pacotes de 120 *bytes*, por segundo; todos os casos utilizaram IPs forjados.

Quanto à configuração do Siege e do *script*, tem-se:

- Siege: simula cerca de 300 conexões concorrentes, durante 5 minutos, com um intervalo entre uma requisição e outra que poderia variar entre 1 e 5 segundos.



**Figura 5.4: Topologia III**

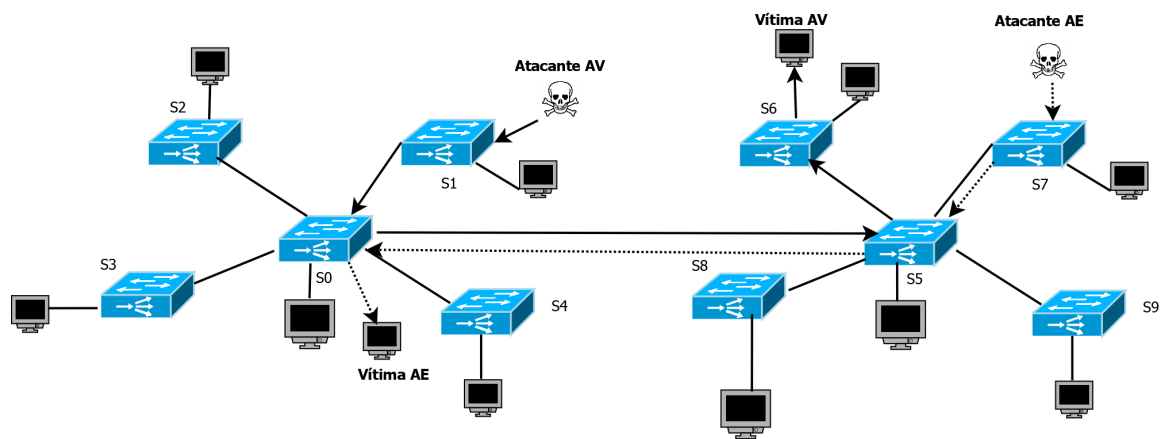
Fonte: Elaborada pela autora.

- *Script*: 100 clientes concorrentes, com cada um abrindo conexões em modo *multithread*, mantendo 6 conexões com o servidor (similar aos navegadores na configuração padrão).

Com o objetivo de evitar padrões durante a detecção dos ataques por parte do FuzDetect, não só os parâmetros dos ataques, topologias e tráfego sofreram alterações ao longo dos testes, procurou-se também alterar o fluxo do tráfego (origem e alvo) por experimento.

Em todas as topologias, o tráfego de ataque volumétrico é representado por direcionadores com linha contínua; já para ataques de exaustão, direcionadores pontilhados; e para tráfego não malicioso, simplesmente uma linha contínua, sem direcionador, conforme Figuras 5.2, 5.3, 5.4 e 5.5. Os atacantes, as vítimas e os clientes legítimos estão devidamente apresentados nas figuras mencionadas.

Assim, na topologia I, Figura 5.2, durante os ataques volumétricos e de exaustão, o tráfego malicioso possuiu como origem o *switch* S1 e como alvo o *switch* S3. O mesmo vale para a topologia II, Figura 5.3. Por sua vez, na topologia III, Figura 5.4, nos experimentos I, II e III (ataques volumétricos), o ataque se origina no *switch* S1, tendo como alvo o *switch* S5. Nos ataques de exaustão, experimentos IV, V e VI, o ataque se origina no *switch* S5, com o *switch* S0 como alvo. Por fim, na topologia IV, Figura 5.5, nos experimentos I, II e III (ataques volumétricos), o ataque se origina no *switch* S1 e o alvo é o *switch* S6. E nos ataques de exaustão, experimentos IV, V e VI, o ataque se origina no *switch* S7 e tem como alvo o *switch* S0.



**Figura 5.5: Topologia IV**  
Fonte: Elaborada pela autora.

# Capítulo 6

## RESULTADOS E DESEMPENHO

---

---

Neste Capítulo será apresentada a motivação acerca da configuração do PSO utilizada, bem como a motivação para o uso de alguns parâmetros. Logo em seguida, serão apresentados detalhes acerca do processo de classificação, em sequência, os resultados gerados por meio do FuzDetect e uma ressalva sobre eles.

Reforça-se que o PSO é utilizado para obter possíveis valores para o estabelecimento dos conjuntos *fuzzy* de forma dinâmica à rede analisada e validando-os em função do tipo de tráfego esperado, uma vez que eles são previamente identificados. Só em seguida, a classificação propriamente dita com tráfego desconhecido pelo FuzDetect será iniciada.

Por último, na Seção 6.4, as análises de performance do FuzDetect, incluindo uma análise sobre a qualidade dos resultados, serão apresentadas também.

### 6.1 Otimização

Para fins de performance e qualidade dos resultados, realizou-se um estudo comparativo com configurações do PSO variando quantidade de partículas e iterações (demais parâmetros como cognição, inicialização da velocidade, parâmetro *weight* e outros foram configurados de acordo com a literatura vigente ([55], [27] [10])), assim como os tipos de conexões com a vizinhança: global e local, em uma topologia de médio porte. Cerca de 10 execuções por configuração foram utilizadas, resultando em um total de 30 execuções do PSO, e, a partir desses dados, extraíram-se médias. O tempo de execução do algoritmo e a qualidade das soluções encontradas foram utilizados como métricas.

Nas Tabelas 6.1 e 6.2 a seguir são apresentadas as médias extraídas por configuração, as quais incluem variações na quantidade de partículas, na quantidade de iterações e por vizi-

nhança. Na Tabela 6.1, são apresentadas as médias pelo tempo de execução, em minutos, em que, quanto menor o tempo, melhor (minimização). Já na Tabela 6.2, as médias são calculadas a partir da qualidade das soluções, por meio da função *fitness*. Aqui, quanto maior o valor do resultado, melhor (maximização). Assim, em busca do equilíbrio entre tempo de execução e qualidade das soluções, optou-se por usar 50 partículas e com 15 iterações, utilizando a vizinhança padrão, global.

É importante salientar que essa escolha de configuração é relativa em função do tamanho da topologia. Quanto menor a mesma for, mais a taxa de acertos é impactante, uma vez que a quantidade de *switches* é menor e os acertos vão de 0% a 100% por *switch*. Já, nos casos em que a topologia for complexa e com mais *switches*, o custo x benefício em sacrificar alguns pontos da taxa de acerto, onde a diminuição da precisão seria mais suave (considerando que a distribuição da perda também seria entre todos os *switches*), por conta da quantidade de *switches*, seja compensado pelo ganho de velocidade na convergência.

**Tabela 6.1: Resultados em tempos de execução (em minutos) com diferentes instâncias do PSO**

PSO GLOBAL	PSO LOCAL
Tempo de Execução	
1º - 20 partículas e 5 iterações	
22.6	25.1
2º - 50 partículas e 15 iterações	
99.3	136
3º - 80 partículas e 15 iterações	
148	212

**Tabela 6.2: Resultados da função *fitness* em diferentes instâncias do PSO**

PSO GLOBAL	PSO LOCAL
<i>Fitness</i> dos melhores resultados encontrados	
1º - 20 partículas e 5 iterações	
1427	1398
2º - 50 partículas e 15 iterações	
1599	1532
3º - 80 partículas e 15 iterações	
1596	1564

## 6.2 Classificação

Trinta e oito experimentos serão apresentados e, em todos eles, o tráfego malicioso e o tráfego legítimo passaram simultaneamente. Os resultados foram divididos por topologia e por experimento, conforme Tabelas 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9 e 6.10. Nessas tabelas, os



resultados estão separados por experimento, podendo ser AV - ataques volumétricos ou AE - ataques de exaustão, e por *switch*. Para cada experimento/*switch*, o valor da variável de saída esperada é destacado (devendo o valor desta ser maior que o das demais, em todos os casos, o que demonstra a precisão do classificador). Por exemplo, na Tabela 6.3, no experimento I (AV), foi esperado que no *switch* S0 o tráfego de maior predominância identificado tenha sido o de ataque volumétrico, e assim ocorreu, com 75% de chances. Já no experimento III (AE) da mesma tabela, esperou-se que em S1 o tráfego de ataque por exaustão tenha sido o de maior predominância, e assim ocorreu, com 76% de chance de acontecer.

É importante ressaltar que, nos resultados apresentados nas Tabelas 6.3, 6.4, 6.7 e 6.8, referentes às topologias I, II, III e IV, respectivamente, a limitação dos conjuntos *fuzzy* foi configurada de modo estático, empiricamente, sem o auxílio do PSO. Já nos resultados apresentados nas Tabelas 6.5, 6.6, 6.9 e 6.10, também referentes às topologias I, II, III e IV, respectivamente, foi de forma dinâmica pelo PSO, a fim de demonstrar quão mais eficiente e precisa é a classificação, quando os conjuntos *fuzzy* são configurados previamente, pelo otimizador.

Para verificar o fluxo do tráfego por topologia, consultar a Seção 5.3.

**Tabela 6.3: Resultados - Topologia I - Sem PSO - Experimentos I, II, III, IV**

Topologia I						
Experimento I - AV			Experimento II - AV			Switch
AV	AE	TNM	AV	AE	TNM	
<b>75%</b>	50%	24.3%	<b>76%</b>	28%	40%	S0
<b>76%</b>	50%	22%	<b>74%</b>	50%	24%	S1
<b>57%</b>	42.9%	38.2%	<b>70%</b>	36.9%	40.5%	S2
<b>60%</b>	40.9%	40%	<b>75%</b>	35%	50%	S3
Experimento III - AE			Experimento IV - AE			
AV	AE	TNM	AV	AE	TNM	
40%	<b>78%</b>	47%	50%	<b>70%</b>	24%	S0
23%	<b>76%</b>	50%	22%	<b>77%</b>	50%	S1
56%	<b>42.9%</b>	57%	25.9%	<b>35%</b>	70%	S2
50%	<b>38%</b>	79%	45%	<b>60%</b>	77%	S3

## 6.3 Resultados

De acordo com os resultados apresentados nas Tabelas 6.3, 6.4, 6.7 e 6.8, por meio dos valores destacados nas variáveis de saída, onde nestes resultados a configuração dos conjuntos se manteve estática, sem o uso do PSO, houve erros quanto a falsos negativos e falsos positivos, em que ali passou tráfego de ataque durante a coleta, mas que foi identificado com maior chance de ser legítimo e tráfego legítimo identificado com maior chance de ser de ataque, res-

Tabela 6.4: Resultados - Topologia II - Sem PSO - Experimentos I, II, III, IV

Topologia II						
Experimento I - AV			Experimento II - AV			Switch
AV	AE	TNM	AV	AE	TNM	
<b>78%</b>	60%	40%	<b>74%</b>	50%	24%	S0
<b>75%</b>	50%	24%	<b>76%</b>	50%	23%	S1
50%	42.9%	<b>60%</b>	40%	40%	<b>55%</b>	S2
<b>38%</b>	40%	60%	<b>55%</b>	50%	70%	S3
55%	40.9%	<b>78%</b>	48%	55%	<b>80%</b>	S4
Experimento III - AE			Experimento IV - AE			
AV	AE	TNM	AV	AE	TNM	
50%	<b>70%</b>	43%	48%	<b>80%</b>	20%	S0
23%	<b>76%</b>	40%	21%	<b>77%</b>	50%	S1
56%	42.9%	<b>57%</b>	25.9%	35%	<b>70%</b>	S2
42%	<b>36%</b>	70%	47%	<b>60%</b>	77.6%	S3
48%	30%	<b>85%</b>	50%	55%	<b>70%</b>	S4

Tabela 6.5: Resultados - Topologia I - Com PSO - Experimentos I, II, III, IV

Topologia I						
Experimento I - AV			Experimento II - AV			Switch
AV	AE	TNM	AV	AE	TNM	
<b>74.6%</b>	24.8%	24.9%	<b>74%</b>	25.4%	26.5%	S0
<b>17.6%</b>	81.7%	50%	<b>74%</b>	25.4%	26.4%	S1
<b>75.3%</b>	24.0%	24.1%	<b>74.3%</b>	25.1%	25.8%	S2
<b>74%</b>	50%	50%	<b>74.7%</b>	50%	50%	S3
Experimento III - AE			Experimento IV - AE			
AV	AE	TNM	AV	AE	TNM	
17.7%	<b>81.5%</b>	50%	18.9%	<b>80.4%</b>	50%	S0
16.8%	<b>82.4%</b>	50%	16.8%	<b>82.4%</b>	50%	S1
22.5%	<b>76.8%</b>	50%	22.3%	<b>77%</b>	50%	S2
50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	50%	S3

pectivamente (é destacado em negrito o valor aproximado por tráfego esperado). Isso pode ser visto, de forma mais específica, na Tabela 6.3, topologia I, experimentos III e IV, *switches* S2 e S3; durante os experimentos I, II, III e IV da topologia II, *switch* S3, conforme Tabela 6.4; nos experimentos I, II, III e IV da topologia III, *switches* S1, S3, S4, S5 e S7, conforme Tabela 6.7 e nos experimentos I, III e VI da topologia IV, *switches* S0, S1, S6, S8 e S9, conforme Tabela 6.8.

Para acompanhar o fluxo do tráfego nos *switches*, durante os experimentos, consultar Figuras 5.2, 5.3, 5.4 e 5.5.

Já em relação aos resultados apresentados nas Tabelas 6.5, 6.6, 6.9 e 6.10, nas quais houve configuração prévia dos conjuntos *fuzzy* de forma adaptada ao tráfego da rede por meio do PSO,

**Tabela 6.6: Resultados - Topologia II - Com PSO - Experimentos I, II, III, IV**

Topologia II						
Experimento I - AV			Experimento II - AV			Switch
AV	AE	TNM	AV	AE	TNM	
<b>72.8%</b>	26.6%	29.4%	<b>68.7%</b>	30.7%	38.5%	S0
<b>80.8%</b>	18.4%	49.5%	<b>72.2%</b>	27.2%	32.8%	S1
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S2
<b>73.6%</b>	25.8%	28.2%	<b>74.7%</b>	24.7%	50%	S3
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S4
Experimento III - AE			Experimento IV - AE			
AV	AE	TNM	AV	AE	TNM	
28.7%	<b>70.3%</b>	46.7%	33.5%	<b>65.5%</b>	45.7%	S0
25.2%	<b>73.7%</b>	46.8%	23.7%	<b>75.4%</b>	47.5%	S1
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S2
23.9%	<b>75.1%</b>	47.4%	24.6%	<b>74.8%</b>	50%	S3
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S4

os resultados foram mais precisos em todos os casos, exceto um caso isolado, na Tabela 6.5, experimento I, *switch* S1, em que o ataque volumétrico foi identificado como ataque de exaustão (de toda forma, não deixou de identificar o tráfego como de ataque). É possível verificar que o tráfego de fato gerado foi o identificado com mais notabilidade em todas as outras classificações, nos *switches* dos demais experimentos das Tabelas previamente citadas. Logo, exclusivamente os *switches* que receberam tráfego de ataque, foram identificados com tráfego malicioso durante a execução do classificador, o mesmo também vale para os *switches* que não fizeram parte do fluxo de ataque e que tiverem o tráfego não malicioso com maior destaque, assim, nenhum *switch*, de forma exclusiva com tráfego legítimo, durante tais experimentos, foi identificado como participante de fluxo de ataque, o que configuraria falso positivo, demonstrando, assim, a capacidade do FuzDetect em identificar com mais assertividade, quando executado com o auxílio do otimizador, o tráfego e classificá-lo.

Para acompanhar o fluxo do tráfego nos *switches* quanto a estes resultados, também consultar Figuras 5.2, 5.3, 5.4 e 5.5.

Na Seção 6.4 será apresentado o impacto desses resultados a nível de desempenho, por meio de valores estatísticos.

Reforça-se, a partir do exposto, que, inclusive para quando as topologias são mais simples ou mais complexas, caso mudanças mesmo que pouco significativas possam ocorrer no tráfego, corre-se o risco de não serem perceptíveis quando a configuração dos conjuntos *fuzzy* não se adéqua à rede envolvida nem às suas demandas, por meio da otimização.

É possível verificar também que não existe um padrão numérico proporcional em relação

Tabela 6.7: Resultados - Topologia III - Sem PSO - Experimentos I, II, III, IV, V e VI

Topologia III						
Experimento I - AV			Experimento II - AV			Switch
AV	AE	TNM	AV	AE	TNM	
<b>74.5%</b>	24.9%	25.2%	<b>73.8%</b>	25.6%	26.9%	S0
<b>24.0%</b>	75.4%	50%	<b>74.2%</b>	25.2%	26.1%	S1
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S2
50%	50%	<b>74.7%</b>	24%	75.5%	<b>50%</b>	S3
<b>74.9%</b>	24.5%	24.6%	<b>50%</b>	50%	74.7%	S4
<b>75%</b>	24.4%	24.6%	<b>50%</b>	50%	74.7%	S5
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S6
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S7
Experimento IV - AE			Experimento V - AE			
AV	AE	TNM	AV	AE	TNM	
17.3%	<b>81.9%</b>	50%	24.6%	<b>74.8%</b>	53%	S0
50%	50%	<b>74%</b>	74%	25.4%	<b>26.4%</b>	S1
50%	50%	<b>74%</b>	50%	50%	<b>74%</b>	S2
50%	50%	<b>74%</b>	50%	50%	<b>74%</b>	S3
16.8%	<b>82.5%</b>	50%	18.4%	<b>80.9%</b>	50%	S4
16%	<b>82.6%</b>	50%	18.6%	<b>80.7%</b>	50%	S5
50%	50%	<b>74%</b>	50%	50%	<b>74%</b>	S6
23.7%	75.7%	<b>50%</b>	50%	50%	<b>74%</b>	S7

a taxa de classificação entre os tráfegos por *switch* e nem por amostra. Independente de ser o tráfego esperado ou não, com PSO ou não. Por exemplo, na topologia III, Tabela 6.3, de acordo com a Figura 5.2, nos *switches* S0 e S1, percebe-se que as taxas para o tráfego não malicioso (TNM) possuem variações altas entre si, isso ocorre pois a quantidade de clientes legítimos mudava por geração de tráfego. O mesmo vale para os ataques (como já mencionado anteriormente, acerca de suas variações por meio de parametrizações). De qualquer forma, mesmo tal variação ocorrendo pra mais ou pra menos por tráfego, a classificação aliada a otimização apresentou bom resultado, isso provavelmente se deve ao fato que além da otimização, as regras utilizadas, conforme Tabela 4.3, utilizam variáveis estabelecidas através de funções estatísticas que não discriminam a quantidade por si só de determinado dado, mas alguns desvios do mesmo ao longo do tráfego, por meio da mediana e variância, por exemplo.

## 6.4 Performance FuzDetect

No ambiente onde o FuzDetect funciona, nas etapas de coleta, extração/inserção na base, classificação e otimização, não foram apresentados carga ou consumo de memória que saturassem as máquinas virtuais envolvidas, tanto em topologias menores como nas mais complexas, conforme as Figuras 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7 e 6.8. De acordo com essas figuras, o tempo

**Tabela 6.8: Resultados - Topologia IV - Sem PSO - Experimentos I, II, III, IV, V e VI**

Topologia IV									
Experimento I - AV			Experimento II - AV			Experimento III - AV			Switch
AV	AE	TNM	AV	AE	TNM	AV	AE	TNM	
<b>22.5%</b>	76.9%	50%	<b>74.6%</b>	24.8%	25.3%	<b>24.3%</b>	75.1%	50%	S0
<b>19.6%</b>	79.7%	50%	<b>74.3%</b>	25.1%	26%	<b>21.9%</b>	77.4%	50%	S1
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S2
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S3
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S4
<b>74.6%</b>	24.8%	25.01%	<b>74.5%</b>	24.8%	25.3%	<b>74.4%</b>	25%	25.6%	S5
<b>74.6%</b>	24.8%	25.1%	<b>74.7%</b>	24.7%	25.1%	<b>50%</b>	50%	74.7%	S6
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S7
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S8
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S9
Experimento IV - AE			Experimento V - AE			Experimento VI - AE			
AV	AE	TNM	AV	AE	TNM	AV	AE	TNM	
20.8%	<b>78.5%</b>	50%	19.5%	<b>79.8%</b>	50%	20.7%	<b>78.6%</b>	50%	S0
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S1
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S2
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S3
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S4
20.1%	<b>79.2%</b>	50%	17.7%	<b>81.5%</b>	50%	16.8%	<b>82.4%</b>	50%	S5
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	S6
19%	<b>80.3%</b>	50%	18.0%	<b>81.3%</b>	50%	16.9%	<b>82.3%</b>	50%	S7
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	24.8%	74.6%	<b>50.1%</b>	S8
50%	50%	<b>74.7%</b>	50%	50%	<b>74.7%</b>	24.3%	75.1%	<b>50%</b>	S9

de processamento calculado corresponde a todo o tempo que o processo passou em execução ao longo de sua existência, em percentual (salientando que o ápice de processamento é de 400%, uma vez que as máquinas onde os módulos eram executados possuem 4 cores, cada uma). Quanto ao consumo de memória, este é representado pela relação entre a memória residente do processo e a memória total do sistema operacional, também em percentual.

Durante a classificação, o tempo em processamento do processo foi alto, porém descendente ao longo da execução, tanto em topologias mais simples como nas maiores, conforme Figuras 6.1 e 6.2, respectivamente. Durante a execução do PSO, entretanto, o pico de tempo em processamento chegou ao máximo de 30%, bem menor do que na classificação, como pode ser visto nas Figuras 6.3 e 6.4, separadas por porte de topologia. Já nas etapas de coleta e extração/armazenamento, o tempo em processamento não ultrapassou 26% da vida útil do processo, para ambos os tipos de topologias: simples e complexa, de acordo com as Figuras 6.5, 6.6, 6.7 e 6.8, respectivamente.

Com relação à memória, na classificação, o consumo foi maior do que nas outras etapas

Tabela 6.9: Resultados - Topologia III - Com PSO - Experimentos I, II, III, IV, V e VI

Topologia III									
Experimento I - AV			Experimento II - AV			Experimento III - AV			Switch
AV	AE	TNM	AV	AE	TNM	AV	AE	TNM	
<b>64.5%</b>	31.8%	53%	<b>74.7%</b>	24.7%	24%	<b>71.7%</b>	27.6%	35.3%	S0
<b>74.7%</b>	24.7%	24.7%	<b>74%</b>	24%	24%	<b>73.3%</b>	26%	34%	S1
57%	42.9%	<b>59.2%</b>	50%	36.9%	<b>62.5%</b>	50%	50%	<b>75%</b>	S2
50%	50%	<b>59.2%</b>	57.5%	42.1%	<b>59.2%</b>	50%	50%	<b>75.7%</b>	S3
<b>74.4%</b>	24.7%	24.1%	<b>74.7%</b>	25%	24%	<b>59.4%</b>	40.1%	54.4%	S4
<b>73%</b>	22%	53.7%	<b>76.6%</b>	19%	52.9%	<b>68%</b>	31.3%	42.6%	S5
56.1%	43.5%	<b>62%</b>	50%	50%	<b>76.1%</b>	50%	50%	<b>76.3%</b>	S6
44.4%	36.1%	<b>63.3%</b>	50%	50%	<b>75.1%</b>	50%	50%	<b>75.6%</b>	S7
Experimento IV - AE			Experimento V - AE			Experimento VI - AE			
AV	AE	TNM	AV	AE	TNM	AV	AE	TNM	
24.7%	<b>74.7%</b>	50%	24.7%	<b>74.7%</b>	50%	24.7%	<b>74.4%</b>	50.3%	S0
25%	18%	<b>76%</b>	30%	16%	<b>77%</b>	20%	25%	<b>78%</b>	S1
24.75%	50%	<b>74%</b>	25%	50%	<b>75%</b>	22%	50.7%	<b>76%</b>	S2
28%	17%	<b>78.7%</b>	30%	18%	<b>78%</b>	35%	20%	<b>79%</b>	S3
19.09%	<b>80.2%</b>	50%	20.1%	<b>79.2%</b>	50%	19.3%	<b>80%</b>	50%	S4
16.6%	<b>82.6%</b>	50%	16.6%	<b>82.6%</b>	50%	16.6%	<b>82.4%</b>	50%	S5
20%	50%	<b>86%</b>	25%	50%	<b>80%</b>	29%	50%	<b>70%</b>	S6
24.75%	50%	<b>74.74%</b>	28.3%	50%	<b>75%</b>	24.7%	50%	<b>74.7%</b>	S7

(máximo de 9.2%), mas não trouxe saturação a ela. O consumo em percentual de memória do classificador, por tamanho da topologia, pode ser visualizado nas Figuras 6.1 e 6.2. O comportamento com relação à memória foi similar durante a otimização, conforme Figuras 6.3 e 6.4. Tal consumo foi ainda menor nas etapas de coleta e extração/inserção, como pode ser visto nas Figuras 6.5, 6.6, 6.7 e 6.8, também separados por tipo de topologia.

Na Tabela 6.11, é apresentado o tempo de execução do otimizador em uma topologia de pequeno porte e em uma topologia de grande porte. Embora a otimização leve uma quantidade de tempo considerável, ela só ocorre uma vez, antes de o classificador ser iniciado. Além disso, mesmo a quantidade de *switches* aumentando por conta de mudanças de topologia, ou o impacto de algumas instâncias de ataque sendo maiores que outras, a janela de detecção de ataque, levando-se em consideração as etapas de: coleta, extração/armazenamento e classificação, no pior caso (topologias de maior porte), foi de aproximadamente 6.16 minutos. É possível verificar o tempo tomado pelas etapas de coleta, extração/armazenamento e classificação durante as topologias menores e maiores, respectivamente, nas Tabelas 6.12, 6.13 e 6.14. Importante destacar ainda que a detecção propriamente dita, sem considerar a coleta, a extração e o armazenamento, levou aproximadamente 1 minuto, no pior caso.

Uma das motivações acerca do uso de PSO, como afirmam Shi e Eberhart [10], é que

**Tabela 6.10: Resultados - Topologia IV - Com PSO - Experimentos I, II, III, IV, V e VI**

Topologia IV									
Experimento I - AV			Experimento II - AV			Experimento III - AV			Switch
AV	AE	TNM	AV	AE	TNM	AV	AE	TNM	
<b>72.7%</b>	26.6%	32.34%	<b>71.27%</b>	28.1%	36.5%	<b>71.4%</b>	27.9%	34.2%	S0
<b>75.2%</b>	24.2%	24%	<b>76.6%</b>	50%	50%	<b>52%</b>	49%	37%	S1
50%	49.8%	<b>76.3%</b>	57.2%	42.4%	<b>76.7%</b>	53%	46%	<b>68%</b>	S2
38.8%	30.2%	<b>69.7%</b>	55%	39%	<b>59.6%</b>	40%	31.4%	<b>68.0%</b>	S3
50%	50%	<b>76.3%</b>	56.3%	43.3%	<b>61.3%</b>	50%	50%	<b>75.5%</b>	S4
<b>71.8%</b>	27.5%	35.8%	<b>72.6%</b>	26.8%	32.3%	<b>70%</b>	29.4%	36.8%	S5
<b>70.8%</b>	47.5%	52.3%	<b>52%</b>	49%	37%	<b>75.2%</b>	24.2%	24.2%	S6
55.1%	39.6%	<b>59.9%</b>	53%	46.5%	<b>67.8%</b>	50%	50%	<b>76.4%</b>	S7
55.2%	39.7%	<b>59.8%</b>	40%	31.5%	<b>67.9%</b>	42.3%	32.3%	<b>67%</b>	S8
54.6%	45.1%	<b>64.9%</b>	50%	50%	<b>76.1%</b>	50%	50%	<b>75.7%</b>	S9
Experimento IV - AE			Experimento V - AE			Experimento VI - AE			
AV	AE	TNM	AV	AE	TNM	AV	AE	TNM	
24.7%	<b>74.2%</b>	50.4%	24.7%	<b>74.6%</b>	50.1%	24.7%	<b>70%</b>	53%	S0
20%	50%	<b>77%</b>	24.7%	24.7%	<b>74.7%</b>	50%	28%	<b>75%</b>	S1
24.7%	20%	<b>74.7%</b>	20%	28.3%	<b>71%</b>	50%	29%	<b>70%</b>	S2
18%	40%	<b>80%</b>	28%	50%	<b>82%</b>	28%	50%	<b>86%</b>	S3
50%	25.7%	<b>75%</b>	24%	24%	<b>74.7%</b>	42%	50%	<b>70.3%</b>	S4
17.1%	<b>82.2%</b>	50%	17%	<b>82.3%</b>	50%	16.7%	<b>82.6%</b>	50%	S5
50%	26%	<b>85%</b>	50%	22%	<b>80%</b>	50%	28%	<b>77%</b>	S6
16.6%	<b>82.6%</b>	50%	16.6%	<b>82.6%</b>	50%	16.6%	<b>82.6%</b>	50%	S7
20%	50%	<b>75%</b>	22%	50%	<b>75%</b>	22%	28%	<b>74%</b>	S8
22%	50%	<b>76%</b>	50%	50%	<b>76%</b>	17%	40%	<b>76.4%</b>	S9

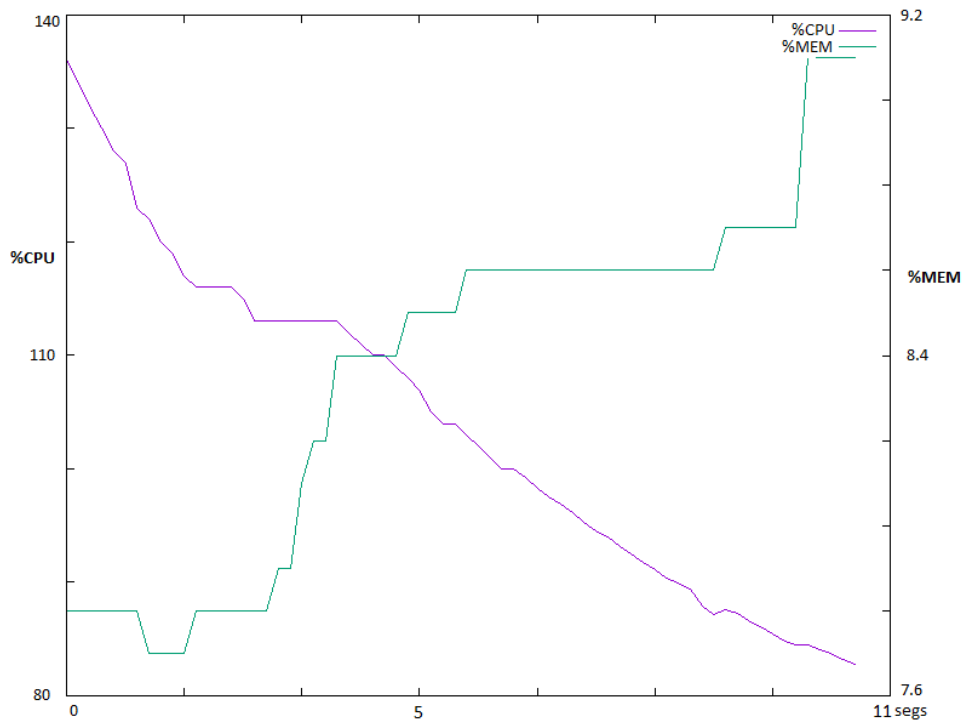
diferentemente das operações de mutação, totalmente aleatórias em versões tradicionais de Algoritmos Genéticos, em PSO, o voo da partícula, em função da velocidade, acontece de forma mais consciente e conseqüentemente mais rápida (em direção a uma melhor área).

**Tabela 6.11: Resultados em tempos de execução do otimizador por porte da topologia**

TEMPO DE EXECUÇÃO PSO (em minutos)	
TOPOLOGIA DE PEQUENO PORTE	84
TOPOLOGIA DE MAIOR PORTE	127

Com o objetivo de mensurar a qualidade dos resultados, apresentam-se os resultados de *precision* e *recall* [56], por topologia, sendo que *precision* representa a relação entre a quantidade de amostras identificadas como ataque pelo FuzDetect e, dentre estas, as que de fato são de ataque; e *recall* corresponde à taxa de acertos entre as amostras identificadas como de ataque e o total de amostras de fato de ataque.

Nas Equações 6.1 e 6.2 a seguir, são apresentados, respectivamente, os cálculos de *Precision* e os de *Recall*, de modo que: TP (*True Positive*) representa a taxa de ataques corretamente



**Figura 6.1:** Tempo em processamento e consumo da memória do classificador em topologias menores

Fonte: Elaborada pela autora.

**Tabela 6.12:** Resultados em tempos de execução da coleta de tráfego por porte da topologia

TEMPO DE EXECUÇÃO DA COLETA (em minutos)	
TOPOLOGIA DE PEQUENO PORTE	3.5
TOPOLOGIA DE MAIOR PORTE	5

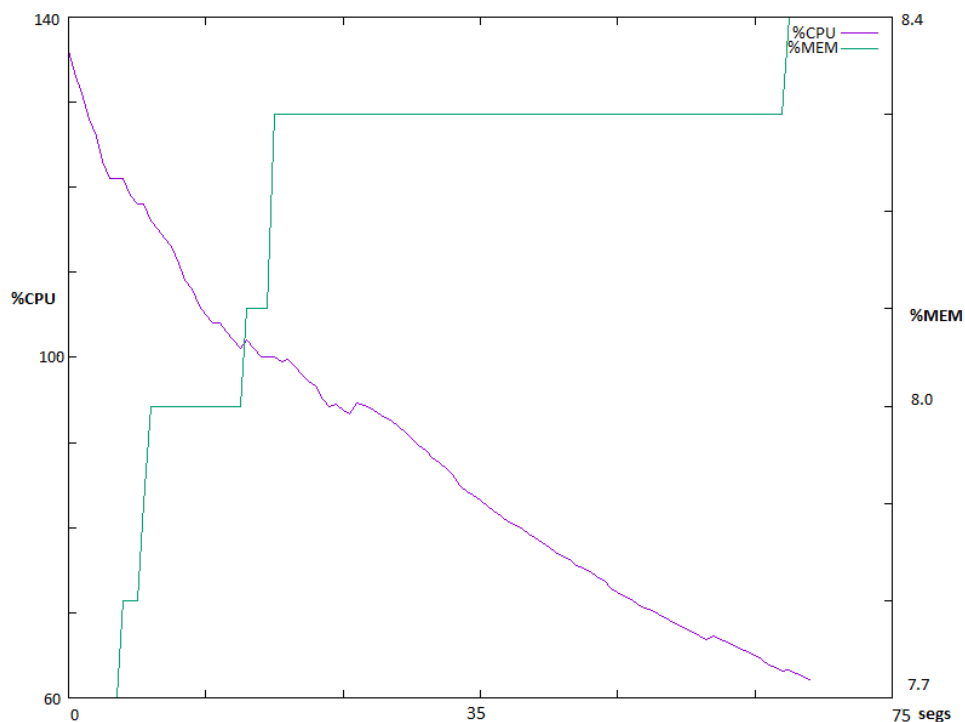
detectados pelo FuzDetect, FP (*False Positive*) é a taxa de ataques erroneamente classificados pelo FuzDetect e FN (*False Negative*) corresponde à taxa de ataques, do total de amostras geradas, que não foram detectados pelo FuzDetect.

$$Precision = \frac{TP}{TP + FP} \quad (6.1)$$

$$Recall = \frac{TP}{TP + FN} \quad (6.2)$$

Nesse sentido, *Precision* define quão assertivo é o resultado levando em conta o tráfego erroneamente identificado como de ataque (falso positivo). *Recall*, por seu turno, quão completo o resultado é de acordo com a taxa de falsos negativos, tráfego de ataque identificado erroneamente como não malicioso (falso negativo). Os resultados se encontram na Tabela 6.15 e estão separados por topologia e com/sem otimização. É possível verificar que, nas topologias I, II, III





**Figura 6.2:** Tempo em processamento e consumo da memória do classificador em topologias maiores

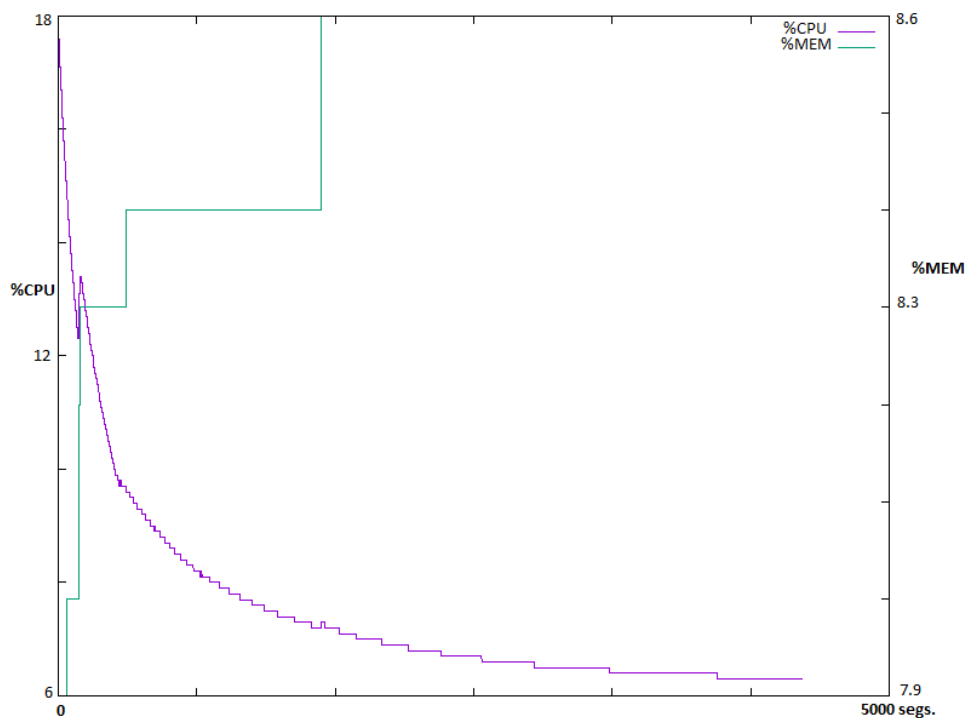
Fonte: Elaborada pela autora.

**Tabela 6.13:** Resultados em tempos de execução da extração e armazenamento dos dados por porte da topologia

TEMPO DE EXECUÇÃO DA EXTRAÇÃO (em segundos)	
TOPOLOGIA DE PEQUENO PORTE	2
TOPOLOGIA DE MAIOR PORTE	3.5

e IV, quando a otimização não foi utilizada, a quantidade de falsos negativos e falsos positivos foi significativa e influenciou nos resultados de *Precision* e *Recall*.

Vale salientar que, os falsos positivos também foram contabilizados a partir de classificações inconsistentes em que o tráfego de determinado tipo de ataque foi identificado como de outro tipo de ataque. Por exemplo, tráfego de ataque volumétrico identificado como ataque por exaustão.



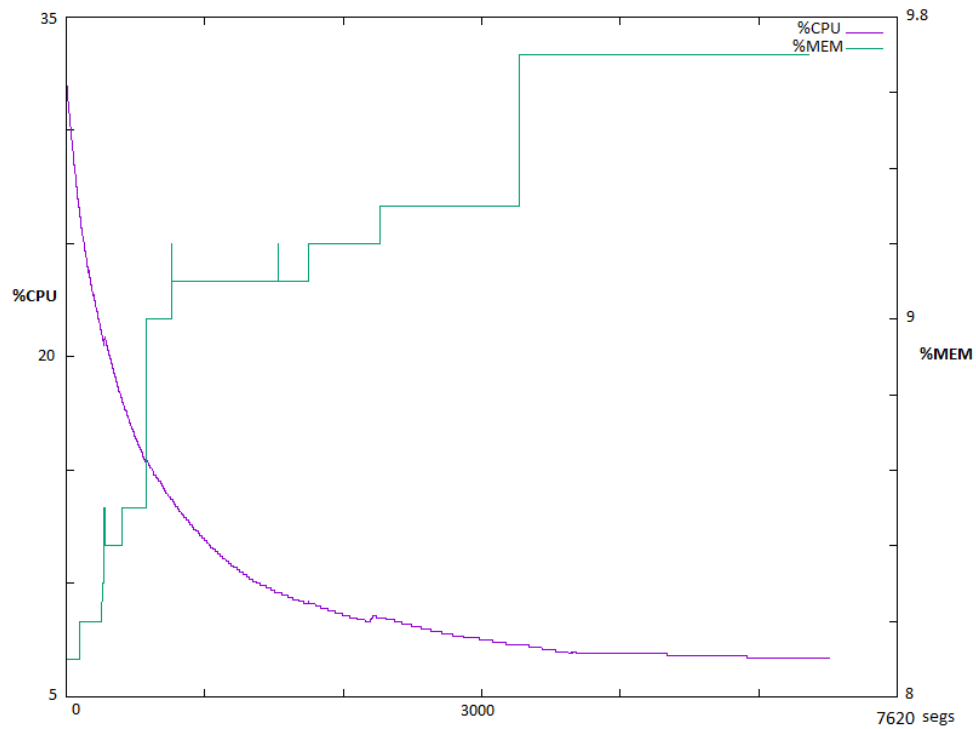
**Figura 6.3:** Tempo em processamento e consumo da memória do PSO em topologias menores  
 Fonte: Elaborada pela autora.

**Tabela 6.14:** Resultados em tempos de execução da classificação por porte da topologia

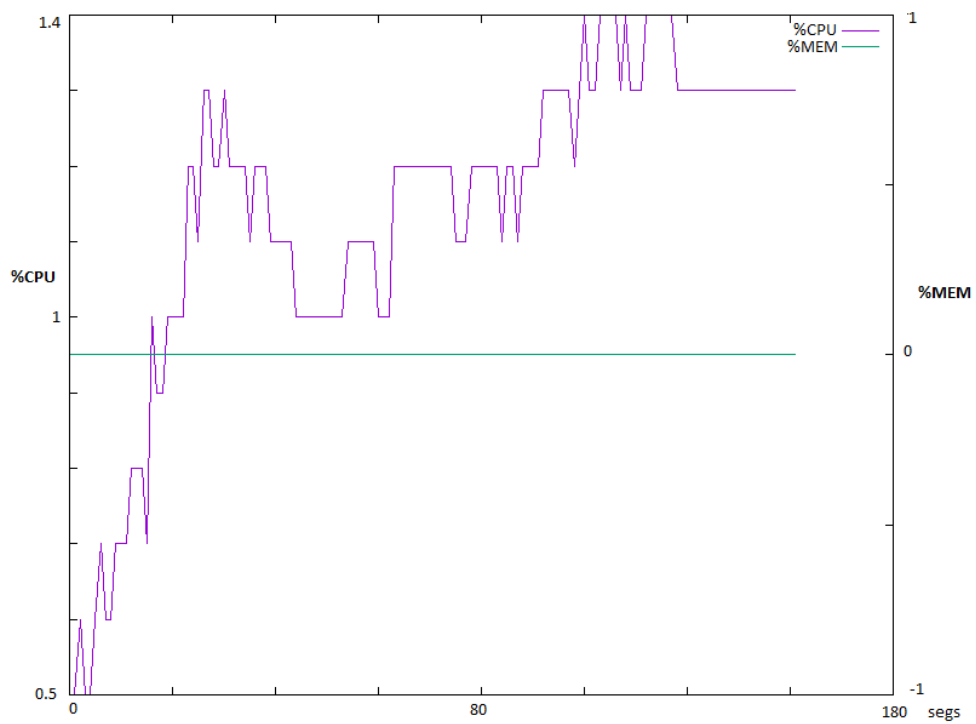
TEMPO DE EXECUÇÃO DA CLASSIFICAÇÃO (em segundos)	
TOPOLOGIA DE PEQUENO PORTE	6
TOPOLOGIA DE MAIOR PORTE	73

**Tabela 6.15:** Precision e Recall por topologia

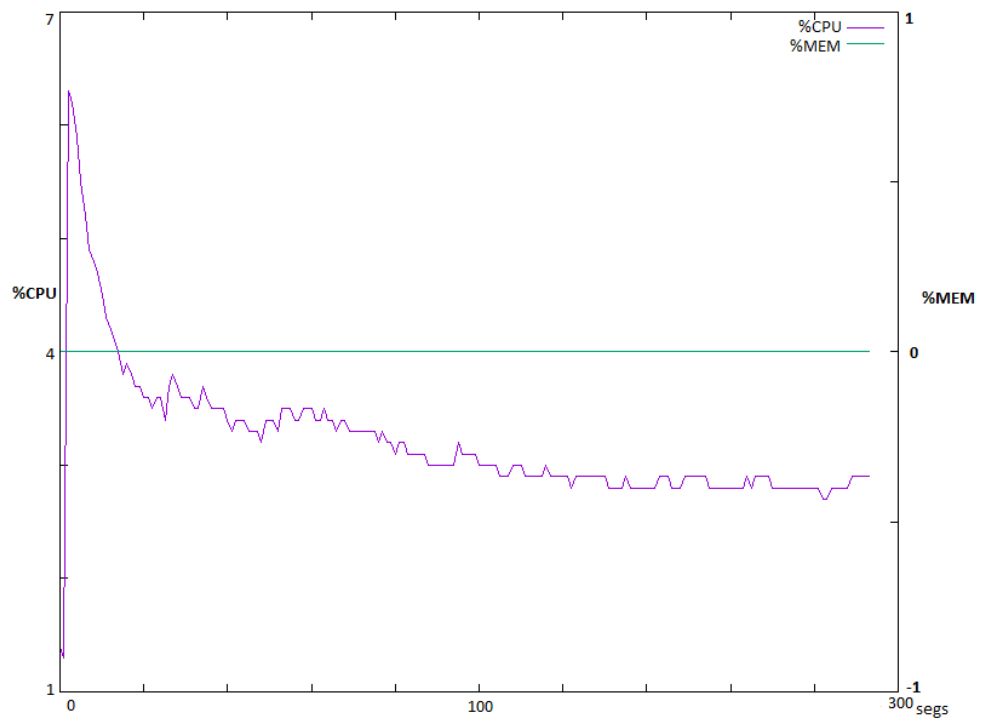
Topologia	Precision	Recall
<b>I - Sem PSO</b>	100%	75%
<b>II - Sem PSO</b>	100%	66%
<b>III - Sem PSO</b>	73%	84%
<b>IV - Sem PSO</b>	72%	94%
<b>I - Com PSO</b>	93%	100%
<b>II - Com PSO</b>	100%	100%
<b>III - Com PSO</b>	100%	100%
<b>IV - Com PSO</b>	100%	100%



**Figura 6.4:** Tempo em processamento e consumo da memória do PSO em topologias maiores  
Fonte: Elaborada pela autora.

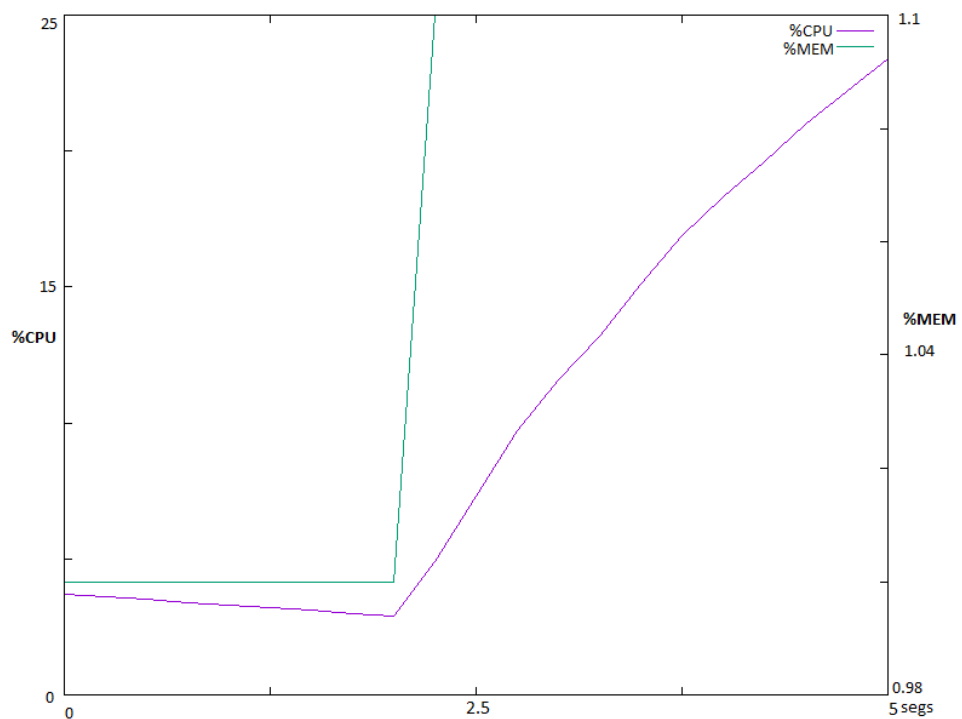


**Figura 6.5:** Tempo em processamento e consumo da memória do FuzDetect durante a coleta de tráfego em topologias menores  
Fonte: Elaborada pela autora.



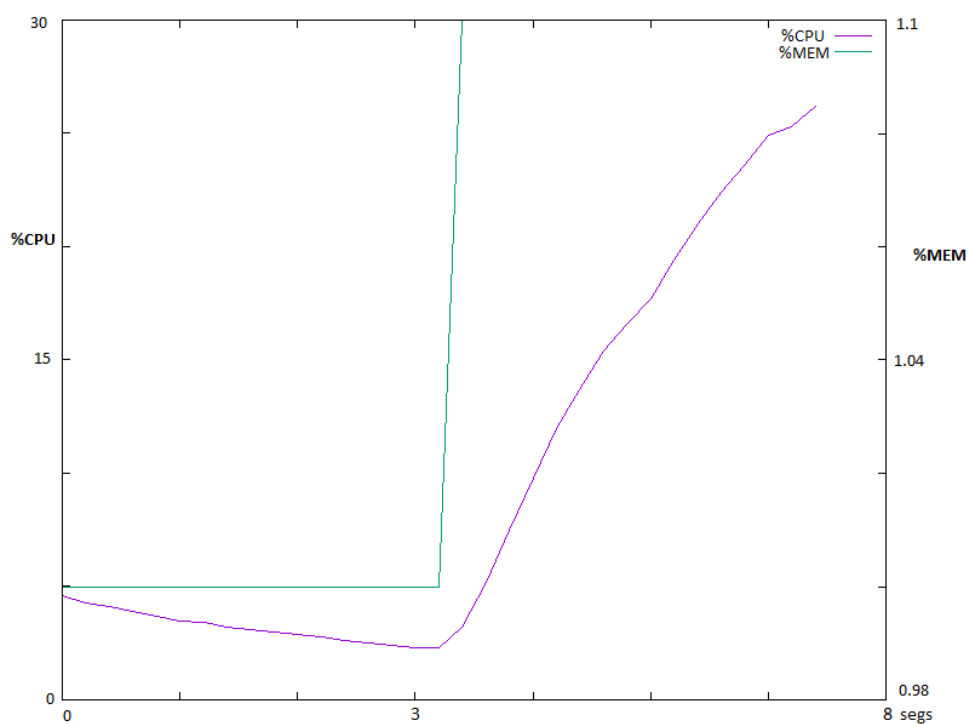
**Figura 6.6:** Tempo em processamento e consumo da memória do FuzDetect durante a coleta de tráfego em topologias maiores

Fonte: Elaborada pela autora.



**Figura 6.7:** Tempo em processamento e consumo da memória do FuzDetect durante extração e armazenamento dos dados de tráfego em topologias menores

Fonte: Elaborada pela autora.



**Figura 6.8:** Tempo em processamento e consumo da memória do FuzDetect durante extração e armazenamento dos dados de tráfego em topologias maiores

Fonte: Elaborada pela autora.

# Capítulo 7

## CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

---

---

Com o crescente aumento no número de dispositivos conectados à Internet, o que deu origem ao termo IoT, ficou bastante acessível a possibilidade em se montar e implementar um ataque DDoS por meio desses equipamentos, visto que boa parte deles não possuem nenhum aparato de segurança em sua implementação.

Os ataques DDoS podem ser categorizados de acordo com a sua natureza, como: **Ataques à Infraestrutura**, geralmente focados em saturação de banda ou esgotamento de *link* (amplificações UDP é um exemplo); e **Ataques à Camada de Aplicação**, mais complexos, pois buscam atacar alguma vulnerabilidade específica da aplicação, podendo ocasionar vazamento de dados, por exemplo.

Nesse contexto, o presente trabalho propõe a criação do FuzDetect, um sistema de detecção de ataques DDoS que é capaz de definir ataques à infraestrutura em duas vertentes: os volumétricos, cujo objetivo principal é o esgotamento de *link* e a capacidade de tráfego da interface, e ataques de exaustão de *hardware*, capazes de causar estouro de memória, esgotamento de *threads* ou sessões a nível de aplicação. O FuzDetect também é capaz de se adaptar ao tráfego da rede alvo, tornando, dessa forma, o classificador mais eficiente.

No decorrer da apresentação dos resultados, é possível verificar que, quanto às topologias menores, embora mais simples, com o uso dos limitadores dos conjuntos configurados de forma estática, o classificador não apresentou eficiência máxima, uma vez que alguns falsos negativos foram gerados. Nas topologias mais complexas, por sua vez, em que o otimizador foi executado previamente, o classificador se adaptou bem às mudanças de tráfego ao longo da execução das ferramentas de geração de tráfego, ressaltando, dessa forma, a capacidade do sistema em

identificar os subtipos do tráfego de ataque, corretamente.

A respeito dos trabalhos futuros, embora tendo sido descrito, na Seção 6.4, que, ao longo dos experimentos, o FuzDetect se manteve dentro de uma janela de resposta esperada, quanto ao nível de eficiência da sua proposta e quanto ao nível de performance, é importante acrescentar a necessidade de um provável refinamento nas inteligências artificiais envolvidas no FuzDetect no futuro, a fim de prover a detecção de outras vertentes de ataque e alguns *tunings* em parâmetros do PSO, para torná-lo mais rápido.

Além disso, a implantação de um sistema de mitigação aliado ao FuzDetect é um projeto que já vem ganhando espaço, conforme o trabalho apresentado por Corrêa et al. [57] e que deve usufruir do grande benefício desta solução: a sua capacidade de definir ataques. Logo, diante do que foi apresentado, é provável que, mais à frente, possa-se também testar o FuzDetect em redes não virtualizadas e ao menos de médio porte, aumentando ainda mais a sua credibilidade.

## REFERÊNCIAS

---

---

- [1] HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP. *Internet of Things - IoT*. 2018. Disponível em: <<https://www.hpe.com/br/pt/what-is/internet-of-things.html>>. Acesso em: 10 mar. 2018.
- [2] GUBBI, J., BUYYA, R., MARUSIC, S., & PALANISWAMI, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7), p. 1645-1660, 2013.
- [3] ARBOR NETWORKS. *Ddos attacks in 2017 activities*. 2018. Disponível em: <<https://www.arbornetworks.com/blog/insight/2017-ddos-attack-activity/>>. Acesso em: 25 ago. 2018.
- [4] GLOBAL DOTS. *DDoS – Distributed Denial of Service Explained*. 2017. Disponível em: <<https://www.globaldots.com/ddos-distributed-denial-service-explained/>>. Acesso em: 15 jul. 2018.
- [5] MIRKOVIC, J.; REITHER, P. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, Nova Iorque, EUA, 34(2), p. 39-53, 2004.
- [6] CERTBR. *Ataques Distribuídos de Negação de Serviço (DDoS)*. 2016. Disponível em: <<https://www.cert.br/docs/whitepapers/ddos>>. Acesso em: 2 abr. 2018.
- [7] BRAGA, R.; MOTA, E.; PASSITO, A. Lightweight DDoS Flooding Attack Detection using Nox/Openflow, *Local Computer Networks (LCN)*, IEEE 35th Conference, Denver, CO, EUA, p. 408-415, out., 2010.
- [8] ROSS, T. J. *Fuzzy Logic: With Engineering Applications*. 3. ed. West Sussex, United Kingdom: WILEY, 2010.
- [9] COSTA, A. D.; RODRÍGUEZ, A.; SIMAS, E.; ARAÚJO, R. Lógica Fuzzy: conceitos e aplicações. *Workshop de Software Livre*, 4, 2003.
- [10] SHI, Y.; EBERHART, R. C. Empirical study of particle swarm optimization. *In Evolutionary computation, 1999*. CEC 99. Proceedings of the 1999 Congress on 3, p. 1945-1950, IEEE, 1999.
- [11] NADEAU, T. D.; GRAY, K. *SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies*. 1. ed. Gravenstein Highway, Sebastopol: O'Reilly Media, Inc., 2013.



- [12] MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; TURNER, J. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, Nova Iorque, EUA, 38(2), p. 69-74, 2008.
- [13] XIAOHUI, H. *Particle swarm optimization*. 2006. Disponível em: <<http://www.swarmintelligence.org/tutorials.php>>. Acesso em: 12 out. 2018.
- [14] SEZER, S.; SCOTT-HAYWARD, S.; CHOUHAN, P. K.; FRASER, B.; LAKE, D.; FINNEGAN, J.; RAO, N. Are we ready for SDN? Implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7), p. 36-43, 2013.
- [15] SHIN, M. K.; NAM, K. H.; KIM, H. J. Software-defined networking (SDN): A reference architecture and open APIs. *ICT Convergence (ICTC)*. International Conference, IEEE, p. 360-361, out., 2012.
- [16] OLIVEIRA, G. D.; MOURA, R. K. G. Gestão da segurança da informação: perspectivas baseadas na tecnologia da informação (TI). *Múltiplos Olhares em Ciência da Informação*. 3(2), ISSN 2237-6658, 2014.
- [17] GOUVEIA, L. B. *Gestão da Segurança da Informação: Conceitos básicos e introdução ao tema da Gestão da Segurança da Informação no contexto do digital e dos Sistemas e Tecnologias de Informação*, Versão 1.1, 2017.
- [18] WETHERALL, J.; TANENBAUM, A. S. *Redes de computadores*. 5. ed. Rio de Janeiro: Editora Campus, 2011.
- [19] CAMPOS, A. *Sistema de segurança da informação*.. 2 ed. São Paulo: VisualBooks, 2007.
- [20] LEWIS, D. *State of the internet / security - Q2 2017 Report*. Akamai, v.4, n. 2, 2017. Disponível em: <<https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q2-2017-state-of-the-internet-security-report.pdf>>. Acesso em: 19 maio 2018.
- [21] KHALIMONENKO, A.; KUPREEV, O.; ILGANAIEV, K. *Ddos attacks in Q3 2017*. Nov., 2017. Disponível em: <<https://securelist.com/ddos-attacks-in-q3-2017/83041/>>. Acesso em: 13 maio 2018.
- [22] ZADEH, L. A. Fuzzy logic = computing with words. *IEEE Transactions on Fuzzy Systems*, 4(2), p. 103-111, 1996.
- [23] MAMDANI, E. H. Application of fuzzy logic to approximate reasoning using linguistic synthesis. Proceedings of the sixth international symposium on Multiple-valued logic. *IEEE Computer Society Press*, p. 196-202, maio, 1976.
- [24] LEE, C. C. Fuzzy logic in control systems: fuzzy logic controller. I. *IEEE Transactions on systems, man, and cybernetics*, 20(2), p. 404-418, 1990.
- [25] FULLÉR, R. *Neural fuzzy systems*. Abo Akademi University, p. 1-348, abr., 1995.
- [26] CHENCI, G. P.; RIGNEL, D. G.; LUCAS, C. A. Uma introdução à lógica Fuzzy. *Revista Eletrônica de Sistemas de Informação e de Gestão Tecnológica*, 1(1), 2011.

- [27] RINI, D. P.; SHAMSUDDIN, S. M.; YUHANIZ, S. S. Particle swarm optimization: technique, system and challenges. *International Journal of Computer Applications*, 14(1), p. 19-26, 2011.
- [28] CHIA-NAN, K.; CHIA-JU, W. A PSO-Tuning Method for Design of Fuzzy PID Controllers. *Journal of Vibration and Control*, Los Angeles, CA, EUA, 14(3), p. 375-395, mar., 2008.
- [29] DOCTOR, S.; VENAYAGAMOORTHY, G. K. & GUDISE, V. G. Optimal PSO for collective robotic search applications. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, (2), p. 1390-1395, jun., 2004.
- [30] DICKERSON, J. E.; DICKERSON, J. A. Fuzzy network profiling for intrusion detection. *Fuzzy Information Processing Society. NAFIPS. 19th International Conference of the North American*, IEEE, p. 301-306, 2000.
- [31] SAINI, L.; SURYAWANSHI, N. Y. A System for Denial-of-Service Attack Detection using MCA and IDS-based on Fuzzy Logic. *International Journal of Computer Applications*, 141(2), 2016.
- [32] MONDAL, H. S.; HASAN, M. T.; HOSSAIN, M. B.; RAHAMAN, M. E.; HASAN, R. Enhancing secure cloud computing environment by Detecting DDoS attack using fuzzy logic. *Electrical Information and Communication Technology (EICT). 3rd International Conference*, IEEE, p. 1-4, dez., 2017.
- [33] TUNCER, T.; TATAR, Y. Detection SYN flooding attacks using fuzzy logic. *Information Security and Assurance - ISA. International Conference*, IEEE, p. 321-325, abr., 2008.
- [34] DANG-VAN, T.; TRUONG-THU, H. A Multi-Criteria based Software Defined Networking System Architecture for DDoS-Attack Mitigation. *REV Journal on Electronics and Communications*, 6(3-4), 2017.
- [35] TAKAGI, T.; SUGENO, M. Fuzzy identification of systems and its applications to modeling and control. *IEEE transactions on systems, man, and cybernetics*, (1), p. 116-132, 1985.
- [36] SHIAELES, S. N.; KATOS, V.; KARAKOS, A. S.; PAPADOPOULOS, B. K. Real time DDoS detection using fuzzy estimators. *Computers & security*, 31(6), p. 782-790, set., 2012.
- [37] SANTOS, A. F. P.; SILVA, R. S. R. *Detecção de Ataques DDoS com Gráficos de Controle e Bases de Regras Nebulosas*, Rio de Janeiro-RJ, 2009.
- [38] DUCLOS, E. *Contribution à la maîtrise statistique des procédés: cas des procédés non normaux*. 1997. Doctoral dissertation, Chambéry.
- [39] DUCLOS, E.; PILLET, M.; AVRILLON, L. The l-chart for non-normal processes. *Quality Technology & Quantitative Management*, 2(1), p. 77-90, 2005.
- [40] XIA, Z.; LU, S.; LI, J.; TANG, J. Enhancing DDoS flood attack detection via intelligent fuzzy logic. *Informatica*, 34(4), 2010.

- [41] RODRIGUEZ, J. C. C.; BRIONES, A. P.; NOLAZCO, J. A. FLF4DoS. Dynamic DDoS Mitigation based on TTL field using fuzzy logic. *Electronics, Communications and Computers*. CONIELECOMP 07. 17th International Conference, IEEE, p. 12-12, fev., 2007.
- [42] KHAJURIA, A.; SRIVASTAVA, R. Knowledge Based System for Detection and Prevention of DDoS Attacks using Fuzzy logic. *International Journal of Enhanced Research in management and computer applications*, (2), p. 33-42, maio, 2013.
- [43] SANCHEZ, M. C. E. Controle por Aprendizado Acelerado e Neuro-Fuzzy de Sistemas Servo - Hidráulicos de Alta Frequência. Tese. *Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO*. Rio de Janeiro. p. 59-72. 2018.
- [44] LIM, S.; HA, J.; KIM, H.; KIM, Y.; YANG, S. A SDN-oriented DDoS blocking scheme for botnet-based attacks. *Ubiquitous and Future Networks (ICUFN)*. Sixth International Conf., IEEE, p. 63-68, jul., 2014.
- [45] TSUNODA, H.; OHTA, K.; YAMAMOTO, A.; ANSARI, N.; WAIZUMI, Y.; NEMOTO, Y. Detecting DRDoS attacks by a simple response packet confirmation mechanism. *Computer Communications*, 31(14), p. 3299-3306, 2008.
- [46] MOHAMMADI, R.; JAVIDAN, R.; CONTI, M. Slicots: an sdn-based lightweight countermeasure for TCP syn flooding attacks. *IEEE Transactions on Network and Service Management*, 14(2), p. 487-497, 2017.
- [47] SKFUZZY. Disponível em: <<https://pythonhosted.org/scikit-fuzzy/overview.html>>. Acesso em: 5 out. 2018.
- [48] BORGES, L. E. *Python para Desenvolvedores*. 2 ed. Rio de Janeiro:Novatec Editora, 2010.
- [49] OCTOPRESS. Mininet, 2018. Disponível em: <<http://mininet.org/>>. Acesso em: 14 jan. 2018.
- [50] RYU SDN FRAMEWORK. Ryu controller SDN, 2017. Disponível em: <<https://osrg.github.io/ryu/>>. Acesso em: 12 jan. 2018.
- [51] NOCTION NETWORK INTELLIGENCE. *DDoS Amplification Attacks*. 22 mar. 2018. Disponível em: <<https://www.noction.com/blog/ddos-amplification-attacks>>. Acesso em: 19 nov. 2018.
- [52] RATIONALLY PARANOID. *Hping usage examples*. 6 jun. 2009. Disponível em: <<https://www.rationallyparanoid.com/articles/hping.html>>. Acesso em: 10 out. 2018.
- [53] KROUT, Elle. *Load Testing Web Servers with Siege*. Linode, 18 fev. 2015. Disponível em: <<https://www.linode.com/docs/tools-reference/tools/load-testing-with-siege/>>. Acesso em: 5 out. 2018.
- [54] URLLIB3 DOCUMENTATION. 2018. Disponível em: <<https://urllib3.readthedocs.io/en/latest/>>. Acesso em: 19 ago. 2018.
- [55] CHEN, S.; MONTGOMERY, J. Selection strategies for initial positions and initial velocities in multi-optima particle swarms. *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, ACM, p. 53-60, jul., 2011.

- 
- [56] GOOGLE DEVELOPERS. 2019. Disponível em: <<https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall>>. Acesso em: 20 nov. 2018.
- [57] CORRÊA, J. H.; AGUIAR, E.; BRUNETTI, P.; NIGAM, V.; FONSECA, I. E.; GUIMARAES, R. L., ... & VILLAÇA, R. S. Dispositivos conectados a serviço web em nuvem: complementariedade entre infraestrutura e seletividade para proteção contra DDoS. *In Anais do Workshop de Segurança Cibernética em Dispositivos Conectados (WSCDC-SBRC 2018)*, SBC, (1), may., 2018