
UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MILTON VINICIUS MORAIS DE LIMA

# BEHOLDER: Um Sistema de Detecção e Prevenção Contra Intrusão em Ambientes da Internet das Coisas Baseado em Processamento de Eventos Complexos

Recife

2022

MILTON VINICIUS MORAIS DE LIMA

# BEHOLDER: Um Sistema de Detecção e Prevenção Contra Intrusão em Ambientes da Internet das Coisas Baseado em Processamento de Eventos Complexos

Tese de Doutorado apresentada ao Centro de Informática da Universidade Federal de Pernambuco em cumprimento parcial dos requisitos para o título de Doutor em Ciência da Computação. Área de Concentração: Engenharia de Software e Linguagens de Programação.

**Orientador** Dr. Ricardo Massa Ferreira Lima (Universidade Federal de Pernambuco, Brasil)

**Co-orientador** Dr. Fernando Antônio Aires Lins (Universidade Federal Rural de Pernambuco, Brasil)

Recife

2022

Catalogação na fonte
Bibliotecária Nataly Soares Leite Moro, CRB4-1722

L732b Lima, Milton Vinicius Morais de
Beholder: um sistema de detecção e prevenção contra intrusão em ambientes da internet das coisas baseado em processamento de eventos complexos / Milton Vinicius Morais de Lima. – 2022.
111 f.: il., fig., tab.

Orientador: Ricardo Massa Ferreira Lima.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2022.
Inclui referências.

1. Engenharia de software e linguagens de programação. 2. IoT. 3. IDS. 4. IPS. 5. CEP. I. Lima, Ricardo Massa Ferreira (orientador). II. Título

005.1 CDD (23. ed.) UFPE - CCEN 2023 – 23

**Milton Vinicius Morais de Lima**

**“Beholder: Um Sistema de Detecção e Prevenção Contra Intrusão em Ambientes da Internet das Coisas Baseado em Processamento de Eventos Complexos”**

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Engenharia de Software e Linguagens de Programação

Aprovado em: 13/12/2022.

_____

**Orientador: Prof. Dr. Ricardo Massa Ferreira Lima**

**BANCA EXAMINADORA**

_____

Prof. Dr. Nelson Souto Rosa
Centro de Informática / UFPE

_____

Prof. Dr. Divanilson Rodrigo de Sousa Campelo
Centro de Informática / UFPE

_____

Prof. Dr. Eduardo Luzeiro Feitosa
Instituto de Computação / UFAM

_____

Prof. Dr. Obionor de Oliveira Nóbrega
Departamento de Computação / UFRPE

_____

Prof. Dr. Michel Sales Bonfim
Universidade Federal do Ceará / Campus Quixadá

Dedico esse trabalho a minha avó Amara Maria Vasconcelos de Lima. Suas orientações jamais serão esquecidas. IN MEMORIAN.

## AGRADECIMENTOS

Primeiramente agradeço a Deus e a seu filho Jesus Cristo pelo amor infinito. À minha avó Amara Maria (IN MEMORIAN), meus pais, Milton Vasconcelos e Vera Lúcia (IN MEMORIAN), e meus familiares.

Ao Dr. Ricardo Massa Ferreira Lima (por me estender a mão) e ao Dr. Fernando Antônio Aires Lins (por me ensinar a "vender o peixe"), vocês são muito mais que orientadores!

A minha rainha e companheira de todos os momentos Mônica Moreira Vieira.

Aos demais docentes do programa de Pós-Graduação do Centro de Informática da UFPE pela competência com que transmitiram os conteúdos e ensinamentos.

Agradeço a todos os amigos que tive a oportunidade de conhecer no Centro de Informática, que, por muitas vezes me ajudaram durante todo o meu doutorado.

A todos os meus amigos que de alguma forma contribuíram e participaram na realização deste trabalho.

Ao CAPES e ao CIn – UFPE, pelo apoio financeiro para realização desta pesquisa.

A todos, os meus mais sinceros agradecimentos.

"...mas aqueles que esperam no Senhor renovam as suas forças. Voam alto como águias; correm e não ficam exaustos, andam e não se cansam."(IS.40.31, 2023).

## RESUMO

A Internet das Coisas (Internet of Things - IoT) caracteriza-se por um conjunto de objetos inteligentes que se comunicam entre sí. Na IoT, os protocolos da camada de aplicação formam a base das comunicações entre aplicativos e serviços. A segurança nos dispositivos e comunicações é considerado um aspecto de design opcional nos protocolos da camada de aplicação, levando os desenvolvedores a negligenciar configurações de segurança disponíveis, ou mesmo não terem mecanismos de segurança em sua configuração padrão. Os Sistemas de Detecção e Prevenção de Intrusões (IDPS) podem melhorar a segurança da IoT, identificando possíveis problemas de segurança, alertando administradores ou agindo automaticamente para proteger o ambiente contra ameaças. Entretanto, as técnicas de IDPS não são projetadas para dispositivos com recursos limitados e precisam ser adaptadas. Esta tese apresenta o Beholder, um Sistema de Detecção e Prevenção de Intrusão (IDPS) para prevenir ataques aos protocolos da camada de aplicação da IoT. O Beholder inova ao ser o primeiro IDPS que explora a tecnologia Complex Event Processing (CEP) para detectar ataques da camada de aplicação da IoT. Este trabalho apresenta um cenário real para o processo de avaliação, realizando ataques em uma casa inteligente. Inicialmente, avaliamos a qualidade do nosso IDS, comparando-o com o conhecido Snort IDS. Os resultados demonstraram que Snort atingiu três 9s de precisão, enquanto Beholder atingiu uma precisão de quatro 9s. Adicionalmente, o experimento para avaliar a precisão de Beholder para os protocolos MQTT, CoAP e AMQP revelou que Beholder obteve quatro 9s de precisão para High QoS Flood (MQTT), 100% de precisão Error Flood (CoAP) e dois 9s para o AMQP Brute Force.



**Palavras-chaves**: IoT; IDS; IPS; CEP; segurança da informação.

## ABSTRACT

The Internet of Things (IoT) is characterized by a set of intelligent objects that communicate with each other. In the IoT, application layer protocols form the basis of communications between applications and services. Device and communications security is considered an optional design aspect of application layer protocols, leading developers to neglect available security settings, or may not even have security mechanisms in their default configuration. Intrusion Detection and Prevention Systems (IDPS) can improve IoT security by identifying potential security issues, alerting administrators, or automatically taking action to protect the environment from threats. However, IDPS techniques are not designed for devices with limited resources and need to be adapted for this type of device. This thesis presents Beholder, an Intrusion Detection and Prevention System (IDPS) to prevent attacks on IoT application layer protocols. Beholder breaks new ground by being the first IDPS that exploits Complex Event Processing (CEP) technology to detect IoT application layer attacks. This work presents a real scenario for the evaluation process, carrying out attacks on a smart home. Initially, we evaluated the quality of our IDS, comparing it with the well-known Snort IDS. The results showed that Snort hit three 9s of accuracy, while Beholder achieved an accuracy of four 9s. Additionally, the experiment to evaluate the accuracy of Beholder for the MQTT, CoAP and AMQP protocols revealed that Beholder obtained four 9s of accuracy for High QoS Flood (MQTT), 100% of accuracy Error Flood (CoAP) and two 9s for the AMQP Brute Force .



**Keywords**: IoT; IDS; IPS; CEP; information security.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| **AMQP** | Advanced Message Queuing Protocol |
| **BPMN** | Business Process Management Notation |
| **CEP** | Processamento de Eventos Complexos |
| **CoAP** | Constrained Application Protocol |
| **HTTP** | Hypertext Transfer Protocol |
| **IDPS** | Sistema de Detecção e Prevenção de Intrusão |
| **IoT** | Internet of Things |
| **MQTT** | Message Queue Telemetry Transport |
| **NAT** | Network Address Translation |
| **NFC** | Near-field Communication |
| **RFID** | Radio Frequency Identification |
| **SBC** | Simple Boarding Computing |
| **TCP** | Transmission Control Protocol |
| **UDP** | User Datagram Protocol |
| **WSN** | Wireless Sensor Network |

## SUMÁRIO

# 1 INTRODUÇÃO

Este capítulo apresenta o contexto da Internet of Things (IoT) e sua segurança. Ele descreve também as principais motivações para realização deste trabalho, detalhando o problema, hipótese, questão de pesquisa, objetivos almejados, metodologia, contribuição e escopo negativo. Por fim, o capítulo sintetiza como está estruturado o restante desta tese.

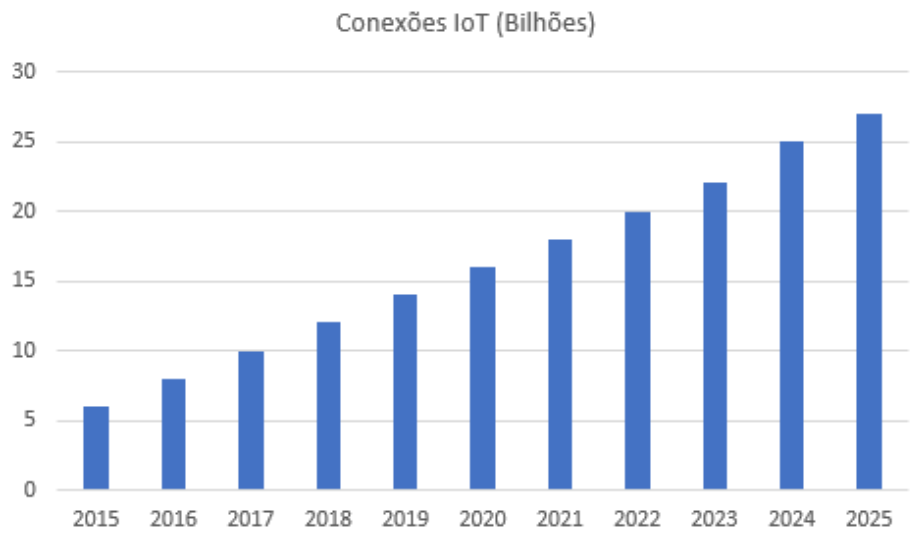
## 1.1 CONTEXTO

A Internet das Coisas (Internet of Things - IoT) não é um conceito novo, mas tem enormes potencialidades para o desenvolvimento e construção de aplicações inteligentes em diversos ambientes (GARTNER, 2020). A IoT Trata-se de uma tecnologia que conecta diferentes dispositivos, que produzem e trocam dados para gerar informações e atuar de acordo com as informações obtidas (COLLINA et al., 2014) (ZHOU; ZHANG, 2014a). Na literatura corrente, é possível encontrar várias definições para descrever a IoT. Uma delas afirma que a IoT consiste, essencialmente, de sensores responsáveis por captar os mais variados tipos de dados sobre o meio ambiente e atuadores que executam tarefas de acordo com o que foi captado pelos sensores (ULLAH; SHAH, 2016) (ALDEEN; SALLEH, 2019).

Segundo Zhou e Zhang (ZHOU; ZHANG, 2014b), a IoT se insere no mundo em que tudo está conectado, podendo ser gerenciado de forma remota e possibilitando o aumento na eficiência das corporações e redução nos custos de produção. Ao longo da última década, estimou-se que o número de conexões IoT iria crescer de 6 milhões em 2015 para 27 bilhões em 2025 (GARTNER, 2020), conforme pode ser observado na Figura 1.

O mercado da IoT permanece complexo e fragmentado com mais verticalização e muitos fornecedores buscando novas funções e ofertas. As oportunidades comerciais que envolvem dados de monetização são desafiadas por padrões semânticos (extração de conhecimento dos objetos na IoT) fracos e questões de propriedade intelectual. Os riscos como a segurança cibernética aumentarão e a IoT está se tornando sujeita a mais regulamentações, diretrizes técnicas e certificações. Mas as oportunidades superarão em muito os desafios, e os recursos em rápido crescimento da IoT estão impulsionando a transformação digital em uma ampla variedade de setores (GARTNER, 2022).

Figura 1 – Conexões IoT globais projetadas.




**Fonte:** Autor.

A conectividade entre diferentes tipos de dispositivos, a falta de padronização de segurança, o rápido crescimento dos ambientes de IoT e a limitação dos recursos computacionais em dispositivos de IoT levaram a um aumento significativo do número de ataques cibernéticos. Tais ataques podem afetar a segurança dos ambientes nos quais eles estão inseridos, como casas inteligentes, cidades inteligentes e indústrias (RAZA; WALLGREN; VOIGT, 2014)(MIDI et al., 2017)(SHERASIYA; UPADHYAY, 2016)(RIZVI; KURTZ; RIZVI, 2016).

Os tipos de ameaças incluem ataques de roteamento, físicos, de transmissão de dados, criptográficos, ataques a gateways da IoT, ataques a dispositivos de borda ou exploração de vulnerabilidades a protocolos de comunicação (BOSTANI; SHEIKHAN, 2017). Por exemplo, mais de 1 bilhão de ataques de IoT ocorreram em 2021, quase 900 milhões dos quais foram ataques de phishing relacionados à IoT. Houve cerca de 62 milhões de ataques DDoS em 2021. Os dispositivos mais vulneráveis foram roteadores (46%), pontos de acesso e extensores (17%-17% cada), NAS (5%), VoIP (4%), câmeras e dispositivos domésticos inteligentes (3%-3% cada) (MARTON, 2022).

## 1.2 MOTIVAÇÃO

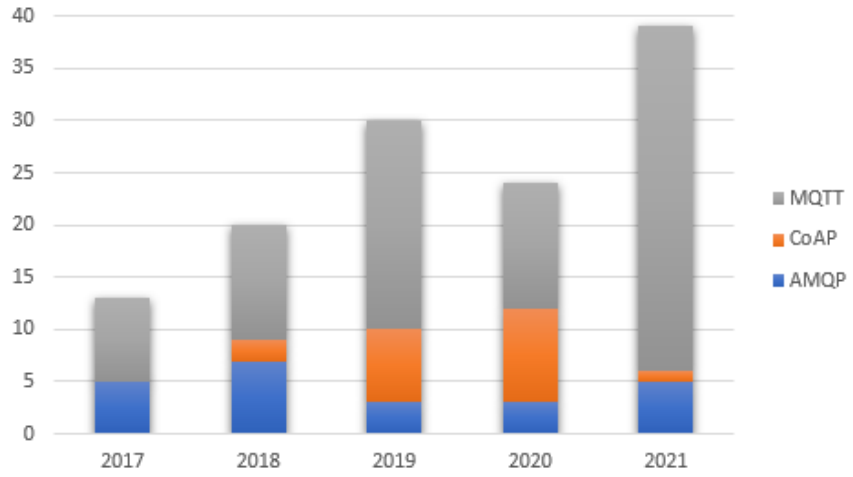
A presença de muitos objetos IoT inseguros e sem tratamento de segurança por seus designers, fabricantes e até proprietários podem ser facilmente explorados por criminosos. Por exemplo, uma falha de segurança em um dispositivo médico implantado pode representar

sérios riscos para os pacientes. Um ataque cibernético distribuído em carros conectados pode facilmente engarrafar cidades inteiras (NEBBIONE; CALZAROSSA, 2020).

O cenário de ameaças da IoT é extremamente amplo e complexo. Gartner prevê que mais de um quarto de todos os ataques cibernéticos contra empresas serão baseados em IoT até 2025 (GARTNER, 2022). Infelizmente, o mercado prioriza conveniência e preço em vez de segurança, que raramente é considerada no design das aplicações (NEBBIONE; CALZAROSSA, 2020).

Nesse cenário, os protocolos da camada de aplicação IoT, como Advanced Message Queuing Protocol (AMQP), Constrained Application Protocol (CoAP), Message Queue Telemetry Transport (MQTT), possuem pontos de vulnerabilidade bem conhecidos. Por exemplo, negação de serviço e força bruta (HUSNAIN et al., 2022; LARMO; RATILAINEN; SAARINEN, 2019). Estes protocolos são a base das comunicações entre aplicativos e serviços executados em diferentes dispositivos IoT (RAZA; WALLGREN; VOIGT, 2014; NEBBIONE; CALZAROSSA, 2020; AKASIADIS; PITSILIS; SPYROPOULOS, 2019). Assim, os criminosos podem explorar vulnerabilidades em tais protocolos para interromper e acessar a comunicação de informações confidenciais entre dispositivos IoT (MAGGI; VOSSELER; QUARTA, 2018). A Figura 2 apresenta a quantidade de vulnerabilidades encontradas dos protocolos AMQP, CoAP, MQTT nos últimos 5 anos de acordo com o CVE do *National Institute of Standards and Technology* (NIST) (STANDARDS; TECHNOLOGY, 2022).

Figura 2 – Quantidade de Vulnerabilidades encontradas nos últimos cinco anos.




**Fonte:** O Autor.

Portanto, é importante a existência de técnicas e ferramentas que possam detectar e mitigar ataques especificamente na camada de aplicação da IoT.

Trabalhos recentes têm usado técnicas como o Processamento de Eventos Complexos (CEP), fuzzy e aprendizagem de máquina para detectar comportamentos anormais em IoT. Roldan et al. (ROLDÁN et al., 2020) propuseram e implementaram um sistema inteligente, combinando CEP e aprendizado de máquina para detectar ataques de segurança IoT. Jun e Chi (CHEN; CHEN, 2014) propuseram uma arquitetura IDS para ambientes IoT baseada em Processamento de Eventos Complexos. Cardoso et al. (CARDOSO et al., 2018) apresentam uma estratégia para prevenir ataques DDoS (Distributed Denial of Services), executando em Raspberry Pi 3B - um dispositivo com poder computacional reduzido. Haripriya et al. (HARIPRIYA; KULOTHUNGAN, 2019) propuseram um método que utiliza um sistema baseado em lógica fuzzy para detectar comportamentos maliciosos em um ambiente IoT com a ajuda de um mecanismo de interpolação de regras fuzzy.

As estratégias de Jun e Chi (CHEN; CHEN, 2014) e Cardoso et al. (CARDOSO et al., 2018) não conseguem identificar ataques em nível de camada de aplicação. Além disso, as técnicas de Haripriya et al. (HARIPRIYA; KULOTHUNGAN, 2019) e Roldan et al. (ROLDÁN et al., 2020) são restritas ao protocolo MQTT e não são projetadas para lidar com outros protocolos na camada de aplicação.

## 1.3 PROBLEMA E QUESTÃO DE PESQUISA

Conforme mencionado, os protocolos da camada de aplicação da IoT contêm vulnerabilidades que precisam ser tratadas. Estes protocolos carecem de mecanismos de segurança, além de apresentarem falta de padronização de segurança, o que pode facilitar a exploração de vulnerabilidades por cyber criminosos (CIKLABAKKAL et al., 2019). Assim, os invasores podem explorar falhas em infraestruturas e comunicações que usam esses protocolos da camada de aplicação IoT (MAGGI; VOSSELER; QUARTA, 2018). Esses pontos de falhas podem envolver ataques do tipo negação de serviço (DoS/DDoS), *spoofing*, *man in the middle*, dentre outros. Para evitar ou mitigar esses ataques, é fundamental o desenvolvimento de sistemas de detecção e prevenção de intrusão (IDPS) (REPCEK, 2016).

Os IDPS's podem complementar os meios de segurança disponíveis para identificar possíveis eventos de segurança, alertar equipes de segurança ou agir automaticamente para bloquear as ameaças. Entretanto, as técnicas de IDPS normalmente não são projetadas para dispositivos com recursos limitados (memória e processamento), e não são adequadas para todos os cenários de aplicação pretendidos (KRIMMLING; PETER, 2014)(ROLDÁN et al., 2020).

Neste contexto, uma proposta de um IDPS que detecte ataques na camada de aplicação de sistemas IoT, até o presente momento, é considerada uma questão aberta. Além disso, de acordo com o mapeamento sistemático da literatura descrito no Capítulo 3, nenhum outro trabalho na literatura forneceu um IDPS que possa detectar ataques que explorem mais de um protocolo de aplicação da IoT. Devido à importância desses protocolos para os dispositivos (NEBBIONE; CALZAROSSA, 2020), esta tese pretende lidar com esta questão.

Diante do exposto, o problema que permeia esta pesquisa é: **Como projetar um IDPS que atue nos protocolos da camada de aplicação em IoT?**.

Assim, com base no problema citado acima, será investigada a seguinte hipótese:

- **Hipótese:** Considerando que a IoT é uma tecnologia que produz uma grande quantidade de dados, o uso de técnicas CEP para filtrar, categorizar e detectar padrões dos dados brutos dos protocolos da camada de aplicação melhora a precisão na detecção de ataques.

Depois de estabelecer essa hipótese, 2 (duas) questões de pesquisa (QP) serão respondidas no decorrer do trabalho:

- **QP1:** melhorar a detecção de ataques que exploram vulnerabilidades dos protocolos da camada de aplicação IoT?
- **QP2:** técnicas CEP fornecem alta precisão na detecção de ataques?

## 1.4 OBJETIVOS

### 1.4.1 Objetivo Geral

Esta tese visa melhorar a segurança de aplicações IoT através do desenvolvimento, um sistema chamado Beholder, para detectar e prevenir intrusões direcionadas aos protocolos da camada de aplicação da IoT.

### 1.4.2 Objetivos Específicos

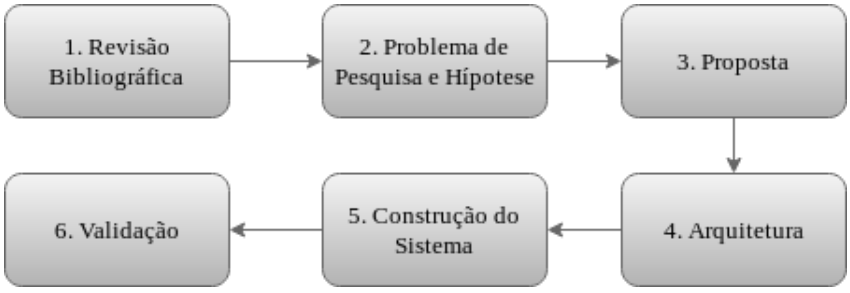
- **Projetar e implementar o Beholder Tracer**: trata-se de um analisador de pacotes brutos e mensagens para a camada de aplicação da IoT. Este componente permitirá a analise dos dados brutos das mensagens transmitidas;

- **Projetar uma arquitetura de IDPS para que seja flexível/extensível**: além da camada de aplicação da IoT e dos protocolos de transporte, o Beholder Tracer deve ser extensível a outros protocolos da camada de aplicação;
- **Desenvolver um Sistema de Detecção e Prevenção de Intrusão (IDPS) baseado na CEP *engine***: para detectar ataques, o IDPS utilizará processamento complexo de eventos (CEP), permitindo analisar informações relacionadas a diferentes eventos, como causalidade, temporalidade, sequências, agregações e composições para detectar as intrusões;
- **Avaliar desempenho na detecção**: para avaliar ao desempenho de detecção, esta tese irá utiliza duas métricas de avaliação na detecção de ataques: Precisão e Recall;
- **CPU e memória RAM**: trata-se da avaliação do consumo de CPU e memória RAM durante a execução do Beholder.

## 1.5 METODOLOGIA

Esta tese utiliza uma metodologia de pesquisa científica baseada em uma abordagem metodológica Hipotético-Dedutivo (NEWMAN; BENZ, 1998). Nesse método, a investigação científica visa construir e testar uma possível resposta ou solução para um problema (ANHEMBI, 2015). A Figura 3 apresenta a metodologia adotada.

Figura 3 – Metodologia de Pesquisa.




**Fonte:** O Autor.

Uma vez realizada a revisão da literatura, define-se o problema e objetivos com base na hipótese apresentada. Depois, a solução proposta é elaborada, arquitetura definida, e o sistema é implementado. Por fim, um conjunto de experimentos são realizados em um cenário real para avaliar a taxa de detecção dos ataques que exploram as vulnerabilidades dos protocolos

da camada de aplicação. Neste trabalho será avaliada a detecção de ataques nos protocolos AMQP, CoAP e AMQP em dispositivos IoT.

## 1.6 CONTRIBUIÇÕES

A principal contribuição deste trabalho é o desenvolvimento de um IDPS baseado em CEP chamado Beholder para detectar e prevenir intrusões e com baixo consumo de memória RAM e CPU, que exploram vulnerabilidades em protocolos da camada de aplicação IoT. A capacidade de detectar ataques no protocolo TCP obtida através dos experimentos realizados é semelhante ao conhecido Snort IDS (CISCO, 2020). Porém, o Beholder avança e permite previnir ataques contra outros protocolos, como AMQP, CoAP e MQTT com alto nível de precisão.

O Beholder inclui dois componentes principais:

- **Beholder Tracer.** Este componente analisa os pacotes da camada de aplicação IoT para selecionar os pacotes AMQP, CoAP e MQTT e enviá-los para a análise do CEP Engine.
- **CEP Engine.** Este componente é responsável pela detecção dos ataques. O mecanismo CEP analisa, em tempo real, eventos para identificar padrões que possam representar ataques aos protocolos da camada de aplicação.

## 1.7 ESCOPO NEGATIVO

Essa seção descreve o que este trabalho não se propõe a pesquisar, analisar ou resolver. A lista seguinte enumera alguns itens relacionados à temática abordada e que não fazem parte do escopo investigado, uma vez que a pesquisa prioriza as questões relacionadas à precisão de detecção de ataques à camada de aplicação da IoT

1. Privacidade de dados: Está fora do escopo desta tese investigar aspectos associados à privacidade dos dados na IoT. A privacidade dos dados se concentra nos direitos dos indivíduos, na finalidade de coleta, processamento de dados, nas preferências de privacidade e na maneira como as organizações controlam os dados pessoais.
2. Bloqueio dos ataques. Por se tratar de um trabalho que analisa a taxa de detecção dos ataques, se o módulo de bloqueio do Beholder estivesse ativado, bloquearia a comunicação

do atacante com a vítima no instante da primeira detecção. Desta forma, impediria a continuidade dos experimentos.

## 1.8 ORGANIZAÇÃO DO DOCUMENTO

As demais partes deste documento estão organizadas da seguinte maneira:

- **Capítulo 2 - Fundamentos:** este capítulo introduz os conceitos fundamentais utilizados nesta tese, tais como abordagem de detecção de intrusão, camadas de comunicação da internet das coisas, protocolos AMQP, CoAP, MQTT e processamento de eventos complexos (CEP).
- **Capítulo 3 - Trabalhos Relacionados:** apresenta os trabalhos relacionados da seguinte forma: detecção de intrusão baseado em técnicas CEP, IDS com técnicas de Inteligência Artificial, IDS com técnicas Fuzzy, IDS que utilizam técnicas tradicionais para detecção, e IDS que utilizam outras técnicas.
- **Capítulo 4 - Beholder**: detalha a solução proposta e seus principais componentes: Beholder Tracer e CEP *engine*.
- **Capítulo 5 - Avaliação e Resultados:** apresenta a avaliação executada e os resultados alcançados.
- **Capítulo 6 - Conclusão:** apresenta as considerações finais desta tese e os trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais conceitos necessários para o entendimento das contribuições desta tese. Inicialmente, são abordados conceitos sobre princípios de sistemas de detecção de intrusão, camadas da arquitetura IoT, os principais protocolos da camada de aplicação e, por fim, *Complex Event Processing* (CEP).

## 2.1 PRINCÍPIOS DE DETECÇÃO E PREVENÇÃO DE INTRUSÕES

Detecção de intrusão é o processo de monitorar eventos que ocorrem em um sistema ou rede de computadores e analisá-los em busca de possíveis incidentes, violações ou ameaças iminentes a políticas de segurança de computadores, políticas de uso aceitável ou padrão de práticas de segurança (SCARFONE; MELL, 2007)(SCARFONE; MELL, 2012). As principais metodologias de detecção de intrusão são conhecidas como: detecção baseada em anomalias, assinatura, especificação e híbrida.

### 2.1.1 Detecção Baseada em Anomalias

A detecção baseada em anomalias analisa eventos observados para identificar desvios em relação ao comportamento considerado normal. Um IDPS (Intrusion Detection and Prevention System) que usa detecção baseada em anomalias possui perfis para representar o comportamento normal de usuários, *hosts*, conexões de rede ou aplicativos. O principal benefício dos métodos de detecção baseados em anomalias é que eles podem ser úteis na detecção de ameaças ainda desconhecidas (SCARFONE; MELL, 2007) (MIDI et al., 2017). A desvantagem da detecção de anomalias é que um alarme é gerado sempre que o tráfego ou atividade se desvia dos padrões de tráfego ou atividade "normais"definidos. Isso significa que cabe ao administrador de segurança descobrir por que um alarme foi gerado. Podendo também gerar um número alto de falsos positivos (MIDI et al., 2017).

### 2.1.2 Detecção Baseada em Assinatura

Esse tipo de IDPS compara o comportamento de um sistema ou um conjunto de assinaturas que está armazenada numa base de dados internos do IDPS. A detecção baseada em assinatura

é muito eficaz na detecção de ameaças conhecidas, mas é altamente ineficaz na detecção de ameaças desconhecidas, ameaças mascaradas pelo uso de técnicas de evasão e muitas variantes de riscos conhecidos (SCARFONE; MELL, 2007) (RAZA; WALLGREN; VOIGT, 2014).

### 2.1.3 Detecção Baseada em Especificação

As abordagens baseadas em especificação detectam invasões quando o comportamento da rede se desvia do comportamento previamente especificado. Portanto, a detecção baseada em especificação tem o mesmo objetivo que a detecção baseada em anomalia: identificar desvios da rotina. Porém, nesta abordagem, um especialista precisa definir manualmente as regras para detectar os ataques em um ambiente específico. Além disso, os sistemas de detecção baseados em especificações não precisam de uma fase de treinamento, pois podem começar a funcionar imediatamente após a configuração das especificações (ZARPELÃO et al., 2017).

### 2.1.4 Detecção Híbrida

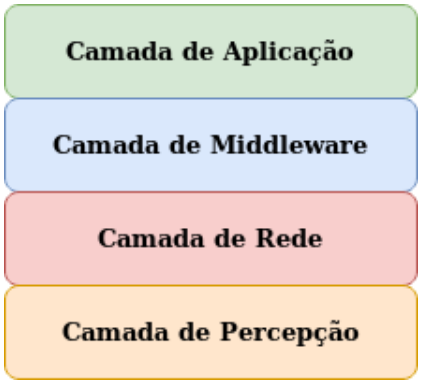
As metodologias de detecção híbridas são técnicas e conceitos baseados em assinatura, especificação e/ou anomalia. Este tipo de método tem por objetivo maximizar as vantagens e minimizar o impacto das desvantagens de cada método (SCARFONE; MELL, 2007) (MIDI et al., 2017).

## 2.2 ARQUITETURA DE CAMADAS IOT

A literatura existente oferece diferentes formas de categorizar as camadas da arquitetura IoT (ZHU et al., 2010), (WANG et al., 2010), (ATZORI; IERA; MORABITO, 2010), (KHAN et al., 2012), (GUBBI et al., 2013), (XU; HE; LI, 2014), (LI; XU; ZHAO, 2015), (ARA; SHAH; PRABHAKAR, 2016), (HESSEL, 2016), (ALLOUCH; AMECHNOUE; ACHATBI, 2017), (SETHI; SARANGI, 2017), (ALDOAIES, 2018), (APPROACH et al., 2018).

Após analisar essas diferentes visões para identificar interpretações comuns nas camadas da arquitetura da IoT, esta tese assumiu uma arquitetura com quatro camadas principais: percepção; rede; middleware; e aplicação. A Figura 4 ilustra as camadas da arquitetura IoT utilizada nesta tese.

Figura 4 – Camadas da Arquitetura IoT.




**Fonte:** O Autor.

- **Camada de percepção**: consiste em objetos físicos e dispositivos como sensores (Radio Frequency Identification (RFID), Bluetooth, leitores de infravermelho, coletor de dados de umidade DHT11, comunicação de campo próximo (Near-field Communication (NFC)), sensores (Wireless Sensor Network (WSN)), etc.). Essa camada identifica e coleta informações de objetos através de sensores. Após reunir as informações necessárias (temperatura, calor, geolocalização, presença, umidade etc.), o dispositivo as envia para a camada de rede (KHAN et al., 2012) (ARA; SHAH; PRABHAKAR, 2016) (ALLOUCH; AMECHNOUE; ACHATBI, 2017).

- **Camada de rede**: a camada de rede transfere as informações adquiridas na camada de percepção. A comunicação pode usar as tecnologias sem fio, 2.5G, 3G, 4G, Wifi, Bluetooth, LoRaWAN e ZigBee. Essa camada gerencia serviços de rede e executa protocolos de roteamento (KHAN et al., 2012) (ARA; SHAH; PRABHAKAR, 2016) (ALLOUCH; AMECHNOUE; ACHATBI, 2017).

- **Camada de middleware**: o objetivo da camada de middleware é promover a transparência na comunicação de sistemas distribuídos de forma que a complexidade da distribuição seja ocultada dos desenvolvedores de aplicações (ATZORI; IERA; MORABITO, 2010) (KHAN et al., 2012) (ARA; SHAH; PRABHAKAR, 2016) (ALLOUCH; AMECHNOUE; ACHATBI, 2017) (SCHMIDT; BUSCHMANN, 2003).

- **Camada de aplicação**: a camada de aplicação é responsável por gerenciar os aplicativos em ambientes da IoT, como em residências, cidades inteligentes, carros independentes, indústrias, computação em nuvem etc (KHAN et al., 2012) (ARA; SHAH; PRABHAKAR, 2016) (ALLOUCH; AMECHNOUE; ACHATBI, 2017).

## 2.3 PROTOCOLOS DA CAMADA DE APLICAÇÃO DA INTERNET DAS COISAS

Os protocolos citados na literatura da camada de aplicação IoT (YASSEIN; SHATNAWI; ALZOUBI, 2016) são: Advanced Message Queuing Protocol (AMQP), The Constrained Application Protocol (CoAP), Restful, Data Distribution System (DDS), Message Queuing Telemetry Transport (MQTT), Extensible Messaging and Presence Protocol (XMPP) e WEBSOCKET. Dentre os protocolos citados, segundo a ADLINK (CORSARO, 2020), os protocolos da camada de aplicação mais discutidos e aplicados para IoT são os protocolos AMQP, CoAP, DDS e MQTT. Esta tese se concentra nos protocolos AMQP, CoAP e MQTT. O DDS não foi incluído por ser um protocolo proprietário, o que iria inviabilizar seu uso no cenário e experimentos desta tese.

### 2.3.1 AMQP (*Advanced Message Queuing Protocol*)

O AMQP é um protocolo que realiza transferência binária sendo projetado para aplicativos corporativos e comunicação servidor-para-servidor. É um protocolo para enviar e receber mensagens de forma assíncrona. Essas mensagens, quando enviadas, não recebem uma resposta imediata. O AMQP pode ser equivalente ao Hypertext Transfer Protocol (HTTP), com um cliente capaz de se comunicar com um broker (PATIERNO, 2018) (OASIS, 2012).

O AMQP suporta a comunicação Publisher/Subscriber, possuindo alguns níveis de serviço de entrega como até uma vez, ao menos uma vez e exatamente uma vez, semelhante ao protocolo MQTT. As mensagens trocadas entre os publishers e subscribers. ocorrem por meio do Transmission Control Protocol (TCP), e usa uma conexão ponto-a-ponto confiável (PATIERNO, 2018) (OASIS, 2012).

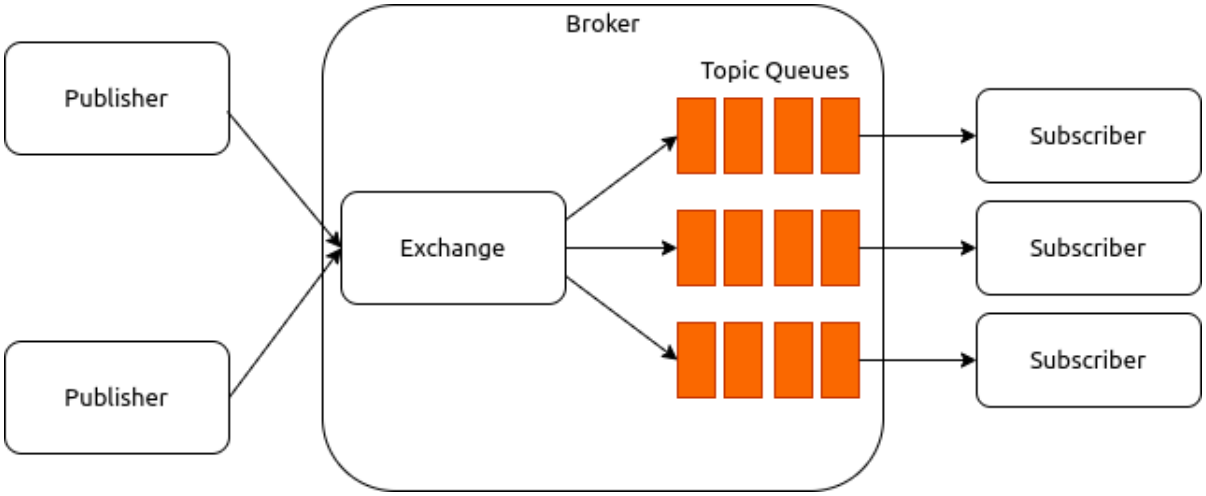
#### *2.3.1.1 Comunicação*

A comunicação do AMQP é realizada e por composta por publishers, broker e subscribers. Os Publishers, por exemplo, podem ser sensores de umidade e temperatura que enviam mensagens para um broker. O broker AMQP é constituído por um *Exchange* e *Queues*. O *Exchange* tem por objetivo receber as mensagens dos *Publishers* e rotear as mensagens para as *Queues* (OASIS, 2012).

Por sua vez, as *Queues* têm por objetivo armazenar as mensagens recebidas pelo *Exchange*

em memória ou em disco. Em seguida, as mensagens são enviadas em sequência para os *subscribers* (OASIS, 2012). A Figura 5 ilustra a comunicação AMQP.

Figura 5 – Comunicação AMQP




**Fonte:** O Autor.

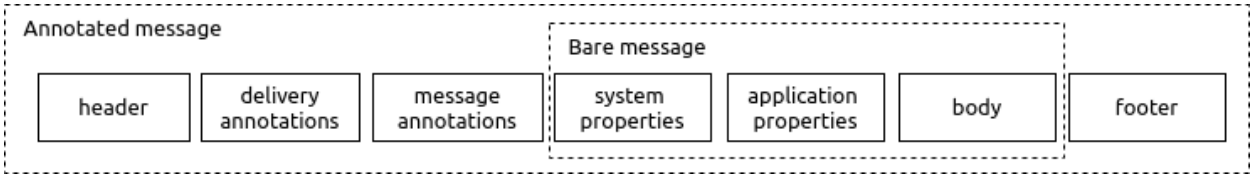
#### *2.3.1.2 Formato da Mensagem*

As aplicações baseadas no protocolo AMQP enviam mensagens que contêm várias informações como políticas de armazenamento, informações de entrega, propriedades da aplicação e seus dados, além de outras informações (OASIS, 2012). Essas informações são apresentadas na Figura 6.

Assim, o quadro de mensagem tem uma estrutura composta de duas partes principais (OASIS, 2012):

- *Bare Message*: é a parte imutável do, e nenhum intermediário pode alterar seu conteúdo;
- *Annotated Message*: Parte da mensagem que pode ser usada e alterada por intermediários entre remetente e destinatário. A *Annotaded Message* contém o corpo e dois tipos de coleções. A primeira é para propriedades padrões e que são definidas pelo AMQP; e a segunda parte permite a definição de propriedades específicas do aplicativo que podem ser adicionadas e alteradas pela aplicação.

Figura 6 – Mensagem AMQP



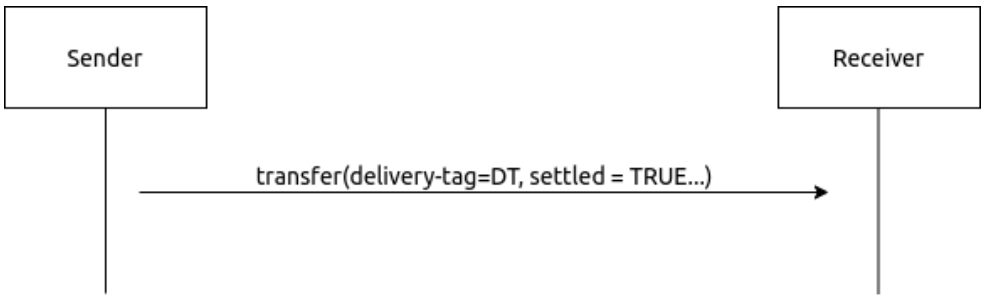

**Fonte:** Adaptado de (OASIS, 2012).

#### *2.3.1.3 Qualidade de Serviço*

O protocolo AMQP tem um mecanismo para garantir o recebimento da mensagem. A qualidade de serviço no AMQP ocorre durante a transmissão da mensagem usando uma tag de entrega. Com essa tag é possível o rastreamento da entrega da mensagem (OASIS, 2012). O protocolo AMQP fornece três níveis de qualidade de serviço:

**At-most-once**: ocorre quando a mensagem é entregue no máximo uma vez. Pode acontecer que a mensagem esteja perdida na rede e não chegue ao receptor. O remetente não recebe nenhuma informação sobre o recebimento da mensagem e não a reenvia. A Figura 7 ilustra essa comunicação.

Figura 7 – *At-most-once*

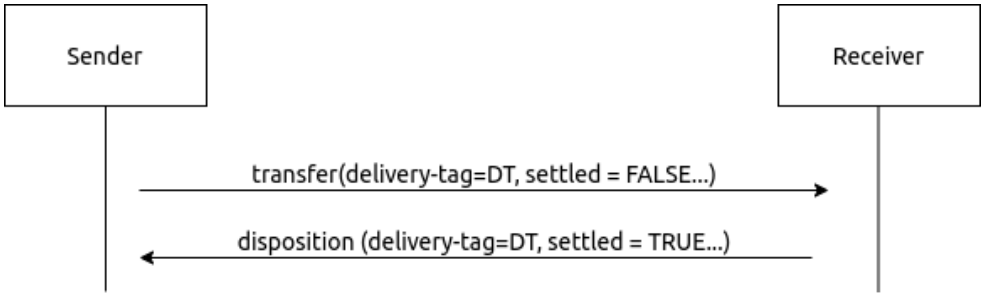



**Fonte:** Adaptado de (OASIS, 2012).

Na entrega *At-most-once*, o remetente envia a mensagem, mas não espera confirmação do seu recebimento (settled = TRUE); ou seja, a mensagem pode ser recebida ou não, mas não se espera confirmação.

**At-least-once**: neste cenário, a mensagem pode ser entregue uma ou mais vezes. Para cada mensagem, o remetente deve receber uma confirmação pelo destinatário. Se o destinatário receber a mensagem, mas a confirmação for perdida, o remetente reenvia a mensagem. A Figura 8 ilustra essa comunicação.

Figura 8 – *At-least-once*




**Fonte:** Adaptado de (OASIS, 2012).

O remetente envia a mensagem em status não estabelecido (settled = FALSE) porque deseja receber uma confirmação do destinatário. Durante a transferência, a mensagem ainda não foi resolvida no mapa interno do remetente. Uma vez recebida a mensagem, o destinatário responde com uma mensagem como *settled*; recebendo a confirmação, o remetente altera o status de *settlement* para *settled* e a mensagem não está mais sendo transmitida.

**Exactly-once**: ocorre quando o remetente envia uma mensagem com *settled=True*. Todas as mensagens serão entregues exatamente uma vez.

### 2.3.2 CoAP (*Constrained Application Protocol*)

É um protocolo projetado para comunicação entre dispositivos com recursos de memória e processamento limitados. O CoAP permite a comunicação Web, usada no M2M com requisitos restritos, troca de mensagens assíncronas, URI (*Uniform Resource Identifier*) e suporte ao tipo de conteúdo e recursos de proxy e cache (SHELBY; STUREK; FRANK, 2010).

Alguns recursos do CoAP são muito semelhantes ao HTTP. Porém, o CoAP não pode ser considerado um protocolo HTTP compactado. O CoAP foi projetado especificamente para a IoT e para comunicações M2M (Machine-to-machine – no português, máquina-para-máquina).

#### *2.3.2.1 Métodos*

O CoAP faz uso dos métodos GET, PUT, POST e Delete, de forma semelhante ao HTTP. Esses métodos são modificados para atender a requisitos da IoT, como baixo consumo de energia (SHELBY; STUREK; FRANK, 2010)(SHELBY K. HARTKE, 2014). A seguir, as definições dos métodos são apresentadas.

- **GET** - recupera uma representação para as informações que correspondem ao recurso identificado pelo (URI) da solicitação realizada pelo cliente.
- **POST** - solicita que a representação incluída no pedido seja processada, resultando geralmente em uma criação ou atualização de um recurso.
- **PUT** - solicita que um recurso identificado pela solicitação URI seja criado ou atualizado com uma representação em anexo.
- **DELETE** - solicita que o recurso identificado pelo URI seja excluído.

Em cada requisição é retornado um código de resposta. Os códigos são apresentados na Tabela 1.

Tabela 1 – Códigos de Resposta do Protocolo CoAP

| Código de Resposta | Definição |
|---|---|
| 2.xx | **Sucess** |
| 2.01 | Created |
| 2.02 | Deleted |
| 2.03 | Valid |
| 2.04 | Changed |
| 2.05 | Content |
| 4.xx | **Client Error** |
| 4.00 | Bad Request |
| 4.01 | Unauthorized |
| 4.02 | Bad Option |
| 4.03 | Forbidden |
| 4.04 | Not found |
| 4.05 | Method not allowed |
| 4.06 | Not Acceptable |
| 4.12 | Precondition Failed |
| 4.13 | Request Entity Too Large |
| 4.15 | Unsupported Content-Format |
| 5.xx | **Server Error** |
| 5.00 | Internal Server Error |
| 5.01 | Not Implemented |
| 5.02 | Bad Gateway |
| 5.03 | Service Unavailable |
| 5.04 | Gateway Timeout |
| 5.05 | Proxying Not Supported |

**Fonte:** Adaptado de (SHELBY; STUREK; FRANK, 2010).

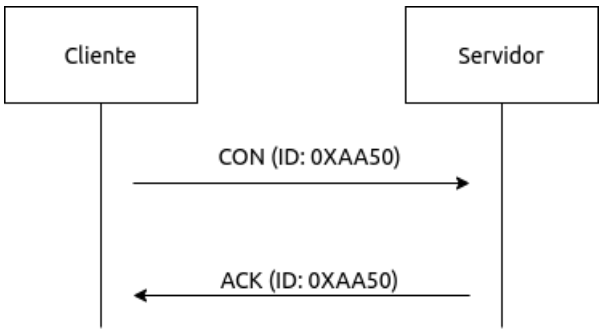
#### *2.3.2.2 Comunicação*

No CoAP existem duas camadas diferentes: *Mensagens* e *Request/Response*. A camada Mensagens lusa o User Datagram Protocol (UDP) e lida com mensagens assíncronas. A camada *Request/Response* gerencia a interação *Request/Response* com base nas mensagens desse tipo. O protocolo CoAP tem suporte a quatro tipos de mensagens: *Conformable*(CON), *Non-Conformable*(NON), *Acknowledgement*(ACK) e *Reset*(RST).

O CoAP utiliza o UDP para a troca de mensagens entre pontos da extremidade. Cada mensagem possui um identificador para identificar duplicatas de mensagem. Uma mensagem CoAP utiliza dois tipos de mensagens: *Confirmable message* e *Non-confirmable message*.

Uma mensagem do tipo *Confirmable* é confiável na troca de mensagens entre dois nós. No CoAP, uma mensagem confiável é obtida usando uma mensagem do tipo (CON). Usando esse tipo de mensagem, o cliente pode saber se a mensagem chegou ou não ao servidor. Uma mensagem CON é enviada repetidas vezes até que a outra parte envie uma mensagem de confirmação (ACK). A mensagem ACK contém o mesmo identificador da mensagem confirmável (CON) (SHELBY; STUREK; FRANK, 2010). A Figura 9 ilustra essa comunicação.

Figura 9 – Troca de Mensagens CON e ACK

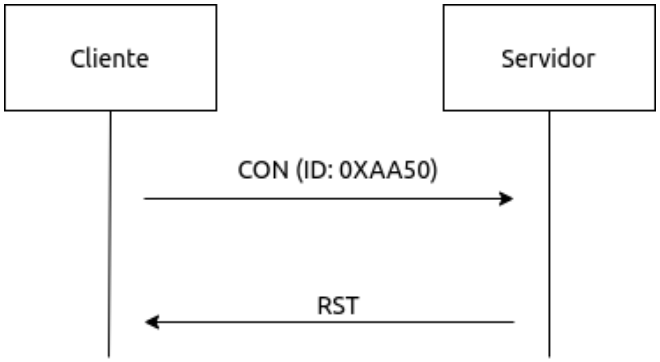



**Fonte:** Adaptado de (SHELBY; STUREK; FRANK, 2010).

Se em algum momento o servidor tiver problemas para retornar a mensagem recebida, poderá enviar uma mensagem do tipo *Reset* (RST - que indica que uma mensagem específica (Confirmável ou Não confirmável) foi recebido, mas falta algum contexto para processá-lo adequadamente) ao invés da mensagem ACK. A Figura 10 ilustra essa comunicação.

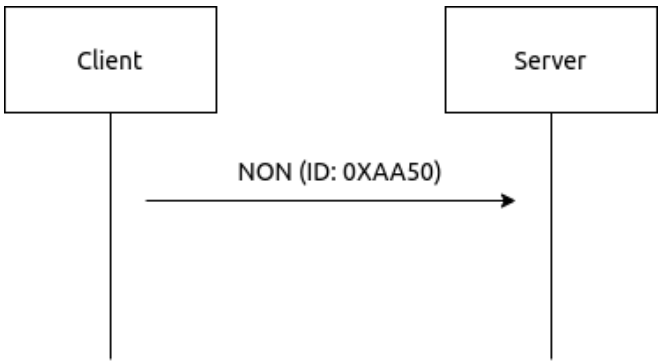
As mensagens do tipo *Non-confirmable* não exigem um reconhecimento pelo servidor. Esses tipos de mensagens não contêm informações críticas. Mesmo que essas não sejam confiáveis, elas possuem ID Exclusivo. A Figura 11 ilustra essa comunicação.

Figura 10 – Troca de Mensagens CON e RST




**Fonte:** Adaptado de (SHELBY; STUREK; FRANK, 2010).

Figura 11 – Troca de Mensagem NON




**Fonte:** Adaptado de (SHELBY; STUREK; FRANK, 2010).

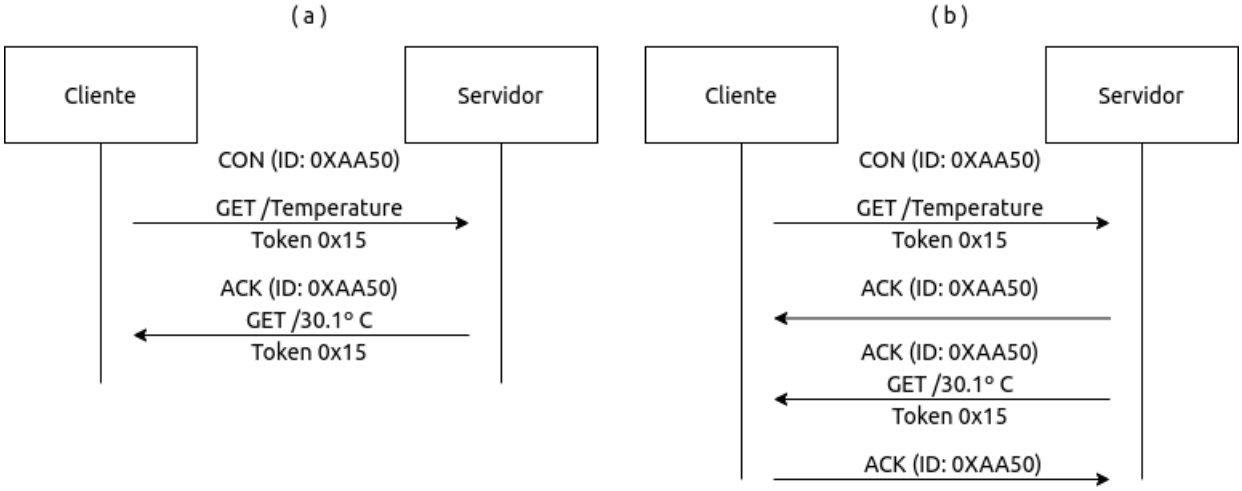
Durante a troca de mensagens, o servidor pode responder imediatamente à solicitação do cliente, caso a solicitação seja realizada usando uma mensagem confirmada (CON). O servidor pode também enviar de volta ao cliente uma mensagem de confirmação contendo código de erro. Na Figura 12a, é possível perceber que a mensagem possui um token.

Caso o servidor não possa responder uma solicitação do cliente imediatamente, ele enviará uma mensagem CON com uma resposta vazia. Assim que a resposta estiver disponível, o servidor envia uma mensagem CON para o cliente. Em seguida, o cliente envia uma mensagem ACK (Figura 12b).

#### *2.3.2.3 Formato da Mensagem*

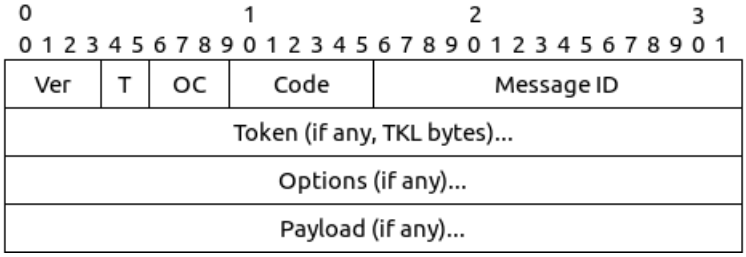
O CoAP usa um modelo de request e response. Cada dispositivo atua como cliente ou servidor e seus recursos podem ser acessados por meio de um URI. Ao contrário do HTTP, a comunicação não ocorre antes da troca da mensagem, mas de forma assíncrona. A Figura 13 ilustra o cabeçalho de mensagem CoAP.

Figura 12 – Troca de Mensagens CON e ACK




**Fonte:** Adaptado de (SHELBY; STUREK; FRANK, 2010).

Figura 13 – Cabeçalho de Mensagem do Protocolo CoAP




**Fonte:** Adaptado de (SHELBY; STUREK; FRANK, 2010).

A seguir, são apresentados os campos do cabeçalho de mensagem CoAP segundo a IETF (SHELBY K. HARTKE, 2014).

- **Version (Ver)**: indica o número da versão do CoAP;
- **Type(T)**: este campo indica se esta mensagem é do tipo Confirmable (0), Non-confirmable (1), Acknowledgement (2), or Reset (3);
- **Code**: indica uma classe que pode realizar uma solicitação (0), uma resposta bem-sucedida (2), uma resposta de erro do cliente (4) ou uma resposta de erro do servidor (5) (Todos os outros valores de classe são reservados);
- **Comprimento do token (TKL)**: indica o tamanho do campo Token de tamanho variável (0 a 8 bytes). Os tamanhos 9 a 15 são reservados, não devem ser enviados e devem ser processados como um erro no formato da mensagem;
- **Message ID**: é utilizado para detectar duplicação de mensagens.

### 2.3.3 MQTT (*Message Queue Telemetry Transport*)

MQTT é um protocolo de transporte de mensagens de publish/subscriber. Este protocolo é conhecido por consumir pouco recurso computacional, ser aberto e fácil de implementar. Essas funcionalidades o tornam útil para uso em vários aplicativos, como ambientes restritos, como Machine to Machine (M2M) e comunicação IoT (INCORPORATING, 2019).

Publicar-assinar é um padrão de mensagens em que os remetentes de mensagens (editores) não especificam o endereço dos destinatários das mensagens. Em vez disso, eles enviam mensagens com um determinado tópico. Todo dispositivo que assinou (assinantes) o tópico pode receber as mensagens.

#### *2.3.3.1 Comunicação*

O MQTT é um protocolo de padrão aberto fácil de implementar devido à sua boa documentação. Ele tem sido usado em várias aplicações, como M2M e comunicação IoT (INCORPORATING, 2019).

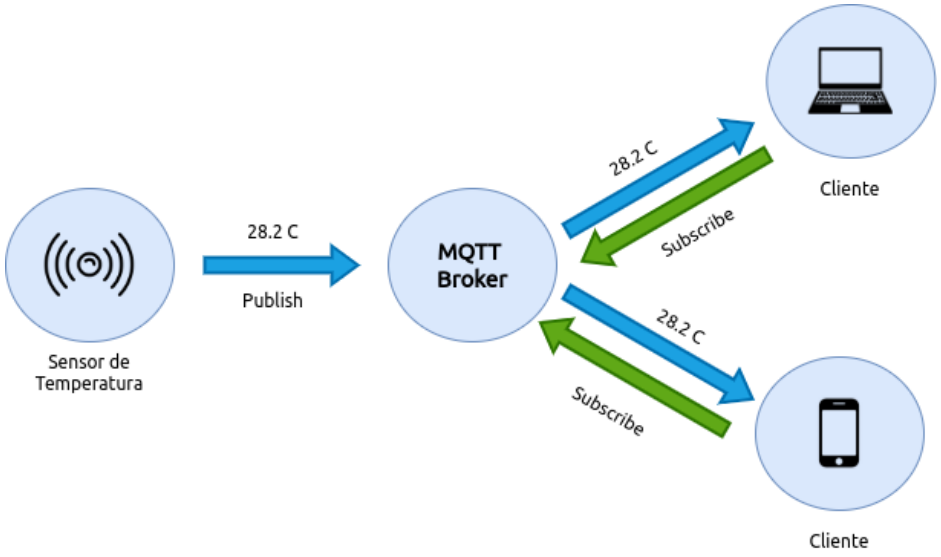
Conforme ilustrado na Figura 14, um cliente MQTT pode ser, por exemplo, um sensor de temperatura em uma casa inteligente. Portanto, os clientes podem se subscrever a um tópico da mensagem (temperatura) disponível no MQTT Broker. Assim, quando o sensor publicar mensagens sobre esse tópico, todos os clientes inscritos são notificados e podem receber a mensagem.

#### *2.3.3.2 Formato da Mensagem*

O cabeçalho fixo possui 2 bytes, o qual pode ser visualizado na Figura 15, onde o byte 1 contém o tipo da mensagem representado por um valor não assinado de 4 bits, sendo esses os bits iniciais (INCORPORATING, 2019).

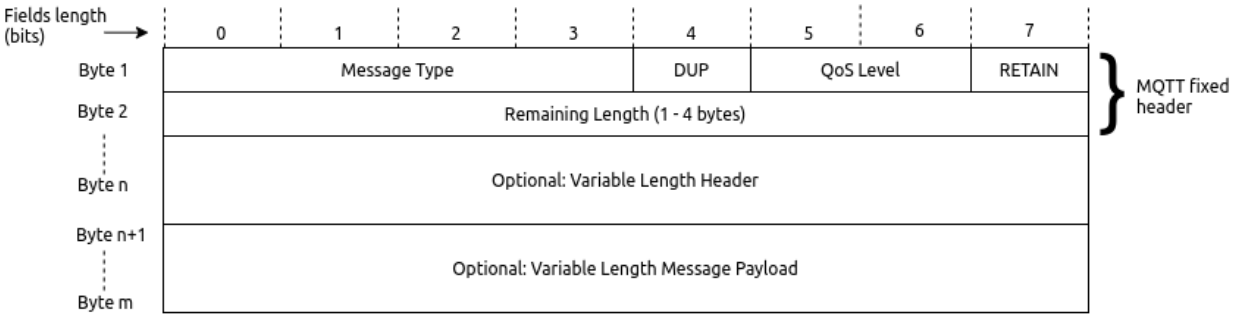
- **Message Type**: indica o tipo de mensagem, que pode ser *CONNECT*, *SUBSCRIBE*, *DISCONNECT*, *PUBLISH*, *PUBACK*, *PUBREC*, *PUBREL*, *PUBCOMP*, *PINGREQ*, *UNSUBACK*, *PINGRESP*, *SUBACK*, *UNSUBSCRIBE*
- **DUP**: é usado para indicar que uma mensagem é duplicada e foi reenviada porque o destinatário não reconhece a mensagem original;

Figura 14 – Comunicação MQTT




**Fonte:** O Autor.

Figura 15 – Formato de Mensagem MQTT




**Fonte:** Adaptado de (INCORPORATING, 2019).

- **QoS Level**: para garantir a entrega, o MQTT é implementado em seu cabeçalho com o campo Quality of Service (QoS). QoS é um contrato entre o remetente de uma mensagem e o seu destinatário;
- **RETAIN**: esse campo indica que a mensagem foi retida. O broker armazena a última mensagem retida e QoS que corresponde a esse tópico. As mensagens retidas ajudam os clientes recém-assinados a obter uma atualização de status de envio de mensagem após assinarem um tópico;
- **Remaining Length**: indica o número de bytes restantes na mensagem atual, incluindo dados no cabeçalho da variável e a carga útil. O esquema de codificação de comprimento variável usa um único byte para mensagens de até 127 bytes;

- **Variable Length Header**: este campo não está presente em todos os pacotes. Ele é usado apenas para fornecer informações adicionais quanto ao tamanho do cabeçalho variável. No cabeçalho da variável, primeiro, deve haver o nome do protocolo. E, para isso, os primeiros 2 bytes indicam o comprimento e o nome do protocolo;
- **Payload**: dados que estão incluídos em alguns pacotes. O pacote MQTT pode conter uma carga útil ou não, podendo variar pelo pacote enviado. A carga útil contém os dados que estão sendo enviados.

O protocolo MQTT define métodos de controle para determinar a ação a ser executada que são distribuídas entre o cabeçalho fixo, cabeçalho variável e payload. Os controles estão listados na Tabela 2.

Tabela 2 – Pacotes de Controle MQTT

| Nome | Valor | Definição |
|---|---|---|
| Reserved | 0 | Reservado |
| CONNECT | 1 | Solicitação do subscriber para se conectar ao broker |
| CONNACK | 2 | Reconhecimento da conexão |
| PUBLISH | 3 | Mensagem publish |
| PUBACK | 4 | ACK da Publicação |
| PUBREC | 5 | Publicação recebida |
| PUBREL | 6 | Publicação liberada |
| PUBCOMP | 7 | Publicação completa |
| SUBSCRIBE | 8 | Solicitação subscribe |
| SUBACK | 9 | ACK subscribe |
| UNSUBSCRIBE | 10 | Solicitação de cancelamento de subscribe |
| UNSUBACK | 11 | ACK de cancelamento de subscribe |
| PINGREQ | 12 | Solicitação PING |
| PINGRESP | 13 | Resposta PING |
| DISCONNECT | 14 | subscriber desconectado |
| Reserved | 15 | Reservado |

**Fonte:** Adaptado de (INCORPORATING, 2019).

- CONNECT: usada para estabelecer uma conexão TCP com o servidor de mensageria;
- CONNACK: mensagem enviada pelo broker em resposta a uma solicitação CONNECT de um cliente;
- PUBLISH: comando usado para publicar uma mensagem. Cada mensagem PUBLISH está associada a um nome de tópico ou assunto;

- PUBACK: uma mensagem PUBACK é a resposta a uma mensagem PUBLISH com QoS nível 1. Uma mensagem PUBACK é enviada por um broker em resposta a uma mensagem PUBLISH de um cliente de publicação e por um subscriber em resposta a uma mensagem PUBLISH do servidor;
- PUBREC: uma mensagem PUBREC é a resposta a uma mensagem PUBLICAR com QoS nível 2. É a segunda mensagem do fluxo de protocolo QoS nível 2. Uma mensagem PUBREC é enviada pelo broker em resposta a uma mensagem PUBLISH de um cliente de publicação ou por um subscriber em resposta a uma mensagem PUBLISH do servidor;
- PUBREL: uma mensagem PUBREL é a resposta de um editor a uma mensagem PUBREC do servidor ou do servidor a uma mensagem PUBREC de um subscriber. É a terceira mensagem no fluxo do protocolo QoS 2;
- PUBCOMP: essa mensagem é a resposta do broker a uma mensagem PUBREL de um editor ou a resposta de um subscriber a uma mensagem PUBREL do servidor. É a quarta e última mensagem no fluxo do protocolo QoS 2;
- SUBSCRIBE. permite que um cliente registre um interesse em um ou mais nomes de tópicos;
- SUBACK: mensagem é enviada pelo broker ao subscriber para confirmar o recebimento de uma mensagem SUBSCRIBE. Uma mensagem SUBACK contém uma lista de níveis de QoS concedidos. A ordem dos níveis de QoS concedidos na mensagem SUBACK corresponde à ordem dos nomes dos tópicos na mensagem SUBSCRIBE correspondente;
- UNSUBSCRIBE: mensagem enviada pelo subscriber ao broker para cancelar a assinatura de tópicos nomeados;
- UNSUBACK: a mensagem enviada pelo servidor ao cliente para confirmar o recebimento de uma mensagem UNSUBSCRIBE;
- PINGREQ: mensagem enviada de um subscriber conectado ao servidor para verificar se existe conexão entre eles;
- PINGRESP: mensagem enviada por um broker a uma mensagem PINGREQ;
- DISCONNECT: mensagem enviada do subscriber para o broker para indicar que ele está prestes a encerrar sua conexão TCP/IP.

#### *2.3.3.3 Qualidade de Serviço (QoS)*

A QoS pode assumir três níveis (HIVEMQ, 2019) (INCORPORATING, 2019):

- No máximo uma vez (0) - não há garantia de entrega; Uma mensagem PUBLISH é enviada com QoS igual a zero e o receptor não envia confirmação de recebimento;
- Pelo menos uma vez (1) - existe uma garantia de entrega, mas com o risco de um pacote ser entregue várias vezes; Uma mensagem PUBLISH é enviada com QoS igual a zero e o receptor não envia confirmação de recebimento;
- Exatamente uma vez (2) - existe uma garantia de que cada mensagem é recebida apenas uma vez pelos subscribers.

  O MQTT usa as seguintes etapas para enviar pacotes de QoS 2:

  1. Uma mensagem do tipo PUBLISH com QoS 2 é enviado e armazenado;
  2. O subscriber responde com um pacote PUBREC para confirmar a mensagem PUBLISH inicial;
  3. Se o publisher não receber um mensagem PUBREC, ele o reenviará novamente com um sinalizador duplicado (DUP) até receber uma confirmação;
  4. Assim que o remetente recebe uma mensagem PUBREC do subscriber, ele pode descartar com segurança o pacote PUBLISH inicial;
  5. O publisher armazena a mensagem PUBREC do subscriber e responde com uma mensagem PUBREL;
  6. Quando o subscriber obtém a mensagem PUBREL, ele pode descartar todos os estados armazenados e responder com um pacote PUBCOMP;
  7. Quando o remetente obtém o pacote PUBCOMP, ele pode descartar todos as mensagens armazenados.

Como pode ser observado, o MQTT QoS 2 é pode acabar utilizando muita memória, processador e quantidade de dados transferidos. É exatamente isso que os criminosos digitais exploram para causar instabilidade ou indisponibilidade nas redes que usam o protocolo MQTT. Um tipo muito comum de ataque que inunda a rede, é o envio massivo de mensagens do tipo PUBLISH QoS 2. Como resultado, todos os recursos de comunicação são desperdiçados com operações de *handshake*.

## 2.4 *COMPLEX EVENT PROCESSING* (CEP)

O CEP é uma tecnologia para processamento de fluxo de dados em tempo real com base no modelo de processo assíncrono (LUCKHAM, 2008). Um sistema CEP recebe fluxos de dados de diferentes fontes, como sensores em uma casa inteligente ou carros em uma rodovia, para detectar padrões e gerar novos eventos. Esses eventos são enviados (como notificações) aos consumidores registrados (BAPTISTA et al., 2013).

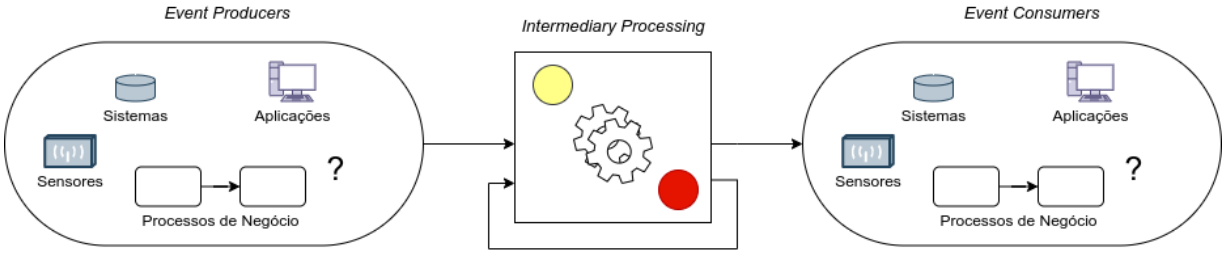
O CEP pode ser aplicado em vários cenários que envolvam tecnologia da informação, dentre eles: monitoramento e detecção maliciosa em redes de computadores; predição de preços de mercado; monitoramento de sistemas computacionais; automação de agentes inteligentes, entre outros.

O CEP provê suporte ao processamento de quantidades elevadas de eventos. Os sistemas de informação que implementam o CEP são capazes de reagir a um conjunto de eventos do nível mais baixo ao mais alto. Isso significa que o CEP é capaz de conectar vários eventos de baixo nível com severidade baixa a um ou vários eventos complexos com gravidade alta e até crítica. Pode-se dizer que o evento complexo é uma agregação de vários eventos simples (REPCEK, 2016).

### 2.4.1 Arquitetura de Processamento de Eventos

Segundo Etzion e Niblett (ETZION; NIBLETT; LUCKHAM, 2011), uma aplicação pode conter um ou mais componentes que geram os eventos. Em geral, existem várias arquiteturas para processamento de eventos, entretanto, a maioria das aplicações se baseia na arquitetura ilustrada na Figura 16.

Figura 16 – Principais Componentes no processamento de eventos




**Fonte:** O Adaptado de (ETZION; NIBLETT; LUCKHAM, 2011).

Os principais componentes são descritos a seguir:

- *Event Producers*: os produtores de eventos podem ser de tamanhos e formas variáveis: sensores que produzem eventos quando detectam certas ocorrências físicas;
- *Intermediary Processing*: o processamento intermediário geralmente ocorre entre os produtores e os consumidores de eventos. A distribuição de eventos é frequentemente realizada usando tecnologia de passagem de mensagem assíncrona;
- *Event Consumers*: componentes que recebem os eventos e normalmente atuam sobre eles. Novamente, suas funções variam. Eles podem, por exemplo, armazenar eventos para uso posterior, exibi-los em um interface do usuário ou executar alguma ação.

### 2.4.2 Eventos no CEP

Segundo (REPCEK, 2016), um evento pode ser definido como ocorrências dentro de um sistema, domínio ou registro de atividades. Recek (REPCEK, 2016) define três características principais de um evento:

- *Form*: informação sobre hora, localização ou outros dados descrevendo as circunstâncias da ocorrência;
- *Significance*: atividade e um formulário do evento contém informações que descrevem algum significado;
- *Relativity*: define o relacionamento entre os eventos como tempo (coleta de dados em tempo real), causalidade (causa ou efeito de um evento) e agregação (análise de vários eventos) (REPCEK, 2016), (ESPERTECH, 2019).

O CEP fornece técnicas para definir a relatividade apropriada entre os eventos, podendo ser aplicado a qualquer tipo de evento que ocorra em qualquer sistema computacional.

O CEP pode utilizar diversas técnicas para execução. Algumas delas são destacadas (LUCKHAM, 2008):

- *Event-pattern detection*: nesta técnica, uma coleção de eventos é avaliada para verificar se eles satisfazem algum padrão existente;
- *Detecting relationships*: esta técnica contém três relações principais entre os eventos. Tempo: tem relação com a ordenação dos eventos através de um relógio de acordo com

o sistema; Causalidade: é uma relação de dependência, ou seja, um determinado evento depende que um outro evento ocorra; Agregação: é um relacionamento onde um evento A consiste a um conjunto de informaçõe de um eventos B;

- *Event abstraction*: esta técnica trata do relacionamento entre um evento complexo e os outros eventos. Esta definição se aplica ao uso de abstração em um contexto de processamento de evento;
- *Event hierarchies*: é um modelo que representa as relações entre eventos que estão em níveis diferentes de abstração entre si;
- *Event driven*: o comportamento de um dispositivo, módulo de software ou outra entidade cuja execução é em resposta à chegada de eventos de fontes externas ou internas como um smartphone ou um sistema operacional.

### 2.4.3 Estrutura de Regras CEP

Os padrões são modelados como regras do CEP. Estas regras, quando injetadas no mecanismo CEP, são analisadas continuamente enquanto o fluxo de dados é obtido. Os principais mecanismos CEP suportam a definição de regras com base na linguagem de consulta estruturada (SQL) (ESPER, 2016),(CQL, 2016). Como exemplo de uma regra CEP, a Listagem 1 refere-se ao retorno do preço médio das ações da IBM nos últimos 30 segundos (REPCEK, 2016).

Código Fonte 1 – Exemplo de retorno do preço médio das ações da IBM

```
1        select avg(price)
         from
3            StockTick.win:time(30 sec)
         where
5            symbol='IBM'
```

**Fonte:** Adaptado de (REPCEK, 2016).

Pelo exemplo da listagem 1, é possível notar a semelhança com a sintaxe utilizada nas consultas SQL.

As regras são executadas assim que os eventos são recebidos. Quando uma ocorrência é detectada, uma notificação chega aos consumidores. Assim, através da análise em tempo real,

o mecanismo CEP combina os eventos (simples) que foram enviados pelas fontes e gera um evento complexo, que representa a interpretação dos eventos simples analisados.

O CEP pode combinar várias fontes de dados IoT em tempo real, realizar baixo acoplamento, facilitando assim a extensibilidade a outros protocolos de camada de aplicação e atualizações de código-fonte. Além disso, com o CEP, é possível processar um grande volume de mensagens com menor consumo de memória e processamento e obter um excelente tempo de resposta para detecção (TERROSO-SAENZ et al., 2019; ROLDÁN et al., 2020).

## 2.5 CONSIDERAÇÕES FINAIS

Este capítulo destacou fundamentos importantes relacionados à temática desta tese.

A Seção 2.1 apresentou as abordagens utilizadas nos sistemas de detecção de intrusão. A abordagem escolhida para esta tese foi a detecção por anomalia. Com ela é possível identificar ataques que ainda não são conhecidos.

A Seção 2.2 apresentou a arquitetura de camadas IoT. Nesta tese foi assumida uma arquitetura de quatro camadas.

A Seção 2.3 descreveu os protocolos e AMQP, CoAP e MQTT.

A Seção 2.4 introduziu conceitos básicos de CEP (Complex Event Processing). O CEP foi escolhido como mecanismo de detecção de intrusão por oferecer uma poderosa linguagem EPL (*Event Processing Language*), que o torna capaz de correlacionar eventos e lidar com milhares de eventos. Em seguida, foi apresentado um exemplo de uma regra CEP que retorna o preço médio das ações da IBM nos últimos 30 segundos.

# 3 TRABALHOS RELACIONADOS

Neste capítulo, são apresentados trabalhos relacionados com esta tese. Mais especificamente, o capítulo mostra o resultado da revisão de literatura relacionada a sistemas de detecção e prevenção de intrusão (IDPS) para ambientes IoT. Os trabalhos descritos são comparados com a abordagem adotada nesta tese.

Para facilitar a análise, os trabalhos relacionados foram classificados em cinco categorias: 1) IDPS com técnicas de CEP; 2) IDPS com técnicas de inteligência artificial; 3) IDPS com técnicas Fuzzy; 4) IDPS com Snort; e 5) IDPS com outras técnicas. Cada trabalho será identificado pela sigla [IDPS+Identificador].

Desta forma, este capítulo está estruturado da seguinte forma:

- A Seção 3.1 - Extração e Seleção dos Trabalhos Relacionados: apresenta como foi realizada a extração e seleção dos trabalhos;
- A Seção 3.2 - IDPS com Técnicas de CEP: apresenta os trabalhos que utilizam regras CEP para detectar intrusão;
- A Seção 3.3 - IDPS com Técnicas de Inteligência Artificial: apresenta os trabalhos que utilizam técnicas de inteligência artificial para detectar intrusões;
- A Seção 3.4 - IDPS com técnicas de Fuzzy: apresenta os trabalhos que utilizam técnicas fuzzy para detectar intrusões;
- A Seção 3.5 - IDPS que utilizam o Snort para detecção de intrusões;
- A Seção 3.6 - IDPS que utilizam outras técnicas de detecção de intrusões;
- A seção 3.7 - apresenta uma análise comparativa;
- A Seção 3.8 - apresenta as considerações finais deste capítulo.

## 3.1 PROTOCOLO DE BUSCA DOS TRABALHOS RELACIONADOS

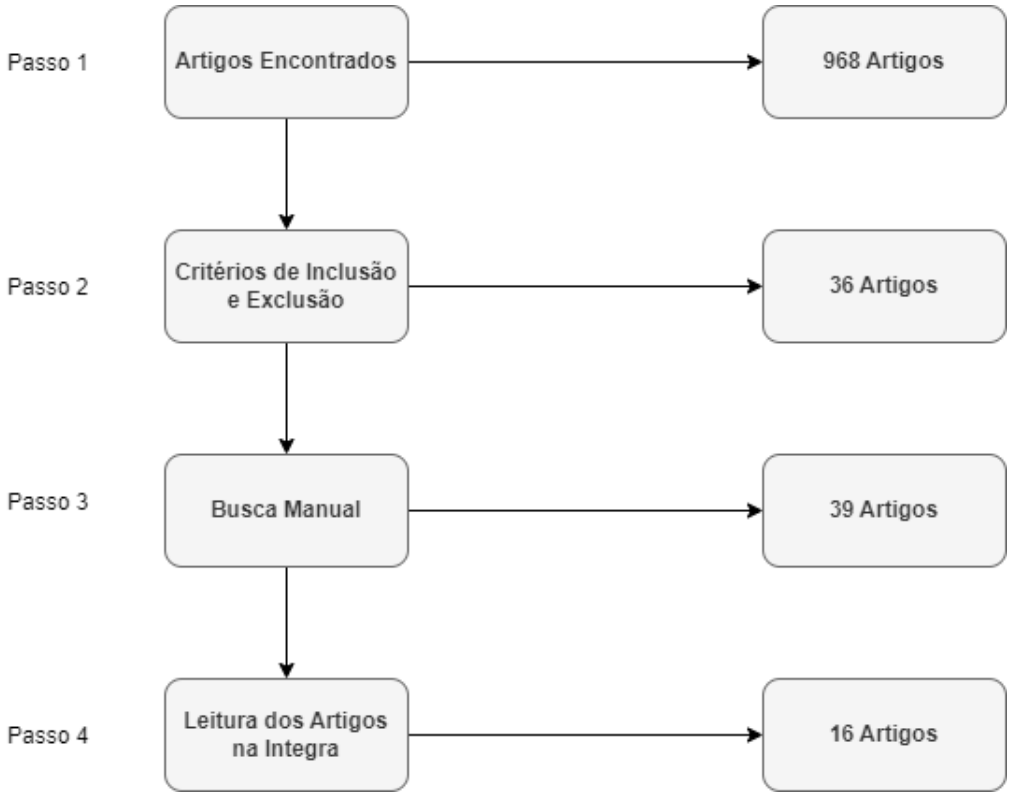
O Mapeamento Sistemático é um método que fornece uma visão geral de um campo de pesquisa, possibilitando identificar, quantificar e analisar tipos de pesquisa e resultados disponíveis (KITCHENHAM et al., 2010). Este processo de revisão da literatura é importante para

o desenvolvimento de uma pesquisa científica. O mapeamento sistemático foi realizado para capturar o estado da arte em sistemas de detecção e prevenção de intrusão para ambientes IoT. O mapeamento sistemático da literatura foi inspirado no trabalho de Kitchenham (KITCHENHAM et al., 2010). Nesse sentido, o protocolo utilizado no atual estudo de mapeamento é apresentado a seguir.

### 3.1.1 Busca automática e Manual (*Ad-hoc*)

Em 2018, foram realizadas buscas no IEEE Explorer [1], ACM Digital Library [2], e Science Direct [3], utilizando como palavras-chave a expressão: *"Intrusion Detection System"AND "Internet of Things"*. A seleção dos artigos seguiu os passos definidos na Figura 17.

Figura 17 – Busca automática e Manual




**Fonte:** Autor.

1. A busca retornou 539 trabalhos no IEEE Xplorer, 305 no Science Direct e 124 trabalhos no ACM Digital Library.

[1] http://ieeexplore.ieee.org/
[2] https://dl.acm.org/
[3] https://www.sciencedirect.com

2. Foram aplicados os seguintes critérios de inclusão e exclusão: limitamos o escopo temporal em cinco anos; foram excluídos editoriais, resumos, tutoriais, *keynote*, relatórios de workshop, opiniões, sumários de conferências, teses, dissertações, relatórios técnicos, livros, estudos secundários ou terciários, estudos primários que não apresentam de forma explícita a aplicação de IDS na internet das coisas, duplicidade de publicação (Indexados) e artigos não escritos em língua inglesa. Como resultado, obtiveram-se 25 trabalhos no IEEE Xplorer, 5 no Science Direct e 6 trabalhos no ACM Digital Library.

3. A fim de buscar possíveis trabalhos relevantes não encontrados, foi realizada uma busca manual no google scholar ainda entre os anos de 2019 e 2022 nos mesmos engenhos de busca supracitados. A busca manual gerou a inclusão de três novos trabalhos.

4. Após a leitura completa dos artigos, restaram 11 trabalhos no IEEE Xplorer, 2 no Science Direct e 3 trabalhos no ACM Digital Library. Estes trabalhos serão apresentados nas próximas seções.

## 3.2 SISTEMAS DE DETECÇÃO DE INTRUSÃO BASEADOS EM TÉCNICAS CEP

### 3.2.1 Design of Complex Event-Processing IDS in Internet of Things [IDPS01]

Jun e Chi (CHEN; CHEN, 2014) desenvolveram uma arquitetura IDPS baseada no processamento de eventos complexos (IDPS baseado em CEP). Os autores propõem a detecção de *Land Attacks* (HADDADI; BEGHDAD, 2018) com base em anomalias do protocolo TCP. Eles consideram que existe uma anomalia quando o ataque é baseado no protocolo TCP, e esta anomalia consiste no envio contínuo pelo invasor de um pacote TCP falsificado com os mesmos endereços IP de origem e destino, o que resulta em um *host* vítima, enviando respostas para si mesmo em um *loop*. Essa arquitetura é composta por um filtro de dados, monitoramento de eventos, análise de eventos e resposta de segurança.

Com base na revisão deste trabalho, alguns pontos são destacados: embora esse IDPS detecte o *Land Attack* com uma boa margem de confiabilidade, ele apenas detecta anomalias encontradas nos protocolos da camada de transporte TCP. Se ocorrerem ataques que explorem vulnerabilidades na camada de aplicação, o IDPS de Jun e Chi (CHEN; CHEN, 2014) não seria capaz de detectar.

### 3.2.2 Real-time DDoS attack detection based on Complex Event Processing for IoT [IDPS02]

O trabalho proposto por Cardoso et. al (CARDOSO et al., 2018) apresenta um IDS baseado na detecção de anomalias usando a técnica CEP, o CEPIDS. Este trabalho estende o artigo de Jun e Chi (CHEN; CHEN, 2014) ao incorporar a identificação de ataques SYN Flood, UDP Flood, ICMP Flood e Port Scan (HADDADI; BEGHDAD, 2018). Os ataques são detectados usando regras EPL (*Event Processing Language*) que analisam o tráfego TCP e UDP e comparam os resultados com o IDS Snort [4] e Bro [5]. Além disso, eles analisam o consumo de CPU e memória RAM.

Após revisar o trabalho de Cardoso et. al (CARDOSO et al., 2018), foi identificado que os autores não apresentam uma arquitetura que possa explicar os componentes do IDS e entender como é realizado o processo de detecção de anomalia. Como nos trabalhos de Jun e Chi (CHEN; CHEN, 2014), o CEPIDS não detecta anomalias que explorem vulnerabilidades dos protocolos da camada de aplicação da IoT.

### 3.2.3 Towards a multilayer strategy against attacks on IoT environments [IDPS03]

O trabalho proposto por Junior et. al (JUNIOR et al., 2019) propõe um IDPS baseado em regras CEP. Entretanto, o foco do trabalho não é a precisão na detecção de ataques, e sim a usabilidade do IDS. Os autores comparam a usabilidade do IDPS proposto com o Suricata IDS (SURICATA, 2020).

Como o trabalho de Junior et. al (JUNIOR et al., 2019) não tem como foco avaliar a precisão de detecção, não se conhece a precisão do IDPS desenvolvido. Outros pontos que precisam ser destacados: as regras do IDPS propostas são construídas com técnicas para detecção de anomalia, porém os autores o comparam com o do Suricata IDS, que detecta intrusão por assinatura, ou seja, são métodos de detecção diferentes, conforme apresentado no Capítulo 2 desta tese.

[4] https://snort.com
[5] https://github.com/topics/bro-ids

### 3.2.4 Integrating complex event processing and machine learning: An intelligent architecture for detecting IoT security attacks [IDPS04]

Roldan et. al (ROLDÁN et al., 2020) propuseram e implementaram um sistema inteligente que combina CEP e aprendizado de máquina capaz de gerenciar padrões para detectar ataques de segurança de IoT.

Este trabalho desenvolveu uma arquitetura que foi aplicada a um protótipo de rede IoT construído em um hospital com o objetivo de detectar ataques (varreduras de portas TCP, UDP, Xmas e DoS) feitos por um dispositivo malicioso. Os resultados mostram que a detecção dos ataques foi realizada com êxito.

Apesar de os dados serem extraídos a partir de uma rede usando MQTT, a solução proposta não detecta ataques direcionados ao MQTT ou a outro protocolo da camada de aplicação. Ou seja, o experimento fica restrito ao protocolo TCP e UDP e entende-se até o momento que a solução não é projetada para lidar com ataques que explorem as vulnerabilidades dos protocolos da camada de aplicação.

## 3.3 IDPS COM TÉCNICAS DE IA (INTELIGÊNCIA ARTIFICIAL)

### 3.3.1 Negative Selection and Neural Network based Algorithm for Intrusion Detection in IoT [IDS05]

Pamukov et. al (PAMUKOV; POULKOV; SHTEREV, 2018) apresentam um algoritmo baseado em Rede Neural de Seleção Negativa (NSNN). Trata-se de um algoritmo de classificação projetado para sistemas de detecção de intrusão da Internet das Coisas. A adição da camada de Seleção Negativa permite treinar uma Rede Neural apenas com base no comportamento normal da rede, sem a necessidade de dados de ataque.

O trabalho de Pamukov et. al (PAMUKOV; POULKOV; SHTEREV, 2018) tem como objetivo desenvolver um algoritmo de IA para o IDPS baseado em rede IoT, que pode ser aplicável a uma ampla variedade de casos de uso da IoT. Entretanto, o artigo não consegue detectar intrusões que explorem vulnerabilidades em protocolos da camada de aplicação. Além disso, a detecção funciona apenas para ataques de negação de serviço.

### 3.3.2 ARTEMIS: An Intrusion Detection System for MQTT Attacks in Internet of Things [IDPS06]

ARTEMIS é um sistema de detecção de intrusão baseado em anomalia para a IoT, mais precisamente para o protocolo MQTT (CIKLABAKKAL et al., 2019). Este IDS processa dados de dispositivos IoT usando aprendizagem de máquina (*Autoencoder, Single-Objective Generative Adversarial Active Learning (SO GAAL), Random Forest, Isolation forest*) para detectar desvios do comportamento normal do sistema e gera alertas.

Neste trabalho foi implementado um protótipo do sistema usando dispositivos IoT inscritos em tópicos em um broker MQTT. Neste trabalho é realizado um experimento para avaliar o IDPS sob ataques de negação de serviço (DoS) (HADDADI; BEGHDAD, 2018) relacionados ao MQTT.

Após análise do ARTEMIS (CIKLABAKKAL et al., 2019), foi possível identificar que esse sistema de detecção de intrusão apenas tem foco no MQTT, não sendo possível dar suporte a outros protocolos da camada de aplicação comumente utilizados.

## 3.4 IDPS COM TÉCNICAS FUZZY

### 3.4.1 Secure-MQTT: an efficient fuzzy logic-based approach to detect DoS attack in MQTT protocol for internet of things [IDPS07]

Neste trabalho, proposto por (HARIPRIYA; KULOTHUNGAN, 2019), é desenvolvido um novo IDPS para aplicações baseadas em MQTT. O sistema proposto identifica as anomalias da rede com a ajuda de variáveis difusas. O grau de comportamento anômalo do nó é determinado a partir dessa imprecisão. O trabalho apresenta sua arquitetura e resultados da precisão na detecção dos ataques.

Este IDPS usa a seleção de recursos de rede baseada em correlação, que seleciona apenas os recursos como largura de banda, pacotes enviados, consumo de memória e processamento, por exemplo. Um mecanismo de inferências baseado em lógica difusa (fuzzy) determina a presença de um dispositivo malicioso. A interpolação de regras difusas permite que o IDPS consuma menos memória e processamento. Não há necessidade de armazenar todas as regras na sua base.

Após revisar o trabalho de Haripriya (HARIPRIYA; KULOTHUNGAN, 2019), foi identificado

que este trabalho apenas dá suporte ao protocolo MQTT. Não é possível identificar se esse IDS pode ser extensível a outros protocolos da camada de aplicação da IoT.

## 3.5 IDPS USANDO O SNORT

### 3.5.1 RPiDS: Raspberry Pi IDS - A Fruitful Intrusion Detection System for IoT [IDPS08]

O trabalho proposto por Sforzin et. al (SFORZIN et al., 2017) apresenta uma arquitetura de detecção de intrusão visando à instalação do tradicional IDS Snort em um dispositivo Raspberry Pi modelo 3B. O estudo avalia o desempenho do Raspberry Pi executando o Snort. Nos experimentos, o Snort, segundo os autores do trabalho, obteve bom desempenho nas detecções, baixo consumo computacional e pode servir como IDS para a IoT.

Sforzin et. al (SFORZIN et al., 2017) realizaram testes extensivos envolvendo o Raspberry Pi atuando como um nó na rede para analisar diferentes tipos de tráfego. Além disso, diferentes configurações do Snort foram utilizadas como mecanismo de detecção e número de regras carregadas. Tais configurações podem influenciar o consumo de CPU e memória RAM do Snort. Os resultados demonstram que o Raspberry Pi é capaz de hospedar o Snort, tornando o RPiDS uma solução viável para detectar anomalias do tráfego TCP e UDP. Porém, o autor não analisou a possibilidade de esse trabalho poder ser extensível para a camada de aplicação IoT.

## 3.6 IDPS USANDO OUTRAS TÉCNICAS

### 3.6.1 Kalis - A System for Knowledge-driven Adaptable Intrusion Detection for the Internet of Things [IDPS09]

O trabalho proposto por (MIDI et al., 2017) apresenta um sistema de detecção de intrusão adaptável. O Kalis é uma abordagem abrangente para detecção de intrusões em ambientes IoT, que não visa a protocolos ou aplicativos individuais e adapta a estratégia de detecção aos recursos específicos da rede.

Com método próprio, o Kalis apresenta uma técnica híbrida (anomalia e assinatura) para detectar as intrusões. O Kalis é constituído a partir de uma base de conhecimento para detecção divididos em três componentes: 1) sistema de comunicação: interage com a comunicação

externa; 2) armazenamento de dados: escuta eventos do sistema de comunicação em pacotes recém-capturados, gerencia o histórico de tráfego recente para acessar os módulos e registra todo o tráfego em disco ou memória, se exigido pelo usuário; 3) base de conhecimento: armazena todas as informações disponíveis sobre os recursos das entidades e redes monitoradas em um local único e centralizado e disponibiliza essas informações para todas as partes que o solicitam, como os módulos de detecção e o gerenciador de módulos.

A avaliação do Kalis demonstrou que ele é eficaz e eficiente na detecção de ataques em redes IoT através. Entretanto, os experimentos apresentam uso de protocolos mais tradicionais como ICMP, TCP e UDP.

### 3.6.2 Denial-of-Service detection in 6LoWPAN based Internet of Things [IDPS10]

O trabalho proposto por Kasinathan et. al (KASINATHAN et al., 2013) apresenta um IDS para detectar ataques de negação de serviço em redes 6LowPAN. Os autores estudam as vulnerabilidades em WSNs (*Wireless Sensor Network*) como foco principal. Além disso, eles propõem a arquitetura final do IDS baseada em 6LowPAN.

Segundo os autores, este IDPS pode detectar ataques de negação de serviço sendo escalável e aplicável à maioria dos cenários da IoT do mundo real. As contribuições mais relevantes do trabalho de Kasinathan et. al (KASINATHAN et al., 2013) são o gerenciador de proteção contra DoS e o próprio IDS, que são integrados ao gerente de segurança de rede.

Analisando o artigo, não foi possível identificar resultados voltados para a precisão na detecção dos ataques, e sim, apenas informações sobre o tráfego de rede.

### 3.6.3 An Intrusion Detection System for Denial of Service Attack Detection in Internet of Things [IDPS11]

O trabalho proposto por Abdelouahab (ABDELOUAHAB, 2017) apresenta o IDS-IoT, projetado para detectar ataques baseados em assinatura. O IDS-IoT visa detectar alguns tipos de ataques de negação de serviço: inundação de SYN, inundação de ICMP, ataque de SMURF e inundação de UDP (HADDADI; BEGHDAD, 2018).

A arquitetura deste IDPS apresenta os seguintes módulos:

- Sensor - responsável por capturar informações da rede;

- IP_Capture - este módulo tem a função de capturar os endereços IP dos dispositivos que estão ativos naquele momento. As informações do endereço IP ativo são usadas pelo módulo Packet_Analyzer para detectar ataques;

- Packet_Analyzer - este módulo seleciona os registros DB_Packs para analisar as características dos pacotes. Ele verifica se os dados capturados pertencem aos dispositivos da mesma rede. Se os pacotes capturados vierem de dispositivos não autorizados, terá início a troca de informações com o Attack_Detection. Para determinar qual tipo de ataque está ocorrendo e o Attack_Detection faz uma análise de suas características. O resultado da análise é registrado na tabela do banco de dados DB_Packs, que possui a funcionalidade de armazenar o histórico dos ataques que o dispositivo sofreu;

- Gereration Rules - o componente usa o histórico dos ataques armazenados na tabela com o histórico de ataques para gerar regras de bloqueio e definir barreiras contra pacotes atacantes do emissor. Para realizar o bloqueio, é utilizado o Iptables, firewall do sistema operacional Linux.

Apesar de este trabalho apresentar resultados relacionados à precisão na detecção e consumo de recursos computacionais, as detecções ocorrem apenas com suporte para TCP e UDP, expondo, assim, uma lacuna, para detecção de ataques em protocolos de camadas superiores da IoT.

### 3.6.4 REATO: REActing TO denial of service attacks in the Internet of Things [IDPS12]

Sicari et. al (SICARI et al., 2018) apresentam um IDS denominado REATO para detectar ataques de DoS em um middleware de IoT. Um protótipo real foi construído para validar o método proposto. O trabalho teve como motivação a necessidade de encontrar uma solução capaz de defender ataques de negação de serviço em objetos inteligentes dentro de ambientes IoT. Este ataque de negação se baseia em uma carga excessiva de CPU levando a uma instabilidade de conexão por um longo período. O REATO foi instalado em um Raspberry Pi para analisar os dados em tempo real e realizar as detecções e bloqueios.

Este trabalho consegue analisar dados específicos de um middleware, e aponta inconsistências como número excessivos de pacotes enviados para o broker MQTT. Entretanto, a detecção

se baseia em pacotes TCP e informações da porta de comunicação, não tendo relação com detecção de ataques que explorem as vulnerabilidades do MQTT.

### 3.6.5 SVELTE: Real-time intrusion detection in the Internet of Things [IDPS13]

O trabalho proposto por (RAZA; WALLGREN; VOIGT, 2014) apresenta o IDS híbrido SVELTE. O objetivo desse IDS é oferecer uma compensação satisfatória entre o custo de armazenamento do método baseado em assinatura e o custo computacional do método baseado em anomalia.

Ele define uma arquitetura e algoritmos de detecção de intrusões para o protocolo 6LowPAN. Sua implementação serve para o contexto de ambientes IoT que utilizem os protocolos RPL, 6LoWPAN. Esse trabalho pode ser extensível para detectar mais ataques. Portanto, outros trabalhos podem surgir a partir do SVELTE com novas técnicas de detecção de intrusão em camadas mais baixas.

Esse IDS apresenta resultados satisfatórios na precisão de detecções no ataque Sinkhole (PATIL; CHEN, 2017). Com este IDS, é possível monitorar pacotes IP e informações de roteamento vindas da camada de rede. Porém, não é possível detectar anomalias em pacotes MQTT e CoAP. Outra limitação do SVELTE é o alto consumo de processamento e memória, conforme (RAZA; WALLGREN; VOIGT, 2014).

### 3.6.6 InDReS: An Intrusion Detection and Response System for Internet of Things with 6LoWPAN [IDPS14]

O trabalho proposto por Sunrendar et. al (SURENDAR; UMAMAKESWARI, 2016) apresenta uma extensão significativa às deficiências do SVELTE (especialmente alto consumo de recursos de processamento e memória). Esse IDS se baseia no modelo de especificação baseado em restrições para detectar ataques sinkhole. Quando o ataque é detectado, os nós que podem ser os atacantes são isolados e a rede é reconstruída isolando esses nós. InDreS apresenta essa nova técnica de detecção que melhora significativamente o desempenho computacional em relação ao SVELTE (RAZA; WALLGREN; VOIGT, 2014).

Alguns pontos podem ser observados. Esse trabalho não apresenta a quais protocolos ele pode ser extensível na camada de aplicação da IoT e métricas de precisão de detecção não são apresentadas.

### 3.6.7 Intrusion Detection in the RPL-connected 6LoWPAN Networks [IDPS15]

O trabalho proposto por (SHREENIVAS; RAZA; VOIGT, 2017) apresenta um IDS que identifica intrusões destinadas a interromper a comunicação o Protocolo de Roteamento para Redes de Baixa Potência e Perdas (RPL). A fim de melhorar a segurança nas redes 6LoWPAN, os autores desenvolvem um sistema de detecção de intrusões para a Internet das Coisas, com um módulo de detecção que usa a métrica ETX (transmissões esperadas). No RPL, o ETX é uma métrica de confiabilidade do link e o monitoramento do valor do ETX pode impedir que um invasor envolva ativamente os nós 6LoWPAN em atividades maliciosas.

Por fim, esse trabalho apresenta um IDS com bons resultados na precisão de detecção de ataques nas métricas EXT e bons consumos de memória RAM e processamento. Como limitação, esse IDS fica restrito a detecções apenas na camada de rede. Portanto, não é capaz de detectar ataques que explorem ataques na camada de aplicação.

### 3.6.8 A Hybrid Intrusion Detection Architecture for Internet of Things [IDPS16]

O trabalho proposto por (SHEIKHAN; BOSTANI, 2016) apresenta um novo modelo de arquitetura para IDS híbrido. Este modelo é baseado na abordagem MapReduce com o objetivo de detecção distribuída. Para fornecer detecção multifacetada (do lado da Internet e das WSNs), o modelo proposto consiste em agentes de detecção de intrusões baseados em anomalias e uso indevido que utilizam o modelo para detecção de intrusão.

Nesse trabalho, os autores conseguem detectar ataques *selective-forwarding attacks* (ZARPELÃO et al., 2017), apresentando resultados satisfatórios de desempenho e precisão de detecção de ataque por meio de simulações. Este IDS tem como foco a detecção de ataques baseados no protocolo 6LoWPAN, não apresentando a possibilidade de suporte para outros protocolos.

## 3.7 ANÁLISE COMPARATIVA

Nesta seção, é apresentada uma análise comparativa entre os trabalhos relacionados. Como forma de analisar os mesmos, foram estabelecidos critérios que são apresentados a seguir.

### 3.7.1 Critérios de Avaliação dos Trabalhos

- **Critério 1 (C1) - Implementação disponível:** avalia se o trabalho proposto apresenta uma implementação prática do IDPS.
- **Critério 2 (C2) - Avaliação da Precisão:** avalia se o trabalho realizou a avaliação de precisão da detecção de intrusão. Segundo a NIST 800-94 (SCARFONE; MELL, 2007), avaliar a precisão é um dos requisitos principais no desenvolvimento de um IDPS.
- **Critério 3 (C3) - Suporte a Protocolos da Camada de Aplicação IoT:** avalia se o trabalho proposto tem suporte a detecção de intrusão em vulnerabilidades exploradas nos protocolos da camada de aplicação IoT.
- **Critério 4 (C4) - Suporte a Protocolos de outras camadas:** avalia se o IDPS tem suporte para detectar intrusão em protocolos de outras camadas (AMQP, CoAP e DDS por exemplo).

### 3.7.2 Comparativo entre os Sistemas de Detecção de Intrusão

A Tabela 3 apresenta um comparativo entre os IDS avaliados neste capítulo. A primeira coluna exibe o nome do trabalho relacionado e as demais colunas exibem os critérios citados na seção anterior. A marcação "+" indica que o trabalho relacionado atende ao critério estabelecido, enquanto a marcação "-" indica que não foi atendido o critério.

Após a análise dos trabalhos, serão agora destacados os principais pontos de avanço desta tese em relação ao estado da arte.

Apenas os trabalhos IDS05 e IDS06 conseguem detectar ataques que explorem alguma vulnerabilidade do protocolo MQTT. Entretanto, o trabalho limita-se apenas ao MQTT, e não são extensíveis a outros protocolos, não sendo possível detectar ataques direcionados a outros tipos de aplicações que usem outros protocolos.

A partir desses dados apresentados, é possível identificar uma lacuna relevante na literatura referente ao desenvolvimento de mecanismos de linha de defesa para a camada de aplicação. A maioria dos IDS apresentados neste capítulo realizaram detecção de ataques baseados em vulnerabilidades exploradas nos protocolos TCP, UDP, IP e outros protocolos mais tradicionais.

Como apontado no trabalho IDS07, já existem IDS's robustos o suficiente para detecção de ataques baseados em vulnerabilidades nos protocolos TCP e UDP e que já são amplamente

Tabela 3 – Comparação entre os trabalhos relacionados

| IDPS | C1 Implementação disponível | C2 Avaliação da Precisão | C3 Suporte a Protocolos da Camada de Aplicação IoT | | | C4 Suporte a Protocolos de outras camadas |
|---|---|---|---|---|---|---|
| | | | MQTT | CoAP | AMQP | |
| IDPS01 | + | - | - | - | - | + |
| IDPS02 | + | + | - | - | - | + |
| IDPS03 | - | - | - | - | - | + |
| IDPS04 | + | + | + | - | - | + |
| IDPS05 | + | + | - | - | - | + |
| IDPS06 | - | + | + | - | - | - |
| IDPS07 | - | + | + | - | - | - |
| IDPS08 | + | + | - | - | - | + |
| IDPS09 | + | + | - | - | - | + |
| IDPS10 | + | - | - | - | - | + |
| IDPS11 | - | + | - | - | - | + |
| IDPS12 | + | + | - | - | - | + |
| IDPS13 | - | + | - | - | - | + |
| IDPS14 | + | - | - | - | - | + |
| IDPS15 | + | + | - | - | - | + |
| IDPS16 | - | + | - | - | - | + |
| **Beholder** | + | + | + | + | + | + |

**Fonte:** O Autor.

usados na indústria. Acredita-se que esta tese possa representar um avanço importante para melhorar a segurança contra ataques que explorem os protocolos da camada de aplicação IoT. O desenvolvimento do Beholder abrange os seguintes avanços:

- **Análise da carga de dados do AMQP, CoAP e MQTT:** através da captura dos pacotes TCP e UDP, o Beholder consegue detectar qual é o protocolo da camada de aplicação através da leitura dos bits que são correspondentes aos campos de cada cabeçalho. Após a leitura, os eventos são criados conforme categorização por tipo de protocolo. Esta leitura dos bits pode ser realizada em ambiente IoT que usem os protocolos AMQP, CoAP e MQTT. Ressalta-se que nenhum dos trabalhos apresentados neste capítulo detectou ataques em aplicações baseadas em AMQP e CoAP;

- **Uso do CEP como mecanismo nos protocolos da camada de aplicação**: o mecanismo CEP na arquitetura do Beholder consegue categorizar os eventos e detectar com boa margem de detecção nos ataques relacionados aos protocolos MQTT, CoAP e AMQP;

- **Extensível a outros protocolos da camada de aplicação**: O baixo acoplamento do CEP facilita que o Beholder possa dar suporte a outros protocolos da camada de aplicação como o XMPP, que é bastante utilizado na indústria e cidades inteligentes, e o DDS que tem seu maior uso em ambientes como mercado financeiro, controle de tráfego aéreo e redes elétricas inteligentes;
- **Extensível a protocolos de outras camadas**: o Beholder pode ser estendido também a protocolos de outras camadas, por exemplo o protocolo IP (*Internet Protocol*), Ethernet, ARP (*Address Resolution Protocol*), entre outros.

## 3.8 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados trabalhos relacionados que abordam sistemas de detecção de intrusão para ambientes IoT entre os anos de 2016 a 2022, descrevendo as principais contribuições de cada trabalho. Para atender à necessidade da pesquisa, uma busca automática e manual baseada em revisão sistemática da literatura foi realizada.

Este capítulo também apresentou uma análise comparativa dos sistemas de detecção de intrusão encontrados na literatura. Por fim, foi realizada uma consolidação de características mais relevantes para avaliar sistemas de detecção de intrusão, os pontos e suas limitações que apoiaram o desenvolvimento desta tese.

# 4 BEHOLDER: UM SISTEMA PARA DETECÇÃO E PREVENÇÃO CONTRA INTRUSÃO BASEADO EM PROCESSAMENTO DE EVENTOS COMPLEXOS EM AMBIENTES IOT

Este capítulo apresenta a principal contribuição desta tese, o Beholder, um sistema de detecção de intrusão e prevenção de ataques aos protocolos da camada de aplicação da IoT.

O capítulo está organizado da seguinte forma: a Seção 4.1 apresenta a visão geral das principais características do Beholder; a Seção 4.2 descreve a metodologia e planejamento para construção do Beholder; a Seção 4.3 demonstra a análise do domínio de aplicação do Beholder; a Seção 4.4 detalha requisitos para desenvolvimento do Beholder; a Seção 4.5 apresenta o método "4+1"para construção de arquitetura de software; A Seção 4.6 especifica a visão lógica da arquitetura; a Seção 4.7 ilustra a visão de processo da arquitetura; A Seção 4.8 apresenta a visão física da arquitetura; A Seção 4.9 detalha a visão de desenvolvimento da arquitetura; A Seção 4.10 contém discussão e considerações finais sobre o capítulo.

## 4.1 VISÃO GERAL DO BEHOLDER

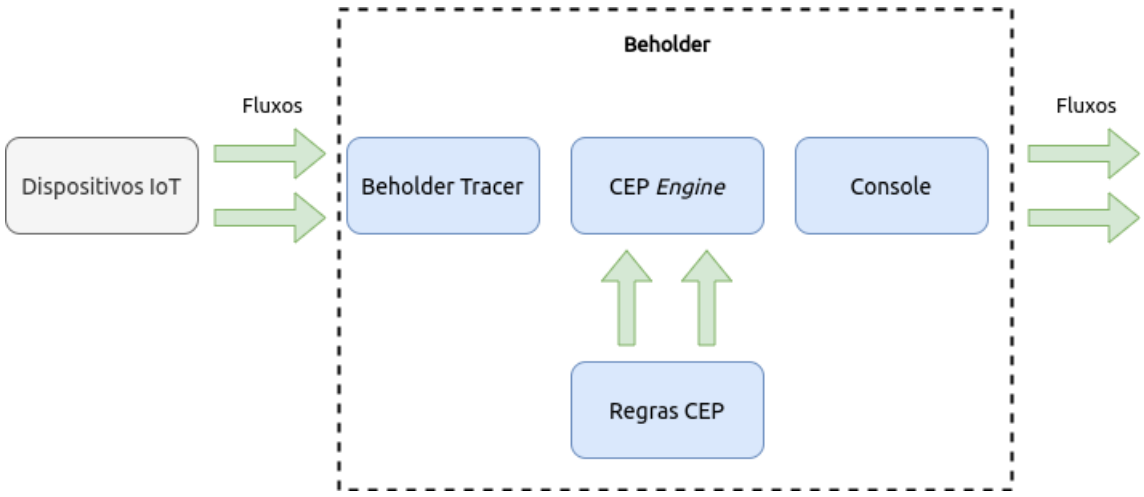
Com o objetivo de propor a detecção de ataques que explorem vulnerabilidades existentes nos protocolos da camada de aplicação, o Beholder fornece uma arquitetura com base em políticas (regras) para o processamento de eventos complexos (REPCEK, 2016). Esses eventos são criados a partir de fluxos de dados que são produzidos por vários dispositivos IoT, como câmeras Wi-fi, sensores de movimento, temperatura e umidade, além de smart tv's e outros objetos inteligentes.

A Figura 18 ilustra a visão geral do Beholder. Os dispositivos IoT produzem e enviam os fluxos para Beholder, que utiliza um gateway IoT para interconectar todos os dispositivos IoT por conexão Wi-fi ou Bluetooth.

Após a chegada dos fluxos de dados, o primeiro componente, o Beholder Tracer, analisa cada *payload* transmitido. Ao identificar os pacotes AMQP, CoAP e MQTT (que são os protocolos suportados atualmente), o Beholder Tracer os envia para a CEP *engine*. A CEP *engine* se encarregará de criar os eventos e executar as regras de detecção dos ataques e gerar os alertas. Por fim, o console mostra os registros do tráfego da rede, e os alertas.

O Beholder utiliza o CEP por consumir menos memória e detectar intrusões. As principais características dos sistemas baseados em CEP são bom desempenho na detecção de anormali-

Figura 18 – Visão Geral do Beholder



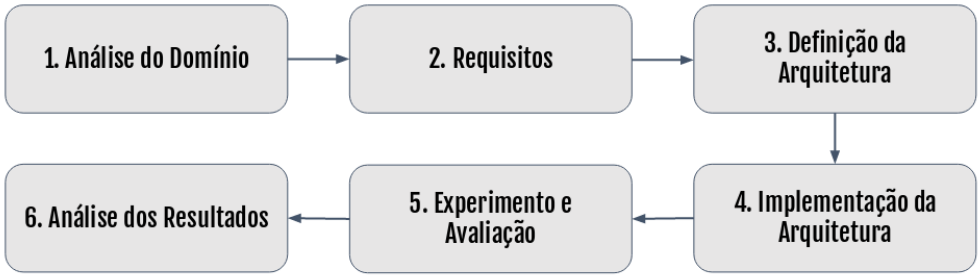

**Fonte:** O Autor.

dades utilizando um mecanismo de processamento de eventos (CHEN; CHEN, 2014). Além disso, o CEP tem característica de baixo acomplamento, o que facilita a extensão da solução para suportar outros protocolos da camada de aplicação.

## 4.2 METODOLOGIA E PLANEJAMENTO

A metodologia e planejamento para construção do Beholder se baseia no modelo proposto por Parson (PARSONS et al., 1999) e Sommerville (SOMMERVILLE; ARAKAKI; MELNIKOFF, 2008). Ela é composta por 6 (seis) atividades, conforme ilustrado na Figura 19.

Figura 19 – Planejamento das atividades




**Fonte:** O Autor.

O objetivo de cada uma dessas atividades são brevemente explanados a seguir:

1. Análise do Domínio: tem por objetivo definir em qual escopo da IoT esta solução será aplicada. Nesta atividade, foram levadas em conta as necessidades apresentadas em ambientes IoT. Esses ambientes utilizam dispositivos para controlar, monitorar e

gerenciar equipamentos inteligentes. Eles podem conter sensores, atuadores e outros objetos inteligentes. Além disso, por parte destes, possuem recursos computacionais limitados.

2. Requisitos: têm como objetivo especificar os requisitos funcionais e não funcionais para a construção do Beholder. De forma resumida, o Beholder detecta ataques aos protocolos da camada de aplicação em sistema IoT, com uma boa precisão e baixo consumo de memória RAM e CPU.

3. Definição da Arquitetura: tem como finalidade projetar a arquitetura da solução em si. Nesta atividade, o Beholder é desenvolvido observando perspectivas arquiteturais diferentes, que são chamadas de visões.

4. Implementação da Arquitetura: tem por objetivo implementar a arquitetura proposta na atividade anterior. Nesta atividade, são realizadas as etapas de desenvolvimento da solução proposta.

5. Experimento e Avaliação: são realizadas avaliações para identificar o nível de precisão de detecção do Beholder e desempenho nos consumos de memória RAM e CPU.

6. Análise dos Resultados: tem como finalidade analisar e apresentar os resultados obtidos.

## 4.3 ANÁLISE DO DOMÍNIO IOT

Segundo Pekar et. al. (PEKAR et al., 2020), a IoT é classificada em oito domínios, que são detalhados a seguir:

- **Edifícios e Moradias inteligentes:** visa aumentar a eficiência energética dos edifícios e melhorar a nossa qualidade de vida. Certamente houve uma demanda significativa por automação neste campo durante os últimos anos e, como resultado, muitas soluções foram propostas.

- **Saúde Inteligente:** visa se concentrar principalmente na capacidade de rastrear suas atividades físicas, enquanto o objetivo social é aumentar a qualidade do tratamento médico, reduzindo os custos de saúde. Esses benefícios podem ser alcançados utilizando dispositivos que são capazes de monitorar a saúde de uma pessoa e fornecer recursos de diagnóstico em movimento.

- **Ambiente inteligente:** este domínio é um esforço para monitorar o espaço ao nosso redor para uma melhor compreensão. Embora não possamos controlar uma força da natureza, por meio de uma observação cuidadosa podemos detectar diferentes fenômenos naturais e reagir a eles de acordo.

- **Cidades Inteligente:** este domínio destina-se à segurança dos cidadãos, organizações, espaços públicos e instalações. O monitoramento geralmente é alcançado usando câmeras estáticas (por exemplo, câmeras de CFTV), bem como câmeras móveis (por exemplo, drones), no entanto, outras fontes de dados, como feeds de mídia social, também foram incluídas para melhorar a qualidade e o valor da informação dos dados.

- **Energia Inteligente:** visa melhorias na distribuição e consumo de serviços públicos, como eletricidade, gás e água. A produção de eletricidade é o ponto focal, pois há um esforço global em direção a fontes de energia renováveis, que produzem energia ecológica, mas flutuante. Devido à sua natureza volátil, a rede elétrica tem que ser muito mais flexível e dinâmica.

- **Transporte e Mobilidade Inteligente:** visa tornar o transporte mais rápido, mais barato e mais seguro, o que pode ser alcançado através da coleta de todos os tipos de dados para analisar e encontrar soluções ideais.

- **Fabricação e Varejo Inteligente:** visa apoiar a produção de produtos personalizados sob alta demanda na fabricação e varejo. Para isso, a IoT acompanha essa tendência, oferecendo períodos de desenvolvimento mais curtos, com grande flexibilidade, eficiência de recursos e novos modelos de negócios inovadores.

- **Agricultura Inteligente:** este domínio visa enfrentar os desafios ligados ao clima, segurança alimentar e seu rápido crescimento da população global e implantação de novas técnicas agrícolas.

Esta tese aplica o experimento do Beholder no domínio de **cidades inteligentes** e aplicado à casas inteligentes em um cenário real.

As casas inteligentes normalmente não dispõem de um gerenciamento de segurança e, por isso, os riscos associados ao uso de dispositivos IoT podem ser maiores que os benefícios que estes podem oferecer.

Portanto, a tecnologia de casa inteligente também apresenta riscos potenciais à segurança. Como os dispositivos domésticos inteligentes processam diretamente os dados gerados pelo usuário, uma vez comprometidos, eles podem apresentar sérias consequências. Por exemplo, a privacidade do usuário pode ser prejudicada; a propriedade pode ser destruída; a segurança da vida e a saúde psicológica também estão ameaçadas (ZHOU et al., 2019).

## 4.4 REQUISITOS DO BEHOLDER

Para construir a arquitetura do Beholder, esta tese se baseou na norma NIST 800-94 (SCARFONE; MELL, 2007), que é referência em normas e padrões em segurança da informação para as empresas públicas federais dos Estados Unidos. Esta norma ajudou a definir os requisitos funcionais e não funcionais com base em características das tecnologias para *Intrusion Detection and Prevention Systems* (IDPS) baseadas em rede, recomendações, design, implementação, configuração, proteção, monitoramento e manutenção (SCARFONE; MELL, 2007).

Os requisitos foram definidos para promover o desenvolvimento de um sistema de detecção centralizado na rede com uma abordagem de detecção de intrusão, conforme recomenda a NIST 800-94 (SCARFONE; MELL, 2007).

### 4.4.1 Requisitos Funcionais

A seguir, são apresentados os requisitos funcionais que foram escolhidos para nortear o desenvolvimento da arquitetura do Beholder:

- **RF01** - **Monitorar tráfego**: permite capturar e analisar o tráfego de rede que passa pelo Beholder, assim como o fluxo de tráfego associado a um firewall. Este requisito define a necessidade em desenvolver o mecanismo de captura de pacotes.
- **RF02** - **Reconhecer ataques de camada de transporte**: detectar ataques em protocolos como TCP e UDP. Este requisito tem suma importância em todo o processo de identificação de ataques. Após o monitoramento do tráfego, o Beholder precisa analisar cada fluxo TCP ou UDP.
- **RF03** - **Reconhecer a camada de aplicação**: define que o Beholder capture os pacotes trafegados e detecte ataques em protocolos como AMQP, CoAP e MQTT. Este requisito permite ao Beholder analisar e categorizar os pacotes AMQP, CoAP e MQTT. Neste

requisito, o desenvolvimento se baseia na leitura dos payloads existentes nos pacotes TCP e UDP.

- **RF04 - Reconhecer a camada de rede**: define a análise dos pacotes em protocolos como ICMP, IP, entre outros desta camada. Com este requisito, é possível atender à necessidade de identificação do endereço lógico de cada dispositivo produtor. Tal requisito é importante para envio do endereço lógico para que seja executada a regra de firewall.

- **RF05 - Identificar hosts**: cria uma lista de hosts na rede organizada por endereço IP ou endereço MAC. A lista poderá ser usada como um perfil para identificar novos hosts na rede. Ao atender esse requisito, será enviada ao console a lista de supostos dispositivos maliciosos, ou até mesmo, a inserção de algum dispositivo numa lista de dispositivos ignorados. Tal lista servirá para relacionar dispositivos que por algum momento podem gerar alerta de falso positivo.

- **RF06 - Identificar características de rede**: coletar informações gerais sobre o tráfego de rede relacionadas à configuração de dispositivos de rede e hosts. Essas configurações são importantes porque podem apresentar informações de tempo, latência, tamanho dos pacotes, tamanho da carga de dados, e se ocorreu algum erro durante a transmissão.

- **RF07 - Executar bloqueio de tráfego**: o mecanismo CEP oferece recursos de firewall que podem ser usados para liberar ou rejeitar atividades suspeitas de rede. Ao gerar o alerta, o Beholder irá executar a regra de firewall para descartar o envio e recebimento dos pacotes do suposto dispositivo malicioso. Desta forma, o Beholder irá atuar também como um sistemas de prevenção de intrusão.

- **RF08 - Monitorar e analisar**: oferecer recursos de gerenciamento, monitoramento, análise e apresentação de dados. Este requisito permite que dados sejam enviados a um console, facilitando, assim, a visualização das informações dos pacotes e mensagens que são enviados pelos dispositivos IoT.

- **RF09 - Permitir atualizações nas regras de detecção**: as atualizações de regras devem adicionar novos recursos de detecção ou refinar os recursos de detecção existentes (por exemplo, reduzindo falsos positivos). Este requisito é importante para melhorar a margem de detecção de intrusão das regras existentes, além de facilitar a inserção de novas regras de detecção.

### 4.4.2 Requisitos Não Funcionais

- **RNF01 - Avaliar a Precisão na Detecção**: conforme a norma NIST 800-94 (SCARFONE; MELL, 2007), avaliar o desempenho da detecção de intrusão é um requisito fundamental para o desenvolvimento de sistemas de detecção de intrusão. Nesta tese são utilizadas duas métricas derivadas da matriz de confusão: *Recall* e *Precision*. O Recall indica a proporção de positivos reais classificados corretamente. O conceito de precisão de detecção se baseia na análise de falsos positivos e verdadeiros positivos (HINDY et al., 2018).

## 4.5 MODELO '4+1'

Uma arquitetura de software pode ser definida como a organização de elementos fundamentais/componentes de um sistema. Ela representa a estrutura ou estruturas do sistemas que compreendem elementos de software, e a relação entre esses elementos (EELES, 2006).

Uma arquitetura de software lida com o design e a implementação do software. Em outras palavras, podemos dizer que uma arquitetura é responsável pela estruturação dos componentes de software para satisfazer exigências ou requisitos (KRUCHTEN, 1995). Ela consiste em todas as decisões de design importantes sobre as estruturas de software e as interações entre essas estruturas que compõem o sistema (EELES, 2006).

A arquitetura de software consiste de uma variedade de artefatos de design típicos de outras disciplinas de engenharia, como diagramas, desenhos e modelos estáticos e dinâmicos. Ela é responsável pela natureza complexa e não material do produto de software e seu relacionamento com as ferramentas e técnicas de gerenciamento de projetos. Os artefatos associados à arquitetura física formam o pacote de dados técnicos do software que é fornecido à equipe de implementação do software para design, codificação, integração e teste da implementação do software (SCHMIDT, 2013).

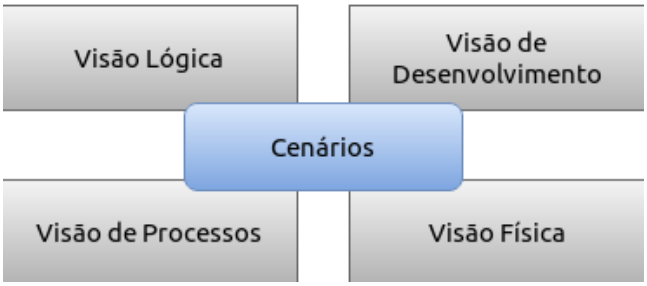
A especificação de uma arquitetura de software contém muitas das decisões de projetos importantes. A escolha de uma definição de arquitetura de software influencia o conteúdo de um ou mais requisitos da arquitetura. A escolha da arquitetura impacta diretamente no processo de desenvolvimento de software (EELES, 2006).

Segundo Tofan (TOFAN; GALSTER; AVGERIOU, 2013), existem muitas pesquisas sobre como documentar uma arquitetura de software no mundo real. Porém, existe uma dificuldade em

documentar uma arquitetura de software de forma legível. Visando facilitar a compreensão, documentação e legibilidade dos componentes, esta tese adotou o método de documentação de arquitetura de software 4+1 de Kruchten (KRUCHTEN, 1995).

O método 4+1 foi escolhido por facilitar a documentação da arquitetura do Beholder em várias visões, abordando o Beholder de uma forma completa, separando as perspectivas arquiteturais e avaliando cada componente do software. A Figura 20 ilustra a integração das visões.

Figura 20 – O modelo de Visão "4+1"




**Fonte:** Adaptado de (KRUCHTEN, 1995).

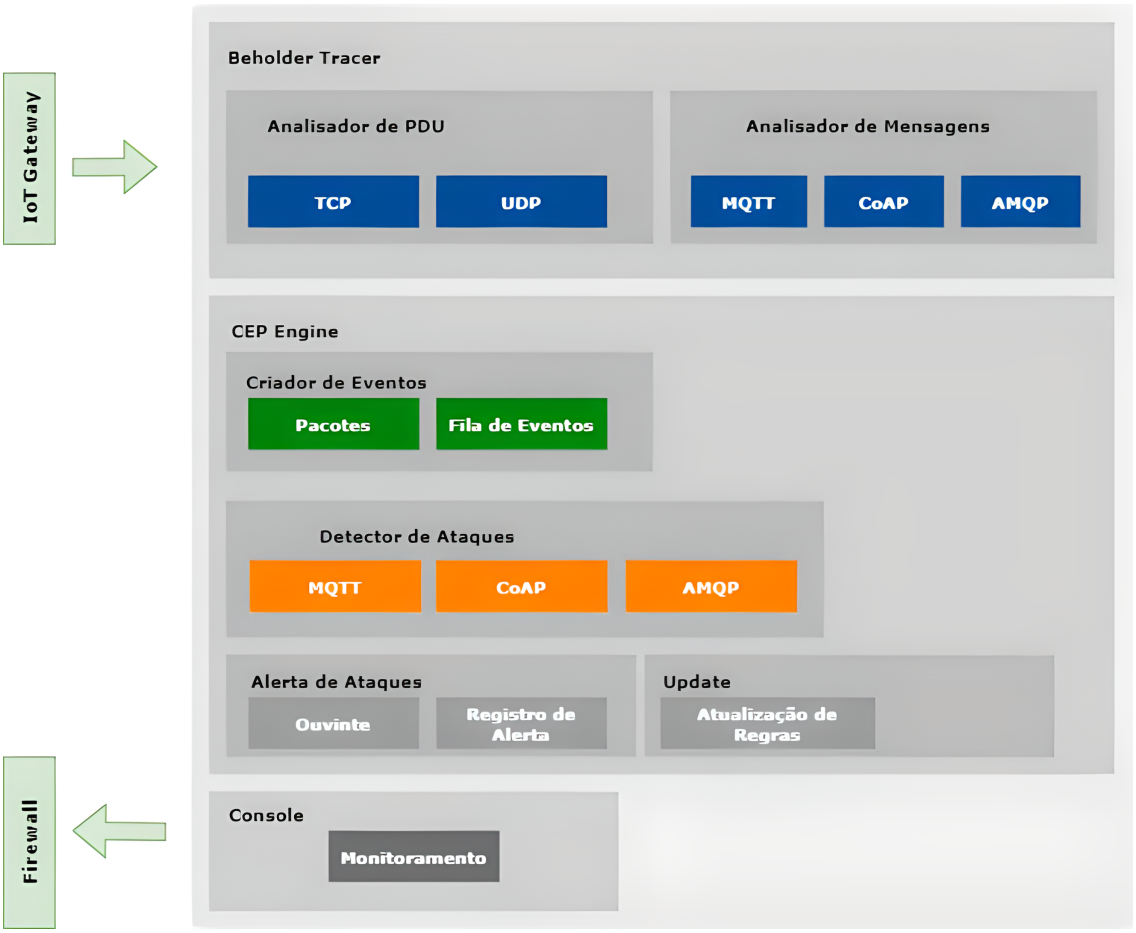
Cada visão é descrita como um projeto usando sua própria notação. Por exemplo, um diagrama de implementação é usado para descrever os componentes da arquitetura. Entretanto, esse modelo é genérico, podendo ser utilizadas outras notações, como diagrama de processos, diagrama de atividades, diagrama de componentes, diagrama UML e etc. Cada visão é brevemente apresentada em seguida.

- **Visão lógica:** descreve componentes e modelos de um software. Ela oferece suporte principalmente, a requisitos funcionais ou como o sistema pode fornecer os serviços aos usuários.
- **Visão de processo:** captura a dependência entre os componentes, concorrência e aspectos de sincronização dos processos do Beholder.
- **Visão de desenvolvimento:** descreve a organização do software utilizado no desenvolvimento, apresentando a linguagem de programação, bibliotecas e API's.
- **Visão física:** esta visão se preocupa com a topologia dos componentes de software na camada física, bem como com as conexões físicas entre esses componentes.
- **Cenários:** apresentam casos de uso significativos em que o Beholder pode ser implantado.

## 4.6 VISÃO LÓGICA

Esta seção apresenta a organização da arquitetura do Beholder a partir das suas funcionalidades. Essas funcionalidades estão ligadas a elementos, módulos e componentes. A Figura 21 apresenta a arquitetura do Beholder do ponto de vista lógico.

Figura 21 – Componentes da Arquitetura do Beholder




**Fonte:** O Autor.

A responsabilidade de cada módulo da arquitetura serão agora destacados.

### 4.6.1 *Beholder Tracer*

Através deste componente, pode-se selecionar a interface desejada para captura de pacotes. Além da camada de aplicação da IoT e dos protocolos da camada de transporte, ele é capaz de capturar outros protocolos, como IPV4, IPV6, ICMP, ARP, PPP, IEEE 802.x, Ethernet, Bluetooth etc. Com base na interface selecionada, o Beholder Tracer utiliza a PCAP (Packet

Capture Library [6]) correspondente para coletar e registrar dados do pacote em tempo real. O Beholder Tracer é constituído dos seguintes componentes:

- **Analisador de PDU (Unidade de Protocolo de Dados)**: captura os fluxos **TCP** e **UDP** em tempo real e suas respectivas cargas de dados. O Beholder Tracer pode ser extensível a outros tipos de PDU, como quadros de enlace.
- **Analisador de Mensagens**: este componente detecta os quadros de mensagem AMQP, CoAP e MQTT. Para tanto, analisa os pacotes capturados pelo **Analisador de PDU**. Em seguida, este componente inspeciona cada campo dos quadros de mensagem AMQP, CoAP e MQTT. O Beholder Tracer pode ser estendido para dar suporte a outros protocolos da camada de aplicação da IoT, como o XMPP (6120, 2011) e DDS (OMG, 2015).

Para executar o **Beholder Tracer**, o arquivo de configuração com a extensão do arquivo.so da biblioteca jnetpcap Versão 2.0 (JNETPCAP, 2022) precisou ser recompilado para se tornar compatível com o processador do dispositivo que utiliza a arquitetura do processador ARM.

### 4.6.2 CEP Engine

#### *4.6.2.1 Criador de Eventos*

Após capturar e analisar os quadros dos protocolos AMQP, CoAP e MQTT, o **Beholder Tracer** envia os respectivos pacotes ao componente **Criador de Eventos**. Esse componente usa a CEP engine para criar um evento, enviando os campos dos quadros de mensagens analisados anteriormente. Em seguida, o evento é adicionado a uma fila de eventos, especificando a que tipo esse evento pertence: fluxo AMQP, CoAP ou MQTT, conforme exemplo da criação de evento AMQP.

Código Fonte 2 – Exemplo de envio de evento AMQP ao CEP

```
1    AMQPPacketSender amqpps = new AMQPPacketSender(method, classc, evilhost,
         srcAddr);
     cepLocal.sendEvent(amqpps);
```

**Fonte:** O Autor.

[6] https://www.tcpdump.org//

A partir desse ponto, todos os fluxos AMQP, CoAP e MQTT são analisados por um conjunto de regras na CEP Engine. Essa regra consiste não apenas na definição de semântica associada ao mecanismo CEP, mas também determina a função dos agentes de processamento de eventos (*Event Processing Agent* - EPA).

O código Java a seguir mostra um trecho da classe MainCEP.java, onde são configurados o recebimento dos fluxos e execução das regras CEP.

Código Fonte 3 – Trecho da Classe Java MainCEP

```
public class MainCEP {
    public static void main(String[] args) {
        try {
            Configuration config = new Configuration();

            config.addEventType("CoAP_Stream", CoapPacketSender.class);
            EPServiceProvider coapCep = EPServiceProviderManager.getProvider("
                esper-coap", config);

            config.addEventType("AMQP_Stream", AMQPPacketSender.class);
            EPServiceProvider amqpCep = EPServiceProviderManager.getProvider("
                esper-amqp", config);

            config.addEventType("MQTT_Stream", MQTTPacketSender.class);
            EPServiceProvider mqttCep = EPServiceProviderManager.getProvider("
                esper-mqtt", config);
```

**Fonte:** O Autor.

Nesse exemplo, o config.addEventType é declarado para permitir que instruções de padrão de eventos e instruções EPL usem o nome do tipo de evento em vez do nome da classe Java Completo. O EPServiceProvider e o método getProvider são usados para enviar eventos em processamento ao mecanismo Esper (ESPERTECH, 2019) para definir e obter valores de variáveis e executar consultas sob demanda.

#### *4.6.2.2 Detector de Ataques*

A detecção de ataques baseada no mecanismo CEP é capaz de executar o processo de comparação de definições executando filtros. Cada atividade pode não ser considerada normal se um determinado comportamento for detectado pela regra CEP. Uma regra do CEP no

Beholder é criada, analisando o fluxo enviado pelo **Beholder Tracer** e o **Criador de Eventos**. As regras também monitoram características dos eventos ao longo do tempo. O Detector de Ataques pode ser extendido a fluxos, conforme análise do **Analisador de PDU**.

#### *4.6.2.3 Alerta de Ataques*

Este componente tem como objetivo registrar alertas relacionados aos eventos detectados. Esses dados podem ser usados para confirmar os alertas emitidos e correlacionar os eventos entre o mecanismo CEP e outras fontes de log. Normalmente, os campos usados em um log podem incluir a data, hora, tipo de evento e o endereço do suposto invasor. Os alertas são armazenados em um arquivo de configuração.

No subcomponente **Ouvinte**, um alerta é enviado para um terminal ou painel com o identificador do suposto atacante. Este identificador pode ser vinculado a um endereço lógico IPv4, IPv6 ou endereço físico (MAC). Em seguida, o endereço é inserido em uma regra do firewall iptables para bloqueio. Por fim, o **Registro de Alerta** insere no arquivo de log o endereço lógico, data, e tipo de ataque detectado.

#### *4.6.2.4 Atualização de Regras*

Este componente tem por objetivo realizar atualizações nas regras de detecção por um usuário administrador do Beholder através de um serviço SSH (*Secure Shell*). Neste componente, as regras de detecção são atualizadas para refinar os recursos de detecção existentes e novas regras de detecção podem ser adicionadas. Os administradores de um IDPS devem revisar o ajuste e as personalizações periodicamente para garantir a precisão dos resultados (SCARFONE; MELL, 2007).

#### *4.6.2.5 Console*

**Console** é um programa que fornece uma interface para os usuários e o administrador do Beholder. Este componente é usado para obter informações e verificação dos alertas e quantidade, pacotes que estão sendo trafegados. O **Console** é composto pelo subcomponente **Monitoramento**. Com esse subcomponente, é possível visualizar em tempo real informações do tráfego de pacotes, qual o protocolo está sendo utilizado e possível ocorrência de ataque.

#### *4.6.2.6 Firewall*

O componente Firewall tem por objetivo servir como mecanismo de prevenção, usando uma lógica de execução de firewall nativo do GNU/Linux para realizar bloqueios e controle de largura de banda com o Firewall Iptables. O Iptables faz parte do projeto Netfilter (WANG; LU; CHANG, 2016). Com ele, é possível criar regras de firewall e Network Address Translation (NAT).

### 4.6.3 Mapeamento de Componentes X Requisitos

A Tabela 4 apresenta a relação dos requisitos funcionais com os componentes da arquitetura Beholder. É possível observar que os componentes descritos atendem a todos os requisitos funcionais da arquitetura Beholder.

Tabela 4 – Mapeamento: Componentes x Requisitos Funcionais do Beholder

| | RF01 | RF02 | RF03 | RF04 | RF05 | RF06 | RF07 | RF08 | RF09 |
|---|---|---|---|---|---|---|---|---|---|
| Beholder Tracer | X | | | | X | X | | | |
| CEP Engine | | X | X | X | | | X | X | |
| Console | | | | | | | | | X |

**Fonte:** O Autor.

## 4.7 VISÃO DE PROCESSOS

A visão de processo apresenta as partes dinâmicas e comportamento do sistema, explicando os processos e como eles se comunicam. Ela também pode apresentar questões de concorrência e distribuição, de integridade do sistema, de tolerância a falhas (KRUCHTEN, 1995).

A Figura 22 apresenta uma notação em Business Process Management Notation (BPMN) que descreve a execução dos processos de cada componente da arquitetura do Beholder.

1. **Gateway IoT**

   **Conectar dispositivos**: inicia quando os dispositivos IoT são conectados a uma rede wi-fi ou bluetooth, gerenciada pelo gateway IoT. Essa rede é gerenciada pelo gateway IoT, utilizando técnicas NAT para que os dispositivos IoT tenham conexão com a rede externa,

com a nuvem e com os servidores. Assim, todos os fluxos enviados pelos dispositivos IoT irão passar pelo gateway IoT.

2. **Beholder Tracer**

   **Capturar carga de dados**: inicia quando os dispositivos IoT conectados ao gateway IoT começam a enviar pacotes TCP e UDP. Esses pacotes são capturados pela biblioteca libpcap a partir do armazenamento do buffer do sistema operacional.

   **Filtrar carga de dados**: inicia quando a carga de dados dos pacotes TCP e UDP é inspecionada. A carga de dados é analisada para que sejam encontrados PDU's da camada de aplicação. Caso a carga de dado contenha o PDU do protocolo AMQP, CoAP ou MQTT, ele será filtrado e enviado para ter seu quadro de mensagem inspecionado.

   **Inspecionar quadros de mensagem**: inicia quando cada quadro de mensagem é analisado para que sejam categorizados, conforme cada protocolo da camada de aplicação IoT.

3. **CEP *Engine***

   **Categorizar quadro de mensagem**: categorizar um processo é identificá-lo e informar a que tipo de evento ele pertence. Esta tarefa no CEP Engine envolve duas atividades: **Enviar quadro inspecionado** e **Enviar evento a uma fila de eventos**.

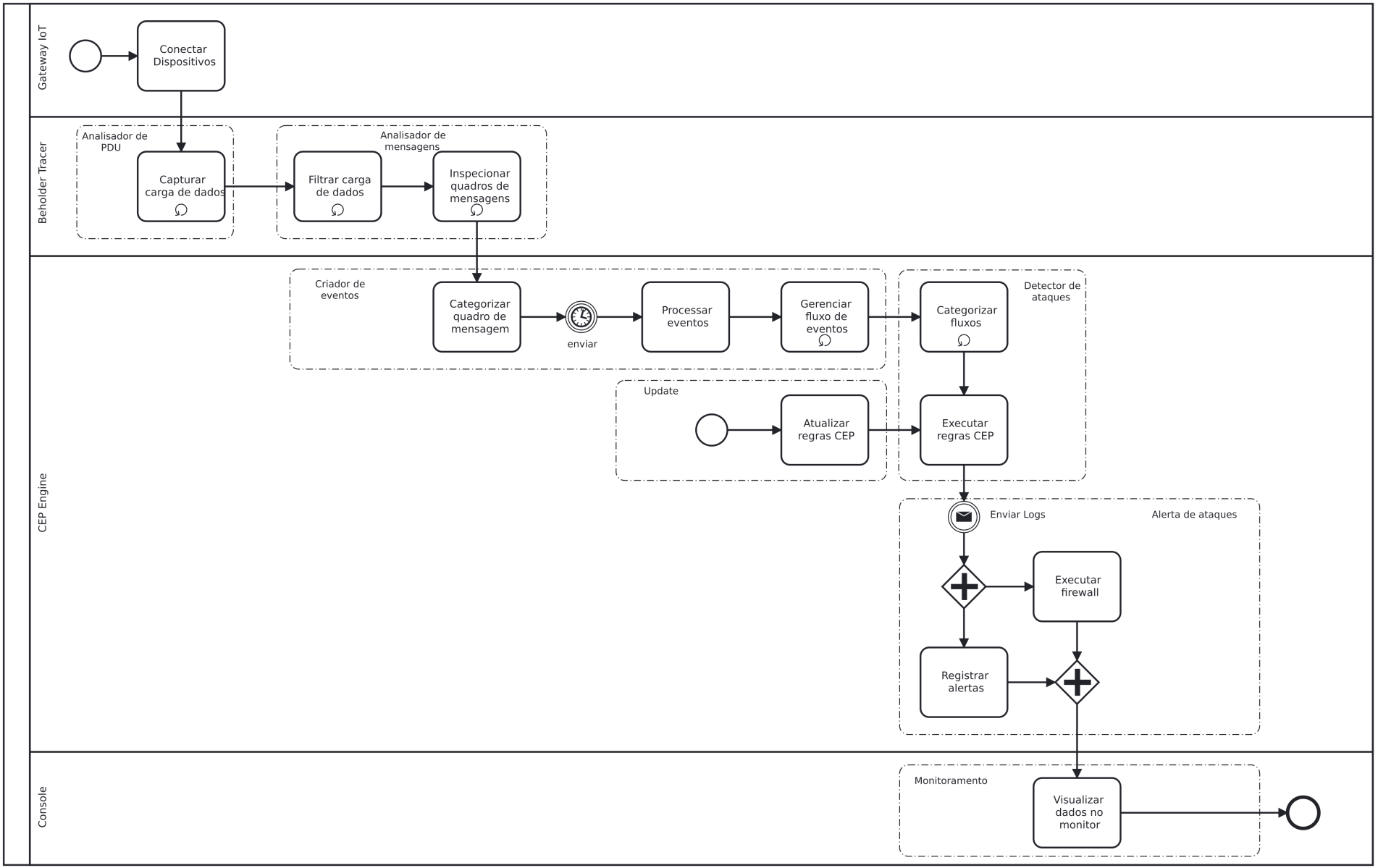
   **Enviar quadro inspecionado**: essa atividade inicia quando for identificado o protocolo da camada de aplicação ao qual o quadro de mensagem pertence (AMQP, CoAP ou MQTT). Na sequência, o quadro é enviado a um evento gerenciado pelo CEP *Engine*.

   **Enviar evento a uma fila de eventos**: essa atividade envia os eventos AMQP, CoAP e MQTT para uma fila de eventos do CEP *Engine*.

   **Processar eventos**: inicia quando os eventos começam a ser processados de forma contínua pelo CEP *Engine*. Nesta atividade, a CEP engine processa as filas de eventos que foram criadas na atividade anterior.

   **Gerenciar fluxos de eventos**: inicia quando os eventos são agrupados pelo tipo de evento. Por exemplo, eventos do protocolo CoAP contêm informações diferentes daquelas contidas nos eventos do protocolo MQTT ou AMQP. Essa diferença é baseada nos campos dos quadros de mensagens e seus comportamentos, que podem incluir aspectos de tamanho da carga de dados, tempo de envio e quantidade de campos.

Figura 22 – Atividades do Beholder




**Fonte:** O Autor.

**Categorizar fluxos**: esta atividade inicia quando os fluxos dos eventos em tempo real são enviados para os filtros de regras CEP já estabelecidos. Cada fluxo de evento é identificado e enviado para o filtro adequado.

**Executar regras CEP**: esta atividade inicia quando cada fluxo de evento for recebido. As regras CEP são executadas apenas nos fluxos de eventos já categorizados. O Beholder processa os dados existentes no fluxo de eventos e compara com o filtro existente na regra CEP para detecção de ataques. Caso algum fluxo de evento caia no filtro, os registros são enviados para duas atividades em paralelo: **Registrar Alertas** e **Executar regras de firewall**.

**Atualizar regras CEP**: inicia quando o administrador do Beholder realiza atualizações nas regras de detecção, podendo refinar as regras existentes ou adicionar novas regras.

**Registrar Alertas**: inicia quando os registros são recebidos pela atividade **Executar Regras CEP**. Essa atividade pode registrar informações do dispositivo como endereços lógicos de origem e destino, endereços físicos de origem e destino, data e hora da ocorrência.

**Executar firewall**: inicia quando o firewall recebe os dados do endereço lógico (IPv4) do dispositivo detectado como ameaça. As regras do firewall poderão realizar bloqueios de envio de pacotes ou controlar a largura de banda do suposto dispositivo malicioso.

4. **Console**

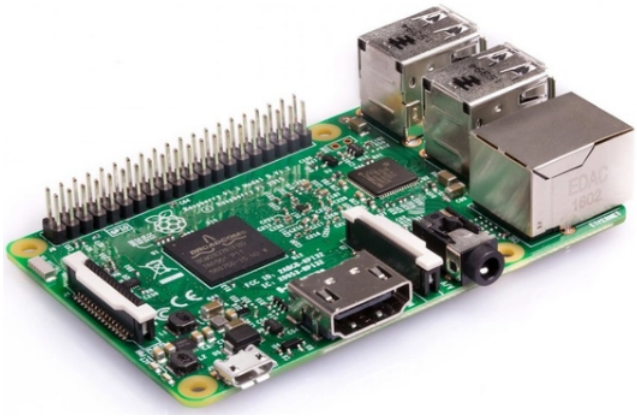
**Visualizar dados no monitor**: inicia quando o console recebe informações fornecidas pelo **CEP Engine**. Nesta atividade são visualizadas informações como pacotes trafegados, endereços físicos e lógicos de origem e destino, informações do firewall e, em caso de detecção de ataque, qual tipo de ataque utilizado.

## 4.8 VISÃO FÍSICA

A visão física leva em consideração os requisitos físicos mínimos para utilização da arquitetura Beholder. Esta visão apresenta o ambiente de execução do sistema, mapeando os artefatos de software ao hardware que os hospedará.

Por se tratar de um ambiente com equipamentos com restrições de memória e processamento, a arquitetura do Beholder utiliza uma infraestrutura baseada em um Simple Boarding Computing (SBC), Raspberry Pi Model 3B (RASPBERRY, 2020), conforme apresentada na Figura 23.

Figura 23 – Raspberry Pi Modelo 3B



**Fonte:** (RASPBERRY, 2020)

A Tabela 5 descreve as principais características de hardware e software dessa infraestrutura do Beholder. Ela também apresenta a topologia dos componentes de software no contexto físico. Os softwares Raspian Stretch, Kura, Java, Esper e Iptables foram escolhidos devido à compatibilidade e constantes atualizações com o hardware Raspberry Pi Modelo 3B.

Tabela 5 – Características de Hardware e Software da Visão Física

| **Hardware** | **Especificação** |
|---|---|
| Raspberry Pi | Modelo 3B |
| Memória RAM | 4Gb |
| Armazenamento | 32Gb |
| Processador | Broadcom BCM2837 (ARM Cortex-A53 Quad Core de 64 bits) |
| **Software** | **Especificação** |
| Sistema Operacional | Raspbian Stretch |
| Gateway IoT | Kura Gateway Versão 4.1.1 |
| Linguagem de Programação | Java Versão 8 |
| CEP *Engine* | Esper Versão 4.4.1 |
| Firewall | Iptables |

**Fonte:** O Autor.

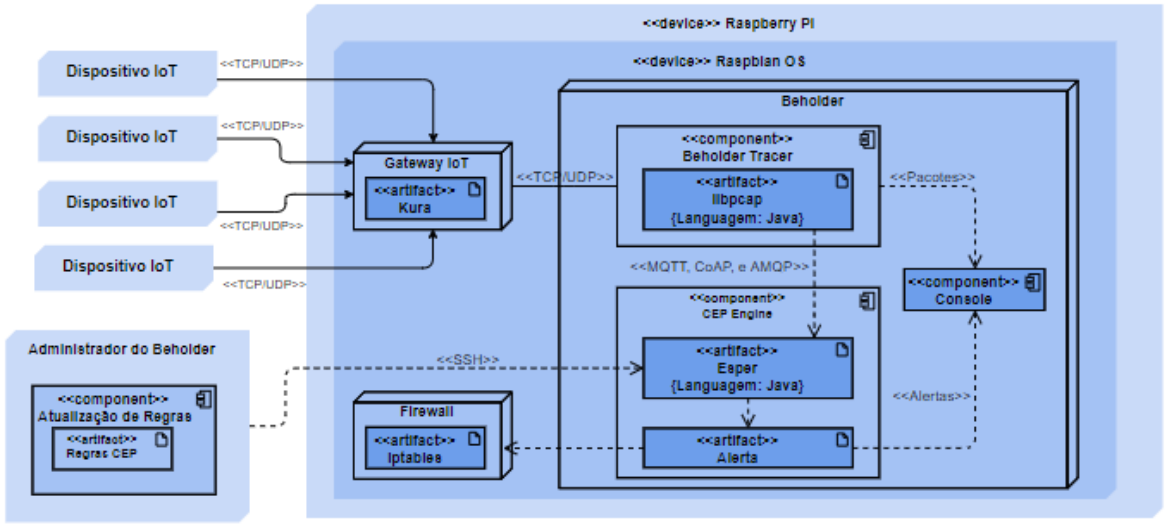
## 4.9 VISÃO DE DESENVOLVIMENTO

A visão de desenvolvimento tem por objetivo realizar uma organização dos módulos de software e hardware. Esta arquitetura pode ser representada por diagramas de módulos e implementação, mostrando relacionamentos entre os componentes.

Esta arquitetura pode levar em consideração requisitos internos relacionados à facilidade no desenvolvimento, gerenciamento de software, reutilização, ou pela linguagem de programação. Portanto, esta visão pode servir como base para alocação de requisitos, planejamento de custos, monitoramento do projeto, portabilidade e segurança do software.

Para apresentar a visão de desenvolvimento do Beholder, foi utilizado um diagrama de implementação, conforme a Figura 24. Em seguida, é apresentada a descrição detalhada da arquitetura, dispositivos e softwares que são executados nesse dispositivo.

Figura 24 – Diagrama de Implementação da Arquitetura Beholder




**Fonte:** O Autor.

Os componentes do Diagrama de Implementação são detalhados a seguir:

1. **Dispositivo IoT**

   Os dispositivos IoT, qualquer objeto, sensores ou outros sistemas digitais, realizam a troca de informações através de conexão com um gateway IoT.

2. **Raspbian OS**

   **Gateway IoT**: o Gateway IoT é responsável pela conexão dos dispositivos IoT. Essa

conexão é realizada pelo software Kura[7]. O Kura Gateway é um software desenvolvido em Java para abstrair objetos IoT e conectá-los a outros serviços. Por exemplo, serviços em nuvem.

**Beholder**: é a principal contribuição desta tese, que tem como principal objetivo realizar detecção e prevenção contra intrusões em ambientes IoT. Conforme o diagrama da Figura 24, esta aplicação possui os componentes (Beholder Tracer, CEP Engine e Console).

**Beholder Tracer**: este componente foi desenvolvido utilizando a API jnetpcap para coleta de pacotes em redes. Esta biblioteca se baseia na biblioteca libpcap para distribuições GNU/Linux.

**CEP Engine**: é responsável por detectar anomalias em ambientes IoT. Este componente foi desenvolvido utilizando a API ESPER[8] para criação, execução e geração de alertas baseadas em regras CEP.

**Alertas**: arquivo de log que recebe os alertas gerados pelo CEP Engine.

**Console**: programa que recebe os registros enviados pela CEP Engine e as informações de tráfego enviadas pelo Beholder Tracer.

3. **Firewall Iptables**
   Firewall nativo do linux, responsável por execução de tarefas referentes ao uso de largura de banda e bloqueio de envio de pacotes.

4. **Administrador do Beholder**
   Dispositivo remoto que possibilita envio de atualizações de regras pelo administrador do Beholder via protocolo de aplicação SSH (*Secure Shell*).

## 4.10 CENÁRIOS

Esta seção detalha os ambientes para *smart home* a serem utilizados como o cenário escolhido para experimentação do Beholder. Entre 2018 e 2019, o número de ataques a ambientes de *smart home* aumentaram sete vezes. Isso mostra que esse tipo de ambiente é um alvo em potencial para inúmeros invasores (FADILPAŠIĆ, 2019).

[7] https://www.eclipse.org/kura
[8] http://www.espertech.com/

No cenário utilizado, a *smart home* contém vários sensores. Esses sensores, por exemplo, podem ser de presença, temperatura luminosidade e umidade, mas podem existir outros. Esses dispositivos podem ser distribuídos entre os vários cômodos da casa. Os fluxos de dados podem ser transmitidos para algum serviço em nuvem, e seus dados serem consumidos em alguma aplicação.

Este cenário contém outros tipos de dispositivos IoT como consoles de jogos eletrônicos, *smart tv*, travas eletrônicas inteligentes, e câmeras IP sem fio. Além disso, vários smartphones pode ser utilizados nesse ambiente.

Por fim, a IoT Security Foundation (FOUNDATION, 2018) sugere que uma *smart home* tenha um equipamento centralizador (gateway IoT) para realizar as conexões entre os dispositivos IoT para a internet. Tais características subsidiaram a definição dos requisitos foram empregados na construção da arquitetura do Beholder.

## 4.11 CONSIDERAÇÕES FINAIS

Este capítulo apresentou a principal contribuição desta tese, que é o Beholder, uma solução para detecção e prevenção de intrusão em ambientes IoT.

O Beholder propõe uma solução para melhorar a segurança de ambientes da IoT, realizando uma detecção de ataques que explorem vulnerabilidades nos protocolos MQTT, CoAP e AMQP, podendo ser expandida a outros protocolos da camada de aplicação. Ela foi projetada com base nos requisitos e recomendações da norma NIST 800-94 (SCARFONE; MELL, 2007) para definição de seus principais componentes.

O Beholder torna possível o suporte a outros protocolos da camada de aplicação IoT. Porém, a maior contribuição desta tese é a detecção de ataques em aplicações que utilizem os protocolos AMQP, CoAP e MQTT. O Beholder utiliza o CEP como um mecanismo para categorizar os eventos AMQP, CoAP e MQTT, criar as regras de detecção de intrusão e gerar os alertas.

# 5 AVALIAÇÃO E RESULTADOS

O capítulo anterior apresentou a principal contribuição desta tese, o Beholder, que é um IDPS que provê suporte a detecção de intrusão na camada de aplicação da IoT.

Este capítulo busca avaliar o Beholder. Para isso, será empregada uma metodologia de avaliação usando como caso de uso um cenário real de uma *smart home*. O intuito é usar o protótipo do Beholder implementado no escopo desta tese e realizar detecções de intrusão em tempo real com base em dados reais.

Nesta avaliação, os dispositivos IoT estarão conectados ao Beholder, que centraliza a conexão entre a intranet e a nuvem, conforme recomendações da IoT Security Foundation (FOUNDATION, 2018) para infraestrutura de *smart home*.

Segundo a NIST 800-094 (SCARFONE; MELL, 2007), a precisão de detecção de intrusão de um IDPS é um dos principais requisitos no desenvolvimento deste tipo de sistema considerando sua importância. Para sistema de IDPS, também é importante avaliar o consumo de memória RAM e CPU. O consumo elevado desses recursos pode levar a perda de pacotes durante a transmissão, o que pode gerar impacto negativo na detecção de ataques.

Para organizar metologicamente a análise, os passos adotados para realizar a avaliação são inspirados em recomendações de Jain (JAIN, 1990): 1) definir o objetivo da avaliação; 2) estabelecer o cenário (Domínio IoT) de avaliação; selecionar métricas; 3) projetar a avaliação; 4) analisar os resultados; 5) apresentar os resultados.

Os passos são apresentados nas seções seguintes.

## 5.1 OBJETIVO DE AVALIAÇÃO

O objetivo desta avaliação é analisar a precisão do Beholder na detecção de ataques em *smart homes*, utilizando um mecanismo de processamento de eventos complexos a partir de dados reais dos dispositivos IoT. Inicialmente, é avaliada a qualidade do Beholder, comparando-o contra o conhecido IDPS Snort, que é um IDPS open source amplamente utilizado pela indústria (ataques contra tráfego TCP foram utilizados em ambos os IDPS) (CISCO, 2020). Em seguida, são avaliadas as detecções dos ataques contra os protocolos AMQP, CoAP e MQTT. Em seguida, será analisando o consumo de memória RAM e CPU.

A estratégia de avaliação consiste em quatro etapas:

- será empregado o ataque do tipo SYN Flood para comparar a precisão de detecção de intrusão utilizando o Beholder e o IDPS Snort (CISCO, 2020);
- será aplicado o ataque *High QoS Flood* e Brute Force, que explora a vulnerabilidades do protocolo MQTT;
- será aplicado um ataque de negação de serviço e injeção com técnica Fuzzing, que exploram vulnerabilidades do protocolo CoAP;
- será aplicado um ataque de negação de serviço e quebra de autenticação, que exploram vulnerabilidades do protocolo AMQP.

## 5.2 APLICAÇÃO DO CENÁRIO (DOMÍNIO IOT)

### 5.2.1 Cenário

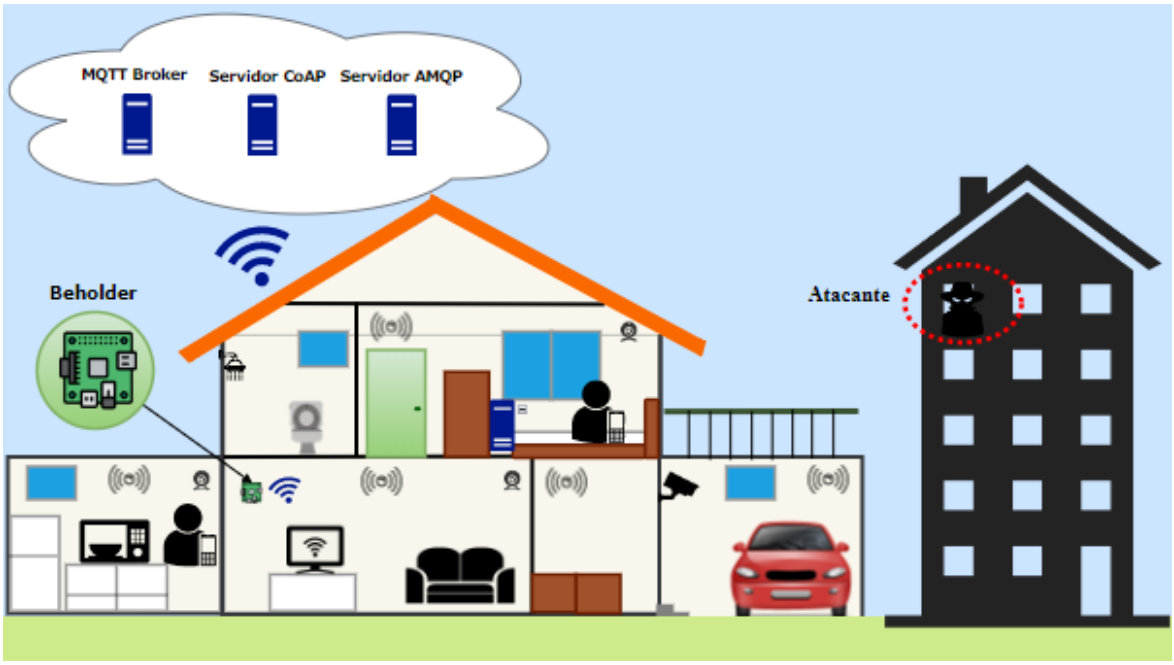
Para avaliar o Beholder, foi usado um ambiente real de IoT com diversos dispositivos em uma *smart home*. Para a análise da precisão de detecção de intrusão foram desenvolvidas aplicações IoT em que o Beholder atua junto com gateway IoT Kura para estabelecer conexão entre os dispositivos e os serviços IoT na nuvem: MQTT Broker (Mosquitto [9]), Servidor CoAP [10]) e Servidor AMQP (RabbitMQ[11]). A Figura 25 ilustra uma visão geral do cenário proposto.

Conforme ilustrado na Figura 25, a *smart home* tem um quarto e um banheiro no primeiro andar. No térreo, fica a sala principal, a cozinha, o armazém e a garagem. A casa está equipada com os seguintes dispositivos:

- 4 câmeras IP sem fio colocadas no quarto, cozinha, sala de estar e garagem;
- 6 smartphones consumindo serviços de mensagens, streaming (vídeo e áudio);
- 1 smart TV conectada continuamente a um serviço de streaming de vídeo;
- 1 computador desktop como cliente AMQP conectado continuamente a um servidor de envio de mensagem.

[9] https://mosquitto.org/
[10] https://coap.technology/
[11] https://www.rabbitmq.com/

Figura 25 – Cenário de avaliação do Beholder




**Fonte:** O Autor.

A avaliação fez uso dos serviços na nuvem da plataforma *open source* IoT Thingsboard (THINGBOARD, 2020) para as aplicações que usam o protocolo MQTT e as aplicações baseadas no protocolo CoAP. No AMQP, o cliente estava conectado a um servidor na nuvem da plataforma da Digital Ocean. Todos os dispositivos locais se conectam ao Beholder por uma conexão Wi-fi (802.11g).

A aplicação MQTT utiliza sensores modelo DHT11 para monitorar a presença de objetos em movimento na cozinha, sala e quartos. A aplicação captura e envia os dados coletados para o serviço IoT na nuvem a cada 5 segundos. Todo tráfego é enviado usando a porta 1883. Aqui, os sensores têm o papel de publishers, enquanto o broker é o subscribers. Cada pacote no protocolo MQTT é definido com a qualidade do serviço de entrega QoS-2, o que garante que cada pacote seja recebido apenas uma vez pelos destinatários pretendidos.

A aplicação CoAP utilizou sensores DHT12 para monitoramento de temperatura e umidade. Tais informações são captadas da cozinha, sala e quartos. O tráfego CoAP é baseado no envio das mensagens a cada 2 segundos, utilizando a porta 5863.

A Aplicação AMQP utiliza um desktop com cliente para envio de mensagens de log do sistema operacional a partir do quarto. O tráfego é baseado no envio de mensagens a cada 1, segundo utilizando a porta 5672.

O hardware do Beholder foi implementado usando o modelo Raspberry pi 3B, executando

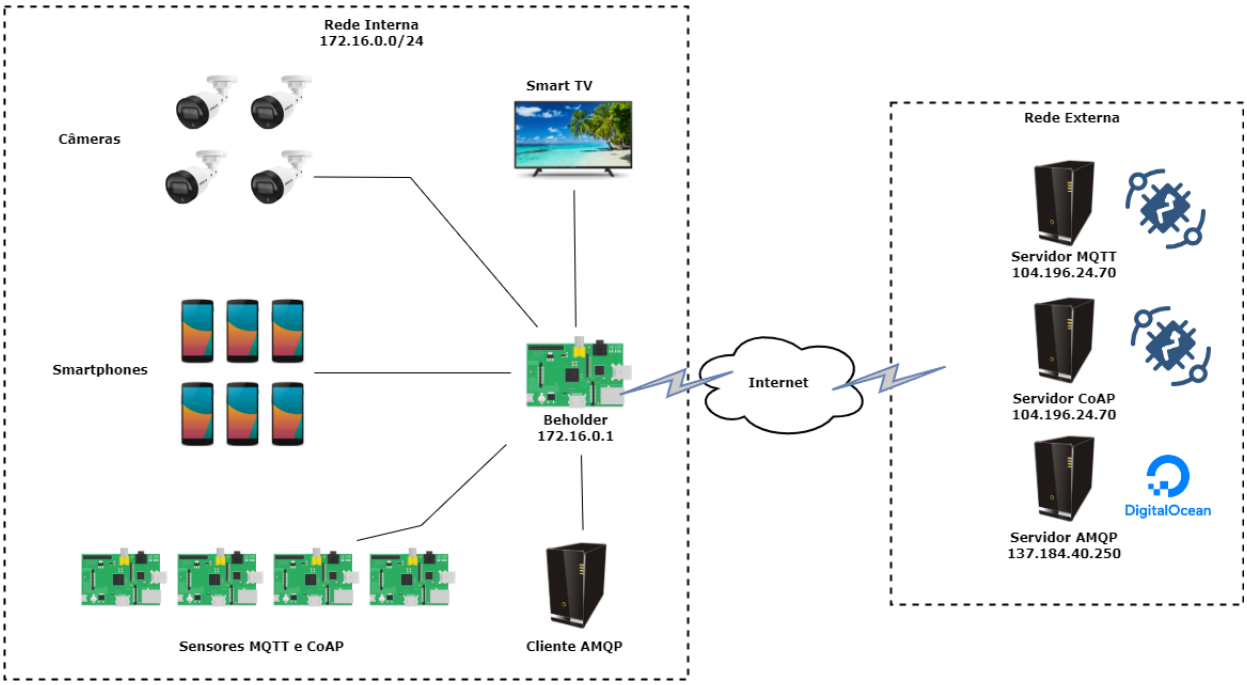
a versão Jessie do sistema operacional Raspian (Kernel 4.14) com armazenamento com cartão microSD de 32 GB e 4 GB de RAM. O gateway IoT Kura conecta todos os dispositivos sem fio para centralizar o tráfego da rede e resolver problemas de heterogeneidade.

Para realização dos ataques, sugere-se como ameaça um possível invasor localizado em um local próximo a *smart home* descrita. O atacante pode se conectar à rede da smart home, por exemplo, por meio de uma antena omnidirecional de alto desempenho, que amplifica o sinal sem fio e os pontos de acesso de 2,4 GHz.

### 5.2.2 Topologia

No cenário adotado, os dispositivos IoT (clientes) estão conectados ao Beholder, que centraliza a conexão entre a intranet e a nuvem dos serviços dos protocolos MQTT, CoAP e AMQP. Os sensores enviam para a nuvem IoT informações de temperatura, movimento e luminosidade. A rede interna é constituída por três Raspberry PI 3B, com seus respectivos sensores. Por fim, a nuvem IoT foi configurada nos ambientes da Digital Ocean [12] e Things Board [13], conforme ilustrado na Figura 26.

Figura 26 – Topologia de Rede



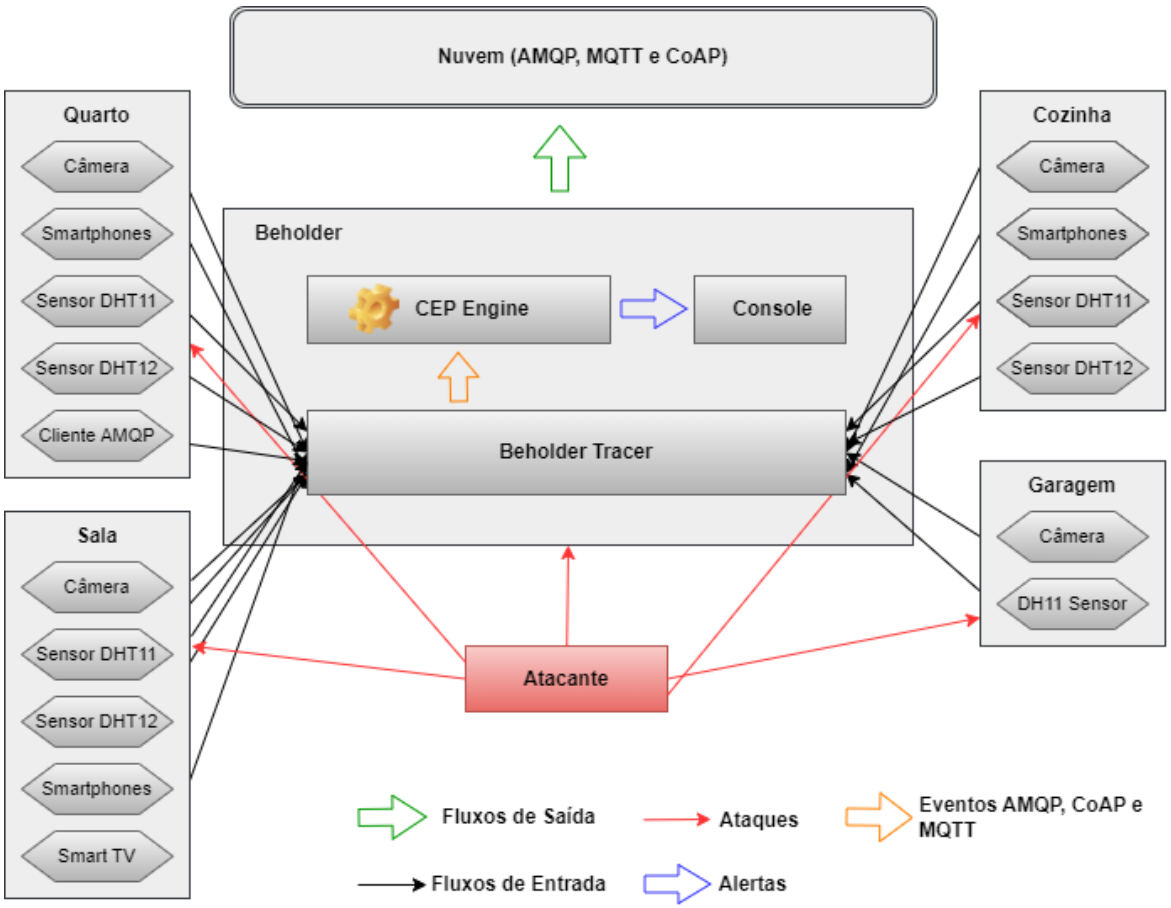

**Fonte:** O Autor.

[12] https://www.digitalocean.com/
[13] https://thingsboard.io/

### 5.2.3 Mapeamento entre o cenário e a arquitetura do Beholder

A Figura 27 apresenta um mapeamento entre o deploy do Beholder e o cenário de avaliação utilizado. É possível identificar os fluxos enviados pelos objetos inteligentes e outros dispositivos ao Beholder. O Beholder Tracer captura e analisa os pacotes AMQP, CoAP e MQTT e os envia para o CEP Engine. Em seguida, o CEP Engine examina os pacotes e os envia para o console do mecanismo. Se o mecanismo de análise do Beholder detectar um ataque, ele gera um alerta.

Figura 27 – Arquitetura de implantação mapeada com o cenário de Smart Home




**Fonte:** O Autor.

## 5.3 MÉTRICAS DE AVALIAÇÃO

A precisão do IDPS pode ser avaliado acompanhando o número de eventos (por exemplo, evento de log ou pacote de tráfego) classificados corretamente e incorretamente. Uma boa taxa

de detecção de intrusão é essencial em um IDPS. Para quantificar a qualidade do Beholder, foram adotadas duas métricas derivadas da matriz de confusão: Recall e Precision.: Recall e Precision (HINDY et al., 2018).

O Recall, ou True Positive Rate (TPR), indica a proporção de positivos reais classificados corretamente e pode ser calculado usando a seguinte equação:

$$Recall = \frac{TP}{TP + FN} \tag{5.1}$$

Precisão é o número de resultados positivos corretos dividido pelo número de resultados positivos previstos pelo classificador.

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

Onde:

- Verdadeiros Positivos (TP). O número de eventos de ataque classificados como ataque;
- Falsos Positivos (FP). O número de eventos normais classificados como ataque;
- Falsos Negativos (FN). O número de eventos de ataque classificados como normais.

## 5.4 PROJETO DA AVALIAÇÃO

O objetivo do experimento é avaliar as métricas (precisão e recall). Para a realização do experimento, diferentes tipos de ataques serão executados.

Na primeira avaliação, foi realizado um ataque sob tráfego TCP, conhecido como *SYN Flood Attack*.

Na segunda avaliação, foi realizado um ataque de negação de serviço e quebra de autenticação, explorando vulnerabilidades do protocolo MQTT.

Na terceira avaliação, foram realizados ataques de negação de serviço e injeção que exploram vulnerabilidades do protocolo CoAP.

Na quarta avaliação, foram realizados ataques de quebra de autenticação e negação de serviço contra o serviço AMQP.

Em toda a avaliação, o módulo de bloqueio (Firewall) do Beholder permaneceu desligado. Note que, se o módulo estivesse ativado, bloquearia a comunicação do atacante com a vítima

no instante da primeira detecção. Desta forma, seria impossível manter por vários segundos o experimento com esse módulo ativado. Em um cenário real, esse módulo ficaria ativado a todo momento.

Nos testes preliminares realizados, foi observado que, no caso de ataques de negação de serviço, os equipamentos travam por completo após 2 (dois) minutos de ataque. Por esta razão, para esse tipo de ataque, ficou estabelecido que serão executadas 30 rodadas com duração de 2 minutos para cada rodada. Além disso, a regras detecção foram desenvolvidas após esses testes preliminares.

Com base em tal constatação e considerando que o estudo enfoca ataques a dispositivos de baixa capacidade computacional, será adotado o tempo padrão de dois minutos de duração para cada ataque. Além disso, o consumo de memória e processamento do Beholder será avaliado através do uso de um script executando em background. O script captura memória livre, memória ocupada, o percentual de memória ocupada e o percentual de uso do processador. O consumo dos recursos da Raspberry PI 3B será avaliado com e sem a execução dos ataques.

A Tabela 6 apresenta a lista dos ataques utilizados e seus respectivos protocolos.

Tabela 6 – Descritivo dos Ataques a serem executados no experimento

| Protocolo | Ataque | Categoria do Ataque | Abordagem | Ferramenta |
|---|---|---|---|---|
| TCP | Syn Flood | Denial of Service | Disponibilidade | hping [14] |
| MQTT | QoS Flood | Denial of Service | Disponibilidade | Malaria [15] |
| | Brute Force | Authentication Broken | Confidencialidade | Mqtt-pwn [16] |
| CoAP | Error Flood | Denial of Service | Disponibilidade | Python Scapy [17] |
| | CoAP Fuzzer | Injection | Confidencialidade | cotopaxi [18] |
| AMQP | Brute Force | Authentication Broken | Confidencialidade | Brutemq [19] |
| | AMQP DDoS | Denial of Service | Disponibilidade | Peniot [20] |

**Fonte:** O Autor.

Os tipos de ataques foram escolhidos com base no documento oficial da Open Web Application Security Project (OWASP)(OWASP, 2018), que apresenta as 10 maiores vulnerabilidades da IoT. A OWASP foi escolhida por ser referência mundial em segurança de aplicações.

[14] http://www.hping.org/
[15] https://github.com/etactica/mqtt-malaria
[16] https://mqtt-pwn.readthedocs.io/en/latest/intro.html
[17] https://scapy.net/
[18] https://github.com/Samsung/cotopaxi
[19] https://github.com/codexlynx/brutemq
[20] https://github.com/yakuza8/peniot

### 5.4.1 Ataque SYN Flood

Um ataque *SYN Flood* (EDDY, 2007) é caracterizado como uma anomalia durante as etapas do estabelecimento de comunicação TCP, conhecido como *three-way handshake*. Normalmente, um cliente, ao tentar se comunicar com um servidor, envia uma solicitação SYN (syncronize). O servidor recebe essa solicitação e responde com um sinalizador SYN-ACK (confirmação de sincronização).

Por fim, o cliente envia um sinalizador ACK. Ao usar o SYN Flood, o invasor envia um grande número de solicitações de SYN, impedindo o servidor de seguir com o SYN-ACK, deixando a comunicação semiaberta, consumindo memória, processamento e largura de banda.

Para realizar a avaliação, foram enviados 10000 pacotes por segundo a cada rodada, em um total de trinta rodadas. O tempo de monitoramento por rodada foi de dois minutos. Foi utilizado esse fator, porque após testes iniciais, este ataques geram impacto negativo ao ambiente.

#### *5.4.1.1 Execução do Ataque*

Para realizar o ataque foi utilizada a ferramenta Hping. Com ela, é possível simular uma inundação TCP com flag SYN. Nesta ferramenta, o seguinte comando foi usado para executar o ataque SYN Flood: *hping3 -V -c Nsp -d Dt -S -flood vIP*, onde:

- *-c Nsp* define o número de pacotes enviados;
- *-d Dt* especifica o tamanho de cada pacote);
- *-flood* especifica que é uma inundação;
- *vIP* representa o endereço IP da vítima (alvo do ataque).

O código Java a seguir mostra a regra CEP do Beholder para detectar o ataque SYN Flood:

Código Fonte 4 – Regra CEP do Beholder para detecção do ataque SYN Flood

```
1 SELECT srcAddr FROM
      TCP_Stream.win:time(1 sec)
3 HAVING COUNT
      (syn) > 100 AND NOT (ack)
```

**Fonte:** O Autor.

A regra do Beholder afirma que, se algum dispositivo enviar mais que 100 pacotes TCP com flag SYN (HAVING COUNT (syn) $>$ 100 AND NOT (ack)), em uma janela de tempo de 1 segundo (TCP_Stream.win:time(1 sec)), o CEP identificará um possível padrão de ataque. // O código abaixo mostra a regra de detecção do Snort para detectar o mesmo ataque SYN Flood:

Código Fonte 5 – Regra do Snort para detecção do ataque SYN Flood

```
alert tcp any any -> $HOME_NET any (msg: "TCP SYN Flooding attack"; flags: S;
    threshold:type threshold, track by_src, count 5, seconds 1; sid:100170;
```

**Fonte:** O Autor.

A regra do Snort afirma que, se algum dispositivo enviar mais que 5 pacotes TCP (count 5) com flag SYN (flags: S), em uma janela de tempo de 1 segundo (seconds 1), será reconhecido um ataque.

### 5.4.2 MQTT: Ataque High QoS Flood

Esta avaliação simula o ataque High QoS Flood para verificar a precisão do Beholder. Esse ataque consiste em inundar a rede com mensagens de QoS-2. Tais mensagens requerem mais recursos computacionais do que as mensagens comuns com baixo nível de QoS (POTRINO; RANGO; SANTAMARIA, 2019).

Nesta avaliação, foram enviadas 10.000 mensagens combinadas a 100 instância de processos com o tempo de monitoramento de dois minutos, combinado com 30 rodadas.

#### *5.4.2.1 Execução do Ataque*

Para realizar o ataque, foi usada a ferramenta *malaria* [21] para simular o High QoS Flood. O comando do software *malaria* para acionar o ataque é o seguinte: *malaria publish -P Np -n Nmp –q 2 -H vIP -T t*, onde:

- *-P Nnp* define o número de instâncias de processos (Nnp) a serem executadas;
- *-n Nmp* especifica o número de mensagens (Nmp) a serem enviadas por processo ;

[21] https://github.com/etactica/mqtt-malaria

- *–q 2* declara que todas as mensagens têm uma QoS de nível 2;
- *-H vIP* representa o endereço IP da vítima (alvo do ataque);
- *-T t* indica a quantidade de mensagens *t* a serem enviadas por segundo.

O código Java a seguir mostra a regra CEP definida para detectar o ataque High QoS Flood:

Código Fonte 6 – Regra CEP de detecção do ataque *High QoS Flood*

```
1       SELECT srcAddr FROM
            MQTT_Stream.win:time(1 sec)
3       HAVING COUNT
            (messageType = 3, qos = 2) > 200;
```

**Fonte:** O Autor.

A regra afirma que, quando um dispositivo enviar mais de 200 mensagens do tipo PUBLISH (messageType = 3), com QoS nível 2 (qos = 2), em uma janela de tempo de 1 segundo (MQTT _Stream.win: time (1 s)), o Beholder reconhecerá a situação como um possível ataque *High QoS Flood*.

### 5.4.3 MQTT: Ataque de Força Bruta

O protocolo MQTT usa um broker centralizado para comunicação entre dispositivos. Este ataque realiza o módulo de força bruta de credenciais que, com um determinado conjunto de nomes de usuário e senhas, tenta autenticar no broker para encontrar credenciais válidas. Esta vulnerabilidade refere-se às categorias de autenticação e autorização, como no caso de clientes que definem seu nome de usuário como "#", contornando os mecanismos de controle de acesso e assinando todos os tópicos MQTT.

#### *5.4.3.1 Execução do Ataque*

Para realizar o ataque foi utilizada a ferramenta mqtt-pwn e seu módulo de força bruta. Nesta ferramenta, o seguinte comando foi utilizado:

Código Fonte 7 – Execução da ferramenta MQTT-PWN

```
    mqtt_pwn/python run.py
2       104.196.24.70:1883 >> bruteforce -uf usernames_file -pf passwors_file
```

**Fonte:** O Autor.

Onde:

- *104.196.24.70* Endereço lógico do broker MQTT (vítima);
- *bruteforce* comando bruteforce da ferramenta;
- *1883* porta e comunicação utilizada pelo serviço MQTT;
- *-uf* representa o parâmetro para uso de lista de usuários;
- *-pf* representa o parâmetro para uso de lista de senhas.

O código Java a seguir mostra a regra CEP do Beholder definida para detectar ataques de força bruta no MQTT:

Código Fonte 8 – Regra CEP de detecção do Bruteforce

```
        SELECT srcAddr FROM
2           MQTT_Stream.win:time(0.5 sec)
        HAVING COUNT (messageType = 2) >= 50;
```

**Fonte:** O Autor.

A regra afirma que, quando um dispositivo enviar mais de 50 mensagens do tipo CONNACK (messageType = 2), em uma janela de tempo de 0.5 segundo (MQTT _Stream.win: time (0.5 s)), o Beholder reconhecerá a situação como um possível ataque de Força Bruta.

### 5.4.4 CoAP: Ataque Error Flood

Esta avaliação simula um ataque de negação de serviço *inundação de mensagens Client Error 4.xx* para avaliar a precisão do Beholder. Esse ataque consiste em inundar a rede com esse tipo de ataque de negação de serviço, enviando uma resposta CoAP do tipo Client Error 4.xx ou Server Error 5.xx, em vez de uma resposta CoAP positiva (do tipo Success 2.xx). Essa

inundação de mensagens diminui consideravelmente o desempenho do CoAP ou pode levar a indisponibilidade do serviço CoAP (SHELBY; STUREK; FRANK, 2010).

Neste ataque, foram enviadas 100 mensagens por segundo, com 100 mensagens por loop.

#### *5.4.4.1 Execução do Ataque*

Para realizar o ataque foi desenvolvido um script na linguagem Python usando a biblioteca Scapy [22], descrito a seguir:

Código Fonte 9 – Script Python - Client Error Flood

```
#! /usr/bin/env python
import sys
from scapy.all import *
load_contrib('coap')

while True:
    p = IP(dst="vIP", id=1111,ttl=99)/UDP(sport=RandShort(), dport=5683)/
        CoAP(code="4.00 Bad Request")
send(p,inter=t, count=Sm)';
```

**Fonte:** O Autor.

Onde:

- *dst="vIP"* representa o endereço IP da vítima (alvo do ataque);
- *UDP(sport=RandShort())* representa um porta de origem randômica;
- *dport=5683* representa uma porta de origem padrão do protocolo CoAP;
- *code="4.00 Bad Request"* representa o código da mensagem de Erro 4.xx;
- *p,inter=t* indica que as mensagens devem ser enviadas por milisegundos;
- *count=Sm* representa quantidade de mensagens enviadas.

O código Java a seguir mostra a regra CEP definida para detectar o ataque *Client Error Flood* (SHELBY; STUREK; FRANK, 2010):

[22] https://scapy.readthedocs.io/en/latest/index.html

Código Fonte 10 – Regra CEP de detecção do ataque *Client Error Flood*

```
SELECT srcAddr, count(code) FROM
    CoAP_Stream(code = 128)#time(1 sec)
HAVING COUNT
    (code) >= 50);
```

**Fonte:** O Autor.

A regra afirma que, quando um dispositivo enviar um número igual ou maior que 50 mensagens (code) >= 50), com o campo code (code = 128), em uma janela de tempo de 1 segundo (CoAP_Stream(code = 128)time(1 sec)), o Beholder reconhecerá a situação como um ataque *Client Error Flood*.

### 5.4.5 CoAP: Ataque Fuzzer

Este ataque se concentra em compreender e descobrir as maneiras pelas quais um nó CoAP pode ser comprometido. Um atacante pode utilizar técnicas de difusão (fuzzing) para descobrir vulnerabilidades nas implementações CoAP disponíveis. O protocolo CoAP torna-se um possível vetor de ataque para injeção fraudulenta de pacotes e a possibilidade de uma entidade maliciosa comprometer um nó para realizar movimentos laterais (MOSENIA; JHA, 2017).

#### *5.4.5.1 Execução do Ataque*

Para realizar o ataque, foi utilizada a ferramenta mqtt-pwn e seu módulo de força bruta. Nesta ferramenta, o seguinte comando foi utilizado: client_proto_fuzzer.py –server-ip SERVER_IP –server-port SERVER_PORT –protocol CoAP –verbose, onde:

- *client_proto_fuzzer.py* Execução do script de injeção de códigos;
- *–server-ip* representa o endereço da vítima;
- *–server-port* representa uma porta do serviço CoAP;
- *–protocol* representa o protocolo a ser utilizado;
- *–verbose* indica o uso do modo verbose.

O código Java a seguir mostra a regra CEP definida para detectar o ataque CoAP Fuzzing:

Código Fonte 11 – Regra CEP de detecção do ataque *Fuzzing*

```
SELECT srcAddr FROM
    CoAP_Stream(code =0, type = 3)#time(1 sec)
HAVING COUNT
    (code = 0, type = 3) >= 50;
```

**Fonte:** O Autor.

A regra afirma que, quando um dispositivo enviar um número igual ou maior que 50 mensagens (code) $>= 50$), com o campo code (code $= 0$) e type $= 3$, em uma janela de tempo de 1 segundo (CoAP_Stream(code $= 128$)time(1 sec)), o Beholder reconhecerá a situação como um possível ataque de Fuzzing.

### 5.4.6 AMQP: Denial of Service

Neste ataque é realizada uma inundação de requisições e solicitações AMQP em um intervalo de tempo em milisegundos. O atacante pode causar indisponibilidade nos serviços e aplicações AMQP. Além disso, Nebbione (NEBBIONE; CALZAROSSA, 2020) alerta que mensagens de comandos de desligamento expostos, podem deixar o ambiente vulnerável a ataques de negação de serviço.

#### *5.4.6.1 Execução do Ataque*

Para realizar este ataque, foi utilizada a ferramenta Peniot, através de seu script que gera uma negação de serviço em aplicações AMQP. Nesta ferramenta, os seguintes parâmetros foram utilizados no script *AMQP Protocol - DoS Attack Module*.

Código Fonte 12 – Script da ferramenta Peniot AMQP Denial of Service

```
host = "137.184.40.250"
queue = "peniot-queue"
exchange = "peniot-exchange"
routing_key = "peniot-routing-key"
body = "peniot-body"
exchange_type = "direct"
timeout = 0.01
```

**Fonte:** O Autor.

Onde:

- *host* representa o endereço IP da vítima (alvo do ataque);
- *queue* representa a fila de mensagem;
- *exchange* representa uma rota para uma fila de mensagens;
- *routing_key* representa a carga útil para determinar quem receberá uma cópia de sua mensagem;
- *body* representa o corpo da mensagem;
- *exchange_type* representa o tipo de exchange como direct.
- *timeout* indica o tempo de conexão.

O código Java a seguir mostra a regra CEP definida para detectar o ataque de negação de Serviço no AMQP:

Código Fonte 13 – Regra CEP de detecção do ataque *AMQP Denial of Service*

```
1       SELECT srcAddr FROM
            AMQP_Stream.win:time(0.5 sec)
3   HAVING COUNT
        (method = 20, classc = 60) > 10
```

**Fonte:** O Autor.

A regra afirma que, quando um dispositivo enviar um número igual ou maior que 10 mensagens (method = 20) com o campo classc (classc = 60), em uma janela de tempo de 0.5 segundo (AMQP_Stream.win:time(0.5 sec)), o Beholder reconhecerá a situação como um possível ataque de negação de serviço contra aplicações AMQP.

### 5.4.7 AMQP: Bruteforce

Este ataque executa uma tentativa de descoberta de senha, carregando várias senhas de um arquivo com várias combinações de letras, números e caracteres especiais (Wordlist). O ataque pode ser realizado manipulando a quantidade de threads por URL de conexão. Com uma configuração incorreta, embora muito comum, o uso de credenciais de login padrão podem ser exploradas por um invasor (NEBBIONE; CALZAROSSA, 2020).

#### *5.4.7.1 Execução do Ataque*

Para realizar o ataque, a ferramenta Peniot foi utilizada através de seu script para negação de serviço. Nesta ferramenta, o seguinte comando foi utilizado: brutemq amqp -d passwords.txt -u admin -e 137.184.40.250:5672/ -t 500, onde:

- *amqp* Execução do comando principal;
- *-d* representa o parâmetro para o uso de uma wordlist de senhas;
- *-u* representa o parâmetro para o uso de uma wordlist de usuários;
- *-e* representa o parâmetro para inserir o endereço da vitíma e porta de destino;
- *-t* indica o envio de 500 mensagens por segundo.

O código Java a seguir mostra a regra CEP definida para detectar o ataque AMQP Bruteforce:

Código Fonte 14 – Regra CEP de detecção do ataque *AMQP Bruteforce*

```
        SELECT srcAddr FROM
2           AMQP_Stream.win:time(0.5 sec)
    HAVING COUNT
4       (method = 10, classc = 10) > 20
```

**Fonte:** O Autor.

A regra afirma que, quando um dispositivo enviar um número igual ou maior que 20 mensagens (method = 10), com o campo classc (classc = 10) em uma janela de tempo de 0.5 segundo (AMQP_Stream.win:time(0.5 sec)), o Beholder reconhecerá a situação como um possível ataque de força bruta contra aplicações AMQP.

### 5.4.8 Avaliação de Consumo de Memória RAM e CPU

Neste experimento, também foi utilizado um script na linguagem Python (Código Fonte 14) para coletar e avaliar os resultados do consumo da memória RAM real e processamento (CPU) em cenários com e sem a execução do Beholder.

Código Fonte 15 – Monitor de CPU e Memória RAM

```
#! /usr/bin/env python
import psutil
import pandas as pd
import time

from datetime import datetime
def monitor(csv='details.csv'):
  try:
    info=[]
    while True:
      all_dic=dict(psutil.virtual_memory()._asdict())
      all_dic['cpu_percent']=psutil.cpu_percent()
      try:
        all_dic['temperature']=psutil.sensors_temperatures()
      except:
        pass
      all_dic['time']=datetime.now().strftime("%H:%M:%S")
      info.append(all_dic)
      time.sleep(1)
  except:
    df=pd.DataFrame(info)
    df=df.set_index('time')
    df.to_csv(csv)
    print(f'saved in {csv}')
if __name__ == "__main__":
    monitor()
```

**Fonte:** O Autor.

## 5.5 RESULTADOS

Esta seção apresenta os resultados das execuções no cenário proposto. Inicialmente, é feita uma comparação entre o Beholder e o Snort na detecção do ataque SYN Flood. Em seguida, é apresentado o resultado da precisão na detecção do ataque de negação de serviço (*High QoS Flood*) e para quebra de autenticação (*Brute Force*) no protocolo MQTT. Posteriormente, é apresentado o resultado de detecção do ataque de negação, que envia uma inundação de mensagens do tipo *Cliente Error e Fuzzing* no protocolo CoAP. Por fim, é apresentado o resultado de detecção dos ataques de negação de serviço e *Brute Force* contra o protocolo AMQP.

### 5.5.1 Resultados da Detecção do Ataque SYN Flood (TCP)

Comparando o Beholder e o Snort (Tabela 7), os resultados demonstraram que ambos os IDPS's são equivalentes quanto à precisão de detecção, utilizando o mesmo hardware, sistema operacional e cenário.

É possível perceber que o Beholder obteve uma precisão de quatro 9's na precisão e cinco 9s no recall. Quanto ao Snort, ele obteve uma precisão de três 9s na precisão e quatro 9's para o recall. Para ambos os IDPS, foram executadas 30 rodadas, com o tempo de monitoramento igual dois minutos. Portanto, com o Beholder, é possível realizar detecção de intrusão com uma taxa de precisão equivalente ao do Snort.

Tabela 7 – Precisão na Detecção do ataque SYN Flood do Attack utilizando o Beholder e Snort

| IDPS | Precision | Recall | Tempo de Monitoramento | Rodadas |
|---|---|---|---|---|
| **Beholder** | 0,99998 | 0,99999 | 2 minutos por rodada | 30 |
| Snort | 0,99965 | 0,99996 | 2 minutos por rodada | 30 |

**Fonte:** O Autor.

### 5.5.2 Resultados no Protocolo MQTT

Os resultado na 8 monstrou que a detecção contra o ataque High QoS Flood obteve uma precisão de três 9's. Para o recall, O Beholder obteve um resultado de dois 9's. Posteriormente, no ataque de brute force, os resultados monstraram que o Beholder oferece uma precisão de 96% na precisão e três 9's no recall.

Tabela 8 – Precisão na Detecção do ataque *High QoS Flood e Brute Force*

| Ataque | Precision | Recall | Tempo de Monitoramento | Rodadas |
|---|---|---|---|---|
| MQTT: High QoS Flood | 0,99962 | 0,99674 | 2 minutos por rodada | 30 |
| MQTT: Brute Force | 0,96502 | 0,99975 | 2 minutos por rodada | 30 |

**Fonte:** O Autor.

### 5.5.3 Resultados no Protocolo CoAP

Os resultados na Tabela 9 monstraram que o Beholder consegue detectar ataques em aplicações CoAP com uma alta taxa de precisão. De fato, em todas as rodadas o Beholder conseguiu detectar o ataque *Error Flood* com taxa zero de falso-positivo. Para o ataque de Fuzzing, o Beholder obteve 97% na precisão e 95% no recall. Para ambos os ataques, foram executadas 30 rodadas de experimento, com tempo de monitoramento igual a 2 minutos.

Tabela 9 – Precisão na Detecção do Ataque *Error Flood* e *Fuzzing*

| Ataque | Precision | Recall | Tempo de Monitoramento | Rodadas |
|---|---|---|---|---|
| CoAP: Error Flood | 1,00000 | 0,99965 | 2 minutos por rodada | 30 |
| CoAP: Fuzzing | 0,97823 | 0,95436 | 2 minutos por rodada | 30 |

**Fonte:** O Autor.

### 5.5.4 Resultados no Protocolo AMQP

Os resultados na Tabela 10 demonstraram que o Beholder consegue detectar ataques em aplicações AMQP com 97% na precisão e quatro 9's no Recall no ataque de negação de serviço. Para o ataque de brute force, o Beholder obteve um precisão de dois 9's e quatro 9's no recall. Para ambos os ataques, foram executadas 30 rodadas de experimento, com tempo de monitoramento igual a 2 minutos.

Tabela 10 – Precisão na Detecção do Ataque Negação de Serviço e *Brute Force*

| Ataque | Precision | Recall | Tempo de Monitoramento | Rodadas |
|---|---|---|---|---|
| AMQP: DoS | 0,97589 | 0,99463 | 2 minutos por rodada | 30 |
| AMQP: Brute Force | 0,99555 | 0,99996 | 2 minutos por rodada | 30 |

**Fonte:** O Autor.

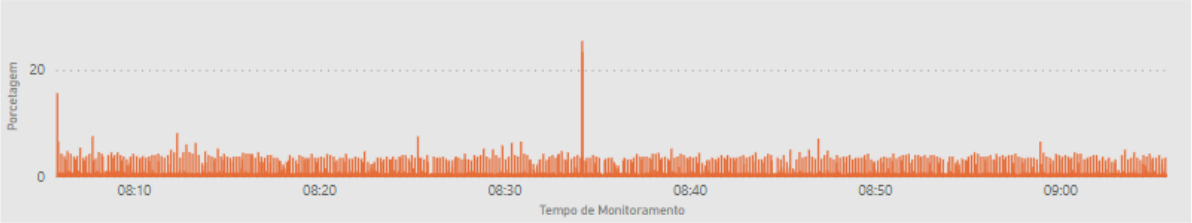
### 5.5.5 Resultados das Análises de Consumo de Memória RAM e Processamento(CPU)

Com o objetivo de analisar se o Beholder causaria impacto negativo no dispositivo Raspberry pi 3B, foi executado um script na linguagem Python em background que analisou o consumo de memória RAM e processamento (CPU), com e sem a execução do Beholder. Ambas as análises tiveram a duração de 1 (uma) hora.

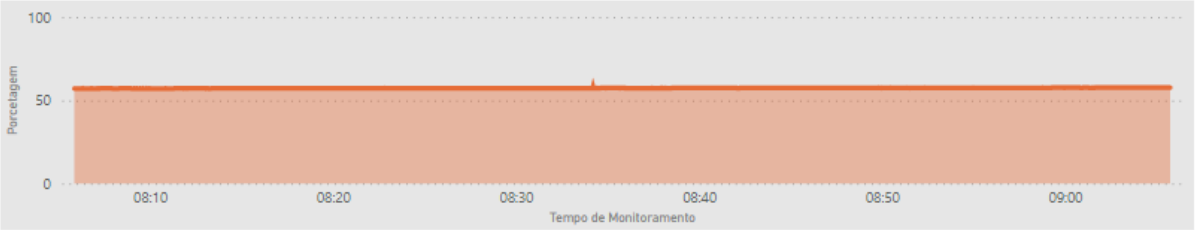
Em ambas as avaliações de consumo dos recursos computacionais, todos os dispositivos apresentados no início deste capítulo estavam conectados gerando fluxo de dados. Outro detalhe importante é que nas análises, os ataques não foram executados. A maioria dos ataques podem facilmante gerar indisponilidade, (overrall) de memória e causar instabilidade nos serviços, o que poderia inviabilizar todo o processo de análise do consumo de recursos computacionais.

A análise com a execução do Beholder ocorreu entre 8:00 de 9:00 da manhã. As Figuras 29(a) e 29(b) mostram os consumos de Memória e Processamento da Raspberry pi 3B.

Figura 28 – Consumo de Recursos Computacionais com o Beholder em execução




(a) Consumo de CPU em % com a execução do Beholder



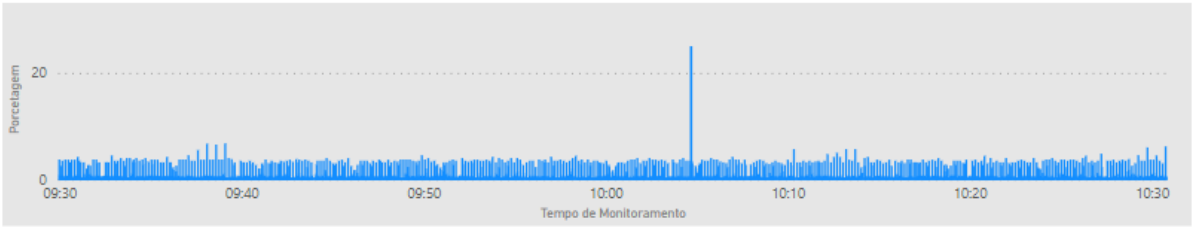

(b) Consumo de Memória com o Beholder em execução

**Fonte:** O Autor.

Na Figura 29(a), é possível identificar um aumento momentâneo no uso do processamento no início da análise. Acredita-se que esse aumento seja devido a execução do Beholder. Nota-se que, após esse evento, o consumo de processamento passa a ficar estabilizado.Já a Figura 29(b), apresenta um consumo de memória RAM moderado e estável durante a execução. Os resultados mostraram o baixo consumo de recurso computacional do Beholder em execução com uma média 0,8% para o processamento e 57,5% para a memória RAM. Na segunda análise foi executado o script em Python, sem a execução do Beholder durante 1(uma) hora (9:30 às 10:30), Figuras 30(a) e 30(b).

Os resultados apresentados nas Figuras 30(a) e 30(b) evidenciam que o consumo de processamento de 0,6% e 52,9% para a memória RAM. Comparando as duas análises, é possível identificar que o consumo de processamento (CPU) durante a execução do Beholder elevou a taxa percentual em apenas 0,2% e 4,6% no consumo de memória RAM.

Figura 29 – Consumo de Recursos Computacionais sem a execução do Beholder




(a) Consumo de CPU em % sem Utilização do Beholder


(b) Consumo de Memória em % sem Utilização do Beholder

**Fonte:** O Autor.

Por fim, os resultados apresentaram que a execução do Beholder não gerou impacto negativo significativo no consumo de recursos computacionais. Acredita-se que o uso dos recursos computacionais pelo Beholder são satisfatórios ao se considerar os benefícios advindos por detecção de ataques.

## 5.6 CONSIDERAÇÕES FINAIS

A primeira avaliação permitiu estabelecer uma equivalência na taxa de precisão de detecção de intrusão entre o Beholder e o Snort. Nessa análise, o Beholder mostrou-se ser um mecanismo robusto para detecção de ataques no ambiente de *smart homes*.

A segunda etapa da avaliação utilizou ataque de negação de serviço que afeta o campo QoS da mensagem MQTT. Os resultados apresentaram uma taxa de detecção com precisão de três 9's para o ataque High QoS Flood.

Na terceira etapa de avaliação, o Beholder obteve êxito ao analisar os fluxos CoAP e detectou um ataque numa aplicação que utiliza e envia mensagens a partir de um sensor de temperatura e umidade. O resultado foi uma taxa de 100% de precisão para o ataque Error Flood e três 9's para o recall.

Na quarta etapa, o Beholder obteve uma satisfatória taxa de detecção. Para o ataque de negação de serviço, foi obtido a precisão de 97% e dois 9's para o recall, dois 9's na precisão no

ataque de brute force na aplicação AMQP. Na quinta etapa, foi executado um script em Python que possibilitou avaliar o consumo de memória RAM e processamento (CPU). Os resultados apresentaram que o Beholder não gerou impacto significativo no consumo de memória RAM e CPU durante a sua execução.

Por fim, conclui-se que o Beholder apresenta uma precisão equivalente a IDPS populares como o Snort, já que o mesmo atua fortemente na camada de rede e transporte. Porém, avança ao obter êxito na detecção de ataques com foco na camada de aplicação IoT. Os resultados enfatizam que a CEP engine consegue obter êxito nas detecções e geração de alertas eficientes, proporcionando assim novas oportunidades para detecções de outros ataques a outros protocolos da camada de aplicação IoT.

# 6 CONCLUSÃO

Este capítulo traz uma síntese desta tese, apresentando como os objetivos propostos foram atingidos e quais as contribuições desta pesquisa para a área de segurança da IoT. Em seguida, são apontadas as limitações encontradas ao longo do desenvolvimento do trabalho. Na seção seguinte, é apresentada a publicação de artigo em periódico que esta tese proporcionou. Finalmente, os trabalhos futuros são discutidos, assim como novas linhas de investigação.

Como visto nos capítulos iniciais, a IoT vem criando novas oportunidades de negócios nas empresas e melhorias na vida humana. Entretando, diversos desafios para melhorar a segurança dos aplicativos IoT estão surgindo.

Estudos mostram que protocolos da camada de aplicação possuem vulnerabilidades que podem ser exploradas potencialmente por criminosos. Além disso, a falta de padronização, prioridade na segurança e falta de tratamento por seus designers aumentam as ameaças e riscos de ataques.

## 6.1 CONTRIBUIÇÕES

Esta tese representa um avanço para melhorar a segurança dos ambientes de aplicações IoT, permitindo a análise, em tempo real, dos quadros de mensagens que estão na carga de dados dos fluxos AMQP, CoAP e MQTT. Além disso, permite ainda a categorização desses fluxos e detecção de ataques que exploram vulnerabilidades os protocolos da camada de aplicação, que são protocolos abertos também amplamente usados no nível de aplicação dos sistemas de IoT.

Esta tese explorou a tecnologia de processamento de eventos complexos (CEP) para processar mensagens trocadas entre dispositivos IoT, afim de identificar padrões que representam um ataque cibernético. O sistema de detecção de intrusões proposto, denominado Beholder, foi descrito em detalhes, incluindo sua arquitetura, componentes de software e as funcionalidades.

A principal contribuição deste trabalho é a detecção de ataques que explorem vulnerabilidades nos protocolos de aplicação da IoT. A detecção de um ataque ocorre quando o Beholder Tracer consegue identificar qual protocolo da camada de aplicação pertence a mensagem enviada e, consequentemente, ele realiza a leitura da carga de dados do cabeçalho fixo. Após a leitura, são criados eventos que correspondem ao tipo de protocolo da camada de aplicação da IoT. Por sua vez, os eventos são analisados através de filtros que correspondem às regras CEP.

Foram realizadas três avaliações para analisar a precisão do Beholder na detecção de ataques aos protocolos TCP, AMQP, CoAP e MQTT em um cenário real, representado por uma *smart home*. Na primeira avaliação, comparou-se o Beholder com o IDS Snort para verificar se a precisão do Beholder é compatível com os IDPS mais utilizados na atualidade. A demais avaliações tiveram o objetivo de analisar a precisão do Beholder para detecção de ataques aos protocolos AMQP, CoAP e MQTT.

Os resultados se mostraram promissores. A precisão e recall entre o Beholder o Snort são equivalentes para o cenário avaliado. Ademais, a solução é capaz de identificar os ataques de negação de serviço e quebra de autenticação, que exploram vulnerabilidades do protocolo AMQP, ataque de negação de serviço e injeção com técnica *Fuzzing*, que exploram vulnerabilidades do protocolo CoAP e *High QoS Flood* e *Brute Force*, que exploram as vulnerabilidades do protocolo MQTT, *Error Flood* para o protocolo CoAP com alto grau de precisão. Além disso, a execução do Beholder apresentou uma baixa taxa de consumo de memória RAM e CPU.

De acordo com o mapeamento sistemático realizado neste trabalho, este é o primeiro trabalho que usa CEP para detectar ataques que exploram o protocolo da camada de aplicação e que torna promissor e possível a extensão para detectar ataques outros protocolos de diferentes camadas.

## 6.2 LIMITAÇÕES

O trabalho apresenta o potencial da solução adotada, contudo, mostra também pontos de melhorias e limitações. A lista abaixo enumera o conjunto delas.

- Testar o Beholder em outros cenários da IoT. Para isso, faz-se necessário testá-lo em outros ambientes como indústrias, carros inteligentes, área da saúde ou até mesmo na nuvem IoT.
- Desenvolver regras de detecção para outros ataques. Embora nesta tese tenham sido detectados sete tipos diferentes de ataques, é valido aumentar o range de detecções para outros tipos de ataques.

## 6.3 PUBLICAÇÃO

Esta tese proporcionou a publicação de um artigo em periódico com importante fator de impacto. A publicação ocorreu no final primeiro semestre de 2022, conforme o título do artigo e descrição abaixo sobre a revista.

- Beholder – A CEP-based intrusion detection and prevention systems for IoT environments. Journal - COSE | Computers & Security, ISSN 0167-4048.

## 6.4 TRABALHOS FUTUROS

Como trabalhos futuros, pretende-se:

- propor novas regras de CEP para detectar outros tipos de ataques em aplicações AMQP, CoAP e MQTT;
- realizar detecção de ataques que explorem vulnerabilidade nos protocolos da camada de aplicação: DDS (Data Distribution Service), XMPP (Extensible Messaging and Presence Protocol).
- detectar ataques em outros protocolos de descoberta de serviço, como mDNS e SSDP. O Multicast Domain Name System (mDNS) é um protocolo aberto amplamente utilizado atualmente para descoberta de serviços e resolução de nomes em links locais. Este protocolo, aliado ao DNS-based Service Discovery, oferece a flexibilidade exigida por ambientes onde é necessário integrar automaticamente novos dispositivos e realizar operações do tipo DNS sem a presença de um servidor DNS convencional. Ao contrário dos protocolos de mensagens (AMQP, CoAP, MQTT), o protocolo mDNS não fornece nenhum serviço de segurança integrado. O Simple Service Discovery Protocol (SSDP) é um protocolo aberto amplamente utilizado atualmente para descoberta de serviços e propaganda em redes residenciais ou de pequenas empresas (UPnP Forum).

  No que diz respeito à segurança, assim como o mDNS, o protocolo SSDP tem pontos vulneráveis porque não fornece nenhum mecanismo embutido. Portanto, vários riscos de segurança afetam os dispositivos habilitados para SSDP.
- propor o desenvolvimento de um módulo de Inteligência Artificial de autoaprendizagem, entendendo os padrões normais de vida do ambiente IoT. Além disso, analisar milhares

de métricas para revelar desvios sutis que podem sinalizar uma ameaça em evolução, também conhecido como técnicas desconhecidas.

- propor o desenvolvimento de uma interface por meio de uma interface de usuário (UI) para que o usuário crie regras e padrões de forma intuitiva, facilitando especificamente a usabilidade do Beholder.
- propor a execução de experimentos em outros ambientes IoT, como indústrias, carros inteligentes, saúde, agro, construção cívil, entre outros.

# REFERÊNCIAS

6120, R. *Extensible Messaging and Presence Protocol (XMPP)*. 2011. Disponível em: <https://www.rfc-editor.org/info/rfc6120>.

ABDELOUAHAB, Z. An Intrusion Detection System for Denial of Service Attack Detection in Internet of Things. *ICC '17 Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing*, 2017. Disponível em: <https://dl.acm.org/citation.cfm?id=3018896.3018962>.

AKASIADIS, C.; PITSILIS, V.; SPYROPOULOS, C. D. A multi-protocol iot platform based on open-source frameworks. *Sensors*, v. 19, n. 19, 2019. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/19/19/4217>.

ALDEEN, Y. A. A. S.; SALLEH, M. Privacy preserving data utility mining architecture. In: *Smart Cities Cybersecurity and Privacy*. [S.l.]: Elsevier, 2019. p. 253–268.

ALDOAIES, B. H. Exploitation of the Promising Technology : Using BlockChain to Enhance the Security of IoT . *2018 21st Saudi Computer Society National Computer Conference (NCC)*, IEEE, p. 1–6, 2018.

ALLOUCH, S. A.; AMECHNOUE, K.; ACHATBI, I. Iot middleware architecture for a collaborative supply chain. n. February, 2017.

ANHEMBI. *Metodologia de Pesquisa Científica - O Método Hipotético-Dedutivo*. 2015. Disponível em: <http://www2.anhembi.br/html/ead01/metodologia_pesq_cientifica_80/lu04/lo4/index.htm>.

APPROACH, I.-i.; FORTINO, G.; SAVAGLIO, C.; PUGA, J. S. D.; GHANZA, M.; PAPRZYCKI, M.; MONTESINOS, M.; LIOTTA, A.; LLOP, M. *Integration, Interconnection, and Interoperability of IoT Systems*. [s.n.], 2018. ISBN 978-3-319-61299-7. Disponível em: <http://link.springer.com/10.1007/978-3-319-61300-0>.

ARA, T.; SHAH, P. G.; PRABHAKAR, M. Internet of Things Architecture and Applications: A Survey. *Indian Journal of Science and Technology*, v. 9, n. 45, 2016. ISSN 0974-5645. Disponível em: <http://www.indjst.org/index.php/indjst/article/view/106507>.

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: A survey. *Computer Networks*, Elsevier B.V., v. 54, n. 15, p. 2787–2805, 2010. ISSN 13891286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2010.05.010>.

BAPTISTA, G. L. B.; RORIZ, M.; VASCONCELOS, R.; OLIVIERI, B.; VASCONCELOS, I.; ENDLER, M. On-line Detection of Collective Mobility Patterns through Distributed Complex Event Processing. *Monografias em Ciência da Computação*, PUC-Rio de Janeiro, v. 13, n. DECEMBER, 2013.

BOSTANI, H.; SHEIKHAN, M. Hybrid of anomaly-based and specification-based IDS for Internet of Things using unsupervised OPF based on MapReduce approach. *Computer Communications*, Elsevier B.V., v. 98, p. 52–71, 2017. ISSN 01403664. Disponível em: <http://dx.doi.org/10.1016/j.comcom.2016.12.001>.

CARDOSO, A. M.; LOPES, R. F.; TELES, A. S.; MAGALHAES, F. B. V. Real-Time DDoS Detection Based on Complex Event Processing for IoT. *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, p. 273–274, 2018. Disponível em: <https://ieeexplore.ieee.org/document/8366997/>.

CHEN, J.; CHEN, C. Design of complex event-processing IDS in internet of things. *Proceedings - 2014 6th International Conference on Measuring Technology and Mechatronics Automation, ICMTMA 2014*, p. 226–229, 2014.

CIKLABAKKAL, E.; DONMEZ, A.; ERDEMIR, M.; SUREN, E.; YILMAZ, M. K.; ANGIN, P. ARTEMIS: An intrusion detection system for mqtt attacks in internet of things. *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, IEEE, p. 369–371, 2019. ISSN 10609857.

CISCO. *Snort*. 2020. Disponível em: <https://snort.org/>.

COLLINA, M.; BARTOLUCCI, M.; VANELLI-CORALLI, A.; CORAZZA, G. E. Internet of things application layer protocol analysis over error and delay prone links. In: IEEE. *Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC), 2014 7th*. [S.l.], 2014. p. 398–404.

CORSARO, A. *What is right messaging/data- sharing standard for the IoT?* [S.l.], 2020. Disponível em: <https://www.adlinktech.com/en/index>.

CQL, O. *Oracle CEP CQL Language Reference*. 2016. Disponível em: <https://docs.oracle.com/cd/E16764_01/doc.1111/e12048/intro.htm>.

EDDY, W. *TCP SYN Flooding Attacks and Common Mitigations*. 2007. Disponível em: <https://tools.ietf.org/html/rfc4987>.

EELES, P. *What is a software architecture?* 2006. Disponível em: <https://www.ibm.com/developerworks/rational/library/feb06/eeles/index.html>.

ESPER. *Esper EPL - Event Processing Language*. 2016. Disponível em: <http://www.espertech.com/esper/release-5.2.0/esper-reference/html/epl_clauses.html>.

ESPERTECH. *Complex Event Processing Streaming Analytics*. 2019. Disponível em: <http://www.espertech.com/>.

ETZION, O.; NIBLETT, P.; LUCKHAM, D. *Event Processing in Action*. 2011.

FADILPAŠIĆ, S. Smart home devices are being hit with millions of attacks. 2019. Disponível em: <https://www.itproportal.com/news/smart-home-devices-are-being-hit-with-millions-of-attacks/>.

FOUNDATION, I. S. *The Meltdown and Spectre Brouhaha*. 2018. Disponível em: <https://www.iotsecurityfoundation.org/the-meltdown-and-spectre-brouhaha/>.

GARTNER. *IoT Security Primer: Challenges and Emerging Practices*. 2020. 1-23 p.

GARTNER. *It's Time for a New IoT Strategy to Drive New Business Opportunities*. 2022. Disponível em: <https://www.gartner.com/en/webinars/4014211/it-s-time-for-a-new-iot-strategy-to-drive-new-business-opportunities>.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, Elsevier, v. 29, n. 7, p. 1645–1660, 2013.

HADDADI, M.; BEGHDAD, R. DoS-DDoS: Taxonomies of Attacks, Countermeasures , and Well-Know Defense Mechanisms in Cloud Environment. *Edpacs*, v. 57, n. 5, p. 1–26, 2018. ISSN 19361009.

HARIPRIYA, A. P.; KULOTHUNGAN, K. Secure-MQTT: an efficient fuzzy logic-based approach to detect DoS attack in MQTT protocol for internet of things. *Eurasip Journal on Wireless Communications and Networking*, EURASIP Journal on Wireless Communications and Networking, v. 2019, n. 1, 2019. ISSN 16871499.

HESSEL, F. P. Internet of Things (IoT) in 5G Mobile Technologies. v. 8, n. April, 2016. ISSN 03772217. Disponível em: <http://link.springer.com/10.1007/978-3-319-30913-2>.

HINDY, H.; BROSSET, D.; BAYNE, E.; SEEAM, A.; TACHTATZIS, C.; ATKINSON, R.; BELLEKENS, X. A Taxonomy and Survey of Intrusion Detection System Design Techniques, Network Threats and Datasets. v. 1, n. 1, 2018. Disponível em: <http://arxiv.org/abs/1806.03517>.

HIVEMQ. *Introducing the MQTT Protocol - MQTT Essentials*. 2019. Disponível em: <https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/>.

HUSNAIN, M.; HAYAT, K.; CAMBIASO, E.; FAYYAZ, U. U.; MONGELLI, M.; AKRAM, H.; ABBAS, S. G.; SHAH, G. A. Preventing mqtt vulnerabilities using iot-enabled intrusion detection system. *Sensors*, v. 22, n. 2, 2022. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/22/2/567>.

INCORPORATING, O. S. *MQTT 3.1.1*. 2019. Disponível em: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>.

IS.40.31. *Bíblia Sagrada Online*. 2023. Disponível em: <https://www.bibliaon.com/versiculo/isaias_40_31/>.

JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. [S.l.]: John Wiley & Sons, 1990.

JNETPCAP. *jNetPcap v2*. 2022. Disponível em: <https://www.jnetpcap.com/>.

JUNIOR, D. M.; RODRIGUES, W.; GAMA, K.; SURUAGY, J. A.; Da Goncalves, P. A. S. Towards a multilayer strategy against attacks on iot environments. *Proceedings - 2019 IEEE/ACM 1st International Workshop on Software Engineering Research and Practices for the Internet of Things, SERP4IoT 2019*, p. 17–20, 2019.

KASINATHAN, P.; PASTRONE, C.; SPIRITO, M. A.; VINKOVITS, M. Denial-of-Service detection in 6LoWPAN based Internet of Things. *International Conference on Wireless and Mobile Computing, Networking and Communications*, p. 600–607, 2013. ISSN 21619646.

KHAN, R.; KHAN, S. U.; ZAHEER, R.; KHAN, S. Future internet: The internet of things architecture, possible applications and key challenges. *Proceedings - 10th International Conference on Frontiers of Information Technology, FIT 2012*, n. December, p. 257–260, 2012. ISSN 1556-3669.

KITCHENHAM, B.; PRETORIUS, R.; BUDGEN, D.; BRERETON, O. P.; TURNER, M.; NIAZI, M.; LINKMAN, S. Systematic literature reviews in software engineering-A tertiary study. *Information and Software Technology*, Elsevier B.V., v. 52, n. 8, p. 792–805, 2010. ISSN 09505849.

KRIMMLING, J.; PETER, S. Integration and evaluation of intrusion detection for CoAP in smart city applications. *2014 IEEE Conference on Communications and Network Security, CNS 2014*, p. 73–78, 2014.

KRUCHTEN, P. Architecture blueprintsthe 4+1 view model of software architecture. *Tutorial Proceedings on TRI-Ada 1991: Ada's Role in Global Markets: Solutions for a Changing Complex World, TRI-Ada 1995*, v. 12, n. November, p. 540–555, 1995.

LARMO, A.; RATILAINEN, A.; SAARINEN, J. Impact of coap and mqtt on nb-iot system performance. *Sensors*, v. 19, n. 1, 2019. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/19/1/7>.

LI, S.; XU, L. D.; ZHAO, S. Internet of Things: A survey. *Journal of Industrial Information Integration*, v. 10, n. April 2014, p. 1–9, 2015. ISSN 2452414X.

LUCKHAM, D. *The power of events: An introduction to complex event processing in distributed enterprise systems*. [S.l.]: Springer, 2008.

MAGGI, F.; VOSSELER, R.; QUARTA, D. The Fragility of Industrial IoT's Data Backbone Security and Privacy Issues in MQTT and CoAP Protocols. 2018. Disponível em: <https://documents.trendmicro.com/assets/white{_}papers/wp-the-fragility-of-industrial-IoTs-data-backbone.pdf>.

MARTON, A. *SAM: More than 1 Billion IoT Attacks in 2021*. 2022. Disponível em: <https://iotac.eu/sam-more-than-1-billion-iot-attacks-in-2021>.

MIDI, D.; RULLO, A.; MUDGERIKAR, A.; BERTINO, E. Kalis - A System for Knowledge-Driven Adaptable Intrusion Detection for the Internet of Things. *Proceedings - International Conference on Distributed Computing Systems*, p. 656–666, 2017. ISSN 1063-6927.

MOSENIA, A.; JHA, N. K. A comprehensive study of security of internet-of-things. *IEEE Transactions on Emerging Topics in Computing*, v. 5, n. 4, p. 586–602, 2017.

NEBBIONE, G.; CALZAROSSA, M. C. Security of IoT application layer protocols: Challenges and findings. *Future Internet*, v. 12, n. 3, p. 1–20, 2020. ISSN 19995903.

NEWMAN, I.; BENZ, C. R. *Qualitative-quantitative research methodology: Exploring the interactive continuum*. [S.l.]: SIU Press, 1998.

OASIS. OASIS Advanced Message Queuing Protocol OASIS Standard. *OASIS Standard*, n. October, p. 124, 2012. Disponível em: <http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html>.

OMG. *OMG Data Distribution Service (DDS)*. 2015. Disponível em: <https://www.omg.org/spec/DDS/1.4/About-DDS/>.

OWASP. *OWASP CSRFGuard Project*. 2018. Disponível em: <TheOWASPFoundation>.

PAMUKOV, M. E.; POULKOV, V. K.; SHTEREV, V. A. Negative Selection and Neural Network based Algorithm for Intrusion Detection in IoT. *2018 41st International Conference on Telecommunications and Signal Processing (TSP)*, IEEE, p. 1–5, 2018.

PARSONS, D.; RASHID, A.; SPECK, A.; TELEA, A. A "framework"for object oriented frameworks design. In: *Proceedings Technology of Object-Oriented Languages and Systems. TOOLS 29 (Cat. No.PR00275)*. [S.l.]: IEEE, 1999. p. 141–151.

PATIERNO, P. *AMQP Essentials. The Binary Transfer Protocol for Enterprise Applications and IoT*. 2018. Disponível em: <https://dzone.com/refcardz/amqp-essentials?chapter=1>.

PATIL, H. K.; CHEN, T. M. Chapter 18 - wireless sensor network security: The internet of things. In: VACCA, J. R. (Ed.). *Computer and Information Security Handbook (Third Edition)*. Third edition. Boston: Morgan Kaufmann, 2017. p. 317–337. ISBN 978-0-12-803843-7. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780128038437000181>.

PEKAR, A.; MOCNEJ, J.; SEAH, W. K. G.; ZOLOTOVA, I. Application domain-based overview of iot network traffic characteristics. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 53, n. 4, jul 2020. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3399669>.

POTRINO, G.; RANGO, F. de; SANTAMARIA, A. F. Modeling and evaluation of a new iot security system for mitigating dos attacks to the mqtt broker. In: *2019 IEEE Wireless Communications and Networking Conference (WCNC)*. [S.l.: s.n.], 2019. p. 1–6. ISSN 1525-3511.

RASPBERRY. *Raspberry Pi Model 3B*. 2020. Disponível em: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.

RAZA, S.; WALLGREN, L.; VOIGT, T. SVELTE: Real-time intrusion detection in the Internet of Things. *Ad Hoc Networks*, Elsevier B.V., v. 11, n. 8, p. 2661–2674, 2014. ISSN 15708705. Disponível em: <http://dx.doi.org/10.1016/j.adhoc.2013.04.014>.

REPCEK, S. *Distributed Complex Event Processing for Network Security*. 83 p. Tese (Doutorado) — Masaryk University, 2016.

RIZVI, S.; KURTZ, A.; RIZVI, M. Securing the Internet of Things ( IoT ): A Security Taxonomy and Dashboard. *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, IEEE, p. 163–168, 2016.

ROLDÁN, J.; BOUBETA-PUIG, J.; Luis Martínez, J.; ORTIZ, G. Integrating complex event processing and machine learning: An intelligent architecture for detecting IoT security attacks. *Expert Systems with Applications*, v. 149, 2020. ISSN 09574174.

SCARFONE, K.; MELL, P. Guide to Intrusion Detection and Prevention Systems ( IDPS ) Recommendations of the National Institute of Standards and Technology. *Nist Special Publication*, v. 800-94, p. 127, 2007. Disponível em: <http://www.reference.com/go/http://csrc.ncsl.nist.gov/publications/nistpubs/800-94/SP800-94.pdf>.

SCARFONE, K.; MELL, P. *Guide to Intrusion Detection and Prevention Systems (IDPS) (Draft) Recommendations of the National Institute of Standards and Technology*. 2012. 1-112 p. Disponível em: <https://csrc.nist.gov/CSRC/media/Publications/sp/800-94/rev-1/draft/documents/draft_sp800-94-rev1.pdf>.

SCHMIDT, D.; BUSCHMANN, F. Patterns, frameworks, and middleware: their synergistic relationships. In: *25th International Conference on Software Engineering, 2003. Proceedings.* [S.l.: s.n.], 2003. p. 694–704.

SCHMIDT, R. F. Chapter 3 - software architecture. In: SCHMIDT, R. F. (Ed.). *Software Engineering*. Boston: Morgan Kaufmann, 2013. p. 43 – 54. ISBN 978-0-12-407768-3. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9780124077683000033>.

SETHI, P.; SARANGI, S. R. Cross Layer Design in Internet of Things: Issues and Possible Solutions. *Journal of Electrical and Computer Engineering*, v. 2017, n. December, p. 1–25, 2017. ISSN 2090-0147. Disponível em: <https://www.hindawi.com/journals/jece/2017/9324035/>.

SFORZIN, A.; MARMOL, F. G.; CONTI, M.; BOHLI, J. M. RPiDS: Raspberry Pi IDS - A Fruitful Intrusion Detection System for IoT. *Proceedings - 13th IEEE International Conference on Ubiquitous Intelligence and Computing, 13th IEEE International Conference on Advanced and Trusted Computing, 16th IEEE International Conference on Scalable Computing and Communications, IEEE Internationa*, p. 440–448, 2017.

SHEIKHAN, M.; BOSTANI, H. A hybrid Intrusion Detection System for Internet of Things. *8th International Symposium on Telecommunications*, n. 3, p. 2395–4396, 2016. Disponível em: <www.ijariie.com>.

SHELBY K. HARTKE, C. B. Z. *The Constrained Application Protocol (CoAP)*. 2014. Disponível em: <https://tools.ietf.org/html/rfc7252{\#}page>.

SHELBY, Z.; STUREK, D.; FRANK, B. Constrained Application Protocol (CoAP). *Orgiddraftietfcorecoap01 Txt 0807*, n. January, p. 1–81, 2010. Disponível em: <http://tools.ietf.org/html/draft-ietf-core-coap-08>.

SHERASIYA, T.; UPADHYAY, H. a hybrid Intrusion Detection System for Internet of Things. *8th International Symposium on Telecommunications*, n. 3, p. 2395–4396, 2016. Disponível em: <www.ijariie.com>.

SHREENIVAS, D.; RAZA, S.; VOIGT, T. Intrusion Detection in the RPL-connected 6LoWPAN Networks. *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security - IoTPTS '17*, p. 31–38, 2017. Disponível em: <http://dl.acm.org/citation.cfm?doid=3055245.3055252>.

SICARI, S.; RIZZARDI, A.; MIORANDI, D.; COEN-PORISINI, A. REATO: REActing TO Denial of Service attacks in the Internet of Things. *Computer Networks*, Elsevier B.V., v. 137, p. 37–48, 2018. ISSN 13891286. Disponível em: <https://doi.org/10.1016/j.comnet.2018.03.020>.

SOMMERVILLE, I.; ARAKAKI, R.; MELNIKOFF, S. S. S. *Engenharia de software*. [S.l.]: Pearson Prentice Hall, 2008.

STANDARDS, N. I. of; TECHNOLOGY. *NATIONAL VULNERABILITY DATABASE*. 2022.

SURENDAR, M.; UMAMAKESWARI, A. InDReS: An Intrusion Detection and response system for Internet of Things with 6LoWPAN. *Proceedings of the 2016 IEEE International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2016*, p. 1903–1908, 2016.

SURICATA. *Suricata IDS*. 2020. Disponível em: <https://suricata-ids.org/>.

TERROSO-SAENZ, F.; GONZáLEZ-VIDAL, A.; RAMALLO-GONZáLEZ, A. P.; SKARMETA, A. F. An open iot platform for the management and analysis of energy data. *Future Generation Computer Systems*, v. 92, p. 1066–1079, 2019. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X17304181>.

THINGBOARD. ThingsBoard Open-source IoT Platform. 2020. Disponível em: <https://thingsboard.io/>.

TOFAN, D.; GALSTER, M.; AVGERIOU, P. Difficulty of architectural decisions – a survey with professional architects. In: . [S.l.: s.n.], 2013. p. 192–199.

ULLAH, I.; SHAH, M. A. A novel model for preserving location privacy in internet of things. In: IEEE. *2016 22nd International conference on automation and computing (ICAC)*. [S.l.], 2016. p. 542–547.

WANG, B.; LU, K.; CHANG, P. Design and implementation of linux firewall based on the frame of netfilter/iptable. In: *2016 11th International Conference on Computer Science Education (ICCSE)*. [S.l.: s.n.], 2016. p. 949–953.

WANG, W.; POOLE, B.; MITRA, A.; FALK, S.; FANTUZZI, G.; LUCIA, S.; SCHRIER, R. Role of Leptin Deficiency in Early Acute Renal Failure during Endotoxemia in ob/ob Mice. *Journal of the American Society of Nephrology*, v. 15, n. 3, p. 645–649, 2010. ISSN 10466673.

XU, L. D.; HE, W.; LI, S. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, v. 10, n. 4, p. 2233–2243, 2014. ISSN 15513203.

YASSEIN, M. B.; SHATNAWI, M. Q.; AL-ZOUBI, D. Application layer protocols for the internet of things: A survey. In: *2016 International Conference on Engineering MIS (ICEMIS)*. [S.l.: s.n.], 2016. p. 1–4. ISSN null.

ZARPELÃO, B. B.; MIANI, R. S.; KAWAKANI, C. T.; ALVARENGA, S. C. de. A survey of intrusion detection in Internet of Things. *Journal of Network and Computer Applications*, Elsevier Ltd, v. 84, n. September 2016, p. 25–37, 2017. ISSN 10958592. Disponível em: <http://dx.doi.org/10.1016/j.jnca.2017.02.009>.

ZHOU, C.; ZHANG, X. Toward the internet of things application and management: A practical approach. In: IEEE. *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on*. [S.l.], 2014. p. 1–6.

ZHOU, C.; ZHANG, X. Toward the internet of things application and management: A practical approach. *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, WoWMoM 2014*, 2014.

ZHOU, W.; JIA, Y.; YAO, Y.; ZHU, L.; GUAN, L.; MAO, Y.; LIU, P.; ZHANG, Y. Discovering and understanding the security hazards in the interactions between iot devices, mobile apps, and clouds on smart home platforms. In: *Proceedings of the 28th USENIX Conference on Security Symposium*. USA: USENIX Association, 2019. (SEC'19), p. 1133–1150. ISBN 9781939133069.

ZHU, Q.; WANG, R.; CHEN, Q.; LIU, Y.; QIN, W. IOT gateway: Bridging wireless sensor networks into Internet of Things. *Proceedings - IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, EUC 2010*, p. 347–352, 2010. ISSN 1527-9995.