UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Adeilson Marques da Silva Cardoso

CEPIDS: Um IDS baseado no Processamento de Eventos Complexos para Internet de Coisas

São Luís
2018

Adeilson Marques da Silva Cardoso

CEPIDS: Um IDS baseado no Processamento de Eventos Complexos para Internet de Coisas

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Maranhão como requisito parcial para a obtenção do grau de MESTRE em Ciência da Computação.

Orientador: Rafael Fernandes Lopes

Doutor em Engenharia Elétrica – UFMA

São Luís

2018

Cardoso, Adeilson Marques da Silva

CEPIDS: Um IDS baseado no Processamento de Eventos Complexos para Internet de Coisas / Adeilson Marques da Silva Cardoso. – São Luís, 2018.

91 f.

Orientador: Rafael Fernandes Lopes.

Impresso por computador (fotocópia).

Dissertação (Mestrado) – Universidade Federal do Maranhão, Programa de Pós-Graduação em Ciência da Computação. São Luís, 2018.

1. CEP. 2. Detecção de Intrusão. 3. Internet das Coisas. I. Lopes, Rafael Fernandes, orient. II. Título.

CDU

Adeilson Marques da Silva Cardoso

CEPIDS: Um IDS baseado no Processamento de Eventos Complexos para Internet de Coisas

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Adeilson Marques da Silva Cardoso e aprovada pela comissão examinadora.

Aprovada em 09 de Maio de 2018

BANCA EXAMINADORA

Rafael Fernandes Lopes (orientador)

Doutor em Engenharia Elétrica – UFMA

Francisco José da Silva e Silva

Doutor em Ciência da Computação – IME-USP

Omar Andres Carmona Cortes

Doutor em Ciência da Computação – ICMC - USP

A Deus, minha querida esposa, aos meus pais, meus irmãos, meus amigos e meus professores.

Resumo

Com a expansão da Internet de Coisas (*Internet of Things - IoT*), pode-se observar um aumento no desenvolvimento de serviços e aplicações que utilizam um grande número de sensores e dispositivos sensíveis ao contexto incorporados em ambientes inteligentes. Esses dispositivos produzem continuamente grandes quantidades de dados, aumentando a necessidade de mecanismos de defesa mais rápidos e precisos contra o tráfego malicioso para garantir que os serviços de rede não sejam interrompidos. Processamento de eventos complexos (*Complex Event Processing - CEP*) é uma tecnologia emergente e promissora que permite a análise em tempo real de dados de eventos em fluxo contínuo. Assim a integração de um Sistema de Detecção de Intrusão (IDS) com CEP pode ser usado para lidar com padrões de eventos, processando uma enorme quantidade de dados. Esta dissertação apresenta o CEPIDS, um sistema de detecção de intrusão para IoT em tempo real que usa as regras do CEP para identificar ataques que desencadeiam alertas. O CEPIDS foi projetado para detectar ataques de redes e realizar a detecção de seis tipos de ataques: *"syn flood, udp flood, icmp flood, smurf attack, land attack"* e *"port scan"*. O desempenho do sistema proposto foi avaliado em um Raspberry Pi. Os resultados mostram que CEPIDS pode ser executado em dispositivos com recursos reduzidos e pode efetivamente ser usado como IDS em um sistema distribuído. Além disso, os resultados indicam que o CEP é uma solução viável para melhorar o desempenho em tempo real do IDS no IoT.

Palavras-chave: Detecção de Intrusão, Internet das Coisas, CEP.

Abstract

With the Internet of Things (IoT) expansion, an increase can be observed in the development of services and applications which use large numbers of sensors and context-aware devices incorporated into smart environments. Those devices continuously produce large amounts of data raising the need for faster and more precise defense mechanisms against malicious traffic to ensure that network services will not be disrupted. Complex Event Processing (CEP) is an emerging and promising technology that enables real-time analysis of the event in the continuous data flow. The integration of an Intrusion Detection System (IDS) with CEP can be used to handle event patterns by processing a huge amount of data. This paper proposes the CEPIDS system, a real-time IoT intrusion detection system that uses CEP rules to identify attacks, which trigger alerts. We evaluated the performance of the proposed system in a Raspberry Pi, one of the most used commodity single-board computers, while running the CEPIDS. Results show that CEPIDS can be run on devices with reduced resources and can effectively be used as IDS in a distributed system. Also, the outcomes indicate that CEP is a viable solution to improve real-time performance of IDS in IoT.

Keywords: Intrusion Detection, Internet of Things, CEP.

Agradecimentos

Ao meu Deus em primeiro lugar, pela vida, pela sabedoria que me deste para a realização deste trabalho, pelo que tenho, pelo que sou e pelo que ainda serei em sua presença.

A minha querida esposa pelo amor, carinho, atenção, apoio incondicional e por está sempre presente em todos os momentos me ajudando e orientando nas minhas decisões.

Aos meus filhos, Ezequias, Isabelle e Davi Samuel, vocês são o combustível que me impulsiona a lutar e a conquistar meus objetivos.

A minha família pelo apoio incondicional, pela compreensão e por acreditarem em mim.

Ao meu orientador, o Prof. Rafael Fernandes Lopes, pelo exemplo, apoio, orientação e compreensão. Mesmo muito ocupado, sempre acompanhou o desenvolvimento das atividades.

Ao professor Ariel Soares Teles (IFMA) pelo apoio, orientação e pelo tempo dedicado durante a elaboração dos artigos e escrita desta dissertação. Mesmo não sendo Coorientador regular pelo PPGCC, sua contribuição foi fundamental para a conclusão desta pesquisa.

Ao membros do LSDi-UFMA, com os quais compartilhei alegrias, tristezas, e conhecimento. Registro a importância de Anderson S. Costa, Danne Makleyston G. Pereira, Fernando B. V. Magalhães, Marcelino Mendes S. Neto e Tércio S. S. Sousa para a conclusão desta pesquisa.

Ao meu amigo do IFTO, Cláudio de Sousa Galvão, que não mediu esforços para ajudar quando precisei.

A UFMA e ao PPGCC pela estrutura dada a execução deste trabalho.

"A opinião que você adota a respeito de si mesmo afetará profundamente a maneira pela qual você leva sua vida. Ela pode decidir se você se tornará a pessoa que deseja ser e se realizará aquilo que é importante para você."

Paulo Vieira

Lista de Figuras

Lista de Tabelas

Lista de Siglas

CEP Complex Event Process.

CPU Central Processing Unit.

DoS Denial of Service.

DDoS Distributed Denial-of-Service.

EPL Event Processing Language.

HIDS Host-based IDS.

ICMP Internet Control Message Protocol.

IDS Intrusion Detection System.

IoT Internet of Things.

IP Internet Protocol.

IPv6 Internet Protocol version 6.

NBA Network Behavior Analysis.

NIDS Network-based IDS.

RFID Radio-Frequency IDentification.

TPC Transmission Control Protocol.

UDP UDP User Datagram Protocol.

UFMA Universidade Federal do Maranhão.

UML Unified Modeling Language.

WIDS Wireless-based IDS.

Sumário

1 Introdução

A Internet das Coisas (IoT) é um novo paradigma que combina aspectos e tecnologias provenientes de diferentes abordagens, tais como: computação ubíqua, ciência de contexto, protocolos e tecnologias de comunicação, rede de computadores e dispositivos com sensores e atuadores. Essa integração acontece a partir da mesclagem de milhares de dispositivos endereçáveis e heterogêneos, conhecidos como objetos inteligentes (*smart objects*), que possuem uma representação na Internet e visa a geração de sistemas que estão continuamente interagindo, sendo capazes de compartilhar dados de contexto entre si e com diversas aplicações [5].

Esse paradigma permite pessoas e coisas/objetos inteligentes estarem conectados em qualquer tempo e lugar, com qualquer coisa e pessoa, idealmente usando qualquer caminho/rede e serviço [44]. A IoT propõe a expansão da Internet para uma rede de objetos interligados que colhem informações do ambiente para prestar serviços de transferência de informação, análises, aplicações e comunicações [14]. Estima-se que, por volta de 2020, mais de 20 bilhões de objetos inteligentes estejam conectados à rede[1]. Com o aumento do número de dispositivos conectados, mais oportunidades para a exploração de vulnerabilidades estarão disponíveis.

Existem muitos métodos de intrusão em redes de computadores, os quais configuram uma variedade de ataques que podem comprometer a segurança dos sistemas e a disponibilidades dos dados. Os ataques diferem em termos das técnicas utilizadas e dos recursos técnicos que um atacante tem à sua disposição [13]. Dentre as principais técnicas que afetam a disponibilidade dos dados destacam-se os ataques de inundação (*syn flood, udp flood, land attack e icmp flood*) e os ataques de amplificação tais como (*smurf attack*), que tem como características o envio de requisições forjadas para um endereço IP de *broadcast* [54]. Esses tipos de ataques são conhecidos como ataques de negação de serviço (*Denial of Service - DoS*) e ataques de negação de serviços distribuídos (*Distributed Denial of Service - DDoS*). Como os dispositivos de IoT capturam dados sobre eventos de objetos inteligentes e *gateways* de IoT, esses fluxos de eventos precisam ser identificados, processados e reagir em um curto espaço de

[1] `https://www.gartner.com/newsroom/id/3165317`

tempo [17]. Portanto os mecanismos de segurança devem processar esses fluxos de eventos imediatamente e emitir alertas de segurança.

O Processamento de Eventos Complexos (CEP) é uma tecnologia emergente para filtrar e processar eventos em cenários de tempo real [10]. As informações básicas processadas no CEP são definidas como eventos e os fenômenos ocorridos na rede podem ser modelados como eventos. O CEP permite que os usuários especifiquem os eventos de acordo com seu próprio interesse, como eventos de transmissão em redes de sensores sem fio ou eventos operacionais em aplicativos corporativos. A tecnologia CEP pode ser usada para detectar relacionamentos complexos significativos entre eventos e responder a eles em tempo real, podendo ser aplicada a um amplo espectro de desafios de sistemas de informação, incluindo automação de processos de negócios, processos de cronograma e controle, monitoramento de rede, previsão de desempenho e detecção de intrusão [28]. Assim, um IDS que utiliza Processamento de Eventos Complexos (CEP) pode ser usado para lidar com padrões de eventos e processar grandes volumes de dados, podendo ser uma ótima solução para melhorar o desempenho e identificar ameaças em tempo real em ambientes de IoT.

Este trabalho propõe o sistema CEPIDS, um Sistema de Detecção de Intrusão que usa regras CEP para resolver problemas de segurança em ambientes IoT. O CEPIDS foi projetado para detectar ataques de redes e realizar a detecção de seis tipos de ataques: *"syn flood, udp flood, icmp flood, smurf attack, land attack"*, que têm como característica um grande fluxo de dados e *"port scan"* que permitem a detecção de portas abertas em máquinas alvo. Desta forma, novos ataques podem ser iniciados usando portas abertas. Além disso, o CEPIDS é projetado para ser estendido para detectar outros tipos de ataques, por exemplo, ARP Spoofing e ataques de roteamento. O Sistema foi avaliado em Raspberry Pi, um pequeno computador que dispõe de poucos recursos computacionais, através do qual foi possível mensurar a viabilidade do sistema proposto.

1.1 Metodologia

Esta dissertação objetiva desenvolver um estudo para detectar ataques à segurança em ambientes IoT. O desenvolvimento deste trabalho consiste na seguinte metodologia:

Inicialmente, foi realizado um levantamento bibliográfico em teses, artigos científicos, anais de congressos, dissertações e periódicos, visando à aquisição da fundamentação teórica das áreas de Sistemas de Detecção de Intrusão, Internet das Coisas, Processamento de Eventos Complexos e Raspberry Pi.

O passo seguinte foi a definição das ferramentas e tecnologias a serem utilizadas, a modelagem e a implementação do IDS. Nesta fase foram definidos os requisitos de segurança, a linguagem de programação, as bibliotecas a serem utilizadas, os protocolo de comunicação e os tipos de ataques a serem detectados. A linguagem de programação Java foi utilizada abordando a metodologia de Programação Orientada a Objetos para desenvolvimento do sistema. Foi utilizada uma versão da biblioteca Jpcap disponível para arquitetura ARM para leitura do tráfego. O sistema foi projetado para detectar os ataques de negação de serviço *"syn flood, udp flood, icmp flood, smurf attack, land attack"* e *"port scan"*. Raspberry Pi foi definido como dispositivo de teste para o experimento.

Considerando o baixo poder computacional do *Raspberry Pi*, o sistema proposto foi avaliado quanto ao uso da Unidade Central de Processamento (CPU), da Memória de Acesso Aleatório (RAM), taxa de pacotes perdidos e da Taxa de Detecção de Ataques. Utilizou-se dois conjuntos de dados (*datasets*) para avaliação do sistema: O **DARPA** do Instituto de Tecnologia de Massachusetts que é composto por tráfego TCP/IP e UDP os quais foram obtidos ao longo de sete semanas de coleta sendo composto por tráfego normal e tráfego de ataque de negação de serviço distribuído [26] e o **CTU-13** que é um conjunto de dados do tráfego de uma *botnet*[2] que foi capturado na Universidade CTU, na República Tcheca cujo objetivo era ter uma grande captura de tráfego real de uma rede *botnet* misturada ao tráfego de uma rede normal. O conjunto de dados CTU-13 consiste em treze capturas (denominadas cenários) de diferentes

[2]Rede formada por centenas ou milhares de computadores infectados com bots. Permite potencializar as ações danosas executadas pelos bots e ser usada em ataques de negação de serviço, esquemas de fraude, envio de spam, etc.

amostras de botnets [32]. Os conjuntos de dados utilizados são compostos por tráfego não tratado e por tráfego rotulado.

Para que os *datasets* fossem utilizados a partir do *Raspberry Pi*, foi utilizado o comando *editcap* da ferramenta *wireshark* para dividi-los em arquivos com menor extensão em *megabytes*. Foi definido o total de um cem mil (100000) pacotes para cada arquivo, sendo estes compostos por tráfego de uma rede normal e por tráfego que configuram ataques. Foi realizado um (1) experimento com 10 repetições cada experimento para cada tipo de ataque abordado nesta pesquisa, totalizando seis (6) experimentos. Ao finalizar cada repetição do experimento o *Raspberry Pi* foi reiniciado com o objetivo de finalizar todos os processos em execução no dispositivo e de limpar os dados armazenados em memória. Os experimentos são descrito a seguir:

- O experimento número 1 teve por objetivo analisar o conjuntos de dados DARPA através da extração das características tráfego para identificação da taxa de detecção de ataques do tipo *SYN Flood*. Esse ataque pode ser identificado a partir das informações coletadas dos pacotes capturados, observando a frequência de chegada dos pacotes, o tamanho dos pacotes trafegados, bem como a presença da *Flag SYN* e a ausência da *Flag ACK*. O experimento foi conduzido da seguinte maneira: Ao iniciar o CEPIDS o sistema realiza a coletar os dados do tráfego a partir do conjunto de dados DARPA. Em seguida o CEPIDS faz a extração das características dos pacotes e a análise dos dados verificando se o tráfego apresenta características de ataques. Para cálculo do consumo da CPU e da memória RAM foi desenvolvida uma ferramenta de *script* capaz de registrar todas as oscilações referente ao uso da CPU e da memória RAM e retornar a média de utilização de ambas as métricas. A taxa de detecção de ataque foi calculada a partir dos *logs* gerados pelo CEPIDS durante o experimento, sendo estes gravados em um arquivo de texto para posterior análise.

- O experimento número 2 utilizou o conjuntos de dados DARPA para extraí as características tráfego e teve por objetivo realizar a análise desses dados para a identificação de ataques do tipo *UDP Flood*. *UDP Flood* é caracterizado pelo grande número de pacotes UPD enviados para o sistema da vítima e pode ser identificado a partir das características dos pacotes capturados, observando a frequência de chegada dos pacotes e o tamanho dos pacotes trafegados. O

experimento foi executado da seguinte forma: O CEPIDS foi inciado dando inicio a coleta do tráfego a partir do conjunto de dados DARPA. Após a coleta do tráfego foi realizada a extração das características dos pacotes capturados e a análise dos dados, verificando se o tráfego apresentava características de ataques. Para aferir o uso da CPU e da memória RAM durante a execução do experimento foi desenvolvida uma ferramenta de *script* capaz de registrar todas as oscilações referente ao consumo da CPU e da memória RAM e retornar a média de utilização de ambas as métricas. A taxa de detecção de ataque foi calculada a partir dos *logs* gerados pelo CEPIDS durante o experimento, sendo estes gravados em um arquivo de texto para posterior análise.

- O experimento número 3 utilizou o conjuntos de dados CTU-13 e cenário CTU-11 tendo por objetivo realizar a análise desses dados para a identificação de ataques do tipo *ICMP Flood*. O Cenário CTU-11 é composto por tráfego normal e tráfego de ataque negação de serviço do tipo ICMP. *ICMP Flood* Tem por característica o envio de inúmeros pacotes ICMP para o sistema da vítima e pode ser identificado a partir das características dos pacotes capturados, observando a frequência de chegada dos pacotes e o tamanho dos pacotes trafegados. O experimento foi executado da seguinte forma: Ao iniciar o sistema a coleta do tráfego é realizada a partir do conjunto de dados CTU-13. Após a coleta do tráfego extrai-se as características dos pacotes capturados e a análise dos dados é realizada, verificando se o tráfego apresenta características de ataques. Para aferir o uso da CPU e da memória RAM durante a execução do experimento foi desenvolvida uma ferramenta de *script* capaz de registrar todas as oscilações referente ao consumo da CPU e da memória RAM e retornar a média de utilização de ambas as métricas. A taxa de detecção de ataque foi calculada a partir dos *logs* gerados pelo CEPIDS durante o experimento, sendo estes gravados em um arquivo de texto para posterior análise.

- O experimento número 4 utilizou o conjuntos de dados CTU-13 e cenário CTU-4 tendo por objetivo realizar a análise desses dados para a identificação de ataques do tipo *Smurf Attack*. O Cenário CTU-4 é composto por tráfego normal e tráfego de ataque negação de serviço do tipo ICMP e do tipo UDP. *Smurf Attack* É caracterizado pelo o envio de inúmeros pacotes ICMP para o sistema da vítima assim como no *ICMP Flood*, no entanto é utilizado o endereço de

broadcast para amplificar o ataque. O *Smurf Attack* pode ser identificado a partir das características dos pacotes capturados, observando a origem e o destino dos pacotes, a frequência de chegada e o tamanho dos pacotes trafegados. O experimento foi executado da seguinte forma: Ao iniciar o sistema a coleta do tráfego é realizada a partir do conjunto de dados CTU-13. Após a coleta do tráfego extrai-se as características dos pacotes capturados e a análise dos dados é realizada, verificando se o tráfego apresenta características de ataques. Para aferir o uso da CPU e da memória RAM durante a execução do experimento foi desenvolvida uma ferramenta de *script* capaz de registrar todas as oscilações referente ao consumo da CPU e da memória RAM e retornar a média de utilização de ambas as métricas. A taxa de detecção de ataque foi calculada a partir dos *logs* gerados pelo CEPIDS durante o experimento, sendo estes gravados em um arquivo de texto para posterior análise.

- O experimento número 5 utilizou o conjuntos de dados DARPA tendo por objetivo realizar a análise desses dados para a identificação de ataques do tipo *Land Attack*. O *Land Attack* tem por característica o envio de inúmeros pacotes TCP ou UDP para o sistema da vítima utilizando o mesmo endereço IP de destino e origem e as mesmas portas de destino e origem. O *Land Attack* pode ser identificado observando se a origem e o destino dos pacotes são iguais, a frequência de chegada e o tamanho dos pacotes trafegados. O experimento foi executado da seguinte forma: Ao iniciar o sistema a coleta do tráfego é realizada a partir do conjunto de dados DARPA. Após a coleta do tráfego as características dos pacotes capturados são extraídas e a análise dos dados é realizada, verificando se o tráfego apresenta características de ataques. Para aferir o uso da CPU e da memória RAM durante a execução do experimento foi desenvolvida uma ferramenta de *script* capaz de registrar todas as oscilações referente ao consumo da CPU e da memória RAM e retornar a média de utilização de ambas as métricas. A taxa de detecção de ataque foi calculada a partir dos *logs* gerados pelo CEPIDS durante o experimento, sendo estes gravados em um arquivo de texto para posterior análise.

- O experimento número 6 utilizou a ferramenta *Zenmap security scanner* tendo por objetivo realizar os ataques de *Port Scan* uma vez que os *datasets* utilizados não tinham dados referente a esse tipo de ataque. O *Port Scan* realiza a varredura de

portas no sistema da vítima verificando quais portas estão abertas, facilitando assim o possíveis intrusões. O *Port Scan* pode ser identificado observando a frequência de chegada dos pacotes e as portas de destino dos mesmos. O experimento foi executado da seguinte forma: Montou-se um ambiente de testes controlado e composto por um *desktop*, um *laptop* e três *Raspberry Pi*. A partir do *laptop* foi realizado o ataque de *Port Scan* utilizando a ferramenta *Zenmap security scanner*. Um dos dispositivos *Raspberry Pi* foi configurado para ser um ponto de acesso *wifi* enquanto executa o CEPIDS. Os demais dispositivos *Raspberry Pi* foram configurados utilizando a ferramenta *hping3* para gerar tráfego na rede com o objetivo de simular o tráfego de uma rede normal. Faz parte do cenário para o experimento uma aplicação para medição de temperatura e umidade, no qual foi utilizado um sensor DHT11 para realizar a coleta da temperatura e umidade do ambiente. O cenário abordado faz parte de um projeto de automação para um setor de avicultura, onde o controle da temperatura e da umidade é essencial para garantir o crescimento adequado das aves e produtividade do aviário. Ao iniciar o sistema a coleta do tráfego foi realizada a partir da placa de rede. Após a coleta do tráfego as características dos pacotes capturados são extraídas e a análise dos dados é realizada, verificando se o tráfego apresenta características de ataques. Para aferir o uso da CPU e da memória RAM durante a execução do experimento foi desenvolvida uma ferramenta de *script* capaz de registrar todas as oscilações referente ao consumo da CPU e da memória RAM e retornar a média de utilização de ambas as métricas. A taxa de detecção de ataque foi calculada a partir dos *logs* gerados pelo CEPIDS durante o experimento, sendo estes gravados em um arquivo de texto para posterior análise do número de alertas emitidos.

- A análise estatística dos dados para todos os experimentos abordados nesta pesquisa foi realizada utilizando a ferramenta *PAST*[3] versão 3.20. *PAST* é um software livre para análise de dados científicos, com funções de manipulação de dados, plotagem, estatística univariada e multivariada, séries temporais e análises espaciais, entre outras [16]. Esta ferramenta foi utilizada para normalizar os dados tornando-os paramétricos, bem como calcular o desvio padrão, o intervalo de confiança dos dados e plotar os gráficos referente aos resultados.

[3]https://folk.uio.no/ohammer/past/

1.2 Objetivos

A pesquisa à qual se refere esta dissertação de mestrado tem como objetivo geral propor um IDS que possa ser executado em dispositivos com baixo poder computacional, que seja capaz de realizar a detecção de tráfego malicioso em tempo real em redes IoT utilizando regras CEP e emitir alertas sobre tráfego suspeito.

Os objetivos específicos desta pesquisa são:

- Levantar o estado da arte referente à sistemas de detecção de intrusão e processamento de eventos complexos;
- Prospecção e estudo das principais técnicas de ataques de negação de serviço, com intuito de identificar os meios mais utilizados para realização desse tipo de ataque;
- Desenvolver uma ferramenta para melhorar a segurança para dispositivos em redes IoT, utilizando uma quantidade mínima de recursos possíveis;
- Projetar e implementar regras de processamento de eventos complexos para detecção de tráfego malicioso em tempo real;
- Realizar testes do sistema desenvolvido utilizando um Raspberry Pi;
- Avaliar a solução proposta visando aumentar a confiabilidade no processo de detecção de tráfego malicioso.

1.3 Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma:

- O Capítulo 2 apresenta a fundamentação teórica sobre ataques negação de serviços, ataques de negação de serviços distribuídos, processamento de eventos complexos e bibliotecas para captura de pacotes na rede.
- O Capítulo 3 apresenta uma análise dos principais trabalhos propostos nos últimos anos que mantêm uma relação aos tópicos que serão estudados no contexto desta pesquisa.

- O Capítulo 4 descreve a solução proposta de forma detalhada, dando ênfase a arquitetura do sistema e ao fluxograma de execução. É apresentada ainda a implementação do IDS, destacando as tecnologias e ferramentas utilizadas.

- O Capítulo 5 apresenta os resultados, dando ênfase à performance do Raspberry enquanto executa o sistema proposto e as discussões, comparando o desempenho do CEPIDS com outros IDSs populares e de código aberto.

- No Capítulo 6 é apresentada a conclusão e os trabalhos futuros.

2 Fundamentação Teórica

Este capítulo apresenta um resumo da fundamentação teórica referente a sistemas de detecção de intrusão, ataques DoS e DDoS e referente a algumas bases tecnológicas utilizadas no desenvolvimento da solução proposta os quais são importantes para a compreensão deste trabalho.

2.1 Sistemas de Detecção de Intrusão

Um Sistema de Detecção de Intrusão (IDS), constitui-se de um mecanismo de defesa que tem por finalidade capturar e analisar os pacotes trafegados em uma rede de computadores. O IDS pode ser usado para monitorar o tráfego em determinado computador, dispositivo ou rede, verificando possíveis ameaças. Ele pode atuar como uma segunda linha de defesa que contribui para proteger a rede de possíveis intrusos [2].

Após a análise, caso seja detectado alguma característica de ataque ou anomalia, conforme tipo de IDS utilizado, este tem por objetivo alertar sobre a atividade maliciosa e gerar um histórico de tais atividades para futuras modificações nas políticas de segurança da rede. Com base no histórico gerado, é possível extrair informações que poderão ser utilizadas para melhorar os sistemas de detecção de ataques, tais como: tipo de ataque, IPs de origem e destino dos ataques, taxa de pacote por segundo, etc.

Atualmente existem vários tipos de IDSs, os quais são classificados de acordo com o local onde são implantados para inspecionar o tráfego suspeito e quais tipos de eventos eles podem detectar [25]. Dentre os quais destacam-se: IDS baseado em host (*Host-based IDS* - HIDS), IDS baseado em rede (*Network-based IDS* - NIDS) e (*Wireless-based IDS* - WIDS) e IDSs baseado em Aplicação, o qual realiza a análise do comportamento da rede (*Network Behavior Analysis* - NBA).

Um HIDS monitora e coleta o tráfego dos hosts que contêm informações confidenciais, servidores que executam serviços públicos e atividades suspeitas. Um NIDS captura o tráfego de rede em segmentos de rede específicos através de sensores e, posteriormente, analisa as atividades de aplicativos e protocolos para reconhecer incidentes suspeitos. O WIDS é semelhante ao NIDS, no entanto realiza a captura do tráfego de rede sem fio, como redes ad hoc, redes de sensores sem fio e redes mesh sem fio. Além disso, um sistema NBA inspeciona o tráfego de rede para reconhecer ataques com fluxos de tráfego inesperados [25]. Uma vez que um ataque é detectado, um IDS pode registrar informações sobre ele ou relatar um alarme. O objetivo do IDS é observar as redes, os hosts ou objetos inteligentes, pra detectar possíveis intrusões e uso indevido na rede.

Jyothsna et al. [18] destacam que os mecanismos de detecção em um IDS podem ser classificados em duas categorias, IDS baseado em assinatura ou IDS baseado em anomalia. A seguir são detalhados cada um desses tipos de IDS.

- **IDS Baseado em Assinatura:** As detecções baseadas em assinaturas correspondem ao comportamento atual da rede contra padrões de ataque predefinidos. As assinaturas são pré-configuradas e armazenadas no dispositivo e cada assinatura corresponde a um determinado ataque. Em geral, técnicas baseadas em assinatura são mais simples de usar. Elas precisam, no entanto, armazenar uma assinatura para cada ataque. Isso requer conhecimento específico de cada ataque e os custos de armazenamento crescem com o número de ataques. Essa abordagem é mais estática e não pode detectar novos ataques a menos que sua assinatura seja adicionada manualmente ao IDS [18].

- **IDS Baseado em Anomalias:** Esta técnica também é conhecida como detecção baseada em comportamentos. O IDS tenta detectar anomalias no sistema, determinando o comportamento normal e usá-lo como linha de base. Quaisquer desvios dessa linha de base são considerados anomalias. Por um lado, os sistemas baseados em anomalias têm a capacidade de detectar quase qualquer ataque e adaptar-se a novos ambientes, mas por outro lado essas técnicas têm taxas de falso-positivo bastante altas [18].

2.2 Ataques DoS e DDoS

A Internet tornou-se hoje uma ferramenta essencial e faz parte da vida cotidiana das pessoas. Devido a essa popularidade tecnológica, surgiram muitos problemas que afetam a segurança das aplicações. As ameaças provocadas por ataques a serviços disponíveis na rede podem causar prejuízos, visto que podem comprometer a segurança da informação. Um ataque DoS tenta esgotar os recursos disponíveis para um determinado serviço, computador ou objeto inteligente conectado a rede, enviando pacotes desnecessários e, portanto, impede que os usuários legítimos acessem serviços ou recursos aos quais têm direito [22].

No ataque DDoS, vários computadores atacam um único servidor simultaneamente. Um usuário mal-intencionado tenta invadir computadores remotos explorando as vulnerabilidades dos softwares que estão sendo executados, instala um programa malicioso nos computadores invadidos e os mantém aguardando uma ordem para atacar um servidor alvo. Esses computadores serão controlados remotamente para realizarem o ataque, sem que os usuários saibam que suas maquinas estão sendo utilizadas. Quando o atacante envia um sinal para eles, o ataque começa de forma sincronizada. Ainda que a taxa de requisições para cada computador seja pequena, o tráfego de ataque pode causar sérios danos no servidor vítima quando o número de computadores utilizado é grande.

Além do atacante e das vítimas, um ataque DDoS normalmente inclui dois elementos adicionais: (i) os computadores mestres e (ii) computadores escravos (veja Figura 2.1).

Os mestres são máquinas comprometidas que são comandadas pelo sistema de um invasor ou programadas para lançar um ataque automaticamente em uma determinada data. Os escravos ou zumbis são máquinas comprometidas que são controladas pelos mestres e responsáveis por gerar o tráfego que a vítima será exposta [50].

Existem duas classes principais de ataques de Negação de Serviço: Ataques de redução da largura de banda e ataques de esgotamento de recursos. Um ataque de redução de largura de banda é projetado para inundar a rede com tráfego indesejado que impede que o tráfego legítimo e o funcionamento normal da rede. Um ataque de

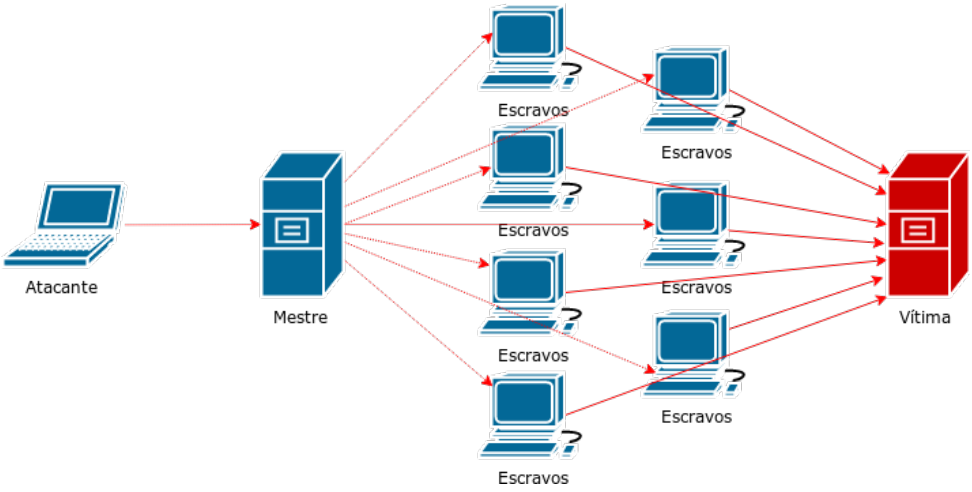

Figura 2.1: Arquitetura de ataque DDoS.

esgotamento de recursos é um ataque que é projetado para esgotar os recursos de um sistema vítima tornando-o incapaz de processar pedidos legítimos dos serviços. São ataques que fazem uso indevido dos protocolos de comunicação de rede ou enviam pacotes de rede malformados. [54].

2.2.1 Ataques por Inundação

Um ataque de inundação envolve computadores zumbis enviando grandes volumes de tráfego com o objetivo de congestionar a largura de banda da rede da vítima. As consequências desse ataque podem diminuir o desempenho do servidor, derrubá-los ou sobrecarregar a rede da vítima, impedindo o acesso por usuários legítimos. Além do protocolo TCP, (acrônimo para o inglês *Transmission Control Protocol*), os ataques de inundação podem ainda utilizar pacotes UDP (*User Datagram Protocol*) ou ICMP (*Internet Control Message Protocol*) [54], realizando ataques *UDP Flood* e *ICMP Flood* respectivamente. Dentre os ataques de inundação, destacam-se os que seguem:

- *UDP Flood:* Nesse ataque um grande número de pacotes UDP são enviados para o sistema da vítima. Esse ataque é caracterizado pelo número de conexões UDP simultâneas a uma taxa maior que 30 pacotes por segundos [19], onde são enviados inúmeros pacotes com o mesmo endereço IP de destino, para diferentes portas, cujo objetivo é saturar a largura de banda [7]. As ferramentas de ataques de negação de serviço utilizam o endereço IP de origem forjado para realizar

ataques. Essa técnica ajuda a esconder a identidade das vítimas secundárias (escravos) dificultando a identificação dos *hosts*[1] que estão formando a *botnet* já que os pacotes da vítima não são enviados de volta para os computadores escravos, mas para endereços falsos [54].

- *ICMP Flood:* Nesse ataque são enviados múltiplos pacotes *echo request* a uma taxa maior que 100 pacotes por segundo [57] até que o limite de requisições por segundos seja ultrapassado. Esses pacotes requerem uma resposta da vítima, causando uma exaustão da banda disponível [21]. Dessa forma o sistema não consegue responder às requisições legítimas, uma vez que a capacidade do enlace foi saturada [37].

- *SYN Flood:* É um dos ataques de negação de serviço em que uma pessoa mal intencionada envia pacotes de sincronização (TCP SYN) a uma taxa maior que 128 pacotes por segundos [57] para um sistema de destino com o objetivo de consumir seus recursos para que o sistema não responda ao tráfego legítimo. Mais detalhes desse tipo de ataque é apresentado na Seção 2.2.3, onde aborda-se sobre ataques por exploração de protocolos.

- *Land Attack:* Esse ataque tem como característica o envio de pacotes TCP SYN em que o endereço de destino e de origem são os mesmos, bem como a porta de origem e destino, fazendo com que o computador vítima ao tentar responder a requisição do cliente entre em loop infinito, uma vez que ele está respondendo a requisição para o mesmo endereço de IP de origem e destino.

2.2.2 Ataques por Amplificação

Também conhecido como ataque reflexivo distribuído de negação de serviço (acrônimo para o inglês *Distribution Reflection Denial of Service (DRDoS)*, os ataques por amplificação são um tipo de ataque DDoS. No entanto, além dos computadores mestres e escravos, esse tipo de ataque é composto também por computadores refletores [40]. A Figura 2.2 representa um ataque DRDoS.

O cenário desse tipo de ataque é o mesmo que os ataques DDoS até um estágio específico. A diferença nesse tipo de ataque é que os computadores escravos

[1]Computador conectado a rede que possui um endereço IP

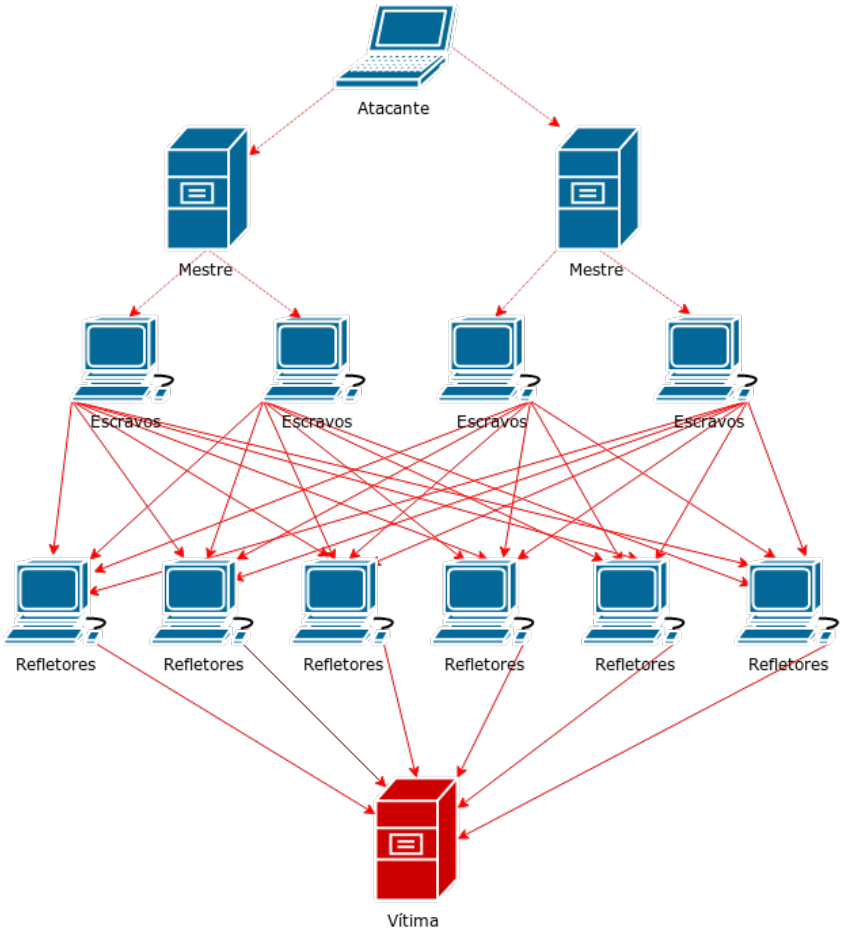

Figura 2.2: Arquitetura um ataque DRDoS.

são programados para enviar um fluxo de pacotes com o endereço IP da vítima como se fosse seu endereço IP de origem para outros computadores não infectados (conhecidos como refletores), fazendo com que todas as respostas ao fluxo de pacotes sejam direcionadas para a vítima, uma vez que o endereço IP que solicitou a abertura de conexão foi o IP da vítima.

Um ataque por amplificação também pode ser caracterizado pelo envio de requisições forjadas para um endereço IP de *broadcast*, também conhecido com *Smurf Attack*, fazendo com que todos os computadores na sub-rede atingidos pelo endereço de *broadcast* enviem uma resposta às requisições ao sistema vítima, (veja Figura 2.3). O recurso de endereço de IP de *broadcast* é encontrado na maioria dos roteadores [54].

Quando uma requisição possui um endereço IP de *broadcast* como o endereço de destino, os roteadores replicam o pacote e o envia para todos os endereços IP dentro do intervalo de endereços de *broadcast*. Neste ataque, o endereço IP de

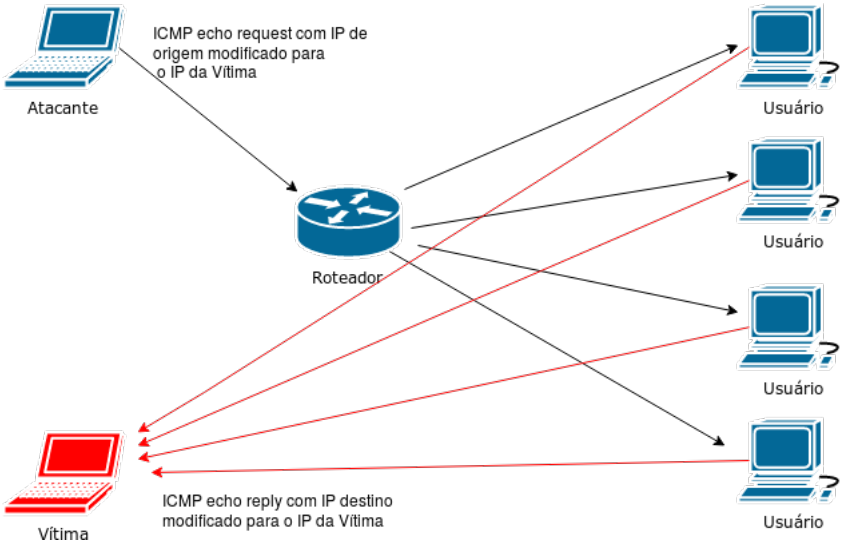

Figura 2.3: Arquitetura de ataque Smurf.

broadcast é usado para amplificar e refletir o tráfego de ataque, e assim reduzir a largura de banda do computador vítima [54] [53].

2.2.3 Ataques por Exploração de Protocolos

Ataques por Exploração de Protocolos são caracterizados por esgotar excessivamente os recursos do servidor vítima. Os principais ataques de exploração de protocolos são caracterizados por uso indevido de pacotes TCP SYN *(Transfer Control Protocol Synchronize)*.

Ataques por inundação TCP SYN, conhecido também como SYN Flood, é um dos ataques de negação de serviço mais usados. Nesse ataque, os invasores enviam tantos pedidos de conexão para um servidor que os usuários legítimos não podem se conectar a esse servidor. Como os invasores podem facilmente colocar servidores em um estado de negação de serviço dessa maneira, cerca de 90% de todos os ataques DoS e DDoS são ataques SYN Flood [34].

No TCP, quando um host local se comunica com um host remoto é estabelecida uma conexão por meio de um processo chamado *handshake* de 3 vias [37]. Nesse processo, o servidor aguarda o recebimento do pacote ACK do cliente em um estado chamado semi-aberto. Como um servidor no estado semi-aberto está usando alguns de seus recursos para o cliente, por exemplo, um buffer alocado, o número de estados semi-abertos deve ser limitado. O número de conexões que ele pode

manter enquanto está no estado semi-aberto é controlado em uma fila de backlog [37]. Pacotes SYN que ultrapassam o número que pode ser mantido na fila de backlog são descartados e o servidor envia pacotes RST para notificar aos clientes que os pacotes foram descartados [55].

Conforme mostrado na Figura 2.4, na comunicação TCP quando um o servidor recebe um pacote TCP SYN, o pacote é interpretado como uma requisição para iniciar uma conexão. Assim, o servidor aloca recursos para guardar informações sobre o estado da conexão e envia um pacote TCP SYN/ACK como resposta para o cliente confirmando o recebimento do pacote SYN e aguarda o recebimento de um pacote ACK do cliente que confirma o recebimento do pacote SYN/ACK. Após esse processo, o transmissor e o receptor podem enfim iniciar a transferência de dados ou finalizar a conexão caso o tempo de espera tenha acabado [55] [37].

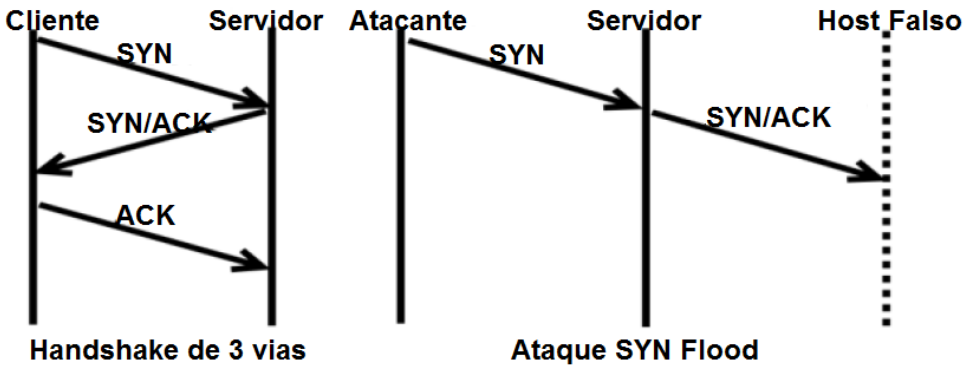

Figura 2.4: Visão geral de um Handshake de 3 vias e um ataque SYN Flood

Em ataques SYN Flood, o servidor recebe uma grande quantidade de pacotes TCP SYN, porém não recebe os pacotes TCP ACK para finalizar o estabelecimento da conexão. Geralmente os pacotes TCP SYN são enviados com um endereço IP de origem forjado [54], assim o servidor irá responder a requisição com um pacote TCP SYN/ACK a um *host* que não fez a requisição de conexão, mantendo assim varias conexões semi-abertas. Com um grande número de conexões semi-abertas, o servidor vítima terá seus recursos sobrecarregados, fazendo com que novas requisições sejam descartadas [37].

2.3 Processamento de Eventos Complexos

Com o surgimento da IoT o número de objetos inteligentes conectados à internet têm crescido de forma considerável. Assim, as pessoas podem acessar qualquer objeto inteligente que esteja conectado, a qualquer momento e de qualquer lugar [4]. Devido a essa popularidade tecnológica, um número cada vez maior de redes de sensores e dispositivos inteligentes coletam continuamente grandes quantidades de dados e realizar a analise desses fluxos de dados em tempo real torna-se um problema crítico.

O processamento de eventos complexos (Complex Event Processing - CEP) tem origem em pesquisas realizadas sobre Simulação de Eventos Discretos, onde o primeiro projeto que utilizou um modelo genérico de linguagem CEP foi desenvolvido na Universidade de Stanford, liderado por David Luckham [24].

O CEP [29] é uma tecnologia emergente que possibilita fazer a correlação de eventos de entrada contínuos e padrões de interesse, onde os resultados podem ser outros eventos complexos, que são derivados dos eventos de entrada. CEP faz o processamento de fluxos de dados sobre eventos que acontecem [15] permitindo inferir uma conclusão a partir deles. Através do processamento de eventos é possível combinar dados de várias fontes para inferir eventos ou padrões que sugerem circunstâncias mais complexas [10].

Em contraste com os Sistemas de Gerenciamento de Banco de Dados, nos quais os dados são primeiro armazenados para depois serem consultados, o CEP armazena consultas contínuas e executa dados através delas, ou seja, ao invés de armazenar os dados, o CEP se concentra em analisar e processar continuamente os dados que passam usando as consultas armazenadas [49]. Assim CEP é usado para processar e analisar fluxos de dados em tempo real, bem como produzir resultados com baixa latência, mesmo em casos onde o fluxo de eventos é grande. CEP permite detectar situações em uma séries de eventos, possibilitando que ações especificas sejam tomadas de acordo com as condições previamente definidas através das regras CEP.

O processamento de eventos complexos pode ser aplicado a um amplo espectro de desafios de sistemas de informação. Exemplos dessa aplicação incluem:

- Aplicações que realizam análise online dos preços das ações para identificar tendências e prever valores futuros [28];
- Aplicações de monitoramento ambiental que processam dados brutos provenientes de redes de sensores para identificar situações críticas [28];
- Inteligência de negócios [9];
- Aplicações para cuidados com saúde [60];
- Detecção de fraudes e segurança [30];
- Sinais de RFID [60];
- Aplicações para sistemas de detecção de intrusão que analisam o tráfego de rede em tempo real para identificar possíveis ataques por meio da especificação de padrões de comportamentos de tráfego suspeito [28];

Desse modo o objetivo do processamento de eventos complexos é identificar eventos significativos (como oportunidades ou ameaças) e responder a eles o mais rápido possível [12].

2.3.1 Engines CEP e Modelo de Processamento de Eventos

Engines CEP são motores de processamento de eventos responsáveis por manipular os fluxos de dados. Existem várias *Engines* CEP disponíveis. Exemplos de *Engines* CEP incluem: Esper [58], WebLogic Event Server [59], Microsoft StreamInsight [1], Apache Flink [6], Red Hat Drools [46] e TelegraphCQ [8]. Dentre as *Engines* citadas, destacam-se as seguintes devido estarem em constante evolução, em relação às funcionalidades e optimização a que são sujeitas.

- **Esper**[2]: Uma das principais *Engines* CEP de código aberto, que está disponível como uma biblioteca Java, que permite que ela seja embutida em qualquer processo Java. Esper utiliza EPL (*Event Processing Language*), uma linguagem declarativa de consulta que estende o SQL com recursos de processamento de fluxo para especificação de consultas contínuas [10].

[2]http://www.espertech.com/esper/

- **WebLogic Event Server (WL EvS)**[3]: É um middleware para o desenvolvimento de aplicações orientadas a eventos baseada em Java que exigem baixa latência. Faz uso da EPL e as regras são definidas em um arquivo XML. O WL EvS baseia-se em uma arquitetura modular onde os componentes e aplicações do servidor são representados através de módulos [59].

- **Microsoft StreamInsight**[4]: É uma plataforma para desenvolver e implantar aplicativos CEP, que tem sua arquitetura e plataforma baseada em .NET Framework. O motor utiliza a memória cache para evitar incorrer em overhead através do armazenamento de dados para processamento e alcança um alto desempenho através da execução de consultas paralelas [1].

Algumas *Engines* CEP estão disponíveis como software livre, enquanto outras são proprietárias. No entanto, o que difere as *Engines* CEP umas das outras são as linguagens utilizadas para definir as primitivas e os tipos de eventos suportados. Quase todas os *Engines* CEP têm sua própria linguagem de processamento de eventos baseada em padrões. Os padrões especificam ações a serem executadas se as condições forem atendidas. As condições são representadas por padrões, que processam os fluxo de dados usando construtores e operadores.

O Modelo de Processamento de Eventos (*Event Processing Model - EPM*) é responsável pela manipulação dos eventos, armazena o repositório de regras e utiliza EPL do Esper [17]. EPL é uma das principais vantagens que o CEP oferece para manipular o processamento de eventos sobre os fluxos de dados [29], uma vez que, EPL é uma linguagem de processamento de eventos adaptada que permite expressar condições ricas e correlações entre eventos, possivelmente abrindo janelas de tempo, minimizando assim o esforço de desenvolvimento necessário para configurar sistemas que possam reagir a situações complexas [58]. EPL é similar ao SQL no uso das cláusulas *select e where*. Entretanto, as instruções EPL ao invés de tabelas utilizam fluxos de eventos. O EPM é composto por componentes específicos nos quais destacam: *Views*, *Event Operators*, *Event Pattern.*

- *Views* (visualizações de eventos): com cláusulas do tipo *win:length e win:time*, semelhante às tabelas em uma instrução SQL, as visualizações definem os dados

[3]http://www.oracle.com/technetwork/middleware/weblogic

[4]https://msdn.microsoft.com/pt-br/library/ee362541(v=sql.111).aspx

disponíveis para consulta e filtragem. Podem representar janelas sobre um fluxo de eventos, bem como podem classificar eventos, derivar estatísticas de propriedades de evento, eventos de grupo ou manipular valores de propriedade de evento exclusivos. A seguir são ilustrados alguns exemplos de regras em EPL.

Listagem 2.1: Exemplo EPL

```
Select src_ip, src_port from
      TCPpacketEvent.win:time(5 sec).win:length(5) having src_port = 80
```

Acima apresentou-se um exemplo de EPL que retorna os 5 últimos endereços IP de origem das conexões na porta 80 em um intervalo de tempo de 5 segundos. A *view* utilizada foi *win.time* que especifica o intervalo de tempo de chegada dos eventos e *win.length* que especifica o comprimento dos eventos chegados, nesse caso os 5 últimos endereços ips conectados.

- *Event Operators* (Operadores de eventos): nele existem cláusulas do tipo (*select, from, where, group by, order by, insert into, update, delete, pattern and output*).

  A seguir apresenta-se outra regra em EPL cuja a função do código ilustrado é calcular o valor médio do tamanho dos pacotes das últimas 100 conexões em uma janela de tempo de 10 minutos.

Listagem 2.2: Exemplo EPL

```
    select avg(length_packet) as avg_length, from
                  TCPpacketEvent.win:length(100).win:time(10 min)
```

- *Event Pattern* (Padrão de eventos): é um padrão que fornece uma correlação de eventos lógicos e temporais. Quando ocorre um evento ou vários eventos que correspondem à definição do padrão, a identificação deste padrão é bem-sucedida [58]. A EPL utiliza expressões de padrão de eventos. Essas expressões podem consistir em expressões de filtro combinadas com operadores de padrões.

  Existem 4 tipos de operadores de padrões:

  1 - Operadores que controlam o inicio e o fim da busca e a repetição de sub-expressões (*every, every-distinct, [num] e until*);

  2 - Operadores lógicos (*and, or, not*),

3 - Operadores temporais que operam a ordem do evento, seguido por (->) e

4 - Operadores *Guards* controlam o ciclo de vida das sub-expressões (*timer:within*, *timer:withinmax*, *timer:interval*, *timer:at*) e expressões *while*.

A cláusula EPL abaixo mostra um exemplo de utilização de padrão de eventos com alguns dos operadores descritos.

Listagem 2.3: Exemplo EPL

```
Select a.src_ip, length_packet from
pattern [every a=TCPpacketEvent -> b=UDPpacketEvent(src_ip = a.src_ip)
where timer:within(1 min)].win:time(30 min) having length_packet>65535
```

A regra apresentada acima realiza a seleção de todos os eventos com pacotes TCP seguidos de eventos com pacotes UDP, em que o tamanho de cada pacote seja maior que 65535 bytes para um mesmo endereço IP dentro de um minuto, ocorrendo nos últimos 30 minutos.

2.3.2 Janelas de Tempo

Janelas de tempo é um conceito importante para processar o fluxo de eventos de entrada em CEP. Precisamente, uma janela de tempo é um contexto temporal [11] [29] que subdivide um fluxo de eventos em intervalos de tempo. As janelas de tempo permitem-nos limitar o número de eventos considerados por uma consulta [58], dividindo o fluxo de eventos usando a *tag timestamp* de cada evento, de tal forma que inclui somente eventos que estão dentro do intervalo dado, por exemplo, *(now - Δ, now)*, onde *now* é o *timestamp* atual. Por exemplo, a declaração de janela de tempo *LocationUpdate* [SLIDE 30 segundos] cria automaticamente uma partição de contexto temporal que retém os últimos eventos *LocationUpdate* (Δ = 30 segundos) [49].

Já foram apresentados exemplos anteriormente que fazem uso de janelas de tempo. No entanto, abaixo é ilustrado outro exemplo prático para melhor entendimento o assunto. Considere a necessidade de calcular o valor médio do consumo de uma CPU em uma janela de tempo de um minuto. A declaração para resolver esse problema é mostrada abaixo.

Listagem 2.4: Exemplo de Janela de Tempo

```
Select avg(uso_cpu) as avg_cpu from
        cpuEvent.win:time(1 min) where uso_cpu > 1.3 avg(uso_cpu)
```

A rega acima irá emitir um evento de alerta todas as vezes que o uso da CPU for maior 30% que média dos eventos em um janela de tempo deslizante de um minuto.

Os dois principais tipos de janelas de tempo em CEP são:

- *Landmark windows (Time Batch):* têm um limite inferior fixo, enquanto o limite superior avança sempre que um novo evento entra no sistema. Esse tipo de janela fornece a capacidade de processar o fluxo de eventos em lote (*batch*). Ele armazena todos os eventos produzidos durante um intervalo de tempo Δ e aplica as consultas contínuas a todo esse conjunto de eventos [49].

  Precisamente, pode haver um atraso entre a hora de chegada do evento e seu tempo de processamento. Por exemplo, na Figura 2.5, enquanto os eventos W1 e W2 chegaram no tempo t=1 e t=3 respectivamente, eles serão processados somente quando o período do lote (*batch*) acabar (t = 4).

  Ao armazenar os eventos em *buffer*, é possível usar funções de agregação, primitivas, como *min* e *max*, para manipular o conteúdo do lote em um evento complexo. No entanto, se os eventos que precisam ser processados em lote são colocados em lotes adjacentes, a correlação entre os eventos não funcionará. Uma maneira de mitigar esta questão é aumentar o período de lote Δ. Porém isso causa uma sobrecarga de atraso adicional para processar um evento.

  O diagrama apresentado na Figura 2.5 ilustra o funcionamento de uma Janela de tempo em bloco. Para o diagrama, abordou-se uma consulta simples conforme mostra a Listagem 2.5.

Listagem 2.5: Exemplo de Janela de tempo em bloco

```
select * from TCPpacketEvent.win:time_batch(4 sec)
```

- *Slide Windows (Time Window):* É uma janela de tempo móvel (deslizante) que se estende até o intervalo de tempo especificado no passado com base na hora

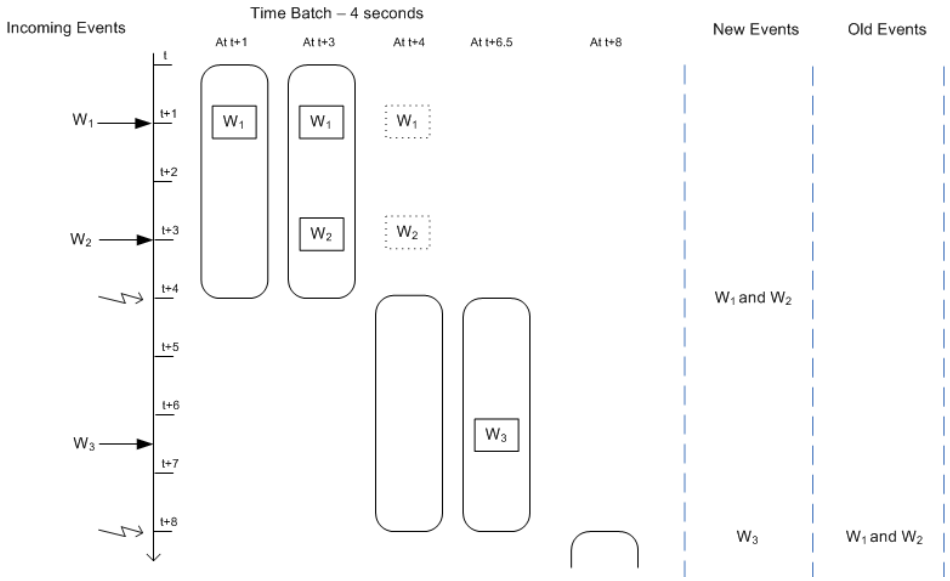

Figura 2.5: Exemplo de saída para uma declaração de Janela de tempo em bloco [58]

do sistema. As janelas deslizantes permitem manipular o número de eventos considerados por uma consulta movendo a janela de tempo junto com os eventos recebidos. Assim, em vez de ter períodos de lote predefinidos, os limites da janela de tempo deslizam para o *timestamp* do evento atual.

Mais especificamente, uma janela deslizante é uma janela de marco móvel que contém eventos nas unidades de tempo do passado Δ [49]. Para ilustrar esse cenário, considere o fluxo de eventos com uma janela de tempo deslizante de 4 segundos na Figura 2.6. O diagrama começa em um determinado momento *t* e exibe o conteúdo da janela de tempo em t = 4 e t = 5 segundos e assim por diante. No exemplo, quando t = 5, a janela de tempo inclui os eventos do segundo passado (t = 4) para o tempo atual.

Este movimento deslizante, ajustando os intervalos da janela de tempo ao evento atual que está sendo analisado, mitiga a questão de ter eventos correlacionados adjacentes em diferentes períodos *batch* (lote), ou seja, através da Figura 2.5 é possível observar que os eventos W2 e W3 não podem ser correlacionados em um mesmo lote, devido a separação em lotes distintos nos tempos t=3 e t=6.5.

Listagem 2.6: Exemplo de Janela de tempo

```
select * from TCPpacketEvent.win:time(4 sec)
```

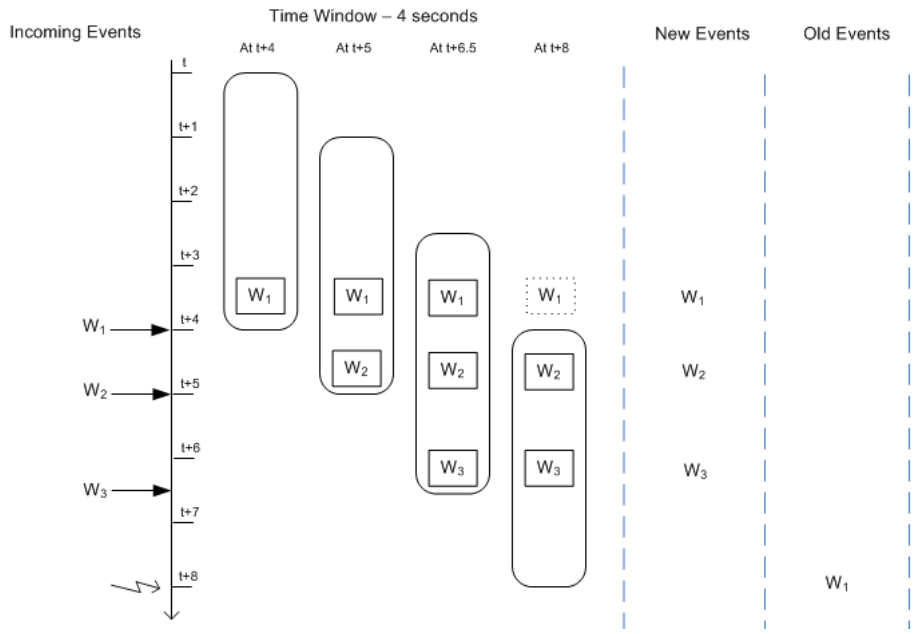

Figura 2.6: Exemplo de saída para uma declaração de Janela de tempo deslizante [58]

A Listagem 2.6 apresenta o funcionamento da Janela de tempo ilustrada na Figura 2.6. Ao deslizar a janela de tempo é possível incluir os eventos adjacentes anteriores que seriam colocados em lotes diferentes. Ao usar uma janela de tempo deslizante com primitivas em tempo real é possível vislumbrar os eventos passados e fornecer processamento de eventos contínuos e próximos ao tempo real [3].

2.3.3 Reconhecimento de Correspondência (Match Recognize)

Além dos padrões EPL disponíveis no EPM, o CEP permite utilizar padrão de Reconhecimento de Correspondência (*Match Recognize*), que apresenta uma maneira alternativa de especificar a detecção de padrões em comparação com o *Event Pattern* apresentado anteriormente. Reconhecimento de Correspondência é um modelo de consulta para correspondência de padrões com base em expressões regulares. O CEP pode aplicar padrões *Match Recognize* em tempo real após a chegada de novos eventos em um fluxo de eventos [58].

A sintaxe utilizada em padrões *Match Recognize* é apresentada a seguir:

Listagem 2.7: Sintaxe do Padrão Match Recognize [58]

```
match_recognize (
measures  measure_expression as col_name [, measure_expression as col_name ]
```

```
pattern ( variable_regular_expr [, variable_regular_expr] [,...] )
[ interval time_period [or terminated] ]
[ define  variable as variable_condition [, variable as variable_condition]]
)
```

Explicando a sintaxe do padrão de Reconhecimento de Correspondência tem-se:

- A palavra-chave *match recognize* inicia a definição do padrão e ocorre logo após as Cláusulas EPL *From e Select*.
- A cláusula *measures* define colunas que contêm expressões sobre as variáveis do padrão. As expressões podem referenciar colunas, variáveis únicas ou agregadas, bem como propriedades indexadas em grupos de variáveis.
- O componente *pattern* é usado para especificar expressões regulares. As expressões regulares são construídas a partir de nomes de variáveis, e podem usar quantificadores, tais como *, +, ?, *?, +?, ??, *repetition e* | *alteration*.
- As palavras-chave *interval* e *or terminated* são opcionais e podem controlar quanto tempo o motor CEP deve aguardar a chegada de eventos adicionais que podem fazer parte de uma sequência de eventos correspondentes antes de iniciar o processamento.
- *Define* é opcional, sendo usado para especificar as condições booleanas que definem alguns ou todos os nomes das variáveis que são declaradas no padrão.

Para exemplificar o uso de CEP com *match recognize*, apresenta-se um cenário para detecção de ataques de negação de serviço, no qual o IDS monitora a porcentagem de uso da CPU, coletando os dados com intervalos de tempo de 1 minuto. Nesse cenário, o IDS define 2 tipos de eventos a serem detectados:

- **Atenção**: Emite um alerta quando o uso total da CPU atingir um valor maior que 80% três vezes consecutivas.
- **Crítico**: Emite um alerta quando o uso total da CPU aumentar subitamente e a leitura inicial for maior que 80%.

Utilizando-se *match recognize* é possível criar duas regras para processar cada um dos padrões de eventos descritos. A seguir as Listagens 2.8 e 2.9 apresentam exemplos de consultas escritas em EPL para os dois tipos de eventos citados.

Listagem 2.8: EPL Match Recognize para detecção de eventos do tipo "Atenção".

```
Select * from cpuEvent.win:time(1 min)
       match_recognize (
           measures A as CPU1, B as CPU2, C as CPU3
           pattern (A B C)
           define
                A as A.uso_cpu > 80,
                B as B.uso_cpu > 80,
                C as C.uso_cpu > 80)
```

Listagem 2.9: EPL Match Recognize para detecção de eventos do tipo "Crítico".

```
Select * from cpuEvent.win:time(1 min)
     match_recognize (
        measures A as CPU1, B as CPU2, C as CPU3, D as CPU4, E as CPU5
        pattern (A B C D E)
        define
             A as A.uso_cpu > 80,
             B as (A.uso_cpu < B.uso_cpu),
             C as (B.uso_cpu < C.uso_cpu),
             D as (C.uso_cpu < D.uso_cpu),
             E as (D.uso_cpu < E.uso_cpu)
             and E.uso_cpu > (A.uso_cpu * 1/100)
```

No Padrão utilizado acima será considerado um estado crítico e o IDS emitirá um alerta quando o consumo da CPU aumentar subitamente atingindo valores maiores que 80% e se houver aumento no consumo cinco vezes consecutivas, sendo o valor deste aumento do evento final, maior que 10% do evento inicial.

2.4 Iptables

Iptables é uma *firewall* que faz parte do projeto *Netfilter* desenvolvido para sistemas operacionais baseado em Linux. Toda a configuração do *Iptables* é feita via

linha de comando, onde os usuários podem criar suas próprias regras de filtragem, para determinar quais pacotes são permitidos trafegar na rede [35]. Iptables suporta os protocolos TCP, UDP e ICMP.

Iptables possui três canais principais que realizam a filtragem dos pacotes: INPUT, FORWARD e OUTPUT. INPUT é responsável por filtrar todos pacotes que então chegando ao dispositivo, com o objetivo de entrar na rede. FORWARD é responsável por filtrar os pacotes que estão passando pelo dispositivo, de uma origem ao destino. E, por fim, OUTPUT, que tem a função de filtrar os pacotes que estão a saindo da rede e passando pelo dispositivo.

2.5 Considerações Finais

Este capítulo apresentou um resumo da fundamentação teórica, dando ênfase aos sistemas de detecção de intrusão, ataques de negação de serviço, *iptables* e processamento de eventos complexos. O entendimento das tecnologias apresentadas são fundamentais para o desenvolvimento da solução proposta.

Sobre sistemas de detecção de intrusão foram apresentados os tipos de sistemas existentes quanto à suas abordagens, dos quais foram citados: IDS baseado em host (*Host-based IDS - HIDS*), IDS baseado em rede (*Network-based IDS - NIDS*) e (*Wireless-based IDS - WIDS*) e IDSs baseados em Aplicação, que realizam a análise do comportamento da rede (*Network Behavior Analysis - NBA*). Sobre os ataques de negação de serviço, foram apresentadas as duas classes principais desse tipo de ataque: Ataques de redução da largura de banda e ataques de esgotamento de recursos, dando ênfase às principais características dos ataques *SYN Flood, UDP Flood, ICMP Flood, Land Atack e Smurf Atack*, os quais compõem essas classes.

Em relação ao CEP foi apresentado como o processamento de eventos complexos pode ser utilizado para detectar padrões de correlação e derivar situações a partir da análise de fluxos de eventos contínuos. Apresentou-se algumas vantagens do uso de CEP tais como sua capacidade de processar eventos em tempo real e com baixa latência. Mostrou-se ainda alguns exemplo de aplicações para CEP e alguns aspectos da linguagem EPL e *engines* utilizadas para o processamento de eventos. Por fim, apresentou-se o *Iptables*, ferramenta utilizada para filtrar e bloquear o tráfego da

rede. Na solução proposta neste trabalho, CEP é a base para o desenvolvimento dos mecanismos de filtragem, monitoramento e detecção de ataques.

3 Trabalhos Relacionados

Muitos trabalhos foram propostos na área de segurança de redes buscando fornecer mecanismos para detecção e prevenção de ataques contra a disponibilidade dos sistemas. [61]. Este capítulo apresenta alguns dos Sistemas de Detecção de Intrusão de código aberto disponíveis, bem como, alguns trabalhos que foram desenvolvidos e apresentados à comunidade científica e que mantêm alguma relação aos tópicos de estudos no contexto desta pesquisa, dando ênfase a detecção de ataques DoS e DDoS para Internet das Coisas.

Três IDSs populares de código aberto são SNORT [45], Bro [42] e Suricata [56]. Os três realizam a captura do trafego da rede através de um *Sniffer* de pacotes. O SNORT e o Suricata realizam a detecção de ataques utilizando uma base de assinaturas enquanto o Bro faz a detecção através de anomalias e também de assinatura [48].

3.1 Snort IDS

Snort é uma ferramenta de Detecção e Prevenção de Intrusão baseada em rede. Realiza o monitoramento de tráfego de pacotes em rede TCP/IP, bem como realiza a análise em tempo real sobre diversos protocolos [52]. O Snort pode ser executado em três modos diferentes:

- **Sniffer**: Realiza a leitura dos pacotes e apresenta em um fluxo contínuo no terminal de saída. Este modo é semelhante a ferramenta de análise de tráfego *tcpdump*;
- **Gravação de pacotes**: Neste modo o Snort realiza a captura do tráfego da rede e grava em um arquivo do tipo *pcap*, possibilitando assim a sua analise posteriormente.
- **Detecção de intrusão**: Neste modo o Snort faz a captura do tráfego da rede e analisa os pacotes buscando assinaturas que configuram algum tipo de ataque.

Snort é modular sendo composto por um decodificador de pacotes, um pré-processador, um motor de detecção e um gerador de alerta. No entanto, Snort não possui um Sniffer nativo, assim, faz uso da biblioteca *libpcap* para realizar a captura dos pacotes. O Snort trabalha com regras específicas, escritas em formato de texto, sendo definidas no arquivo "snort.conf" [20] e utiliza um mecanismo de processamento *single-threaded* [39] podendo utilizar apenas um núcleo de processador.

3.2 Bro IDS

O sistema Bro foi projetado e desenvolvido por Vern Paxson, que continua liderando o projeto em conjunto com uma equipe de pesquisadores e desenvolvedores no *International Computer Science Institute* (ICSI), na Universidade de Berkeley e no *National Center for Supercomputing Applications* na universidade Urbana-Champaign [41] [43]. Bro é um Sistema de Detecção de Intrusão em Redes (NIDS), que foi desenvolvido para sistemas Unix e utiliza o método baseado em anomalias e em assinaturas [43] para monitorar o tráfego de rede em busca de atividades maliciosas.

O IDS Bro está sob a licença BSD (*Berkeley Software Distribution*). Assim como o Snort, o Bro não possui uma biblioteca nativa para realizar a captura dos pacotes na rede, assim, o *libpcap* é utilizada para tal função. Ao ser iniciado, o Bro funciona da seguinte maneira:

- Realiza a captura dos pacotes a partir da interface de rede;
- Os pacotes brutos são ordenados em fluxos de dados que são colocados e analisados "analisadores" específicos para cada protocolo;
- Os registros são gravados no disco em tempo real e são categorizados de acordo com o protocolo;
- Alertas são gerados;

Os principais componentes do Bro são os analisadores. Há analisadores para funções especificas, como por exemplo para: detecção de anomalias, detecção de uso indevido e análise de conexão [48].

3.3 Suricata IDS

O Suricata é um sistemas para detecção de intrusão desenvolvido em 2010 pela *Open Information Security Foundation* (OISF), baseado em regras, o Suricata utiliza uma conjuntos de assinaturas desenvolvidas externamente para monitorar o tráfego de rede e fornecer alertas para o administrador do sistema quando ocorrem eventos suspeitos [56]. Projetado para ser compatível com componentes de segurança de rede existentes, o Suricata possui funcionalidade de saída unificada e oferece opções de biblioteca que aceita chamadas de outras aplicações.

A arquitetura do Suricata é composta dos três módulos a seguir, conforme descrito na Figura 3.1, o Decodificador de Protocolo (*Protocol Decoder*), o Motor de Detecção (*Detection Engine*) e a unidade de Alerta (*Alert*).

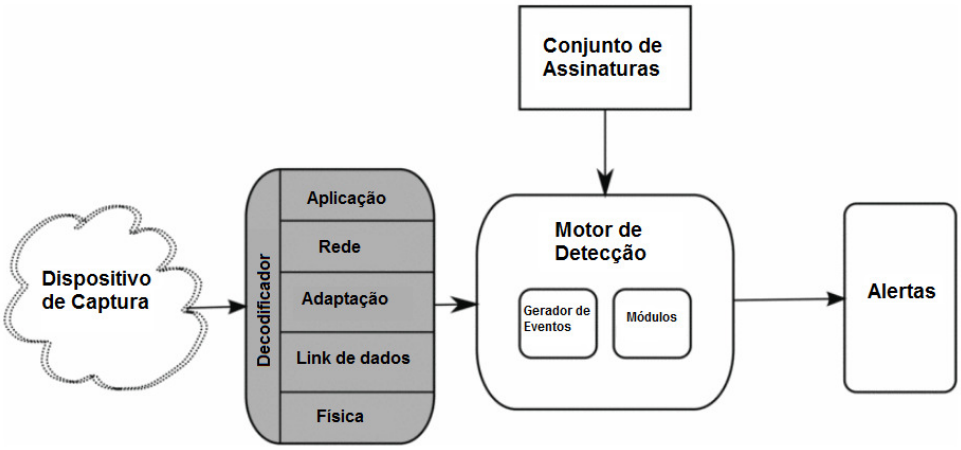

Figura 3.1: Arquitetura do IDS Suricata [19]

A arquitetura do Suricata funciona da seguinte maneira: Após a captura do tráfego os pacotes são interpretados por meio (*Protocol Decoder*). Os pacotes são decodificados e analisados procurando tráfego suspeito. Se algum tráfego suspeito for identificado um evento será registrado no (*Detection Engine*) com as informações do tráfego e este irá consultar a base de dados de assinaturas. Quando ocorre uma correspondência de assinatura, um alerta é gerado pelo sistema informando ao administrador da rede sobre o tráfego suspeito

Diferente do Snort e do Bro, o Suricata possui como vantagem uma arquitetura multi-thread, prometendo maior velocidade e eficiência na análise de tráfego da rede. Além da aceleração de hardware, o motor é construído para utilizar o

aumento do poder de processamento oferecido pelos mais recentes conjuntos de chips de CPU multi-core [51]. Sumariamente as características do IDS Suricata são:

- Detecção de tráfego malicioso em tempo real;
- Analise de tráfego a partir de arquivos offline;
- Processamento Multi-Thread;
- Detecção automática de protocolos;
- Sistema de prevenção de intrusão;
- Suporte IPv6;

Os três IDSs apresentados nesta seção fazem uso de uma grande lista de assinaturas que configuram as políticas para detecção do tráfego malicioso. No entanto, executar uma grande lista de regras pode ser viável para uma rede tradicional utilizando máquinas com alto poder computacional, todavia em pequenas redes, como redes IoT e em dispositivos que dispões de poucos recursos computacionais como um Raspberry PI, essa abordagem pode provocar uma sobrecarga e prejudicar o desempenho do sistema.

3.4 Detecção de Negação de Serviço em Internet das Coisas baseada no 6LoWPAN

Kasinathan [19] propôs uma arquitetura de IDS para detecção de ataques de negação de serviços para ambientes de Internet das Coisas, mas precisamente para redes 6LoWPAN (acrónimo para *IPv6 over Low power Wireless Personal Area Networks*), baseado no projeto *ebbits* da EU FP7[1].

A plataforma *ebbits* (*Enabling business-based Internet of Things and Services*), possui uma Arquitetura Orientada a Serviços (*Service oriented Architecture - SoA*) baseada em protocolos abertos e *middleware*. Fornece resolução semântica para a

[1] ebbits é uma plataforma que possibilita a convergência da Internet das Pessoas (IoP), Internet das Coisas (IoT) e da Internet dos Serviços (IoS) criando a "Internet de Pessoas, Coisas e Serviços (IoPTS)" para fins comerciais. http://www.ebbits-project.eu/

Internet das Coisas e apresenta uma nova ponte entre aplicações empresariais, pessoas, serviços e o mundo físico [47]. A Figura 3.2 apresenta a arquitetura do IDS proposta pelos autores.

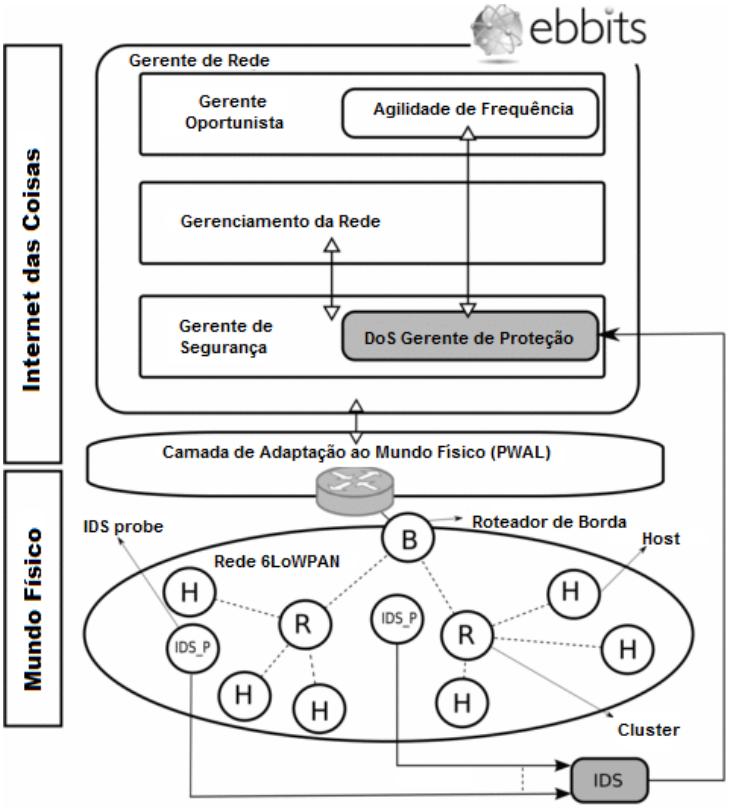

Figura 3.2: Arquitetura do IDS proposto por [19]

Nesta abordagem, o *IDS probe* ajuda o IDS a monitorar o tráfego da rede 6LoWPAN. Os componentes do Sistema são compostos por *Mundo Físico* que é representado pela rede 6LoWPAN e por *Internet das Coisas* que é representado por *Gerente de Rede ebbits*, que por sua vez é composto por três subcomponentes: *Gerenciamento da Rede, Gerente Oportunista e Gerente de Segurança*.

O *Mundo Físico* representa a rede 6LoWPAN, onde existe uma variedade de *hosts* (H) conectados aos seus respectivos *cluster* (R). Os dados trafegados na rede são enviados para o *Gerente de Rede* através do roteador de borda (B). A seguir são apresentados os componentes do *Gerente de Rede ebbits*.

- *Gerenciamento da Rede*: realiza serviços de monitoramento e configuração de rede, através do qual é possível obter informações de desempenho, tais como: nível de interferência, latência e colisões.

- *Gerente Oportunista*: oferece otimização de comunicação para o *framework ebbits*, permitindo que o sistema funcione com a melhor comunicação de rede disponível em todos os momentos. Permite à rede a identificação do nível de interferência analisando os estados de ocupação do canal em tempo real, possibilitando determinar o melhor canal disponível.

- *Gerente de Segurança*: fornece mecanismos de segurança para a comunicação na rede através da execução de politicas de segurança e do uso de criptografia. Este componente possui dois subcomponentes: *DoS Gerente de Proteção* e IDS.

  - *DoS Gerente de Proteção*: O gerenciador de proteção DoS é responsável por receber os alertas gerados pelo IDS quando ocorrem tentativas de intrusão. É também responsável pela extração de informações de outros gerenciadores ebbits, (*gerente oportunista e gerenciamento da rede*), tais como taxa de transferência e taxa de perda de pacotes, para determinar se o tráfego suspeito é realmente um ataque real;
  - IDS: O IDS proposto é um IDS baseado em rede (NIDS). É responsável pelo monitoramento e análise dos pacotes. Este agente IDS é responsável por processar o tráfego capturado por múltiplos IDS_Ps conforme mostrado na Figura 3.2 e verificar os resultados são característicos de ataques.

Os autores utilizaram o Suricata IDS para desenvolvimento da proposta (componente IDS apresentado na Figura 3.2). Para validação do sistema, foi desenvolvida uma aplicação para a realização de ataques *UDP Flood* baseado em IPv6 utilizando o sistema operacional *Contiki*[2]. A Figura 3.3 mostra os resultados dos alertas gerados para as taxas de verdadeiros positivos.

Os ataques foram realizados no período de 1 (um) minuto, sendo realizados por um número progressivamente crescente de Nós (Mp) executando um ataque de inundação em direção a um nó 6LoWPAN alvo. Primeiro foi utilizado apenas um dispositivo (Mp) para a realização do ataque, depois dois dispositivo, até chegar a utilização de cinco (05) dispositivos para a realização do ataque. Os ataques foram repetidos três vezes com tempo de 1 minuto para cada ataque para avaliar o desvio padrão dos alertas para cada Nó Mp: os resultados estão representados na Figura 3.3.

[2]Sistema operacional de código aberto para a Internet das coisas. (http://www.contiki-os.org/)

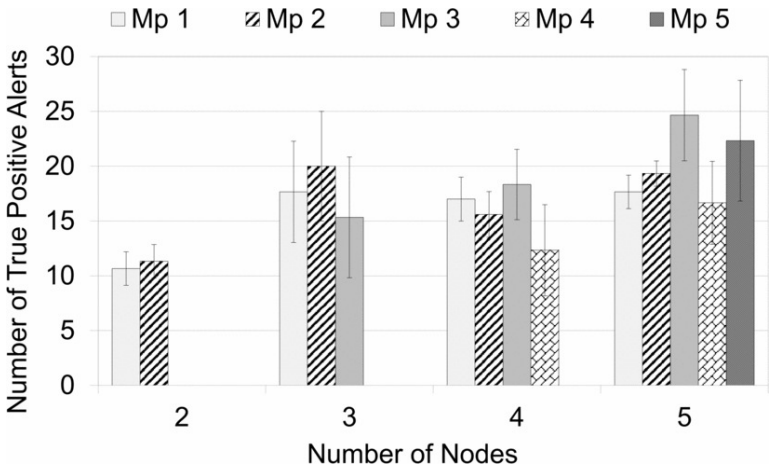

Figura 3.3: Taxa de Verdadeiro-Positivo para *UDP Flood* [19]

Pode-se notar que o IDS proposto é capaz de detectar os ataques de inundação e que a taxa de detecção de cada nó Mp aumenta a medida que mais nós Mp estão envolvidos no mesmo ataque.

De formar resumida as principais contribuições da proposta dos autores é o DoS Gerente de Proteção e o IDS, que estão interligados ao Gerente de Redes, através do Gerente de Segurança. A arquitetura de detecção proposta foi projetada para detectar ataques de negação de serviço, mas precisamente ataques UDP *Flood*. O Gerente de Proteção recebe alertas do IDS quando ocorrem tentativas de invasão. Em seguida ele extrai as informações (taxa de interferência, taxa de queda de pacotes, etc.) de outros gerentes da rede ebbits (Gerenciamento de Rede e Gerente Oportunista) e as analisa para confirmar um ataque real. No entanto, o método utilizado pode gerar altas taxas de consumo de recursos e provocar um baixo desempenho, podendo comprometer eficacia do sistema, uma vez que suricata IDS não foi projetado para dispositivos que dispõem de poucos recursos computacionais, como os que são utilizados em ambientes de IoT.

3.5 Pi-IDS: Avaliação de Sistemas de Detecção de Intrusão de código aberto no Raspberry Pi

Kyaw et al. [23] concentram seus estudos no Raspberry Pi, comparando o desempenho dos IDSs Snort e Bro. Os autores configuraram o Snort e o Bro para usar

apenas os módulos de assinaturas para detecção dos ataques de *Port Scan, SYN Flood e ARP Spoofing*.

A metodologia aplicada na avaliação de desempenho dos IDSs Snort e Bro foi o método de pesquisa experimental, onde os autores formularam as seguintes questões de pesquisa com o objetivo de avaliar o desempenho dos IDSs em execução no Raspberry Pi.

1 - Q1. Um Raspberry Pi é capaz de executar o Snort IDS e o Bro IDS?

2 - Q2. Os IDSs que estão sendo executados no Raspberry Pi são capazes de detectar ataques de redes?

3 - Q3. Pode um Raspberry Pi lidar com todo o tráfego de rede?

4 - Q4. O desempenho do Snort é melhor do que o desempenho do Bro IDS executando em um Raspberry Pi?

Para responder as questões da pesquisa, os autores utilizaram uma arquitetura para o experimento composta por cinco (5) servidores diferentes e um Raspberry Pi conectado a um *switch Cisco Catalyst 2950*, conforme pode-se observar na Figura 3.4.

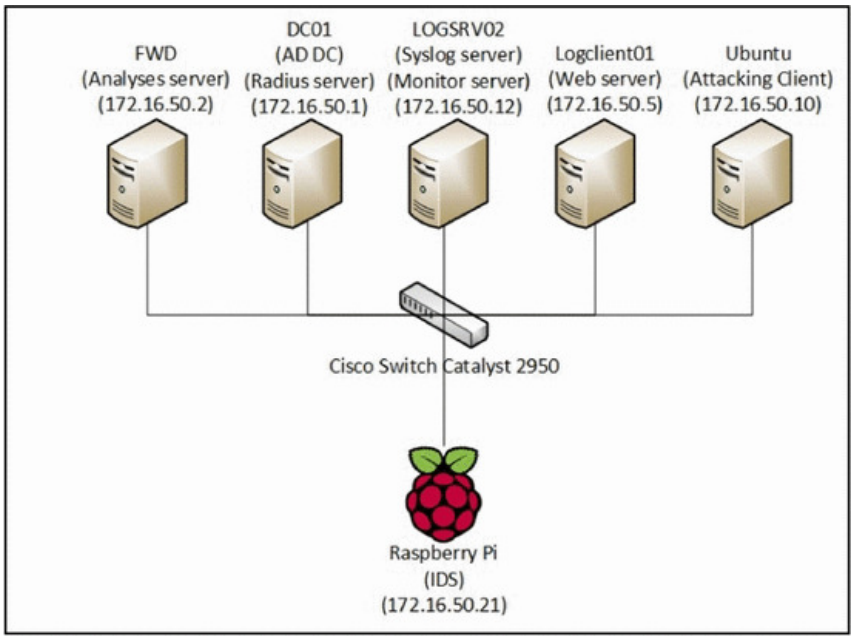

Figura 3.4: Arquitetura utilizada para o experimento. [23]

DC01 é o servidor responsável por controlar o domínio da rede e também executa o serviço RADIUS (*Remote Authentication Dial-In User Service*). O servidor *LOGSRV02* executa o *Splunk*[3] para coletar *logs* de todos os dispositivos da LAN, incluindo o Pi-IDS, enquanto o *Logclient01* é o servidor web. A máquina *Ubuntu* é utilizada para simular os ataques de rede e o servidor *FWD* realiza a analise dos logs gerados pelos IDSs Snort e Bro no Pi-IDS. Após o tráfego da rede ser capturado pelo Pi-IDS, os *logs* gerados com informações dos ataques são coletados pelo servidor de registro do *Splunk* (*LOGSRV02*) e são analisados usando o aplicativo*Wireshark* que é executado no servidor *FWD*. O Snort e o Bro são executados no Rapsberry Pi sendo utilizado para capturar o tráfego na rede e gerar os logs dos ataques.

O experimento é realizado utilizando vários cenários de rede combinando tráfegos em segundo plano com diferentes tamanhos de pacotes. Por exemplo, a Tabela 3.5 mostra que o ataque SYN Flood foi realizado com pacotes de tamanhos 0, 80 e 160 kilobytes e com tráfego em segundo plano de 30%, 60% e 90%. Para os ataques de *ARP Spoofing* e *Port Scanning* utilizou-se apenas o tráfego em segundo plano, uma vez que os tamanhos do pacotes não são configuráveis nesses tipos de ataques.

Ataques de Redes	Tamanho do Pacote	Tráfego em Segundo Plano
Ataque SYN Flood	0	30%
	80	30%
	160	30%
	0	60%
	80	60%
	160	60%
	0	90%
	80	90%
	160	90%
Ataque ARP Spoofing		30%
		60%
		90%
Ataque de Port Scanning		30%
		60%
		90%

Figura 3.5: Cenários de Ataque. [23]

Os autores utilizam a ferramenta *PyLoris* para gerar tráfego em segundo plano com o objetivo de testar a vulnerabilidade do servidor da Web para ataques de

[3]Ferramenta para gerenciamento de logs. (https://www.splunk.com)

inundação como por exemplo o *SYN Flood*. *PyLoris* é uma ferramenta desenvolvida em *Pyton* para testar a vulnerabilidade de um servidor aos ataques de negação de serviço.

Os autores utilizam alguns fatores que consideram importantes ao realizar a comparação da performance dos IDSs, que são: uso CPU, uso de memória RAM, taxa de perda de pacote, e o número de ataques detectados. A Figura 3.6 mostra o desempenho médio do Snort e do Bro referente as métricas citadas anteriormente.

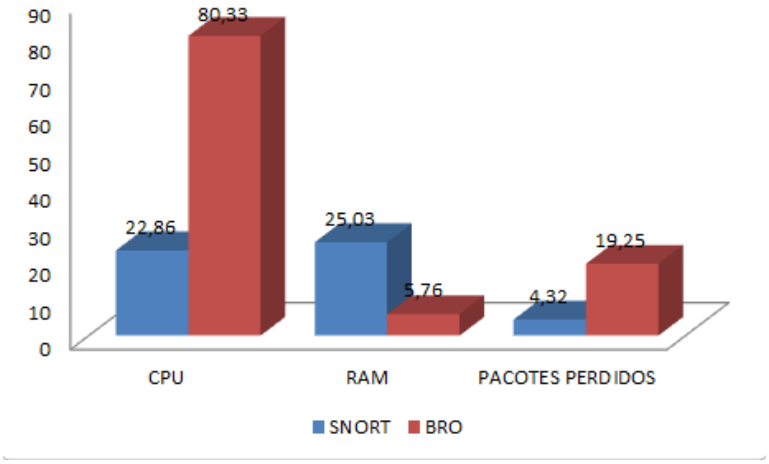

Figura 3.6: Desempenho médio do Snort e do Bro em um Raspberry Pi [23]

Os resultados mostram que o Snort usa 22,86% de CPU e 25,03% de RAM para detectar ataques de rede abordados, enquanto o Bro usa 80,33% de CPU e 5,76% de RAM. Com relação a taxa de pacotes perdidos, o Snort teve 4,32%, uma taxa bem inferior ao Bro que foi de 19,25%. Os autores finalizam o trabalho como as seguintes conclusões referente as questões da pesquisa.

1 - R1. O Snort e o Bro podem ser executados em um Raspberry Pi.

2 - R2. O Snort e o Bro são capazes de detectar todos os ataques de rede abordados nesta pesquisa.

3 - R3. Um Raspberry Pi pode falhar ao processar todo o tráfego de rede enquanto executa os IDSs Snort ou Bro, devido suas limitações referente ao poder de processamento, memória RAM e velocidade de leitura e escrita em disco.

4 - R4. O desempenho do Snort é melhor do que o desempenho do Bro referente ao uso da CPU e ao número de pacotes perdidos. Porém o Bro teve melhor

performance no referente ao uso de memória RAM conforme pode ser observado na Figura 3.6.

A metodologia de teste utilizada nesta dissertação também aborda vários cenários combinando tráfegos em segundo plano, no entanto ela difere deste trabalho a medida que utiliza-se um *Dataset* com dados de ataques de negação de serviço para avaliar o sistema proposto. Além disso, conforme mostra o Capítulo 5, nossos resultados mostram que CEPIDS pode ser executado no Raspberry Pi como sistema de detecção de intrusão em diferentes cenários de rede, enquanto os autores [23] concluem que o Snort e o Bro podem comprometer a performance do *Raspberry Pi* ao desenvolverem essa função.

3.6 Modelo de IDS com Processamento de Eventos Complexos para Internet das Coisas

Chen Jun [17] propôs um IDS baseado em processamento de eventos complexos para resolver problemas de intrusão em tempo real em redes de Internet das Coisas. Nessa abordagem, os autores projetaram a arquitetura IDS com base no Modelo de Processamento de Eventos (*Event Processing Model - EPM*), onde são utilizadas 3 partes principais do modelo. (1) Operadores de eventos, onde existem cláusulas tipo: selecionar como, de, onde, agrupar por e ordenar por. (2) Visualização, que é usado para filtrar e unir os eventos. (3) Integração com banco de dados relacional, que suporta a incorporação de instruções SQL.

O IDS proposto é baseado em regras que são armazenadas no Repositório de Padrões de Regra, usa SQL e EPL como referência. O módulo principal do sitema é o *Event Pattern Repository (EPR)*, que armazena o modelo de processamento de eventos (EPM).

A Figura 3.7 mostra a arquitetura proposta pelos autores. O IDS coleta os eventos de entrada dos dispositivos IoT, em seguida, processa e analisa os dados e produz relatórios de segurança para o motor de ação.

A seguir explicam-se os módulos apresentados na arquitetura.

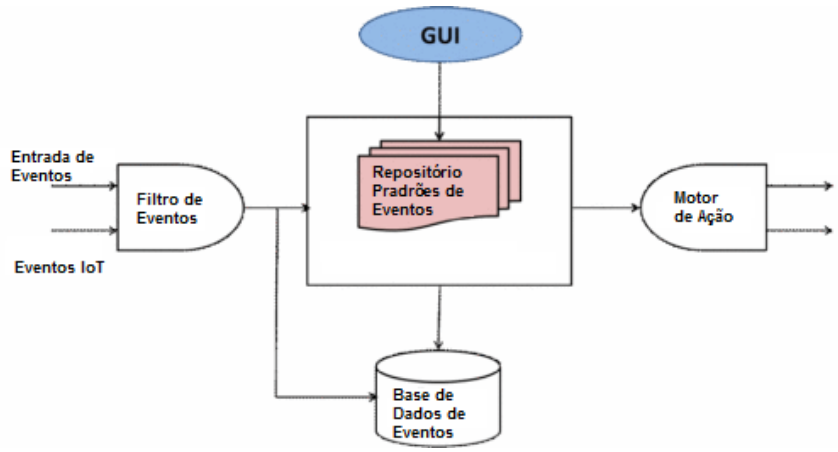

Figura 3.7: Arquitetura IDS para IoT [17]

- **Filtro de Eventos:** componente responsável pela captura dos pacotes trafegados na rede;
- **Processador de Eventos Complexos:** responsável pela manipulação dos eventos do EPM. EPM armazena todas a regras EPL sendo responsável por tratar os modelos de eventos já ocorridos, realizando comparações com novos fluxos de eventos. Caso haja correlação entre os eventos, um novo evento é criado;
- **Motor de Ação**: Responsável pelas ações a serem tomadas conforme os eventos gerados pelas regras CEP;
- **Banco de Dados**: responsável pelo armazenamento dos *logs* dos eventos filtrados e resultados dos eventos processados pelo CEP;

Antes de obter informações úteis dos fluxos de eventos, os dados brutos dos *gateways* IoT ou stream de servidores são filtrados pelo Filtro de Eventos. Em seguida, os eventos são extraídos e formulados a partir de dados IoT por um modelo de estrutura de dados do evento (i.e., Pojo, XML ou ObjectArray), depois é processado pelo motor CEP para análise de tráfego malicioso.

A Figura 3.8 mostra a regra CEP utilizada pelos autores para identificação do tráfego malicioso e significa que quando o comprimento total do pacote de dados é superior a 65536, esse acesso à rede pode ser um ataque do tipo *Land Attack*. Os eventos que acionam a regra CEP são gerenciados pelo *Motor de Ação* que coleta e registra os *logs* para atividades de invasão e adicionam os endereços IP de origem a uma lista de bloqueio.

```
/*Code for Land Attack Listener*/
Configuration config = new Configuration();
config.addEventType("ReadPacket",
ReadPacket.class.getName());
//Creating CEP engine instance
EPServiceProvider cepEngine =
EPServiceProviderManager.getProvider("esper-Land",
config);
//Creating EPMStatement
String rule = "select * from ReadPacket where packetLength
> 65536";
EPMStatement stm =
epService.getEPAdministrator().createEPL(rule);
//Binding event listener to EPM pattern
stm.addListener(new UpdateListener(){
update(EventBean[] newEvent, EventBean[] oldEvent)(){
System.out.println("Receive tag information from
reader007");
}
}
```

Figura 3.8: regra CEP para *Land Attack* [17]

Os autores usam a classe *Configuration* em Java para registrar as consultas do EPM e em seguida criam uma instância correspondente no mecanismo CEP por *EPServiceProviderManager object*. As instruções de consultas do EPM são adicionadas ao mecanismo CEP para que elas estejam trabalhando nos fluxos de eventos de entrada. A proposta inicial dos autores foi projetada para detectar apenas um tipo de ataque de negação de serviço, o *Land Attack*. No entanto, o IDS proposto considera apenas o tamanho do pacote e o endereço de origem do *host* atacante no processo de identificação do tráfego malicioso.

Para verificar o desempenho do IDS proposto, um experimento comparativo com abordagens de IDS tradicional é realizado para ver se o uso do CEP pode atender aos requisitos do ambiente IoT, conforme apresentado na Tabela 3.1. Neste contexto, o IDS tradicional é definido pelo autores para referenciar o mesmo IDS proposto, porém sem a utilização da tecnologia CEP, onde ambos utilizam o mesmo método de detecção, considerando a origem do tráfego e o tamanho dos pacotes trafegados.

Os resultados apresentados pelos autores, mostram que o IDS usando a tecnologia CEP consumiu mais recursos de CPU do que o IDS tradicional ao processar os fluxos de dados de IoT. No entanto, o IDS baseado em CEP consumiu menos memória e levou menos tempo de processamento que o IDS tradicional, o que

Tabela 3.1: Comparação IDS baseado em CEP e IDS tradicional [17]

Tipo de Dados	Escala de Dados	Uso da CPU(%)	Consumo de RAM (MB)	Tempo de Processamento(ms)
IDS CEP	200	48	556	287
	400	50	684	368
	800	62	730	422
IDS Tradicional	200	42	782	477
	400	45	964	2042
	800	57	1064	8688

efetivamente prova que CEP pode melhorar o desempenho em tempo real do sistema IDS em cenários baseados em eventos.

3.7 Kalis - Um Sistema de Detecção de Intrusão Auto-adaptável Baseado em Conhecimento para Internet da Coisas

Midi et al. [33] propõem um IDS para detecção de intrusão em IoT, auto-adaptável baseado em conhecimento. Na proposta dos autores, o sistema coleciona conhecimento sobre os recursos da rede de forma autônoma e utiliza esse conhecimento para configurar dinamicamente as técnicas de detecção.

O sistema proposto utiliza um modelo conceitual para a detecção de intrusão orientada pelo conhecimento baseado nos seguintes conceitos-chave:

- **Observação:** uma informação é analisada observando os eventos disponíveis (i.e., a frequência de um tipo de tráfego, um campo de encaminhamento especial nos pacotes interceptados, uma mudança na força do sinal de um nó, ...);
- **Característica:** observa-se as característica específicas dos dispositivos e redes monitoradas (i.e., redes multi-salto vs. redes de único-salto, rede móvel vs. rede estática, ...);

- **Sintoma:** caso particular de observação que pode ser associado a um incidente de segurança potencial (i.e., perdas de dados ou inconsistências, duplicação ou alteração de pacotes, ...);

- **Técnica de detecção:** Técnicas de detecção de um incidente, ataque ou anomalia de segurança conhecido, realizada de forma proposital ou não, que deve desencadear atividades de resposta, como um alerta para um usuário e/ou ações de resposta automática (como a re-transmissão de pacotes, e isolamento do dispositivo).

A partir desse conceitos, a abordagem dos autores para detecção de intrusão orientada por conhecimento segue o modelo conceitual apresentado na Figura 3.9, na qual *usando a observação O, o sistema pode determinar a característica F dos dispositivos e da rede monitorada. Dado o conhecimento sobre F*, o sistema pode determinar *qual entre as técnicas de detecção {D1, D2, ...Dn}* deverá ser ativada. Quando apenas as técnicas de detecção corretas estiverem ativas, elas processarão as informações disponíveis para detectar os incidentes de segurança pelo(s) sintoma(s) S, melhorando assim a precisão do sistema.

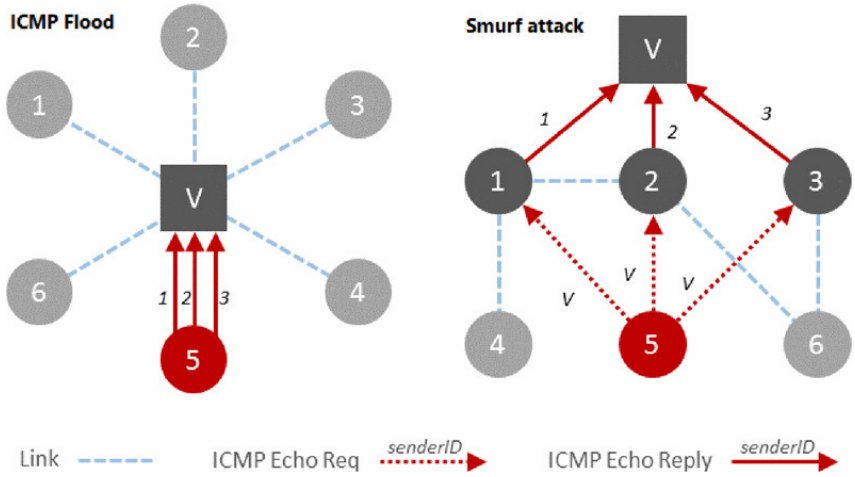

Figura 3.9: ICMP Flood vs. Smurf Atack. [33]

Os autores ilustram o modelo proposto através de um exemplo com dois possíveis ataques: *Ataque ICMP Flood e Smurf Atack* conforme pode ser observado na Figura 3.9. Durante um ataque de inundação ICMP, um único nó atacante envia muitas mensagens *ICMP Echo Request* para a vítima, usando vários endereços IPs diferentes

como remetente. Em um *Smurf Atack*, o atacante envia mensagens *ICMP Echo Request* ao endereço de *broadcast* usando a identidade da vítima como remetente, assim todos os nós da rede responderão com as mensagens *ICMP Echo Reply* dirigidas à vítima. Para um observador externo, esses dois ataques mostram os mesmos sintomas(S): uma grande quantidade de mensagens de resposta *eco ICMP* enviadas para o nó da vítima. No entanto, o *Smurf Atack* não pode ser executado em redes de um único salto, e esse conhecimento pode ser utilizado para conseguir uma detecção precisa.

Os autores explicam o modelo da seguinte maneira: *Considere o ataque e a topologia mostrada no lado esquerdo da Figura 3.9 e suponha que o nó 5 realize um ataque de ICMP Flood no nó da vítima V. Ao observar o tráfego, o sistema pode reconstruir parte da topologia da rede monitorada e determinar que ela é uma rede de um único salto. Dado esse conhecimento, o sistema ativa a técnica de detecção para ataques de ICMP Flood e não para Smurf Atack. Após a detecção de uma alta quantidade de mensagens ICMP Echo Reply para o nó, o único módulo ativo detectará inequivocamente o ataque de inundação ICMP sofrido.*

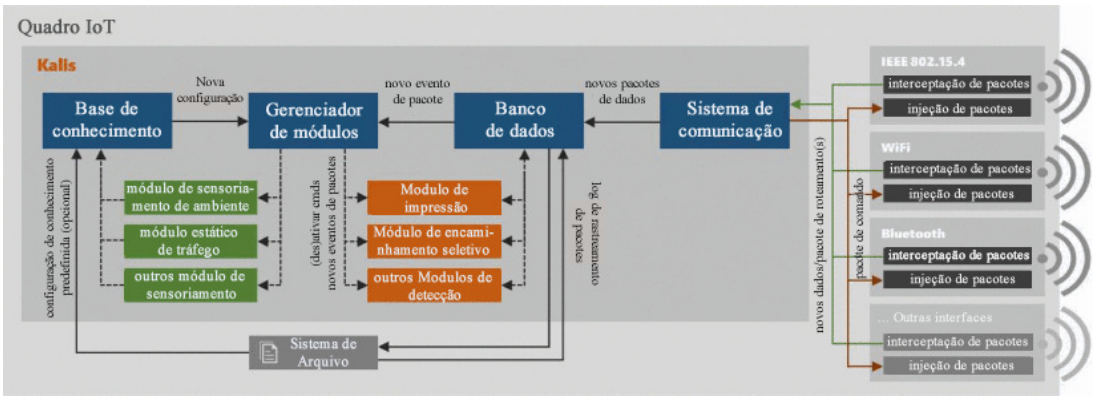

Figura 3.10: Arquitetura do Kalis. [33]

A Figura 3.10 mostra a arquitetura do sistema Kalis, proposta pelos autores [33], onde:

- **Sistema de Comunicação:** Examina todo o tráfego em todas as interfaces suportadas. O modelo atual inclui *ZigBee/XBee/6LoWPAN (on IEEE 802.15.4), WiFi (on IEEE 802.11), and Bluetooth.*
- **Banco de Dados:** escuta os eventos do Sistema de Comunicação em pacotes recém-capturados, administra o histórico do tráfego para acesso dos módulos e opcionalmente registra o tráfego no disco ou memória, conforme configurado pelo usuário.

- **Base de Conhecimento:** Este componente armazena todas as informações disponíveis sobre os recursos dos dispositivos e das redes monitoradas em um único lugar centralizado e disponibiliza essa informação para o módulo de Detecção e o Gerenciador de Módulos.

- **Gerenciador de Módulos:** Este componente coordena todos os módulos, ativando/desativando-os conforme necessário, dependendo das mudanças na *Base de Conhecimento*, roteando novos eventos de pacotes para todas as partes interessadas e coletando alertas sobre incidentes detectados. No sistema proposto, toda funcionalidade específica da rede ou funcionalidade específica de ataque é implementada em um módulo independente.

Os autores comparam a abordagem baseada no conhecimento com a de um IDS tradicional. Para uma equidade em relação às técnicas de detecção, o IDS tradicional é simulado executando o próprio sistema proposto pelos autores sem fazer uso da Base de Conhecimento, utilizando todos os módulos ativos. Os autores também comparam o Kalis com Snort, usando regras personalizadas, juntamente com o conjunto de regras da comunidade padrão do Snort para detectar os ataques simulados. A Tabela 3.2 mostra o desempenho médio do Kalis em comparação a abordagem tradicional e ao Snort.

Tabela 3.2: Média de eficácia e desempenho

	Trad. IDS	**Snort**	**Kalis**
Taxa de Detecção	48%	89%	91%
Acuracia	75%	76%	100%
Uso de CPU	0,22%	6,3	0,19%
Uso de RAM(kb)	23961,06	101978,24	13978,62

Os resultados mostram que Kalis é leve em termos de requisitos de CPU e RAM e que a abordagem baseada no conhecimento proposta neste trabalho supera o IDS tradicional e o Snort em todas as métricas. Os autores apresentam ótimos resultados para o sitema Kalis, no entando, o trabalho não informa quais as configurações de *Hardware* utilizada (i.e., CPU e Mémoria RAM) para realização do experimento, não ficando claro qual a real utilização da CPU.

4 Solução Proposta

Este capítulo tem como objetivo apresentar o CEPIDS, um Sistema de Detecção de Intrusão que usa regras CEP para resolver problemas de segurança em ambientes de IoT. A arquitetura do CEPIDS será apresentada, bem como alguns diagramas UML, aspectos da implementação, cenário de ataque para o experimento e as regras CEP utilizadas para a detecção dos ataques.

4.1 Arquitetura e Funcionalidades dos Componentes

A arquitetura do sistema CEPIDS é composta por três camadas conforme mostra a Figura 4.1. O sistema verifica fluxos de eventos como entrada a partir de informações coletadas do tráfego, processa, analisa, e produz relatórios de segurança para o motor de ação conforme as regras escritas em CEP.

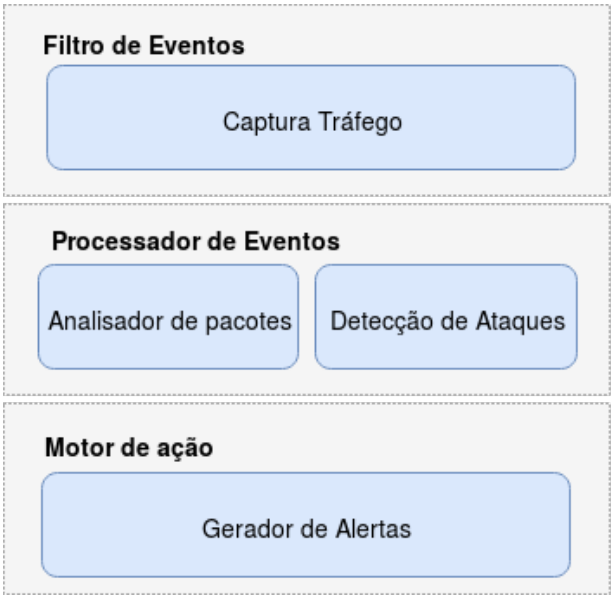

Figura 4.1: Arquitetura CEPIDS

- A camada **Filtro de Eventos** é usada para monitorar e coletar o tráfego da rede e a ocorrência de eventos. O fluxo de dados é capturado a partir de uma placa

de rede do sistema e os dados filtrados são encaminhados para o processador de eventos.

- A camada **Processador de Eventos** é composta pelos módulos *analisador de pacotes* que realiza a análise das características dos pacotes capturados e *detecção de ataques*, que é responsável por determinar qual tipo de ataque está acontecendo. Os resultados do processamento são gerados automaticamente e continuamente.

- O módulo Gerador de Alertas faz parte da camada **Motor de Ação**, sendo responsável por tratar os eventos que desencadearam regras CEP, emitir alertas sobre atividades de intrusão e bloquear o acesso para conexões suspeitas.

A Figura 4.2 apresenta o fluxograma de execução do CEPIDS.

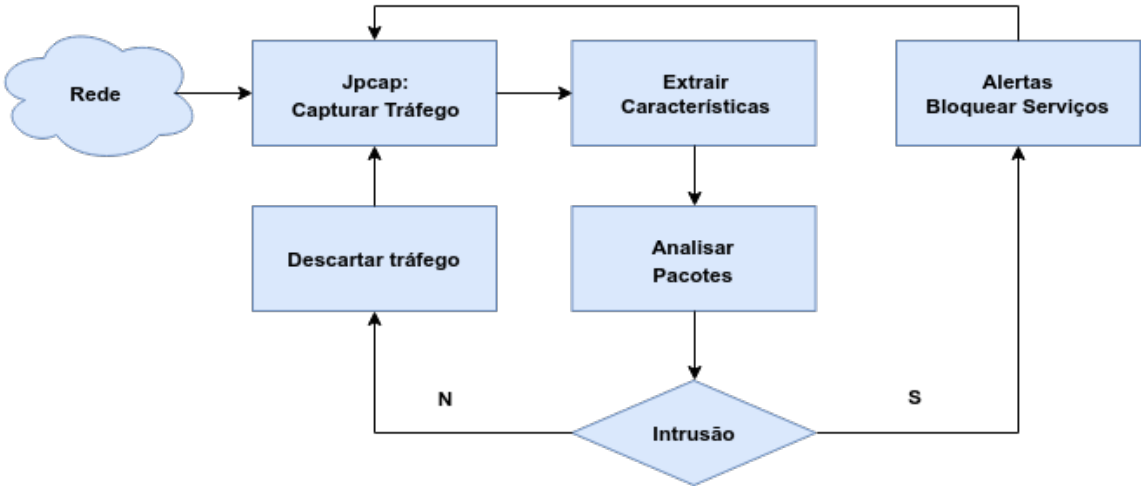

Figura 4.2: Fluxograma de Execução

Quando o sistema é iniciado, uma rotina de captura de pacotes é realizada pelo método *startPacketCapture()* (veja Figura 4.5). Após a captura dos pacotes, as etapas seguintes são:

- Extração das características dos pacotes capturados e análise do fluxo de dados pelo processador de eventos, em que o método *handle()* verifica as características extraídas dos pacotes (i.e., endereços IP de origem e destino, número total de pacotes, tamanho médio de pacote, taxa de pacote por segundo) conforme tipo de protocolo do pacote e determina se o tráfego é malicioso ou não.

- Após a análise ser realizada, é verificado se o resultado caracteriza um ataque. Caso o resultado da análise não detecte nenhuma característica de intrusão, os dados analisados são descartados e o processo de captura do tráfego é reiniciado.

- Caso contrário, o método *handle()* irá fazer a identificação do tipo de ataque que está ocorrendo conforme as regras CEP criadas para cada tipo de ameaça. Uma vez que o tipo de ataque é identificado, o sistema envia um alerta através do método *update()* de acordo com o tipo de ataque detectado. O passo seguinte é realizar o bloqueio do IP atacante para novas solicitações de conexão através do método *Block()*.

A Figura 4.3 apresenta o diagrama de sequência do CEPIDS para detecção de ataques do tipo *SYN Flood* para um melhor entendimento do funcionamento do sistema.

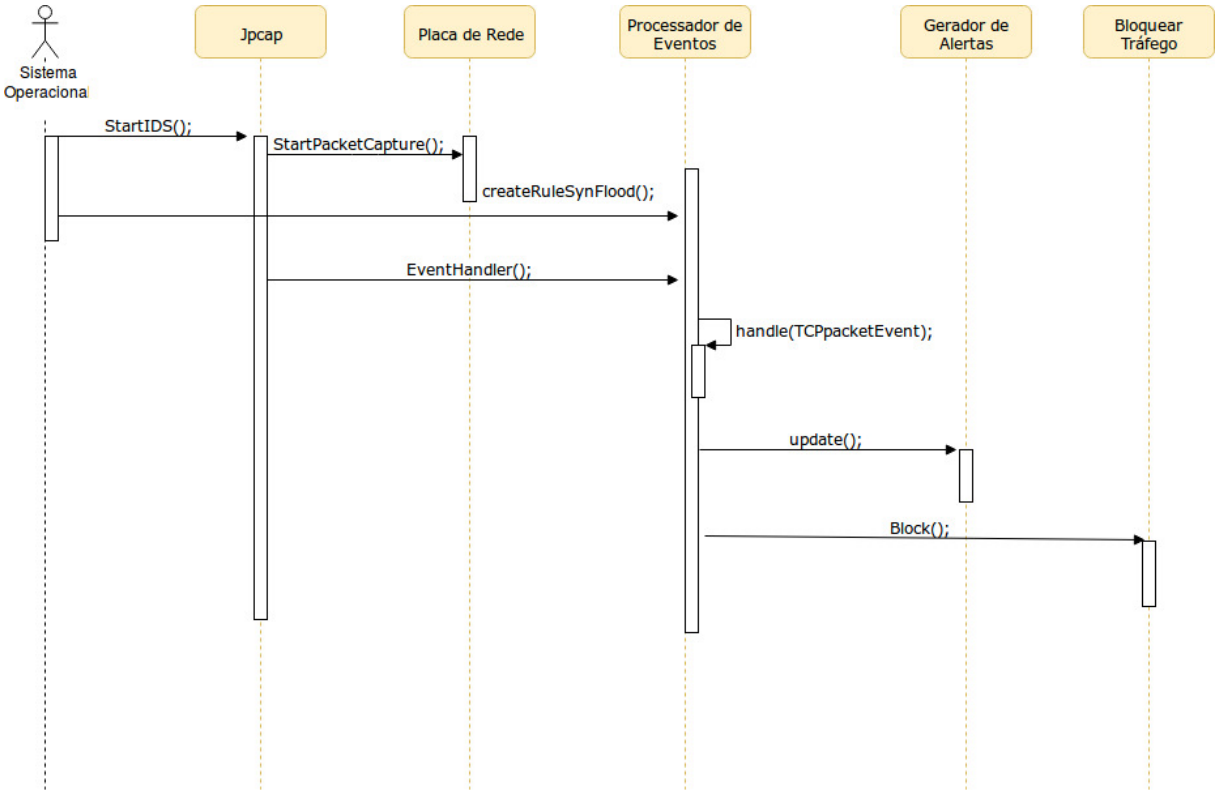

Figura 4.3: Diagrama de Sequência para detecção de ataques *SYN Flood*

O diagrama de sequência para detecção dos demais ataques abordados nesta pesquisa seguem o mesmo modelo do diagrama apresentado na Figura 4.3. Para detectar outro tipo de ataque, é necessário modificar a chamada do método que cria a regra CEP *createRuleSynFlood()* que pertence a classe *EventHandler*, conforme apresentado na Figura 4.5.

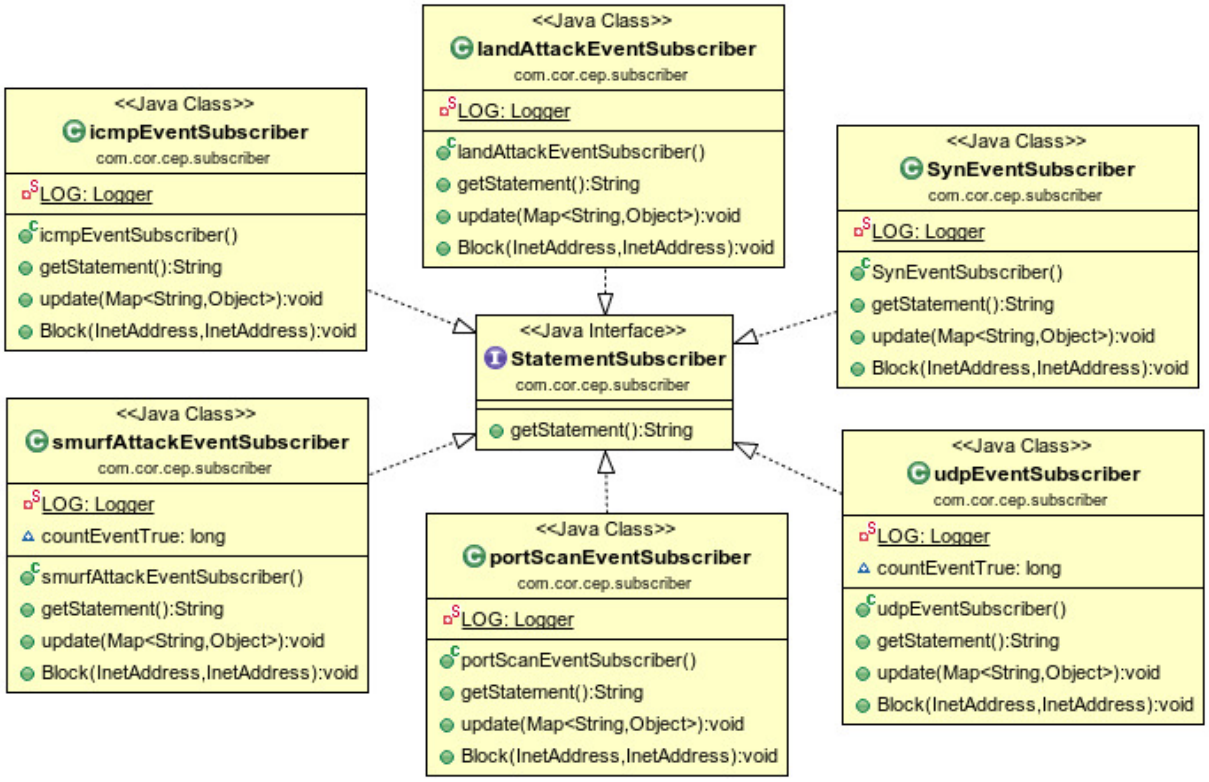

Figura 4.4: Diagrama de Classes do Processador de Eventos

A Figura 4.4 mostra um diagrama de classes que apresenta as classes *icmpEventSubscriber, smurfAttackEventSubscriber, landAttackEventSubscriber, portScanEventSubscriber, udpEventSubscriber* e *SynEventSubscriber* que implementam a interface *StatementSubscriber* e armazenam as regras CEP. Nela é possível observar que, além de outros métodos, existe em cada classe um método *update()* e um método *Block()*. O método *update()* é responsável por atualizar as informações de notificações sobre os eventos que ativaram as regras CEP e enviar um alerta para cada detecção de ataque. O método *Block()* por sua vez, tem a função de executar as regras de bloqueio do sistema.

A Figura 4.5 mostra um diagrama de classes que expressa a relação entre as classes: *PacketEventGenerator*, que é responsável pela captura dos pacotes utilizando a biblioteca *Jpcap*; *TCPpacketEvent, UDPpacketEvent* e *ICMPpacketEvent* que contém os atributos para armazenagem dos eventos (i.e., endereço IP de origem e destino, tamanho do pacotes, ...), e a classe *EventHandler*, responsável por criar e manipular as regras CEP.

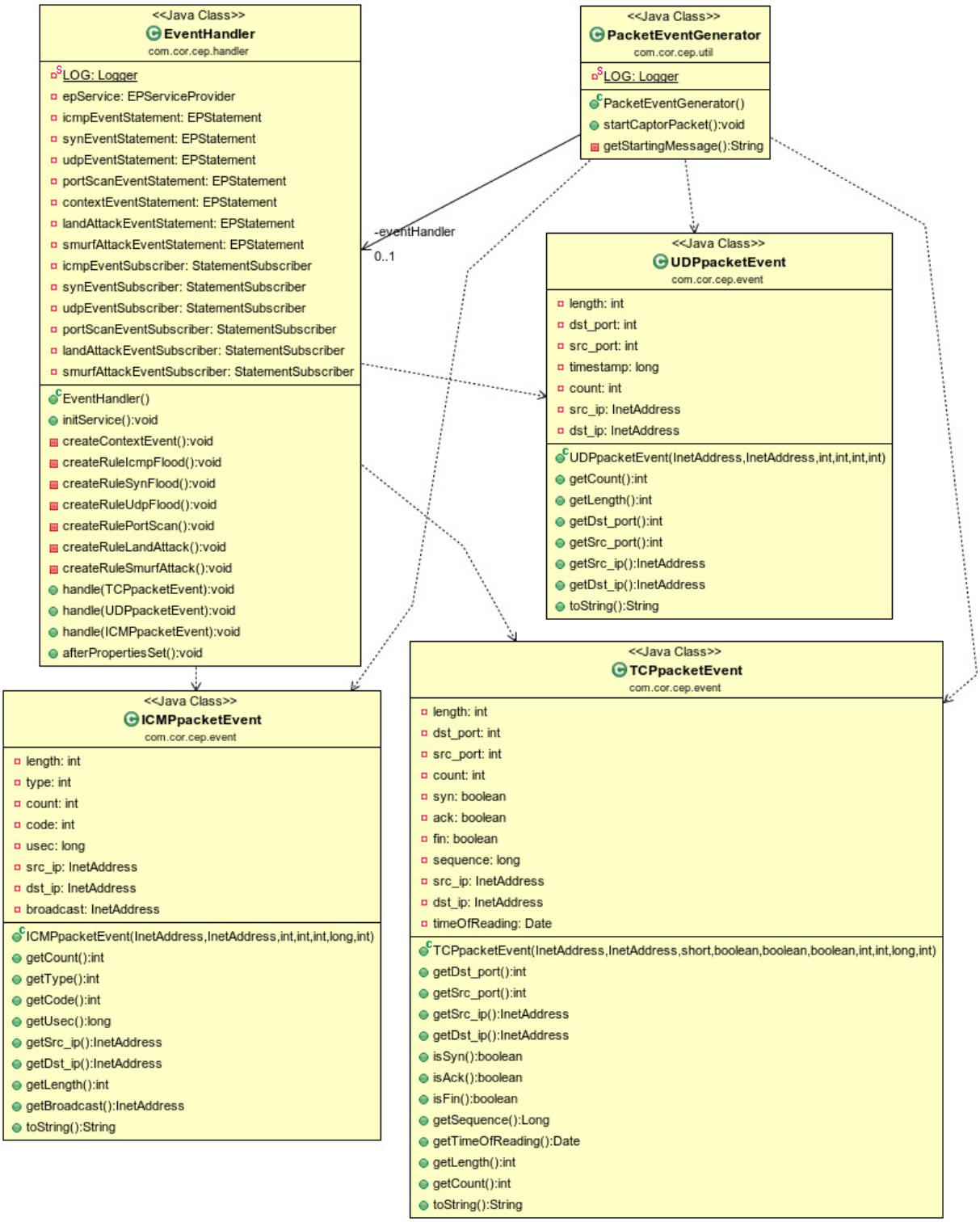

Figura 4.5: Diagrama de Classes do Filtro de Eventos

4.2 Aspectos de Implementação

O CEPIDS foi desenvolvido usando a linguagem Java abordando a metodologia de Programação Orientada a Objetos. A Jpcap é a biblioteca utilizada para capturar o tráfego de rede e consiste em um conjunto de classes Java e interfaces que

implementam chamadas para uma biblioteca do sistema *libpcap*. Jpcap oferece captura de pacotes em tempo real e uma opção para armazenar e ler pacotes capturados em arquivos *offline*, também permite a filtragem de pacotes por tipo de protocolo (IPv4, IPv6, ARP / RARP, TCP, UDP e ICMPv4), que o CEPIDS usa para detectar ataques de Negação de Serviço.

O Sistema CEPIDS foi implementado para plataforma Raspberry Pi um dispositivo com poucos recursos computacionais, que possui 1 gigabyte de memória e um processador ARM quad-core de 1,2 GHz e usa o sistema operacional *Raspbian*. A escolha da plataforma se deu devido às características que o dispositivo oferece, que são:

- (i) pequeno computador de placa única, com interfaces de rede sem fio que permitem que objetos comuns sejam transformados em objetos inteligentes, com capacidades de detecção e atuação;
- (ii) ele é portátil, podendo os usuários transportá-los sem esforços;
- (iii) é versátil, podendo ser utilizado em qualquer lugar, desde casas inteligentes até universidades e praças.

4.3 Avaliações

Seis tipos diferentes de ataques de redes foram selecionados para estressar e avaliar do sistema proposto (*SYN Flood, UDP Flood, ICMP Flood, Land Ataque e Smurf Ataque e Port Scan*). Os cinco primeiros ataques são caracterizados como ataques de inundação, cujo objetivo é esgotar os recursos do sistema da vitima através do envio de uma grande quantidade de pacotes [27]. Assim, ao tentar processar todos os pedidos, a vítima não consegue respondê-los a tempo, causando DoS. O último ataque é utilizado para verificar as portas abertas de uma máquina alvo.

Utilizou-se dois conjuntos de dados (*datasets*) para avaliação do sistema: O **DARPA** do Instituto de Tecnologia de Massachusetts que é composto por tráfego TCP/IP que foram obtidos ao longo de sete semanas de coleta e totalizam 9,87 GB de dados [26] e o **CUT-13** que é um conjunto de dados do tráfego de uma *botnet* que foi capturado na Universidade CTU, na República Tcheca cujo objetivo era ter uma grande

captura de tráfego real de uma rede *botnet* misturada ao tráfego de uma rede normal. O conjunto de dados CTU-13 consiste em treze capturas (denominadas cenários) de diferentes amostras de botnets [32]. Ambos os conjuntos de dados utilizados são compostos por tráfego não tratado e por tráfego rotulado, possibilitando assim a classificação dos ataques por tipo de protocolo.

Os experimentos para detecção dos ataques *SYN Flood, UDP Flood e LAND Attack* foram realizados utilizando o conjuntos de dados DARPA. Já os experimentos para detecção dos ataques do tipo (*ICMP Flood e Smurf Attack*) foram realizados utilizando o conjuntos de dados CUT-13, conforme descrito no Capitulo 1 na seção da metodologia abordada. O experimento para detecção dos ataques de *Port Scan* foi realizado em um ambiente de testes controlado e composto por um *desktop*, um *laptop* e três *Raspberry Pi*. O dispositivo (IP: 192.168.10.1) foi configurado para ser um ponto de acesso *wifi*. Faz parte do cenário para o experimento uma aplicação para medição de temperatura e umidade, no qual foi utilizado um sensor DHT11 para realizar a coleta da temperatura e umidade do ambiente. O cenário abordado faz parte de um projeto de automação para um setor de avicultura, onde o controle da temperatura e da umidade é essencial para garantir o crescimento adequado das aves. Esses fatores precisam está em condições adequadas, uma vez que eles impactam diretamente na produtividade do aviário.

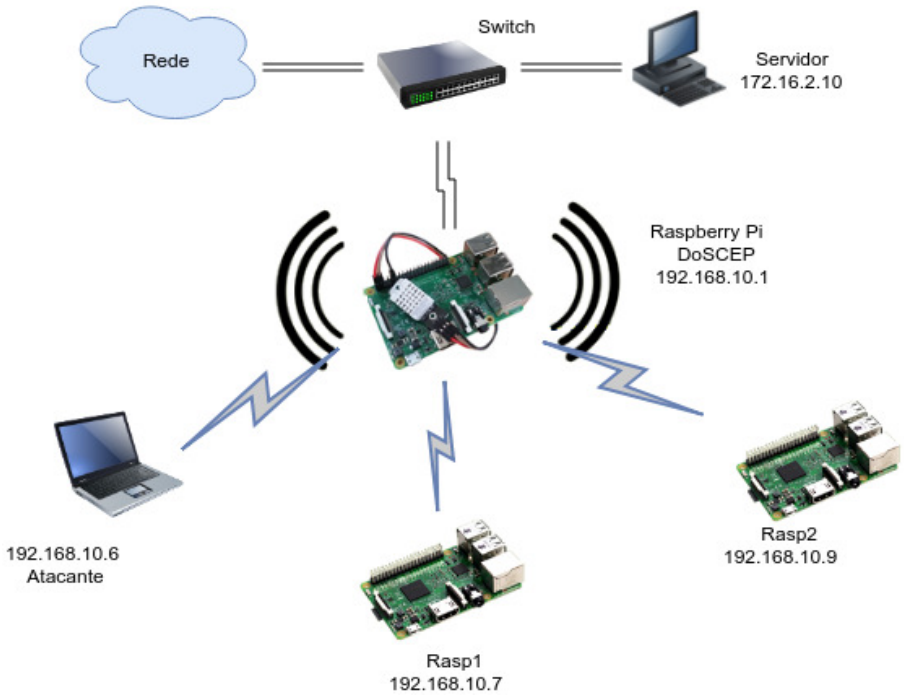

Figura 4.6: Arquitetura para Experimento do Sistema

Para aferir o uso da CPU e da memória RAM durante a execução do experimento foi desenvolvida uma ferramenta de *script* para registrar as oscilações referente ao consumo da CPU e da memória RAM possibilitando assim a obtenção do consumo médio para ambas as métricas avaliadas. A taxa de detecção de ataque foi calculada a partir dos *logs* gerados pelo CEPIDS durante o experimento, sendo estes gravados em um arquivo de texto para posterior análise.

Os experimentos foram realizados no período de 120 segundos com 10 repetições cada experimento para cada tipo de ataque abordado nesta pesquisa [31]. A análise estatística dos dados para todos os experimentos abordados nesta pesquisa foi realizada utilizando a ferramenta *PAST* versão 3.20. *PAST* é um software livre para análise de dados científicos, com funções de manipulação de dados, plotagem, estatística univariada e multivariada, séries temporais e análises espaciais, entre outras [16]. Esta ferramenta foi utilizada para normalizar os dados tornando-os paramétricos, bem como calcular o desvio padrão, o intervalo de confiança dos dados e plotar os gráficos referente aos resultados. Os resultados apresentam um nível de confiança de 95%.

4.4 Regras de Detecção

Para detectar os ataques, foi utilizada uma abordagem baseada na extração de características de pacotes [38] a partir das quais as regras do CEP foram criadas para identificar os ataques. A regra CEP utilizada para detectar o ataque *SYN Flood* é mostrada na Listagem 4.1.

Listagem 4.1: Regra CEP para SYN Flood

```
Select count(syn) as synCount, ack, src_ip, dst_ip from
    TCPpacketEvent.win:time(1 sec) having count(syn) > 128 and not(ack)
```

Na regra acima o IDS emitirá um alerta quando a taxa de pacotes TCP SYN que chegam no dispositivo for maior que 128 pacotes em uma janela de tempo de 1 segundo [57] e quando a *flag ACK* estiver ausente. A regra na Listagem 4.2 é usada para detecção de ataques *UDP Flood*.

Listagem 4.2: Regra CEP para UDP Flood

```
Select count(*) as udpEvent, count, src_ip from
        UDPpacketEvent.win:time(1 sec) having count(*) > 30
```

Para detecção do *UDP Flood* uma das regras utilizadas foi uma abordagem do IDS Suricata [19], onde o sistema emite um alerta se os pacotes UDP chegarem a uma taxa maior que 30 pacotes por segundo. Essa abordagem pode ser utilizada para detecção de diferentes ataques de inundação, levando em consideração a quantidade de pacotes em um limite de tempo. A Listagem 4.3 apresenta a regra para detecção de *Land Atack*.

Listagem 4.3: Regra CEP para Land Atack

```
select count(*) as land_count, src_ip, dst_ip, src_port, dst_port
    from TCPpacketEvent.win:time(1 sec) having src_ip = dst_ip
                        and src_port = dst_port and count(*) > 128
```

Para detecção do *Land Atack* o CEPIDS realiza a contagem dos pacotes do tipo *TCP* em uma janela de tempo de um segundo e verifica se mais de 128 pacotes foram recebidos neste intervalo e se os endereços IPs e as portas de origem e destino são iguais.

Listagem 4.4: Regra CEP para ICMP Flood

```
select istream count(*) as icmpCount, count, src_ip from
        ICMPpacketEvent.win:time_batch(1 sec) having count(*) > 100
```

Para detecção de *ICMP Flood* o CEPIDS utiliza uma abordagem semelhante ao *Smurf Ataque*. Realiza a contagem de todos os pacotes *echo request* verificando se mais de 100 pacotes foram enviados dentro de uma janela de tempo de um segundo para um mesmo endereço IP de destino, em vez de verificar se foram redirecionadas para o endereço de *broadcast*.

Listagem 4.5: Regra CEP para Port Scan

```
select count(syn) as synCount, src_ip, dst_ip from
    TCPpacketEvent.win:time_batch(1 msec) having count(syn) > 10 and
                        count(distinct(dst_port))>=(count(syn))
```

A Regra CEP ilustrada é utilizada para detecção de *Port Scanning* do CEPIDS. A regra realiza a contagem de todos os pacotes *TCP SYN* e retorna o endereço IP de origem das conexões com o servidor, verificando se mais de 10 pacotes *TCP SYN* foram enviados para diferentes portas de um único endereço de destino em uma janela de tempo de 1 milissegundo [36].

4.5 Considerações Finais

Este capitulo apresentou o CEPIDS, um Sistema de Detecção de Intrusão baseado em Processamento de Eventos Complexos para Internet das Coisas. Apresentou-se arquitetura do sistema proposto, bem como o fluxograma de execução e os diagramas UML de classe e de sequência do CEPIDS. Foram apresentados ainda os cenários de ataques, onde foi mostrada a arquitetura utilizada para o experimento e por fim mostrou-se as regras CEP utilizadas para a detecção de cada tipo de ataque abordado nesta pesquisa.

5 Resultados

Neste Capítulo são apresentados os resultados da avaliação do sistema. CEPIDS foi avaliado em um *Raspberry Pi 3 Model B*. Devido as limitações do poder computacional do Raspberry Pi, quatro métricas foram consideradas para realizar a avaliação do desempenho em tempo real do sistema e verificar se o uso do mecanismo de processamento de eventos pode atender aos requisitos do ambiente IoT. As métricas consideradas foram: uso da CPU, uso de memória RAM, taxa de pacotes perdidos e número de ataques detectados.

O Sistema CEPIDS emite um alerta de segurança sempre que o fluxo de dados na rede ativarem uma regra CEP. A Figura 5.1 mostra um alerta para ataques *Syn Flood*, destacando o endereço IP do invasor. O sistema utiliza uma abordagem baseada em janelas de tempo, analisando a intensidade do tráfego na rede o que possibilita um maior número de detecção desse tipo de ataques em relação aos sistemas que não utilizam esse tipo de abordagem.

Figura 5.1: Alerta SYN Flood

5.1 SYN Flood

Conforme abordado no Capítulo 2, o ataque *SYN Flood* é caracterizado pelo envio de pacotes (*TCP SYN*) a uma taxa maior que 128 pacotes por segundos com o objetivo de consumir os recursos do sistema vítima para que este não consiga responder ao tráfego legítimo.

Para avaliar o Sistema referente a detecção do ataque *SYN Flood* foi utilizado o *dataset* DARPA do Instituto de Tecnologia de Massachusetts, sendo este composto por tráfego TCP/IP e tráfego UDP. A Figura 5.2 mostra resultado médio de utilização da CPU e Memória RAM pelo CEPIDS durante a avaliação do sistema referente ao ataque

SYN Flood. Antes de realizar o ataque o CEPIDS estava consumindo 0.5% de CPU e 1.4% de memória RAM.

Ao realizar o experimento, o consumo da CPU e da memória RAM sofreu várias oscilações registrando uma média de 78,31% de uso da CPU e uma média de 12,18% de uso da memória RAM. Acredita-se que tais oscilações devem-se ao grande fluxo de tráfego gerado com milhares de requisições de conexão por segundo a partir do *dataset*. Os resultados obtidos pelo CEPIDS referente ao consumo de CPU e RAM são semelhantes aos resultados apresentados pelo autores [23] ao avaliarem o Snort e o Bro em um Raspberry Pi, tornando satisfatórios os resultados do CEPIDS em relação ao Snort e o Bro, uma vez que estes são sistemas robustos e com uma grande comunidade de desenvolvedores.

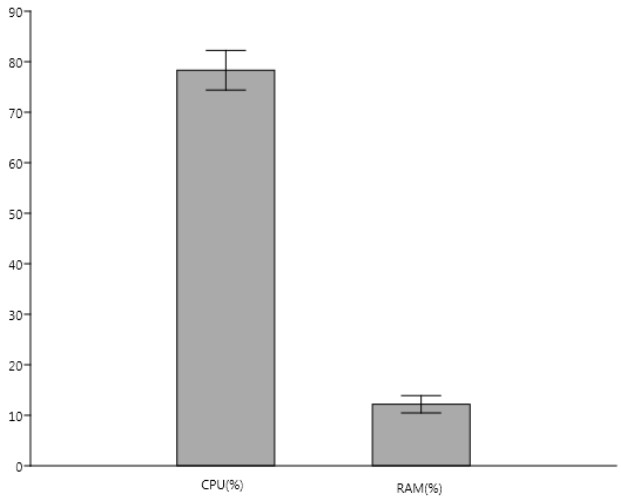

Figura 5.2: Uso da CPU e Memória RAM durante o ataque SYN Flood

Após o CEPIDS identificar os ataques, as seguintes regras de bloqueio foram executadas utilizando *Iptables*:

Listagem 5.1: Comandos utilizados para bloquear o ataque SYN Flood

```
#iptables -A FORWARD  -p tcp -s src_ip -d dst_ip -j DROP

#iptables -A INPUT  -p tcp -s src_ip -d dst_ip -j DROP.
```

Sobre o comando *Iptables* acima temos: *iptables* corresponde ao *firewall* utilizado para bloquear o tráfego; *-A FORWARD* e *-A INPUT* são *chain* que permiti

que os dados recebidos pelo CEPIDS trafeguem até o IP de destino; *-p tcp* corresponde ao tipo de protocolo que será bloqueado; *-s* corresponde ao IP origem do ataque; *-d* corresponde ao IP de destino (vítima); e por fim, *-j DROP* que realiza o bloqueio de todos os pacotes do IP de origem para o destino especificado. O *firewall* foi instalado no *Raspberry Pi* e executa o bloqueio do tráfego malicioso conforme regras definidas no CEPIDS.

5.2 UDP Flood

Para os ataque de *UDP Flood*, que tem como características um grande número de conexões UDP simultâneas, a Figura 5.3, mostra o desempenho médio do CEPIDS para uso da CPU e memória RAM ao analisar o sistema durante o ataque. A avaliação do Sistema foi realizada utilizado o *dataset* DARPA do Instituto de Tecnologia de Massachusetts. Durante a execução do experimento, o consumo da CPU e da memória RAM registraram uma média de 70,67% e de 7,64% de utilização respectivamente.

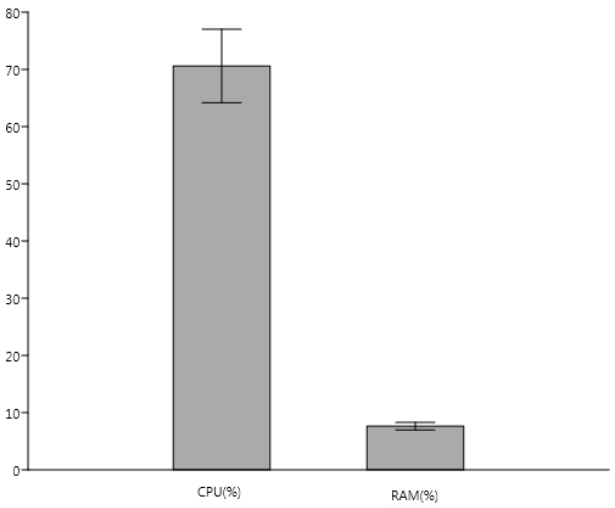

Figura 5.3: Uso da CPU e Memória RAM durante o ataque UDP Flood

A partir da Figura 5.3 percebe-se que o consumo da CPU e RAM para o ataque UDP Flood foi menor em comparação ao consumo durante o ataque SYN Flood. Acredita-se que esta diferença deve-se ao grande número de pacotes gerados para os ataques do tipo SYN Flood, uma vez que o protocolo TCP utilizado durante este

ataque é orientado a conexão e exige uma resposta durante o processo de comunicação (Handshake de 3 vias), conforme abordamos no Capitulo 2 seção 2.2.3. Isso faz com que o número de pacotes TCP trafegados na rede seja maior, podendo provocar alto consumo da CPU e RAM para os ataques que utilizam este protocolo.

As regras de bloqueio executadas pelo CEPIDS são as que seguem:

Listagem 5.2: Comandos utilizados para bloquear o ataque UDP Flood

```
#iptables -A FORWARD  -p udp -s src_ip -d dst_ip -j DROP

#iptables -A INPUT  -p udp -s src_ip -d dst_ip -j DROP.
```

5.3 ICMP Flood

O ataque *ICMP Flood* é caracterizado pelo envio de múltiplos pacotes *echo request* para a vítima até que o limite de requisições por segundos suportado pelo servidor seja excedido, provocando assim a negação de serviço. A Figura 5.4 mostra o desempenho do CEPIDS durante o ataque de *ICMP Flood*.

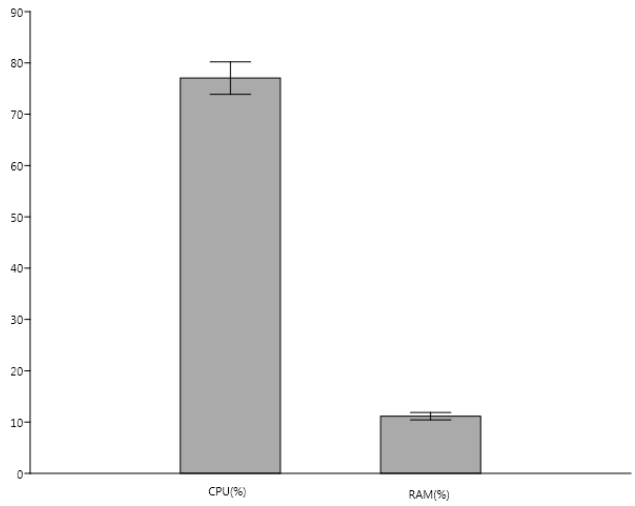

Figura 5.4: Uso da CPU e Memória RAM durante o ataque ICMP Flood

A avaliação do Sistema foi realizada utilizado o *dataset* CTU-13 e cenário CTU-11 da Universidade CTU. A utilização de processamento inicialmente estava em

0,5% e o uso de memória inicial estava em 1,4%. Durante a execução do experimento, o consumo médio da CPU foi de 77,05% e o consumo da memória RAM foi de 11,14%, aproximadamente 114,07 MB de utilização de memória. CEPIDS apresenta desempenho satisfatório durante o ataque de *ICMP Flood* tendo melhor performance referente ao uso da memória RAM em comparação aos autores [17] e em comparação ao Snort e melhor performance referente ao uso da CPU em comparação ao Bro [23].

As regras utilizadas para bloquear os ataques de *ICMP Flood* são:

Listagem 5.3: Comandos utilizados para bloquear o ataque ICMP Flood

```
#iptables -A FORWARD  -p icmp -s src_ip -d dst_ip -j DROP
#iptables -A INPUT  -p icmp -s src_ip -d dst_ip -j DROP.
```

5.4 Land Atack

O *Land Atack* é caracterizado pelo envio de pacotes onde o endereço IP e porta de origem e destino são os mesmos, fazendo com que a vítima ao tentar responder as requisições do cliente entre em loop infinito. O desempenho médio do sistema é apresentado na Figura 5.5, onde é mostrado o consumo da CPU e da memória RAM durante a realização do *Land Atack*.

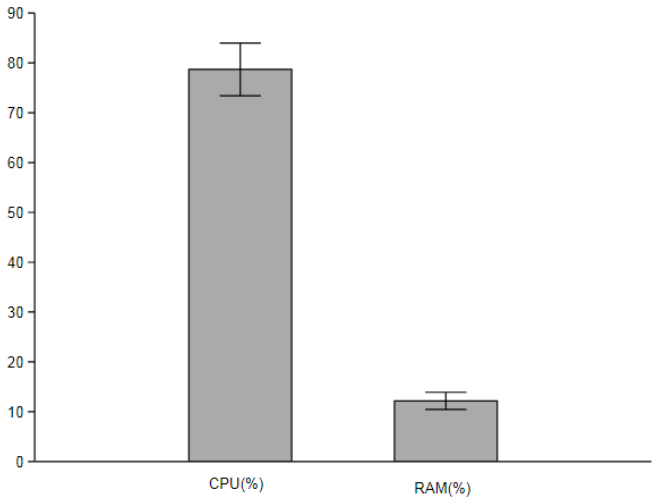

Figura 5.5: Uso da CPU e Memória RAM durante o Land Ataque

Analisando a Figura 5.5 pode-se observar que o desempenho do CEPIDS durante os testes com *Land Atack* foi semelhante aos resultados dos testes do ataque *SYN Flood*, onde a quantidade média de uso da memória RAM foi de 12,16% e o uso da CPU foi de 78,70%. Acredita-se que a semelhança nos resultados desses ataques deva-se ao motivo de estarem utilizando o protocolo TCP para ambos os ataques.

As regras utilizadas para bloquear os ataques do tipo *Land Atack* são:

Listagem 5.4: Comandos utilizados para bloquear o Land ataque

```
#iptables -A FORWARD  -p tcp -s src_ip -d dst_ip -j DROP

#iptables -A INPUT  -p tcp -s src_ip -d dst_ip -j DROP.
```

5.5 Smurf Atack

O *Smurf Atack* é um tipo de ataque que utiliza o protocolo ICMP assim como o ataque *ICMP Flood*. No entanto apesar de ambos utilizarem o mesmo protocolo, eles se diferem em termo da abordagem utilizada durante a execução do ataque. O *ICMP Flood* tem o objetivo de inundar diretamente o sistema vítima, enviando uma grande quantidade de requisições *echo request*, enquanto o *Smurf Atack* por sua vez é realizado através de inúmeras requisições *echo request* para o endereço de *broadcast*, utilizando o endereço IP de origem modificado para o endereço IP da vítima, fazendo com que todos os dispositivos que estão ao alcance do endereço de *broadcast* recebam as requisições e respondam para o sistema vítima, provocando então a negação de serviço. A Figura 5.6 mostra o desempenho médio do CEPIDS referente a utilização de CPU e memória RAM durante o *Smurf Atack*.

A avaliação do Sistema durante o *Smurf Attack* foi realizada utilizado o *dataset* CTU-13 e cenário CTU-4 da Universidade CTU. A utilização de processamento inicialmente estava em 0,5% e o uso de memória inicial estava em 1,4%. Durante a execução do experimento, o consumo médio da CPU e da memória RAM foram de 76,50% e 11,29% respectivamente. Percebe-se que os resultados do Smurf Atack são próximos aos resultados obtidos durante o ataque *ICMP Flood*. Acredita-se que tal proximidade deva-se ao protocolo ICMP utilizado por ambos durante os ataques.

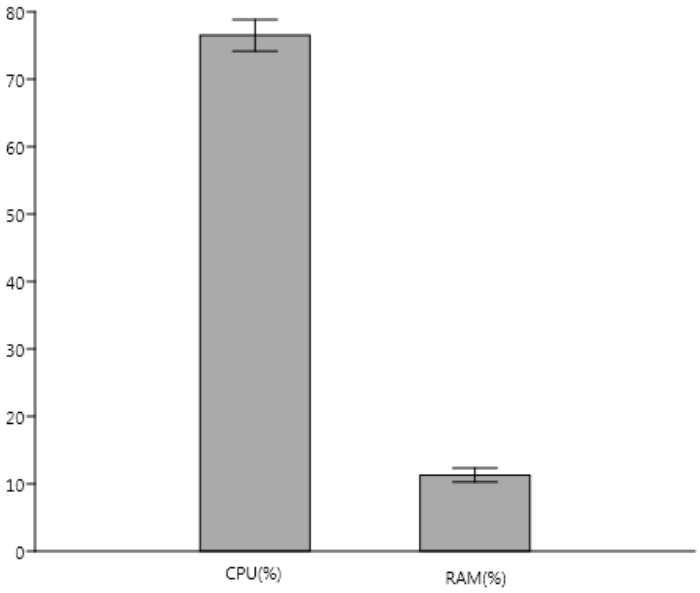

Figura 5.6: Uso da CPU e Memória RAM durante o Smurf Atack

As regras utilizadas para bloquear os ataques *Smurf Atack* são:

Listagem 5.5: Comandos utilizados para bloquear o ataque Smurf Atack

```
#iptables -A FORWARD  -p icmp -s src_ip -d dst_ip -j DROP

#iptables -A INPUT  -p icmp -s src_ip -d dst_ip -j DROP.
```

5.6 Port Scan

O cenário do experimento utilizado para a avaliação do CEPIDS durante os ataques de *Post Scan* usou uma abordagem diferente do cenário utilizado para os ataques de inundação, uma vez que os *datasets* utilizados não tinham dados referente a esse tipo de ataque. Nessas condições foram definidos dois cenários com tráfego em segundo plano configurado para 30% e 90%, com o intuito de avaliar o consumo da CPU e memória RAM pelo CEPIDS.

A Figura 5.7 apresenta o desempenho do sistema referente a utilização da CPU e memória RAM do CEPIDS durante a execução do ataque de *Port Scan*.

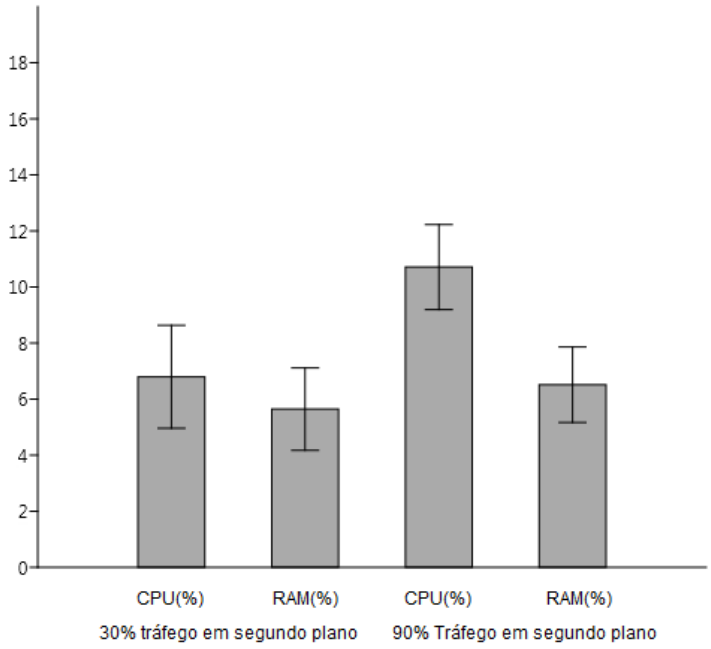

Figura 5.7: Uso da CPU e Memória RAM durante o ataque de Port Scan

O ataque de *Port Scan* foi mensurado primeiro com 30% de tráfego em segundo plano. Quando o ataque foi iniciado, os índices de utilização da CPU e memória RAM eram de 0.5% de CPU e 1.4% de memória RAM e subiram para 6,48% e 5,21% respectivamente. Quando o ataque foi repetido, porém com 90% de tráfego em segundo plano, o consumo da CPU e da memória RAM subiram para 10,25% e 6,57% respectivamente.

5.7 Taxa de Detecção

As seções anteriores apresentaram os resultados de desempenho referente ao uso da CPU e memória RAM pelo CEPIDS. A Tabela 5.1 resume os resultados experimentais para as taxas de detecção de ataques e taxa de pacotes analisados e perdidos pelo sistema para os diferentes tipos de ataques abordados nesta pesquisa.

CEPIDS apresenta uma taxa de detecção de ataque satisfatória nos resultados experimentais, tendo a menor taxa de detecção de 96,80% para *Port Scan* e a maior taxa de detecção de 98,95% para *ICMP Flood*. Para a taxa de pacotes perdidos CEPIDS também mostrou um desempenho aceitável, tendo 0,63% e 0,41% como a

maior e a menor taxa de pacotes perdidos para os ataques *SYN Flood* e *ICMP Flood* respectivamente.

Tabela 5.1: Taxa de detecção

-	SYN Flood	UDP Flood	ICMP Flood	Land Attack	Smurf Attack	Port Scan
TP	100000	100000	100000	100000	100000	-
PA	99367	99428	99589	99367	99495	-
PP	633	572	411	633	505	-
AE	2000	2000	2000	2000	2000	1000
AD	1948	1940	1979	1956	1965	968
TD	97,4%	97,00%	98,95%	97,8%	98,25%	96,80%
TPP	0,63%	0,57%	0,41%	0,63%	0,51%	-

Legenda para siglas da Tabela:

TP - Número total de pacotes

PA - Número de pacotes analisados

PP - Número de pacotes perdidos

AE - Número de Alertas esperados

AD - Número de Alertas detectados

TD - Taxa de detecção de ataques

TPP - Taxa de pacotes perdidos

5.8 Comparação do desempenho com Snort e Bro

Os resultados foram previamente apresentados nas seções 5.1 a 5.7 do capítulo 5. Nesta seção, faremos um abordagem comparativa do desempenho do CEPIDS no Raspberry Pi com o trabalho dos autores [23] que avaliaram a performance do Snort e Bro IDS no Raspberry Pi. Os ataques abordados pelos autores [23] foram *Port Scan, SYN Flood e ARP Spoofing*. Assim, realizaremos uma análise comparativa considerando apenas os resultados para os ataques do tipo *Port Scan*, uma vez que o cenário para os demais tipos de ataques abordados nesta pesquisa diferem do cenário utilizado pelos autores. É importante destacar que o cenário do experimento para o

ataque do tipo *Port Scan* é o mesmo utilizado pelos autores [23], considerando 30% e 90% de tráfego em segundo plano.

A Figura 5.8 mostra o desempenho dos IDSs durante o ataque de *Port Scan* com 90% de tráfego em segundo Plano.

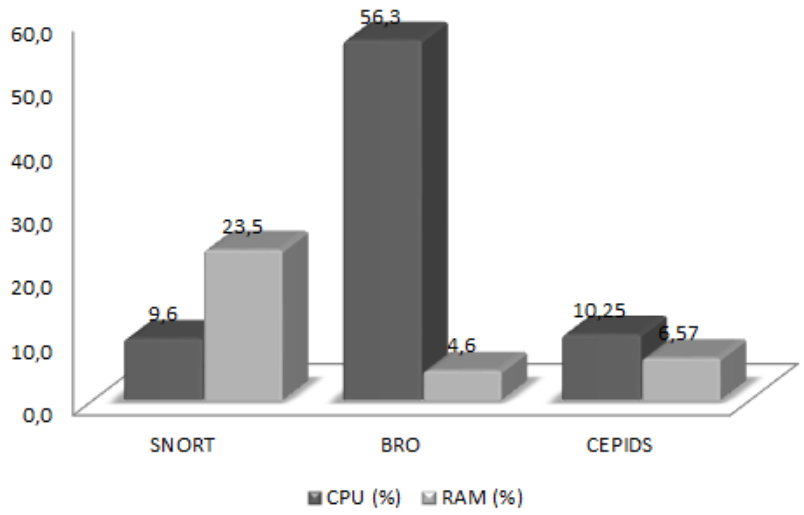

Figura 5.8: Desempenho para ataques *Por Scan* com 90% de tráfego em segundo plano

A Figura 5.8 mostra que CEPIDS teve uma performance satisfatória durante o ataque de *Port Scan*, com baixos consumo de CPU e RAM. Para o cenário abordado CEPIDS foi melhor que Snort no consumo de memória RAM e melhor que Bro no consumo de CPU. No entanto, considerando as pequenas diferenças no uso de CPU em comparação ao Snort e no uso de memória RAM em comparação ao Bro, acredita-se que CEPIDS apresentou melhor desempenho sendo uma boa opção de IDS para o cenário apresentado.

Na Figura 5.9 é apresentado o desempenho médio do CEPIDS em comparação com o desempenho médio dos IDSs Snort e o Bro apresentados pelos autores [23].

A partir da Figura 5.9 é possível observar que CEPIDS teve um desempenho aceitável nos resultados experimentais em comparação com os sistemas Snort e Bro IDS. CEPIDS usou 10,08% de memória RAM, enquanto o Snort utilizou 25.03%. No uso da CPU, CEPIDS apresenta melhor desempenho que o Bro, utilizando 64,99% de processamento, enquanto o Bro utilizou 80.33%. No entanto Snort teve melhor desempenho no uso da CPU, consumindo apenas 22.86%. Para taxa de pacotes

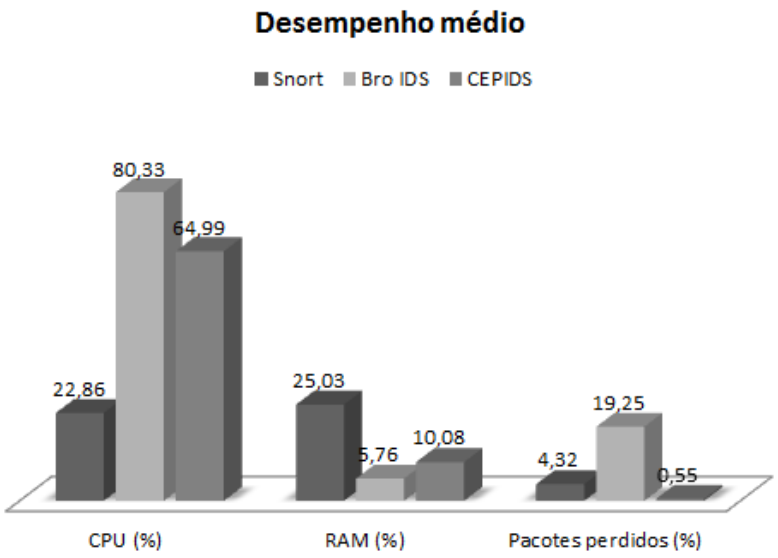

Figura 5.9: Comparação do desempenho do CEPIDS, Snort e Bro IDS

perdidos CEPIDS apresentou melhor desempenho que o Snort e o Bro com uma taxa de apenas 0,55%, enquanto o Snort e o Bro tiveram uma taxa de 4,32% e 19,25% de pacotes perdidos respectivamente. É importante destacar que os resultados referente ao CEPIDS representam a média dos seis (6) tipos de ataques abordados nesta pesquisa, enquanto os resultados apresentados pelos autores [23] abordam apenas três (3) tipos de ataques enquanto avaliavam o desempenho do Snort e do Bro.

6 Conclusões e Trabalhos Futuros

Nesta dissertação é apresentado o CEPIDS, um Sistema de Detecção de Intrusão baseado no Processamento de Eventos Complexos para ambientes de IoT. O CEPIDS foi projetado para detectar ataques de redes e realizar a detecção de seis tipos de ataques: *"syn flood, udp flood, icmp flood, smurf attack, land attack"*, que têm como característica um grande fluxo de dados e *"port scan"* que permitem a detecção de portas abertas em máquinas alvo. Além disso, o CEPIDS foi projetado para ser estendido para detectar outros tipos de ataques, por exemplo, ARP Spoofing e ataques de roteamento.

Testes de avaliação de desempenho foram realizados utilizando dois cenários diferentes, um cenário utilizando dois dataset para detecção de ataques de inundação e um outro cenário utilizando tráfego em segundo plano de 30% e 90% para ataques de varredura portas. O sistema foi avaliado em um Raspberry Pi 3 e os resultados experimentais foram comparados com o desempenho dos IDSs Snort e do Bro [23] mostrando que CEPIDS teve melhor desempenho médio que o Bro para uso de CPU. Melhor desempenho que o Snort para o uso de memória RAM. E melhor desempenho médio que ambos para taxa de pacotes perdidos.

Os resultados mostram que o mecanismo CEP é adequado para aplicações e serviços que precisam processar grandes fluxos de dados, tornando-o uma ferramenta adequada para a detecção de intrusão. CEP mostrou através de avaliações referentes à utilização da CPU, consumo de memória, taxa de pacotes perdidos e taxa de detecção de ataques, ser adequado para aplicativos e serviços que precisam processar grandes fluxos de dados gerado a partir de objetos inteligentes em ambientes IoT. O sistema foi implementado com base no ESPER, que verificou a viabilidade de nossos estudos.

As principais contribuições deste trabalho são:

- A investigação do estado da arte em ferramentas de detecção de intrusão para o apoio ao desenvolvimento das regras de detecção de ataques de negação de serviço utilizando CEP. Através desta investigação foi possível conhecer e entender o funcionamento dos ataques DoS, possibilitando criar regras baseadas

em janelas de tempo com CEP através da extração das características do tráfego da rede. Isso pode servir de base para novos estudos que visam o desenvolvimento de ferramentas ou sistemas para detecção e prevenção de intrusão.

- O desenvolvimento do CEPIDS, um sistema de detecção de intrusão baseado no mecanismo de processamento de eventos complexos para ambientes de IoT. CEPIDS utiliza uma abordagem baseada em processamento de fluxos de dados fazendo uso de janelas de tempo que permite a identificação de ataques em tempo real.

A partir deste trabalho inicial, foi identificado algumas limitações que podem ser resolvidas a partir de trabalhos futuros a serem desenvolvidos, tais como:

- A identificação de ataques por anomalias e por assinaturas a partir da definição de novos padrões criados através da extração de características do tráfego. Essa abordagem poderá resolver problemas de detecção de ataques como *Zero Day*.
- Implementação de novas funcionalidades para detecção de ataques do tipo: *ARP Spoofing, e Selective Forward*. Essa funcionalidade poderá prover segurança para ataques de roteamento e *Man-In-The-Middle*.
- Adaptar o IDS proposto para dispositivos móveis. Essa abordagem poderá prover uma maior segurança aos dispositivos que necessitem se conectar diretamente à Internet através de redes móveis ou domésticas desprovidas de segurança, por exemplo: Carros e Casas Inteligentes.

Os resultados preliminares obtidos durante a realização deste trabalho estão sendo divulgados por meio da seguinte publicação:

Poster Abstract: Real-Time DDoS Detection Based on Complex Event Processing for IoT: Trabalho aceito para publicação na *3rd ACM/IEEE International Conference on Internet of Things Design and Implementation*, realizada em Orlando, Estados Unidos em Abril de 2018.

Referências Bibliográficas

[1] M. Ali, B. Chandramouli, J. Goldstein, and R. Schindlauer. The extensibility framework in microsoft streaminsight. In *2011 IEEE 27th International Conference on Data Engineering*, pages 1242–1253, April 2011.

[2] A. Anand and B. Patel. An overview on intrusion detection system and types of attacks it can detect considering different protocols. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(8), 2012.

[3] A. Arasu, S. Babu, and J. Widom. The cql continuous query language: Semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, June 2006.

[4] R. Benabdessalem, M. Hamdi, and T. H. Kim. A survey on security models, techniques, and tools for the internet of things. In *2014 7th International Conference on Advanced Software Engineering and Its Applications*, pages 44–48, Dec 2014.

[5] E. Borgia. The internet of things vision: Key features, applications and open issues. *Computer Communications*, 54:1 – 31, 2014.

[6] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[7] C. C. Center. Denial of service attacks, 2001.

[8] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. Telegraphcq: Continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 668–668, New York, NY, USA, 2003. ACM.

[9] S. Chaudhuri, U. Dayal, and V. Narasayya. An overview of business intelligence technology. *Commun. ACM*, 54(8):88–98, Aug. 2011.

[10] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, June 2012.

[11] O. Etzion, P. Niblett, and D. C. Luckham. *Event processing in action*. Manning Greenwich, 2011.

[12] V. Govindasamy, R. Ganesh, G. Nivash, and S. Shivaraman. Prediction of events based on complex event processing and probabilistic fuzzy logic. In *2014 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC)*, pages 494–499, April 2014.

[13] C. Gruhl, F. Beer, H. Heck, B. Sick, U. Buehler, A. Wacker, and S. Tomforde. A concept for intelligent collaborative network intrusion detection. In *ARCS 2017; 30th International Conference on Architecture of Computing Systems*, pages 1–8, April 2017.

[14] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, Sept. 2013.

[15] A. Gupta and B. Dhingra. Stock market prediction using hidden markov models. In *2012 Students Conference on Engineering and Systems*, pages 1–4, March 2012.

[16] Ø. Hammer, D. Harper, and P. Ryan. Past-palaeontological statistics. *www. uv. es/˜ pardomv/pe/2001_1/past/pastprog/past. pdf, acessado em*, 25(07):2009, 2001.

[17] C. Jun and C. Chi. Design of complex event-processing ids in internet of things. In *Proceedings of the 2014 Sixth International Conference on Measuring Technology and Mechatronics Automation*, ICMTMA '14, pages 226–229, Washington, DC, USA, 2014. IEEE Computer Society.

[18] V. Jyothsna, V. R. Prasad, and K. M. Prasad. A review of anomaly based intrusion detection systems. *International Journal of Computer Applications*, 28(7):26–35, 2011.

[19] P. Kasinathan, C. Pastrone, M. A. Spirito, and M. Vinkovits. Denial-of-service detection in 6lowpan based internet of things. In *WiMob*, pages 600–607, 2013.

[20] N. Khamphakdee, N. Benjamas, and S. Saiyod. Improving intrusion detection system based on snort rules for network probe attacks detection with association rules technique of data mining. *Journal of ICT Research and Applications*, 8(3), 2015.

[21] S. Kumarasamy and A. Gowrishankar. An active defense mechanism for tcp syn flooding attacks. *arXiv preprint arXiv:1201.2103*, 2012.

[22] P. Kumari, M. Kumar, and R. Rishi. Study of security in wireless sensor networks. *Proceedings of International Journal of Computer Science and Technology*, 1(5):347–354, 2010.

[23] A. K. Kyaw, Y. Chen, and J. Joseph. Pi-ids: evaluation of open-source intrusion detection systems on raspberry pi 2. In *2015 Second International Conference on Information Security and Cyber Forensics (InfoSec)*, pages 165–170, Nov 2015.

[24] N. Leavitt. Complex-event processing poised for growth. *Computer*, 42(4):17–20, April 2009.

[25] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16 – 24, 2013.

[26] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman. Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*, volume 2, pages 12–26 vol.2, 2000.

[27] M. A. Lopez, A. Lobato, D. Mattos, I. Alvarenga, O. Duarte, and G. Pujolle. Um algoritmo não supervisionado e rápido para seleção de características em classificação de tráfego. 2017.

[28] D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, pages 3–3. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[29] D. Luckham. The power of events: An introduction to complex event processing in distributed enterprise systems. In N. Bassiliades, G. Governatori, and A. Paschke, editors, *Rule Representation, Interchange and Reasoning on the Web*, pages 3–3, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[30] D. C. Luckham. *Event Processing for Business: Organizing the real-time enterprise*. John Wiley & Sons, 2011.

[31] F. M. Matos, T. E. de Sousa Araujo, and J. A. Moreira. Intrusion detection systems' performance for distributed denial-of-service attack. In *2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, pages 1–6, Oct 2017.

[32] C. U. MCFP. Malware capture facility project - universidade ctu. https://mcfp.weebly.com/the-ctu-13-dataset-a-labeled-dataset-with-botnet-normal-and-background-traffic.html/.

[33] D. Midi, A. Rullo, A. Mudgerikar, and E. Bertino. Kalis x2014; a system for knowledge-driven adaptable intrusion detection for the internet of things. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 656–666, June 2017.

[34] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. *ACM Transactions on Computer Systems (TOCS)*, 24(2):115–139, 2006.

[35] o. Netfilter. Iptables users' guide. *Online document, www. netfilter. org.* https://www.netfilter.org/.

[36] H. Networks. Configuring attack defense. 2018. https://www.hillstonenet.com/support/4.5/en/ad_config.html.

[37] Y. Ohsita, S. Ata, and M. Murata. Detecting distributed denial-of-service attacks by analyzing tcp syn packets statistically. In *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, volume 4, pages 2043–2049 Vol.4, Nov 2004.

[38] T. T. Oo and T. Phyu. A statistical approach to classify and identify ddos attacks using ucla dataset. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 2(5):pp–1766, 2013.

[39] W. Park and S. Ahn. Performance comparison and detection analysis in snort and suricata environment. *Wireless Personal Communications*, 94(2):241–252, 2017.

[40] C. Patrikakis, M. Masikos, and O. Zouraraki. Distributed denial of service attacks. *The Internet Protocol Journal*, 7(4):13–35, 2004.

[41] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23):2435 – 2463, 1999.

[42] V. Paxson, S. Campbell, J. Lee, et al. Bro intrusion detection system. Technical report, Lawrence Berkeley National Laboratory, 2006.

[43] V. Paxson, R. Sommer, S. Hall, C. Kreibich, J. Barlow, G. Clark, G. Maier, J. Siwek, A. Slagell, D. Thayer, et al. The bro network security monitor, 2012.

[44] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys Tutorials*, 16(1):414–454, First 2014.

[45] M. Pihelgas. A comparative analysis of open-source intrusion detection systems. *Master's thesis, Tallinn University of Technology, Estonia*, 2012.

[46] M. Proctor. Drools: A rule engine for complex event processing. In *Proceedings of the 4th International Conference on Applications of Graph Transformations with Industrial Relevance*, AGTIVE'11, pages 2–2, Berlin, Heidelberg, 2012. Springer-Verlag.

[47] E. Project. Enabling business-based internet of things and services. 2017. http://www.ebbits-project.eu.

[48] Q. Qassim, A. M. Zin, and M. J. Ab Aziz. Anomalies classification approach for network-based intrusion detection system. *IJ Network Security*, 18(6):1159–1172, 2016.

[49] M. Roriz Junior. DG2CEP : An On-line Algorithm for Real-time Detection of Spatial Clusters from Large Data Streams through Complex Event Processing Marcos Paulino Roriz Junior DG2CEP : An On-line Algorithm for Real-time Detection of Spatial Clusters from Large Data Stream. (March), 2017.

[50] H. Safa, M. Chouman, H. Artail, and M. Karam. A collaborative defense mechanism against {SYN} flooding attacks in {IP} networks. *Journal of Network and Computer Applications*, 31(4):509 – 534, 2008.

[51] S. A. R. Shah and B. Issac. Performance comparison of intrusion detection systems and application of machine learning to snort system. *Future Generation Computer Systems*, 80:157 – 170, 2018.

[52] I. Snort. Snort open-source ids. *IPS/NSM engine (https://snort. org/)*, 2014.

[53] I. Somal and S. Virk. Classification of distributed denial of service attacks–architecture, taxonomy and tools, 2014.

[54] S. M. Specht and R. B. Lee. Distributed denial of service: Taxonomies of attacks, tools, and countermeasures. In *ISCA PDCS*, pages 543–550, 2004.

[55] C. Sun, J. Fan, and B. Liu. A robust scheme to detect syn flooding attacks. In *2007 Second International Conference on Communications and Networking in China*, pages 397–401, Aug 2007.

[56] I. Suricata. open-source ids. *IPS/NSM engine (http://suricata-ids. org/)*, 2014.

[57] C. Systems. Configuring attack protection - cisco small business isa500. 2018. https://www.cisco.com/assets/sol/sb/isa500_emulator/help/guide/ag1359280.html.

[58] E. Team. Esper Reference, Copyright 2015 EsperTech Inc. http://www.espertech.com/esper/.

[59] S. White, A. Alves, and D. Rorke. Weblogic event server: A lightweight, modular application server for event processing. In *Proceedings of the Second International Conference on Distributed Event-based Systems*, DEBS '08, pages 193–200, New York, NY, USA, 2008. ACM.

[60] W. Yao, C.-H. Chu, and Z. Li. Leveraging complex event processing for smart hospitals using rfid. *Journal of Network and Computer Applications*, 34(3):799 – 810, 2011. RFID Technology, Systems, and Applications.

[61] S. T. Zargar, J. Joshi, and D. Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE Communications Surveys Tutorials*, 15(4):2046–2069, Fourth 2013.