PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOG E EDGE COMPUTING: UMA ARQUITETURA HÍBRIDA EM UM AMBIENTE DE INTERNET DAS COISAS

MATHEUS CRESPI SCHENFELD

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Fabiano Passuelo Hessel
Co-Orientador: Prof. Dr. Leonardo Albernaz Amaral

Porto Alegre
2017

Ficha Catalográfica

S324f Schenfeld, Matheus Crespi

Fog e Edge Computing : Uma Arquitetura Híbrida em um Ambiente de Internet das Coisas / Matheus Crespi Schenfeld . – 2017.
91 f.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Fabiano Passuelo Hessel.
Co-orientador: Prof. Dr. Leonardo Albernaz Amaral.

1. Internet das Coisas. 2. Fog computing. 3. Edge computing. 4. Middleware. 5. Cloud computing. I. Hessel, Fabiano Passuelo. II. Amaral, Leonardo Albernaz. III. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS com os dados fornecidos pelo(a) autor(a).

MATHEUS CRESPI SCHENFELD

FOG E EDGE COMPUTING: UMA ARQUITETURA HÍBRIDA EM UM AMBIENTE DE INTERNET DAS COISAS

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Ciência da Computação do Programa de Pós-Graduação em Ciencia da Computação, Faculdade de Informática da Pontifícia Universidade Católica do Rio Grande do Sul.

Aprovado em 23 de Março de 2017.

BANCA EXAMINADORA:

Prof. Dr. César Augusto Missio Marcon (PPGCC/PUCRS)

Prof. Dr. Jorge Luis Victória Barbosa (UNISINOS)

Prof. Dr. Fabiano Passuelo Hessel (PPGCC/PUCRS - Orientador)

DEDICATÓRIA

Dedico este trabalho aos meus pais por terem desde a infância me motivado a estudar, obter bons resultados e sempre ter o interesse em buscar mais conhecimento.

AGRADECIMENTOS

Gostaria de agradecer primeiro aos meus pais, Marcos e Claudia, por terem sempre me guiado com os bons exemplos. Meu pai Marcos, desde a infância sempre dedicou seu esforço para que eu conseguisse ter a melhor educação, além de me proporcionar diversas novas experiências em viagens pelo mundo. Minha mãe Cláudia por ter sempre me motivado a estudar, buscar os melhores resultados, me dedicar e ter interesse na busca de novos conhecimentos. Sou imensamente grato por tudo que vocês me proporcionaram durante a vida, todo o investimento e dedicação em uma boa educação baseada sempre nos valores da família.

Gostaria também de agradecer ao meu orientador Dr. Fabiano Passuelo Hessel pela oportunidade de cursar mestrado na PUCRS, confiando no meu trabalho como aluno e pesquisador. Também ao meu Co-orientador Dr. Leonardo Albernaz Amaral por todas as diversas reuniões e todo o suporte dado na produção de artigos e etapas do mestrado. Além disso aos meus amigos e colegas no Grupo de Sistemas Embarcados, Me. Everton de Matos e Me. Ramão Tiago Tiburski, por toda a ajuda durante os dois anos do mestrado, no suporte a realização de diversas tarefas, reuniões, além de toda a ajuda na pesquisa e implementação.

Gostaria de fazer também um agradecimento aos amigos Frederico, Mário e Pedro, representando todas as pessoas as quais conheci na mudança para Porto Alegre que estiveram sempre juntos nos momentos de descontração, compartilhando dos diversos sentimentos de se cursar um mestrado. Meu amigo Matheus Rubert por ter produzido as imagens e vídeos da estufa agrícola, contribuindo fortemente para o desenvolvimento de um bom estudo de caso, além de todos os meus outros amigos que participaram de alguma forma no deselvolvimento do trabalho.

Gostaria também de agradecer a Lorriny Cardoso por ter me acompanhado na etapa final da graduação e durante todo o mestrado, compartilhando dos variados sentimentos e frustrações do mestrado, além de sempre me motivar a estudar, ter perseverança dando o melhor no desenvolvimento de todos os meus projetos.

Por fim, mas não menos importante gostaria de agradecer meu orientador na graduação Dr. Marcelo Trindade Rebonatto, por ter me motivado durante o desenvolvimento do TCC, onde despertei o interesse pela pesquisa científica.

FOG E EDGE COMPUTING: UMA ARQUITETURA HÍBRIDA EM UM AMBIENTE DE INTERNET DAS COISAS

RESUMO

Internet das Coisas (IoT) é considerada uma evolução computacional que preconiza a existência de uma grande quantidade de objetos físicos embarcados com sensores e atuadores, conectados por redes sem fio e que se comunicam através da Internet. Desde o surgimento do conceito até os dias atuais, a IoT é amplamente utilizada nos diversos setores da indústria e também no meio acadêmico. Uma das necessidades encontradas nessas áreas foi a de estar conectado com dispositivos ou subsistemas de IoT espalhados por todo o mundo.

Assim, *cloud computing* ganha espaço nesses cenários, onde existe a necessidade de estar conectado e se comunicando com um *middleware* para realizar o processamento dos dados dos dispositivos. O conceito de *cloud computing* refere-se ao uso de memória, armazenamento e processamento de recursos compartilhados, interligados pela Internet. No entanto, aplicações IoT sensíveis à latência de comunicação, tais como, aplicações médico-emergenciais, aplicações militares, aplicações de segurança crítica, entre outras, são inviáveis com o uso de *cloud computing*, visto que para a execução de todos os cálculos e ações é necessária a troca de mensagens entre dispositivos e nuvem.

Solucionando essa limitação encontrada na utilização de *cloud computing*, surge o conceito de *fog computing*, cuja ideia principal é criar uma camada federada de processamento ainda na rede local dos dispositivos de computação das extremidades da rede. Além de *fog computing* também surge *edge computing* operando diretamente na camada dos dispositivos, realizando algum tipo de processamento, mesmo que de pouca complexidade computacional, a fim de diminuir ainda mais o volume de comunicação, além de colaborar para prover autonomia na tomada de decisões ainda na camada das coisas. Um grande desafio tanto para *fog* quanto para *edge computing* dentro do cenário de IoT é a definição de uma arquitetura de sistema que possa ser usada em diferentes domínios de aplicação, como saúde, cidades inteligentes entre outros.

Esse trabalho apresenta uma arquitetura de sistema para dispositivos IoT capaz de habilitar o processamento de dados nos próprios dispositivos ou o mais próximo deles, criando a camada

de *edge* e *fog computing* que podem ser aplicadas em diferentes domínios, melhorando a Qualidade dos Serviços (QoS) e autonomia na tomada de decisão, mesmo se os dispositivos estiverem temporariamente desconectados da rede (offline). A validação dessa arquitetura foi feita dentro de dois cenários de aplicação, um de iluminação pública em ambiente de IoT e outro simulando uma estufa agrícola inteligente. Os principais objetivos das execuções dos testes foram verificar se a utilização dos conceitos de *edge* e *fog computing* melhoram a eficiência do sistema em comparação com arquiteturas tradicionais de IoT. Os testes revelaram resultados satisfatórios, melhorando os tempos de conexão, processamento e entrega das informações às aplicações, redução do volume de comunicação entre dispositivos e *core middleware*, além de melhorar a segurança nas comunicações. Também é apresentada uma revisão de trabalhos relacionados tanto no meio acadêmico como no da indústria.

Palavras Chave: Internet of Things, Fog Computing, Edge Computing, Middleware, Cloud Computing.

FOG AND EDGE COMPUTING: A HYBRID ARCHITECTURE IN INTERNET OF THINGS ENVIRONMENT

ABSTRACT

Internet of Things (IoT) is considered a computational evolution that advocates the existence of a large number of physical objects embedded with sensors and actuators, connected by wireless networks and communicating through the Internet. From the beginning of the concept to the present day, IoT is widely used in the various sectors of industry and also in academia. One of the needs encountered in these areas was to be connected to IoT devices or subsystems throughout the world.

Thus, cloud computing gains space in these scenarios where there is a need to be connected and communicating with a middleware to perform the data processing of the devices. The concept of cloud computing refers to the use of memory, storage and processing of shared resources, interconnected by the Internet. However, IoT applications sensitive to communication latency, such as medical emergency applications, military applications, critical security applications, among others, are not feasible with the use of cloud computing, since for the execution of all calculations and actions messaging between devices and the cloud is required.

Solving this limitation found in the use of cloud computing, the concept of fog computing arises and whose main idea is to create a federated processing layer, still in the local network of the computing devices of the ends of the network. In addition to fog computing, there is also edge computing operating directly on the devices layer, performing some kind of processing, even with little computational complexity, in order to further decrease the volume of communication, besides collaborating to provide autonomy in decision making yet in the Things layer. A major challenge for both fog and edge computing within the IoT scenario is the definition of a system architecture that can be used in different application domains, such as health, smart cities and others.

This work presents a system architecture for IoT devices capable of enabling data processing in the devices themselves or the closest to them, creating the edge computing layer and fog computing layer that can be applied in different domains, improving Quality of Services (QoS) and autonomy in decision making, even if the devices are temporarily disconnected from the network (offline). The

validation of this architecture was done within two application scenarios, one of public lighting in smart city environment and another simulating an intelligent agricultural greenhouse. The main objectives of the tests were to verify if the use of the concepts of edge and fog computing improve system efficiency compared to traditional IoT architectures. The tests revealed satisfactory results, improving connection times, processing and delivery of information to applications, reducing the volume of communication between devices and core middleware, and improving communications security. It also presents a review of related work in both academia and industry.

Keywords: Internet of Things, Fog Computing, Edge Computing, Middleware, Cloud Computing.

LISTA DE SIGLAS

COMPAAS – Cooperative Middleware Platform as a Service

FAAS4IOT – Fog-as-a-Service for Internet of Things

IOT – Internet of Things

QOS – Qualidade do Serviço

CEP – Processamento de eventos complexos

M2M – Machine-to-Machine

PAAS – Platform-as-a-Service

RFID – Radio-Frequency IDentification

SAAS – Software-as-a-Service

SOA – Service Oriented Architecture

GEI – Generic Enabler Implementation

WOT – Web of Things

XML – eXtensible Markup Language

MQTT – Message Queuing Telemetry Transport

TLS – Transport Layer Security

SSL – Secure Sockets Layer

HSQLDB – Hyper Structured Query Language Database

DTLS – Datagram Transport Layer Security

UDP – User Datagram Protocol

TCP – Transmission Control Protocol

JSON – JavaScript Object Notation

ESB – Enterprise Service Bus

WSDL – Web Services Description Language

UDDI – Universal Description, Discovery and Integration

OTA – Over the Air

IOS – Internet dos Serviços

IOP – Internet de Pessoas

HTTP – Hypertext Transfer Protocol

JS – JavaScript

XMPP – Extensible Messaging and Presence Protocol

GPS – Global Positioning System

REST – Representational State Transfer

SOAP – Simple Object Access Protocol

API – Application Programming Interface

SDK – Software development kit

BCI – Brain-Computer Interfaces

CARS – Cloud-assisted remote sensing

SANS – Redes de Sensores e Atuadores

SLA – Service Level Agreement

IDE – integrated development environment

WS4D-GSOAP – Web Services for Devices based on gSOAP development toolkit

SOC – System-on-a-Chip

COAP – Constrained Application Protocol

SQL – Structured Query Language

DPWS – Devices Profile for Web Services

MS – Milissegundo

HD – Hard Drive

KWH – Quilowatt-hora

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

O termo Internet das Coisas (IoT) foi introduzido em 1998 [5] e definido como o paradigma de computação que permite que pessoas e coisas (dispositivos de computação) possam se conectar a qualquer hora, em qualquer lugar, com qualquer coisa e qualquer um, usando qualquer caminho de rede ou serviço [40] [21]. Conectividade é uma funcionalidade crítica que é necessária para cumprir a perspectiva gerada pela IoT. Nesse sentido, existem estatísticas atuais de mercado e previsões que mostram um rápido crescimento em implementações de dispositivos relacionados a ambientes IoT.

A visão da IoT é poder propagar suas tecnologias e implantá-las em todos os lugares e em diversas áreas de aplicação. Portanto, ela tem se tornado uma tecnologia chave para alavancar soluções relacionadas aos grandes desafios da computação do país e no mundo. Olhando para o futuro, a quantidade de dispositivos IoT tende apenas ao crescimento. Em 2020, estima-se que existirão de 50 a 100 bilhões de dispositivos conectados à Internet [52] [43] e uma das necessidades, tanto meio acadêmico quanto na indústria é a de se estar conectado com diversos dispositivos e subsistemas de IoT independente de localização geográfica.

Nesse sentido, o conceito de *cloud computing* surge da necessidade de, primeiro, colocar sistemas computacionais em ambientes altamente distribuídos e escaláveis em termos de desempenho e utilização de recursos, e, segundo, de tornar simples o ambiente de configuração e reúso dos recursos. *Cloud computing* refere-se à utilização da memória e da capacidade de armazenamento e cálculo de computadores e servidores compartilhados e interligados por meio da Internet [39]. A utilização de um *middleware* para processamento em uma plataforma de nuvem, possibilita a seleção, configuração e atuação nos dispositivos da rede, além do gerenciamento de quais dispositivos atendem as requisições dos usuários e aplicativos [40].

Entretanto, arquiteturas de *cloud computing* não são eficientes para executarem aplicações IoT que requerem uma comunicação frequente entre os dispositivos e uma resposta em tempo real. Esse tipo de arquitetura não é adequada e tem alto custo de comunicação para trafegar grandes fluxos de dados gerados por dispositivos IoT para a nuvem. Isso trouxe problemas para sistemas IoT e aplicações sensíveis a latência de comunicação, tais como, aplicações de segurança crítica, aplicações militares, sistemas médicos emergenciais, entre outros. Além disso, aplicações consumidoras de informações e dispositivos geradores de dados estão em ambientes altamente distribuídos, o que resulta na necessidade de sistemas que melhor gerenciem a alta demanda de dados, mas com uma menor latência na comunicação.

Para tentar solucionar esse problema surge então o conceito de *fog* [7] e *edge computing*[20]. *Fog computing* tem como objetivo principal distribuir serviços e funções para os dispositivos computacionais (como as coisas ou dispositivos da IoT) localizados nas extremidades ou bordas das redes. *Fog computing* tem como uma de suas principais características, criar uma camada federada, ou seja, virtual, que tem como finalidade aproximar de forma eficiente as camadas do sistema, provedoras e consumidoras de informação, e com isso, tentar minimizar o grande volume de comunicação existente na Internet. A constante evolução nas capacidades de processamento e armazenamento,

além da diminuição no tamanho de dispositivos de computação tornam possível o *edge computing*, que realiza o processamento uma camada abaixo de *fog computing*, em uma visão vertical da arquitetura, em outras palavras, diretamente na camada de software responsável pelos sensores e atuadores. Dispositivos *edge* geralmente estão em ambiente de rede local fechada [33], trafegando dados via *Wireless, bluetooth, ZigBee*, entre outros.

Os conceitos e métodos de *fog* e *edge computing* trazem vantagens às aplicações requisitantes, pois ajudam a reduzir o volume de comunicação e a carga de trabalho nos nodos da rede que processam e produzem informação (como é o caso dos sistemas de *middleware* IoT), além de também reduzir o tráfego na rede, latência, utilização de energia, e por fim melhorar a Qualidade do Serviço (QoS) provido às aplicações e usuários da IoT. Além disso, a criação de uma camada de rede fechada é mais eficiente comunicando e distribuindo informações entre dispositivos próximos uns dos outros, onde "próximos" pode ser interpretado tanto no sentido físico, pois os dispositivos estão localizados no mesmo ambiente quanto lógico, em virtude que trafegam dados em ambiente de rede local através de protocolos rápidos de comunicação, citados anteriormente. O processamento diretamente no dispositivo auxilia na tomada de decisão de uma maneira muito mais eficiente, além de abrir espaço para o desenvolvimento de uma gama de aplicações que satisfazem o consumidor de informação mesmo que uma resposta em tempo real seja necessária [62].

Nesse trabalho é apresentado o projeto e desenvolvimento de uma arquitetura híbrida de sistema utilizando os conceitos e abordagens de *edge* e *fog computing* em dispositivos da IoT. Espera-se que com esta arquitetura, os dispositivos sejam capazes de realizar o processamento de dados neles mesmos ou o mais perto deles, permitindo uma melhor Qualidade dos Serviços (QoS) providos e mais autonomia na tomada de decisões, mesmo que os dispositivos estejam momentaneamente desconectados *(offline)*.

1.1 MOTIVAÇÃO

Os sistemas IoT vêm ganhando cada vez mais espaço em um nível global [5] e estão evoluindo constantemente para suportar demandas futuras onde existirão mais de 20 bilhões de dispositivos conectados a Internet [52]. O investimento em Cidades Inteligentes também vem ganhando destaque e cada vez mais empresas e governos têm investido em infraestrutura e tecnologia para tornar isto possível.

O objetivo das Cidades Inteligentes é impulsionar o crescimento econômico e melhorar a qualidade de vida das pessoas, permitindo o desenvolvimento local impulsionado pelas tecnologias habilitadoras da IoT. A aplicação de Soluções Inteligentes permite que as cidades usem tecnologia, informação e dados da própria cidade para melhorar a infra-estrutura e serviços [26].

A aplicabilidade deste conceito esta relacionada a diversos setores da economia e do governo, sem falar nas questões sociais, por isso, é difícil de se padronizar e serem criadas políticas públicas. As políticas públicas de fato, mudam de país para país, uma vez que dependem das carac-

terísticas do contexto urbano de acordo com os âmbitos territoriais, políticos e econômicos [8]. Na Figura 1.1, é possível visualizar seis subáreas que dividem as áreas de cidades inteligentes [44] [49], as quais estão listadas abaixo:

- **Smart Mobility:** acessibilidade, disponibilidade de tecnologias de informação e comunicação, além de sistemas de transporte modernos e sustentáveis (veículos com baixa ou sem emissão de carbono, mobilidade de acordo com a demanda, sistemas de transportes públicos sustentáveis e seguros, redes de sensores urbanos, entre outros);
- **Smart Governance:** compreende aspectos de participação política, serviços para os cidadãos, bem como o funcionamento da administração (governança transparente, estratégias políticas e perspectivas);

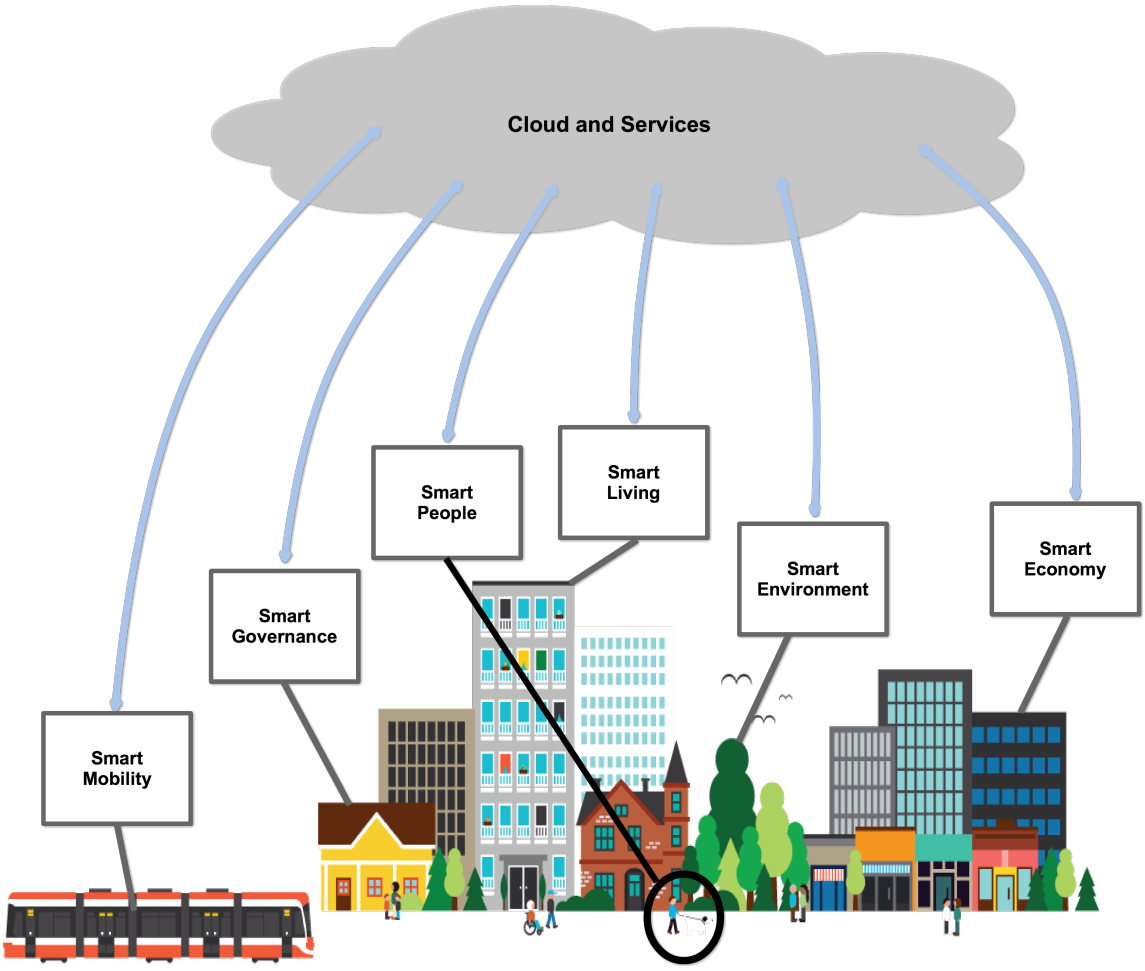

Figura 1.1 – Ambiente de Cidades Inteligentes

- **Smart People:** refere-se ao capital social e humano. É descrito pelo nível de qualificação e educação dos cidadãos e também pela qualidade das interações sociais em relação à integração, à vida pública, além da relação entre países de todo o mundo. Também o acesso às instalações culturais, atratividade turística, entre outros;

- **Smart Living:** condições de saúde, segurança individual, qualidade da habitação, instalações que possibilitam o acesso as informações sobre os recursos hidráulicos, elétricos, entre outros;
- **Smart Environment:** refere-se aos recursos naturais, controle de poluição, gestão de recursos, também por esforços para a proteção ambiental e incentivo ao uso de tecnologia para maximizar as produções;
- **Smart Economy:** competitividade econômica, inovação, empreendedorismo, produtividade e flexibilidade do mercado de trabalho, integração no mercado internacional. Desenvolvimento de espírito inovador, capacidade de transformar, eco e agro-indústria, infra-estruturas de TIC, economia alternativa, entre outros.

Dentro da subárea de *smart economy* estão as aplicações voltadas para o agronegócio, e o Brasil é um país referência mundial na agricultura. Dessa forma, pensando no desenvolvimento econômico futuro, existe a necessidade de investimentos que provejam suporte a pesquisas acadêmicas e também na área comercial, a fim de se melhorar as tecnologias existentes e criar outras, que possibilitem a maximização das produções de diversas culturas.

O crescente investimento em cidades inteligentes, bem como o constante esforço para serem criadas novas tecnologias, serviços e aplicações, que tenham como objetivo melhorar a qualidade de vida de todas as pessoas, criando um ambiente mais eficiente em termos de produção e consumo de energia, mobilidade urbana, entre outros, motiva a pesquisa de métodos que juntos facilitam e melhoram o desenvolvimento de serviços voltados para IoT e cidades inteligentes.

Apesar de já existirem diversas pesquisas em andamento, ainda existem diversas tecnologias a serem desenvolvidas ou melhoradas, tendo como justificativa as necessidades atuais e principalmente as demandas futuras. Pensando em um ambiente como o descrito na introdução, onde existirão bilhões de dispositivos IoT conectados, gerando dados e se comunicando, é fundamental o desenvolvimento de uma arquitetura de sistema com a utilização de técnicas e métodos de *fog* e *edge computing*, a fim de tornar possível o processamento já nos dispositivos ou até mesmo na própria rede que interconecta os dispositivos.

1.2 OBJETIVOS

O objetivo geral deste trabalho é desenvolver uma arquitetura de sistema para dispositivos IoT através da utilização dos conceitos de *fog* e *edge computing*. Esta arquitetura deverá habilitar o processamento de dados já nos dispositivos, além da comunicação entre eles, tentando reduzir a latência de rede, entre outros requisitos.

Os objetivos podem ser divididos em estratégicos e específicos, os quais foram executados como tarefas durante o desenvolvimento do trabalho. Os objetivos estratégicos são:

- Pesquisar e definir as ferramentas necessárias para o sistema a fim de tornar possível o desenvolvimento de uma arquitetura de *fog* e *edge computing*.

- Definir técnicas de processamento para serem utilizadas nos dispositivos.
- Definir a arquitetura para o processamento nos dispositivos.
- Definir o mecanismo e a utilização das regras que suportarão a tomada de decisões nos dispositivos.

Os objetivos específicos são:

- Desenvolver a arquitetura de Middleware, *fog computing* e *edge computing* que trabalhe de modo eficaz em um ambiente de cidades inteligentes.
- Desenvolver o mecanismo que gerencie a confluência de eventos utilizando como base as regras definidas anteriormente.
- Realizar testes e validações com a arquitetura híbrida de *edge* e *fog computing* implementada.

1.3 CONTRIBUIÇÃO

As contribuições deste trabalho são apresentadas a seguir:

- Desenvolvimento de uma arquitetura para suporte ao processamento de dados já nos dispositivos.
- Desenvolvimento do *Fog-as-a-Service for Internet of Things* (FaaS4IoT), criando uma camada virtual de comunicação e processamento mais próxima dos dispositivos, que deverá ajudar a reduzir o volume de comunicação com o *Core middleware*, além de diminuir a latência de comunicação na rede, entre outras vantagens.
- Desenvolvimento da camada de software para os dispositivos capaz de processar e administrar os dados vindos de seus sensores, além de estabelecer um canal de comunicação constante com a camada de *fog computing* FaaS4IoT. Isso é possível pois, o poder de processamento e armazenamento dos dispositivos têm aumentado cada vez mais.
- A proposta também atende a algumas das tendências futuras para Sistemas IoT [51], sendo elas: prover inteligência e autonomia aos dispositivos, arquitetura dirigida a eventos, suporte a grande volume de dispositivos e comunicação.

1.4 ORGANIZAÇÃO DO TEXTO

Os capítulos a seguir estão organizados da seguinte forma: Capítulo 2 apresenta o referencial teórico, introduzindo alguns dos termos e conceitos que serão abordados no decorrer do

trabalho, como por exemplo, IoT, *cloud computing*, *fog computing*, entre outros. O capítulo também apresenta alguns trabalhos relacionados, visando mostrar a relevância do assunto, bem como algumas limitações e tendências futuras que precisam ser solucionadas. O capítulo 3 apresenta a proposta de arquitetura explanando suas características e funções. No capítulo 4 é apresentado o cenário de aplicação dos testes, bem como sua implementação e validação, apresentando seus resultados. Por fim, o capítulo 5 apresenta a conclusão e os trabalhos futuros.

2. REFERENCIAL TEÓRICO E TRABALHOS RELACIONADOS

Nesse capítulo são brevemente apresentados alguns conceitos comumente citados no decorrer do trabalho. Em 2.1 é apresentada a definição de Internet das Coisas. Na seção 2.2 é apresentado o conceito de *cloud computing*. Logo após, a seção 2.3 apresenta a definição de *fog computing* seguindo para a seção2.4, que apresenta o conceito de *edge computing*. Por fim, a subseção2.5 apresenta uma relação entre arquiteturas híbridas em ambientes de cidades inteligentes.

2.1 Internet das Coisas

A criação da Internet revolucionou o funcionamento tradicional das sociedades modernas, e com o uso de protocolos de endereçamento, tornou-se possível a comunicação entre dispositivos separados fisicamente, rompendo diversos paradigmas da distância e do tempo. A possibilidade de quaisquer dispositivos - como, Identificação por Radio Frequência (RFID), *tags*, sensores, atuadores, *smartphones*, entre outros - conectados à rede, em qualquer lugar e a qualquer hora, iniciou a era da ubiquidade, principal característica e pilar fundamental para o surgimento da Internet das Coisas ou em inglês, *Internet of Things* (IoT) [54] [14].

O conceito da IoT é uma evolução de diversos elementos tecnológicos (sensores, *hardware*, semântica, armazenamento, processamento, comunicação, entre outros), que ao serem unidos em um mesmo ambiente, representam o futuro da computação e das comunicações [53]. A IoT tem o objetivo final de criar um mundo melhor para todas as pessoas, onde objetos a nossa volta tenham conhecimento e autonomia, agindo de forma inteligente sem instruções explícitas [40].

Uma das funcionalidades essenciais para o funcionamento de um sistema IoT é a camada de *middleware* [5]. Dentre as suas características, a principal é a de abstrair os protocolos técnicos, tais como, protocolo de comunicação, tipo de porta de comunicação, entre outras, das diferentes tecnologias encontradas nos dispositivos, tornando assim a tarefa do programador de aplicações algo mais simples. O *middleware* tem ganhado muito mais importância nos últimos anos, visto que facilita o desenvolvimento de novos serviços e a integração de tecnologias já existentes com as novas em desenvolvimento [4]. Isto tira a necessidade do programador de ter o conhecimento específico dos dispositivos, que atualmente têm variados tipos de protocolos de comunicação e funcionalidades [47].

Entretanto, a possibilidade de poder estar conectado a diversos dispositivos e independente de localização, assim como prover para o desenvolvedor de sistemas a chance de poder utilizar recursos que facilitam o ambiente de configuração, adaptação e monitoramento de serviços, motivou a utilização da tecnologia de *cloud computing* para melhorar ainda mais a IoT.

2.1.1 ARQUITETURA ORIENTADA A SERVIÇOS

As arquiteturas de *middleware* propostas e desenvolvidas nos últimos anos têm geralmente seguido a abordagem de Arquitetura Orientada a Serviços (SOA). A utilização dos princípios SOA permitem a decomposição de sistemas complexos e monolíticos em aplicações, num ambiente com componentes simples e bem definidos, porém consistentes. Isso facilita a interação entre partes de um sistema, além de permitir fácil adaptação e reutilização tanto de *software* como *hardware*, pois, não impõe a utilização de uma tecnologia especifica para a implementação de um serviço. A Figura 2.1 apresenta a arquitetura SOA para *middleware* IoT.

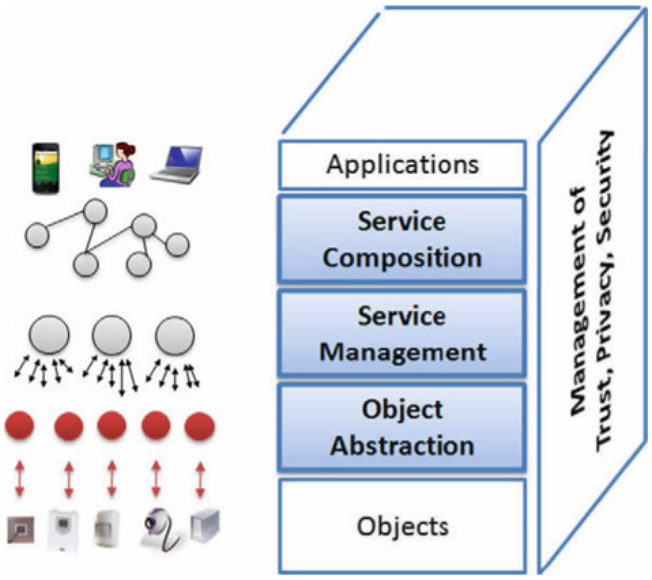

Figura 2.1 – Arquitetura orientada a serviços para Middleware IoT [5].

As funcionalidades específicas de cada camada são apresentadas a seguir:

- **Aplicação:** Camada localizada no topo da arquitetura. Exporta todas as funcionalidades do sistema para o usuário final, entretanto, não é considerada parte do *middleware*. Através dos padrões consolidados dos *Web Services* [48], consegue realizar de forma confiável a integração entre os sistemas distribuídos e as aplicações.

- **Composição de Serviços:** Esta camada esta normalmente presente nas arquiteturas de *middleware* baseadas em SOA. Provê as funcionalidades para a composição dos serviços oferecidos pelos objetos da rede para construir aplicações específicas. Nesta camada não se tem noção dos dispositivos, apenas dos serviços oferecidos por eles.

- **Gerenciamento de Serviços:** É nesta camada que são fornecidas as principais funcionalidades, que tornam possível o gerenciamento de objetos no cenário de IoT. Dentre as funcionalidades fornecidas podem ser listadas: Descoberta de dispositivos, monitoramento de status, serviços de configuração, entre outros. Algumas características de *middleware* também são

encontradas nesta camada, tais como, gerenciamento de contexto, serviços de repositórios com informações dos objetos e suas respectivas funções, entre outros.

- **Abstração de Objetos:** Tendo em vista que os dispositivos conectados a rede são os mais heterogêneos possíveis, é esperado que cada um tenha características próprias, sejam protocolos de conexões, tipos de dados disponibilizados ou linguagem de comunicação. É necessário que exista uma abstração desses protocolos, para tornar o acesso aos dispositivos mais harmônico e essa é a função específica desta camada.

- **Confiabilidade, privacidade e segurança:** A evolução das tecnologias de comunicação autônoma entre objetos representa um perigo para o nosso futuro. Dispositivos espalhados por todo ambiente, desde etiquetas *RFID, Smartphones* recolhendo informações de localização e deslocamento, ou quaisquer outros tipos de monitoramento, geram dados que ao serem processados contextualizam toda a nossa vida. O *middleware* então deve ter a capacidade de gerenciar essas informações de forma confiável, privada e sobretudo segura.

A arquitetura SOA não é uma técnica pronta para uso, mas sim um modelo, ou, em outras palavras, um padrão [62]. Entretanto, são necessárias técnicas prontas para implementar SOA, incluindo, *Web Services* e ESB.

Web Services são um dos principais possibilitadores do padrão SOA nestes últimos anos. Eles são compostos de quatro partes [60]: XML, SP, WSDL, e UDDI. O XML é usado como formato de troca de mensagens. O protocolo SP para acesso remoto de objetos, como o serviço WSDL, trabalhando como uma linguagem de descrição para o prestador de serviços, descrevendo o seu serviço e por fim, o UDDI [28] para o registro de serviço e detecção.

Enterprise Service Bus (ESB) é o componente funcional básico em SOA. Ele funciona como o corretor de serviço e é o intermediário entre o prestador de serviços e o solicitante. O ESB permite que os serviços encontrem um ao outro rapidamente, e faça chamadas de uma forma segura, assíncrona, além de também permitir a transferência de dados e notificação de eventos entre aplicações [10].

2.2 Cloud Computing

Cloud Computing ou Computação em Nuvem, é o resultado da evolução e adoção de tecnologias e paradigmas computacionais consolidados [63]. A principal tecnologia que permite a computação em nuvem é a virtualização, que separa um dispositivo físico de computação em um ou mais dispositivos virtuais, fazendo com que cada dispositivo possa ser facilmente utilizado e gerenciado para executar tarefas. Com a virtualização em nível de sistema operacional, a criação de um sistema escalável de vários dispositivos de computação independentes, além de recursos computacionais ociosos, podem ser atribuídos e utilizados de forma mais eficiente. O objetivo da

computação em nuvem é permitir que os usuários tirem benefício de todas essas tecnologias, sem a necessidade de conhecimento profundo ou experiência com cada uma delas [22].

Entretanto, mesmo com diversas vantagens, *cloud computing* tem apresentado algumas limitações [39], fazendo com que aplicações sensíveis a latência de comunicação sejam afetadas, visto que aplicações consumidoras de informações e dispositivos geradores de dados estão em ambientes distribuídos. Problemas com largura de banda e consequentemente os custos decorrentes disso também fazem parte das limitações. Para solucionar estes e outros problemas surge o conceito de *fog computing* [7].

2.3 Fog Computing

Fog Computing surge como uma infraestrutura de computação na qual aplicações e serviços podem ser tratados tanto em servidores *cloud*, como na própria rede [59]. O conceito *fog* surge visando atender a três objetivos principais:

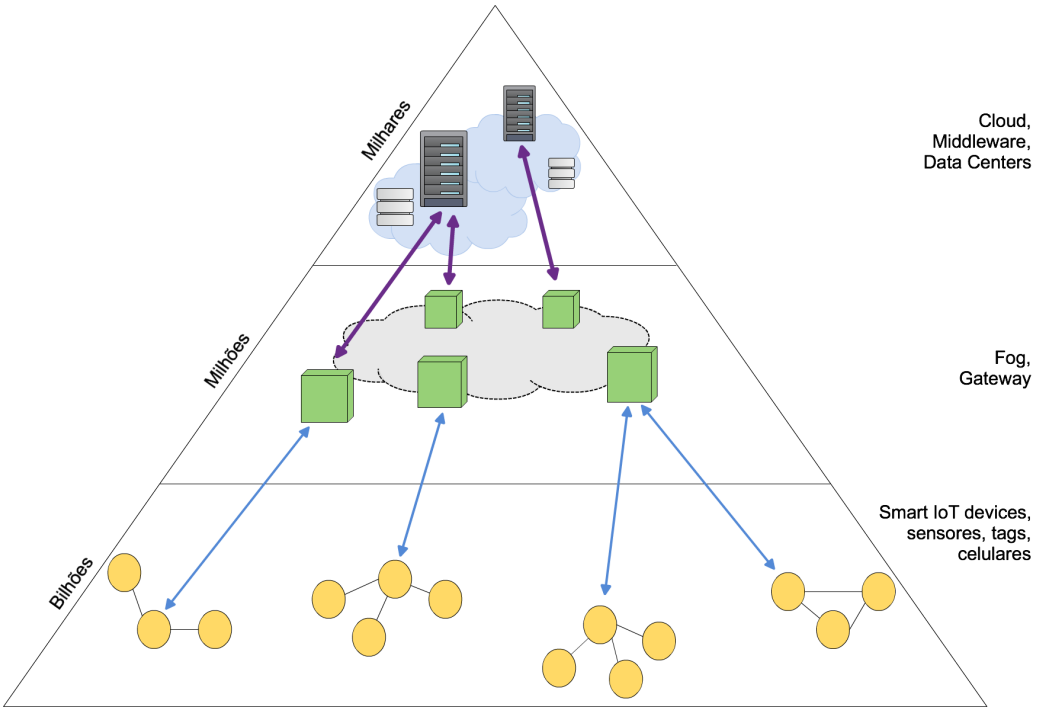

Figura 2.2 – Camada virtual de fog computing.

- Melhorar a eficiência e reduzir a quantidade de dados que precisam ser transmitidos para que seja feito o processamento, análise e armazenamento.

- Aproximar o consumidor de informações com o provedor de dados.

- Prover segurança e conformidade para a transmissão de dados.

Em *fog computing*, grande parte do processamento ocorre nos próprios dispositivos que geram os dados, tendo em vista que nos últimos anos o seu poder de processamento e armazenamento teve significativa evolução [35]. A arquitetura *fog* cria uma plataforma virtual que fornece serviços de processamento e de armazenamento entre a nuvem *cloud* e os dispositivos. Em outras palavras, cria uma camada federada, isto é, a união de diversas funcionalidades que formam uma camada virtual em ambiente descentralizado e mais próximo dos dispositivos de borda (ou *edge devices*), reduzindo a latência na rede e largura de banda, solucionando ao menos parcialmente os problemas encontrados em *cloud computing* [27]. A Figura 2.2 ilustra a camada virtual criada em *fog computing* aproximando os dispositivos e a camada *cloud computing*.

Processamento, armazenamento e os recursos de rede são componentes fundamentais para a construção de sistemas *cloud* e *fog*. Entretanto, diversas características fazem com que o desenvolvimento de uma arquitetura de *fog computing* a partir de uma plataforma de *cloud computing* não seja uma tarefa trivial [7]. Alguns dos desafios existentes estão listados abaixo:

- Conhecimento de localização e baixa latência de comunicação.
- Dispositivos localizados em ambiente altamente distribuído.
- Grande volume de dispositivos, como uma consequência da distribuição geográfica.
- Suporte a mobilidade, como por exemplo, carros conectados a rede, *smartphones, smartwatches* e etc.
- Processamento em tempo real.
- Acesso predominantemente via *Wireless*.
- Heterogeneidade. Dispositivos *fog* tendem a ser os mais variados, com diferentes tipos de dados gerados, protocolos de comunicação, entre outros.
- Interoperabilidade e virtualização. Algumas situações, como por exemplo transmissão de vídeo em tempo real, requerem a cooperação de diferentes provedores. Nesse sentido, *fog computing* precisa ser capaz de interoperar e os serviços precisam ser virtualizados entre os diferentes domínios.

2.4 Edge Computing

Edge Computing surge como uma metodologia que coloca o processamento (dados, cálculos, serviços, entre outros) fora de um ponto central (*Core Middleware*) para as extremidades da rede [9]. Funciona replicando fragmentos e informações do *middleware* através da rede para os *edge-devices*, localizados em ambiente distribuído, incluindo inclusive subsistemas IoT fora da rede principal [38].

Processamento nas camadas *edge*, reduzem significativamente o volume de comunicação entre dispositivos e o core de processamento, consequentemente o tráfego na rede, os custos decorrentes disso, latência, além de melhorar a qualidade do serviço (QoS) [9]. O *edge computing* elimina, ou pelo menos reduz a carga de trabalho do *middleware*, que por ser centralizado, torna-se um ponto único de falha. Como o volume de dados é menor, o mesmo tem a possibilidade de ser encriptado para ser trafegado na rede, o *middleware* por sua vez descriptografa os dados. Em caso do tipo de criptografia não ser reconhecido, o mesmo é descartado, dada a possibilidade de ser um ataque [27]. Com isso, a segurança do sistema como um todo também é beneficiada. Além disso, outra vantagem esta na escalabilidade do sistema, que por estar em ambiente distribuído, também é beneficiada.

O *edge computing* está relacionado com *grid computing*, onde é feita a divisão de tarefas em diversas máquinas. Na maior parte das vezes essa divisão é feita de forma manual com alteração de código fonte. O *edge computing* por sua vez, ao replicar fragmentos do código, além de informações pertinentes, cria um *template*, facilitando a distribuição das tarefas entre os dispositivos [38]. Prover autonomia aos dispositivos também se refere a *Autonomic Computing*, onde dentre as diversas definições, existem algumas seções de ambiente de processamento autônomo distribuído, incluindo:

- **Automático:** Significa ter a possibilidade de controlar a si mesmo, incluindo funções e operações.

- **Adaptativo:** Ter a possibilidade de mudar a sua operação, bem como status, dado algum evento.

- **Consciente:** Um sistema autônomo deve ser capaz de se automonitorar, além de monitorar o contexto em que está localizado, o que influencia diretamente na adaptabilidade do sistema.

2.5 FOG E EDGE COMPUTING NUM AMBIENTE DE CIDADES INTELIGENTES

Arquiteturas que têm a nuvem como elemento principal de processamento, como mostrado no lado esquerdo da Figura 2.3, não são adequadas em termos de eficiência e desempenho, para darem suporte a serviços de Internet das Coisas que requerem uma comunicação frequente entre os dispositivos e uma resposta em tempo real. Esse tipo de arquitetura não é eficiente em virtude do fluxo de dados que são gerados nos dispositivos IoT que precisam ser trafegados pela rede para serem armazenados e analisados. Esse modelo de arquitetura exige alto poder de computação na nuvem para analisar os dados para responder as aplicações do mundo real com resultados confiáveis e consistentes em tempo ineficiente para aplicações que exigem uma resposta imediata [11].

Recentemente, novos paradigmas para sistemas IoT, como *edge computing* e *fog computing*, têm atraído atenção crescente executando o processamento de dados dentro ou à margem da rede, mitigando a carga de trabalho do servidor em nuvem. Assumindo que o poder de computação

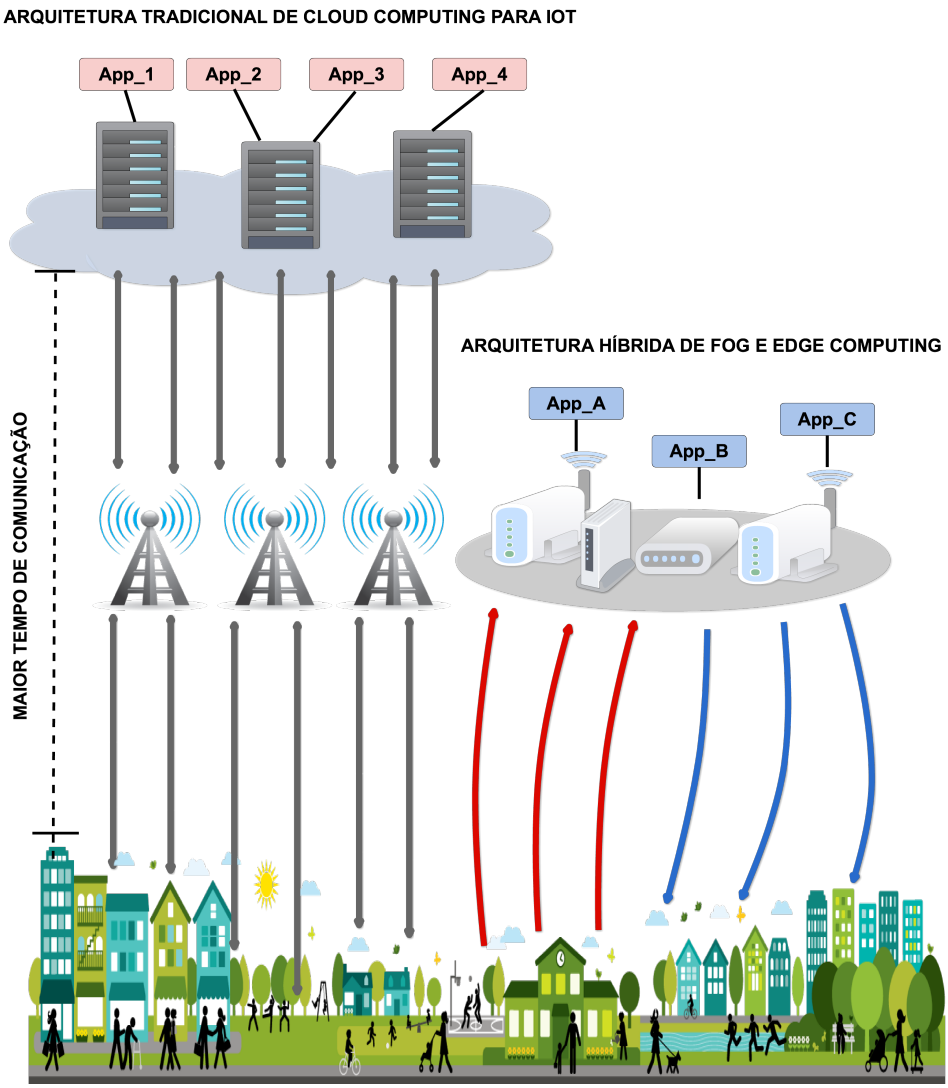

Figura 2.3 – Arquitetura de Cloud, Edge e Fog Computing.

e a capacidade de memória dos dispositivos IoT têm aumentado de ano para ano, faz com que esses dispositivos sejam capazes de processar fluxos de dados por si mesmos como mostrado no lado direito da Figura 2.3. Sendo possível delegar o processamento de tarefas de servidores *cloud* para sistemas distribuídos e independente de proximidade habilitando a execução de aplicações em tempo real em um ambiente distribuído [30].

A utilização de *fog e edge computing* dentro do cenário de cidades inteligentes, possibilita a criação de uma vasta gama de aplicações, como por exemplo, sistemas de iluminação publica, sistemas médicos-emergenciais, sistemas de segurança, sistemas de suporte ao agronegócio, entre outros [45]. As cidades inteligentes são definidas em múltiplas dimensões, como a economia, a mobilidade, as pessoas, governança, entre outros. Nestas dimensões, a tecnologia e a Internet em particular desempenham um importante papel de apoio, baseando-se em três eixos fundamentais: a Internet das Coisas (IoT), a Internet dos Serviços (IoS) e a Internet de Pessoas (IoP), onde Pessoas são parte das redes inteligentes e onipresentes que têm o potencial de se conectar, interagir e trocar informações sobre si, seu contexto social e ambiente [32]. O objetivo final é de criar uma cidade

inteligente e eficiente, provendo aos cidadãos informações sobre seus recursos, disponibilizar serviços com a intenção de facilitar a tomada de decisão com qualidade, confiabilidade e segurança.

2.5.1 COOPERATIVE MIDDLEWARE PLATAFORM AS A SERVICE (COMPaaS)

Cooperative Middleware Platform as a Service (COMPaaS) é um *middleware* IoT baseado em SOA desenvolvido pelo grupo de pesquisa GSE/PUCRS [4]. Fornece uma infra-estrutura de serviços simples e bem definida. Além desses serviços, há um conjunto de camadas de sistema que lida com os requisitos de usuários e aplicativos, por exemplo, solicitação e notificação de dados, descoberta e gerenciamento de dispositivos físicos, canais de comunicação e gerenciamento de dados.

COMPaaS é composto de dois sistemas principais: *middleware core* e *logical device*. O *middleware core* é o sistema responsável por abstrair as interações entre aplicativos e dispositivos e abstrai do desenvolvedor toda a complexidade envolvida nessas atividades. O *logical device* é o sistema responsável por ocultar toda a complexidade dos dispositivos físicos e abstrai suas funções para a camada *middleware core*.

A Figura 2.4 apresenta a arquitetura de *middleware* do COMPaaS e destaca alguns detalhes sobre a integração entre Aplicativos, *middleware core*, *logical device* e dispositivos, bem como o fluxo principal de informações. Cada camada é descrita nos itens seguintes:

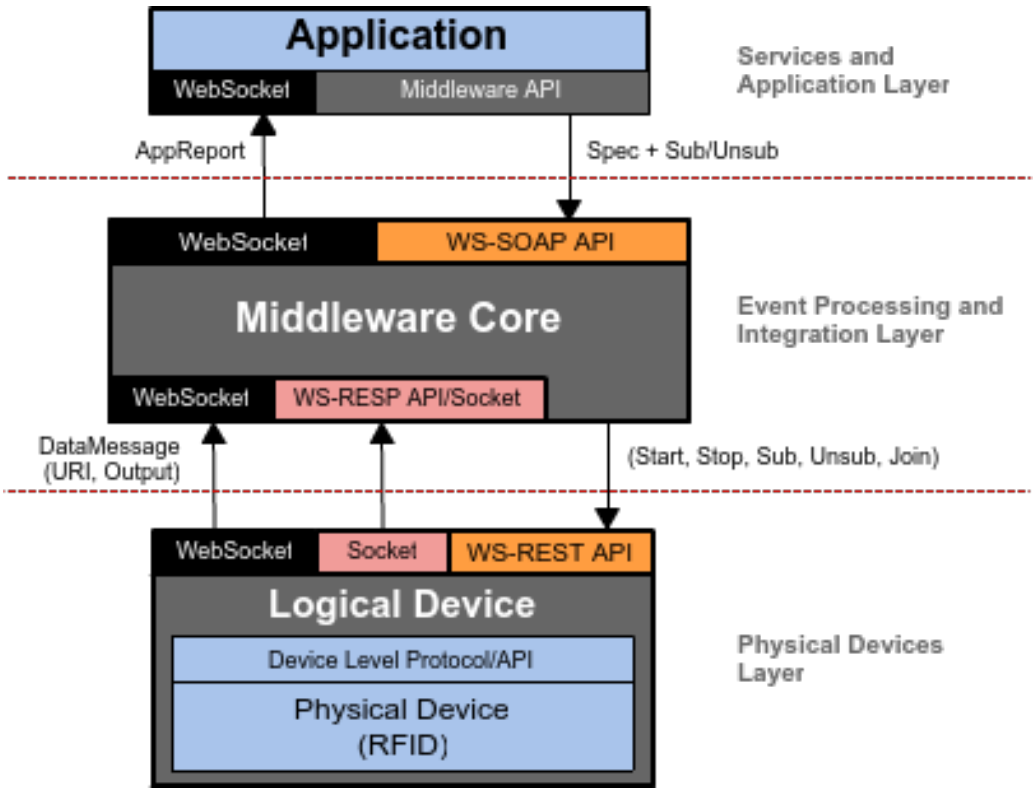

Figura 2.4 – COMPaaS Middleware

- **API:** A API é disponibilizada pela camada de *middleware* e é responsável por gerenciar o acesso aos serviços do COMPaaS. Ele deve ser incorporado nas aplicações, onde envia dados através dos serviços *web* SOAP ao COMPaaS e recebe dados através de um canal de *socket* baseado em TCP.

- **Middleware Core:** As principais funções do Middleware Core variam de gerenciamento de dados a integração de dispositivos e atendem à prestação de serviços de alto nível para aplicativos. Esta camada fornece um serviço *web* SOAP, que é usado por aplicativos. Os dispositivos podem fazer o processo de registro através de *sockets* baseados em UDP. A camada *middleware core* envia comandos para dispositivos através dos serviços *web* REST do dispositivo e recebe dados através de um canal de *socket* baseado em TCP.

- **Logical Device:** Esta camada fornece a abstração para dispositivos físicos e deve ser embarcada nos próprios dispositivos. Os *logical device* são descritos através de um arquivo XML chamado "perfil". Cada perfil de dispositivo contém atributos para caracterizar o dispositivo físico, como nome, fabricante, função, modelo, tipo de dados, URI, etc. Além do perfil, o dispositivo possui métodos que expõem suas interfaces e recursos à camada *middleware core* através de um *Web Service* REST. O *logical device* envia dados para o Middleware Core através de *sockets* baseados em TCP.

2.6 Trabalhos Relacionados

No âmbito da Internet das Coisas, tanto na área comercial, como na acadêmica, diversas plataformas de *cloud computing* podem ser utilizadas para facilitar o desenvolvimento e a implantação de sistemas e aplicações para IoT. Exemplos conhecidos dessas plataformas são: ThingWorx, OpenIoT, Xively, Nimbits, Axeda, entre outras. A Tabela2.1 resume algumas das características mais importantes destas e de outras plataformas de *cloud computing* para IoT (na tabela, "+" significa suporta e "-" significa não suporta).

As métricas de avaliação utilizadas na Tabela 1 incluem: suporte a *fog e edge computing*. Gerenciamento, que diz respeito ao suporte à descoberta de dispositivos, entrega das informações, configuração e ativação de aplicações e serviços, garantindo a autonomia. Suporte a qualidade e confiabilidade de aplicações e serviços (Veracidade). E por último, o suporte de protocolos de comunicação padronizados.

Xively[41], representa uma das primeiras plataformas para disponíveis IoT na *web*. Xively tem como objetivo conectar os dispositivos com as aplicações, garantindo a segurança em tempo real. Xively fornece uma Plataforma como Serviço (PaaS) para os desenvolvedores de aplicações IoT, além dos prestadores de serviços. Ela é capaz de integrar os dispositivos com a plataforma através do uso de bibliotecas prontas (como ARM mbed, *Eletric Imp* e iOS/OSX) e também de facilitar a comunicação via HTTP(S), ou *Sockets/WebSocket* [57]. É possível também integrar Xively com outras plataformas que utilizam bibliotecas Java, JS, Python e Ruby.

Tabela 2.1 – Plataformas Cloud IoT e Suas Características

Plataforma	Fog	Edge	Gerenciamento	Veracidade	REST	MQTT
Arkessa	-	+	-	+	+	-
Axeda	+	-	+	+	+	-
Nimbits	-	-	-	-	+	-
RealTime.io	+	-	+	-	+	-
EMPaaS	+	+	+	+	-	+
EEG	+	+	-	+	-	+
Cars	+	+	+	-	+	-
NurseSystem	+	+	+	-	-	+
Thingworx	-	+	-	+	+	-
Xively	+	+	+	+	+	-

Nimbits[15] é uma Plataforma como Serviço (PaaS) de código aberto que conecta dispositivos inteligentes com a nuvem. Ele também realiza a análise de dados na nuvem, gerando alertas, além de se conectar com redes sociais e planilhas. Além disso, ele se conecta a sites e pode armazenar, compartilhar e recuperar dados dos sensores em diversos formatos, incluindo numérico, texto, GPS, JSON ou XML. A troca de dados ou mensagens XMPP [56] é um serviço embutido no Nimbits. O núcleo do Nimbits é um servidor que fornece serviços *Web* REST para o registro e recuperação de dados brutos e processados.

Axeda [6] é uma outra plataforma para integração de dados e desenvolvimento de aplicações, bem como para conexão de dispositivo e outros serviços comumente oferecidos em serviços de *cloud computing*. Essa plataforma oferece suporte a *edge-gateway* desde que os dispositivos a serem conectados atendam a algumas especificações de *firewall*. Axeda também dá suporte ao controle de eventos em tempo real. Além disso, ela também provê ao desenvolvedor uma interface de alto nível, tornando o ambiente de configuração e reutilização de recursos facilitado. A plataforma também dispõe de serviços e aplicações com funções de gerenciamento de dados, juntamente com aplicações *web* e APIs que tem como finalidade estender a plataforma usando Java SDK.

EEG *Tractor Beam* [61] é um jogo de monitoramento do estado cerebral para *smartphones*. O jogo proposto é baseado em *Brain-Computer Interfaces* (BCI), onde o cérebro do usuário interage com o jogo através de um fone de ouvido EEG. O princípio do jogo é bastante simples. Todos os jogadores são mostrados em um círculo que cerca um objeto alvo. O objetivo de cada jogador é puxar o alvo para si concentrando-se. O *smartphone* do usuário troca as informações de jogo e os fluxos de dados brutos gerados pelos fones de ouvido através dos dispositivos da área de fog. Especificamente, a classificação contínua em tempo real do estado cerebral é realizada nos dispositivos. Tal classificação é altamente intensiva do ponto de vista computacional. Os modelos de classificação são enviados para a nuvem para fins de processamento adicionais. EEG *Tractor Beam* não fornece ferramentas e nem APIs para desenvolvimento. Para implantação e execução, baseia-se em um protocolo padrão de publicação, com assinatura *Machine-to-Machine (M2M)* e, *Message Queuing Telemetry Transport (MQTT)*, como mecanismo no transporte de dados entre servidores *cloud e fog*.

O artigo de [3] em um primeiro momento descreve um cenário médico-hospitalar, chamado de *Nursing Home Patient Monitoring System* (NurseSystem), onde em uma UTI existem diversos dispositivos conectados, tanto de monitoramento de pacientes como também de controle da sala - Luz, temperatura, etc. Esses dispositivos podem ser monitorados em um único computador ou acessados diretamente pelos dispositivos móveis de pessoas autorizadas. Posteriormente busca soluções já existentes no mercado para torna-lo possível. Para realizar o controle da sala é utilizado o *SmartThings* da Samsung, que é composto de lampadas gerenciáveis, sistemas de segurança de portas e janelas, entre outros. E para simular os equipamentos médicos utiliza do kit de desenvolvimento em Arduino, *E-Health*, composto por mais de 9 tipos de sensores de funções vitais, incluindo, monitor multiparamétrico, oximetria, ventilação pulmonar e etc. Os dados produzidos por esses sensores são publicados em um *broker* MQTT, onde os mesmos quando solicitados podem ser buscados pelo cliente que subscrever o *broker*, conectado à internet através de uma conexão LTE-A, provida pelo *gateway* da Cisco 819 M2M.

Cloud-assisted remote sensing (CARS) [1] é um sistema que permite acessar e manipular dados em tempo real a partir da rede de sensores distribuídos. O sistema consiste em quatro camadas: A camada de *fog* que fornece ao sistema recursos de detecção e atuação interligados pela Internet. A camada Stratus inclui os recursos da nuvem. Esses recursos são basicamente redes de sensores e atuadores (SANs). A camada Alto-cumulus mapeia e orquestra várias nuvens de Stratus. Finalmente, a camada Cirrus fornece as interfaces para interagir com clientes CARS. A arquitetura proposta fornece um conjunto de APIs para desenvolver aplicativos que abstraem as SANs físicas. Por outro lado, várias considerações arquitetônicas são mencionadas como desafios de pesquisa, como o design da interface e as interações entre entidades CARS. Consequentemente, as interfaces críticas necessárias para dar suporte a implementação e operação das aplicações (por exemplo, interfaces de comunicação entre *cloud e fog*) não são fornecidas. Além disso, os autores discutiram que novos algoritmos são necessários para garantir SLAs/QoS. Eles, no entanto, nem fornecer qualquer algoritmos nem soluções para lidar com isso.

A arquitetura proposta em [58] é arquitetura disponibilizada como uma plataforma como Serviço (PaaS) que tem como objetivo principal automatizar o provisionamento de aplicativos em um ambiente híbrido *cloud/fog*. O Cloud Foundry, uma PaaS de código aberto, é usado como base para sua implementação. Como um caso de uso, o PaaS proposto foi empregado para prover uma aplicação IoT baseada em componentes simples que detectam o fogo e acionam robôs para que se deslocam para combater o incêndio. Para ser estabelecida a comunicação entre *fog* e *cloud*, os princípios de REST são utilizados. Também disponibiliza ao desenvolvedor uma IDE para desenvolvimento, facilitando a utilização de kits de desenvolvimento, bibliotecas e APIs

O trabalho explanado em [2] propõe o desenvolvimento de um sistema para gerenciamento do consumo de energia em *Smart Home* (EMPaaS) e define as arquiteturas de *hardware* e *software* para tornar isso possível. A arquitetura de hardware é composta por cinco elementos: a) Roteadores, dando suporte a conectividade. b) *Gateway*, estabelecem a compatibilidade de conexão (*ZigBee, Bluetooth, Ethernet*) entre diversos dispositivos. c) Sensores que monitoram o ambiente, por exem-

plo, temporizador, luminosidade, custo da energia, entre outros. d) Atuadores. O sistema decide se atua em dispositivos de acordo com as mudanças do ambiente. Esses dispositivos gerenciáveis são considerados atuadores e podem ser controlados localmente ou remotamente. e) *Computing*, que são dispositivos que armazenam, processam e analizam os dados no sistema, além disso, devem ter a capacidade de implementar controles sofisticados para manipular os atuadores. A arquitetura de *software* por sua vez, utiliza do TinyOS, sistema operacional que exige até 8 kilobytes de memória e é flexível, principalmente na implementação de redes de sensores. Além disso, a comunicação entre os dispositivos ocorre através da utilização do DPWS, que implica em SOAP-over-UDP, SOAP, WSDL e XML. A implementação dessa parte do sistema torna-se facilitada utilizando o kit de desenvolvimento WS4D-gSOAP.

2.7 Discussão dos Trabalhos Relacionados

Como pode ser visto nos trabalhos relacionados da Tabela 1, mesmo que existam diversas soluções até mesmo comerciais, elas ainda não atendem por completo aos requisitos de uma arquitetura de *cloud* e *fog computing* explanados em Al-Fugaha et al. [3]. Em cinco plataformas, não há suporte para *edge* ou *fog*, o que limita que um conjunto de aplicações possa ser executada na plataforma, em alguns cenários de aplicações existe a necessidade de utilizar as capacidades de uma arquitetura híbrida, por exemplo, sensores e atuadores espalhados dentro de uma estufa agrícola, criando a camada de *edge computing*, comunicando-se com a central controladora do sistema, software embarcado em um SoC, criando a camada de *fog computing*.

A arquitetura híbrida de *edge* e *fog computing* desenvolvida neste trabalho (Figura 3.1) apresenta várias vantagens em comparação com trabalhos similares encontrados. Entre elas podem-se destacar:

- **Conectividade:** conexão com diversos tipos de equipamentos e dispositivos através do uso de diferentes redes de comunicação, como por exemplo, WiFi, *Ethernet*, serial, entre outras, garantindo o suporte a heterogeneidade do ambiente.
- **Protocolo de comunicação:** utilização do MQTT [31] como uma das alternativas de protocolos de aplicações também pode ser citada, tendo em vista que representa um método simplificado para a transferência de dados entre servidores e consumidores através do HTTP [17]. O MQTT é ideal para arquiteturas distribuídas em cenários de IoT, visto que pode ser executado em dispositivos com pouca capacidade de energia, processamento e armazenamento, como por exemplo o módulo ESP8266, que possui 512 kb para armazenamento em memória, 256 kb de memória RAM e um processamento de 80 *MegaHertz*.
- **Segurança na transferência de dados:** dentre os trabalhos apresentados, principalmente os trabalhos acadêmicos, deixam de lado uma das principais requisitos de sistemas IoT, não garantindo a segurança e privacidade nos dados trafegados pela rede. A não utilização de

protocolos de segurança colocam em risco todo o sistema. Grande parte dos sistemas *fog* estão em ambiente aberto e muito suscetíveis a ataques, como em postes de luz, roteadores, entre outros, sendo assim, a adoção de um protocolo confiável de segurança um requisito imprescindível em arquiteturas de IoT. A arquitetura apresentada neste trabalho tem a garantia de segurança na transferência de dados, tanto entre dispositivo e *fog*, como entre *fog* e *core middleware* através da utilização do protocolo de segurança TLS.

- **Armazenamento de dados eficiente:** Muito dos trabalhos apresentados, não dão suporte ao armazenamento de dados, inviabilizando o seu posterior processamento para produção de conteúdo. Outros trabalhos relacionados executam os testes em ambiente simulado utilizando máquinas virtuais com bancos de dados tradicionais, que em ambiente real de IoT seriam ineficientes em termos de busca, tratamento e disponibilidade dos dados. Para solucionar essa limitação será utilizado um banco de dados *lightweight* [23] que permite o armazenamento eficiente de dados em ambiente de IoT, além da possibilidade de ser executado em dispositivos com baixa capacidade de processamento.

As capacidades do COMPaaS, plataforma responsável pela camada de *middleware*, também tornam-se uma vantagem, pois, garantem a qualidade e confiabilidade do sistema. Entre as capacidades pode-se citar o suporte a descoberta, gerenciamento, configuração e adaptação, tanto de dispositivos como do contexto dos mesmos. Além disso, a utilização de *System-on-a-Chip* (SoC) com diferentes capacidades como dispositivos para embarcar o *edge-gateway* e o FaaS4IoT, torna-se um ponto positivo devido ao baixo custo e ao alto poder computacional.

Além das vantagens técnicas apresentadas, a arquitetura também está preparada para atender algumas das tendências futuras para IoT, como por exemplo, estar preparada para o tamanho da rede, visto que algumas pesquisas mostram que até 2020 existirão entre 50 e 100 bilhões de dispositivos conectados a internet [52].

3. ARQUITETURA DO SISTEMA

É proposto o planejamento e desenvolvimento de uma arquitetura híbrida com suporte para *fog computing* e *edge computing* em arquiteturas de *middleware* Orientados a Serviços (SOA). A arquitetura provê mais autonomia para os dispositivos, pois permite que os mesmos continuem executando as suas tarefas mesmo que estejam momentaneamente desconectados da rede *(Offline)*.

O objetivo é solucionar os problemas da alta taxa de transferência de dados em sistemas com arquiteturas tradicionais IoT, onde o processamento acontece majoritariamente em um ponto centralizado da rede, criando a necessidade de alta largura de banda e latência de comunicação.

A arquitetura está dividida em três partes principais e independentes umas das outras. A primeira parte é a plataforma de *middleware* utilizada como responsável por operar o topo da arquitetura, gerenciando todos os dispositivos *fog* e demais dispositivos conectados a rede, bem como os dados gerados por eles. A plataforma utilizada como base para o desenvolvimento é o *Cooperative Middleware Plataform-as-a-Service* (COMPaaS) [4], pois é uma plataforma que foi desenvolvida dentro do nosso grupo de pesquisa o que facilita a extensão e reutilização de códigos, além de ser flexível em termos de adaptabilidade em diferentes cenários de aplicação.

A segunda parte da arquitetura é a camada de *fog computing*, intitulada, *Fog-as-a-Service for Internet of Things* (FaaS4IoT). Ela é responsável por gerenciar e operar todos os sensores e atuadores de todos os dispositivos conectados a esta camada, criando uma camada virtual mais próxima dos dispositivos das bordas da rede, atuando como suporte ao processamento e armazenamento dos dados recebidos. O desenvolvimento foi feito priorizando a utilização de protocolos já padronizados tanto de segurança, quanto de comunicação, visando garantir a qualidade e confiabilidade nas operações.

Por fim, a terceira parte é a camada de *edge computing*, embarcada nos próprios dispositivos das bordas da rede que atua como API de comunicação com os sensores e atuadores do ambiente. *Edge computing* possibilita, a criação de métodos de processamento, mesmo que de baixa capacidade, nos próprios dispositivos, diminuindo ainda mais o tempo entre a produção do dado, tratamento, transporte, processamento, armazenamento, até a entrega da informação nas aplicações requisitantes. Além disso também possibilita a alta disponibilidade na execução do sistema, pois cada dispositivo é autônomo e tem conhecimento de suas funções, o que garante que ele continue em funcionamento mesmo quando desconectado da rede.

A arquitetura apresentada une as vantagens obtidas com a utilização dos conceitos de *fog e edge computing* em uma arquitetura híbrida que pode ser aplicada em diferentes domínios.

A arquitetura proposta foi submetida e aceita para publicação [46] e pode ser vista na Figura 3.1. Ela é constituída por cinco camadas: Camada de Aplicações, responsável por entregar os dados processados; Camada de *middleware*, que executa as principais funções da arquitetura; Camada de *Fog e Edge Computing*, elementos principais da pesquisa desse trabalho. Além da camada de Comunicação, responsável pelas conexões entre diferentes plataformas.

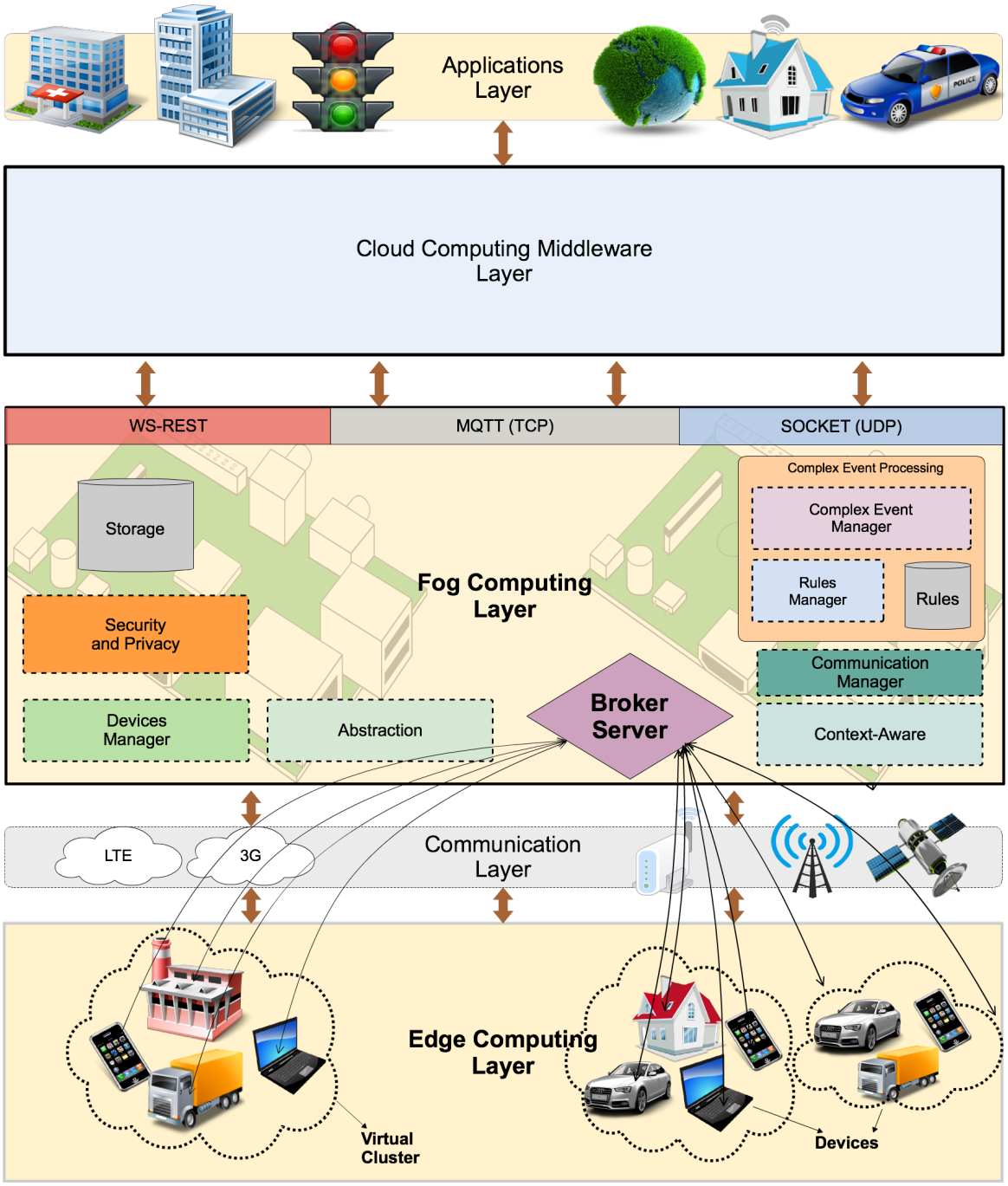

Figura 3.1 – Arquitetura de fog computing

3.1 Descrição das Camadas da Arquitetura

No topo da arquitetura, cumprindo o papel de elemento principal de controle, gerenciamento e armazenamento, seja uma plataforma de *cloud computing* ou uma plataforma de *middleware*, é necessária a existência de protocolos bem estabelecidos de comunicação, segurança e armazenamento de dados. Além disso, também é necessário que se tenha conhecimento de cada dispositivo conectado a sua rede, tendo a possibilidade de operar sobre cada um, ter conhecimento das diferenças de dispositivos *fog* de dispositivos *edge*, ter conhecimento sobre cada *fog* e os disposi-

tivos conectados a ele. Por fim, ainda é necessário que disponibilize de forma eficiente as informações para o consumidor.

Na camada de *fog computing* é necessário que se tenha conhecimento de cada dispositivo conectado em sua rede local, com a possibilidade de operar sobre os dispositivos, seus sensores e atuadores. Também é necessário que exista um método eficiente de armazenamento dos dados, visto que a camada de *fog computing* geralmente está em ambiente distribuído e em dispositivos de computação sem grande poder computacional, dessa forma, diversos bancos de dados *lightweight* estão disponíveis com suporte para diversas linguagens de programação. Por se estar em ambiente distribuído, um requisito fundamental é a segurança no canal de comunicação entre dispositivos e a camada de *fog*, bem como *fog* com *core middleware*, tendo em vista que são canais mais suscetíveis a ataques. Além de seguro o protocolo de comunicação precisa ser eficiente para não prejudicar o tempo total entre o recolhimento do dado, seu processamento, até ser disponibilizado para o consumidor.

Na camada de *edge computing*, é necessário que sejam estabelecidos os métodos de comunicação e parâmetros de segurança. Além disso também que seja possível executar algum tipo de processamento, mesmo que de baixa complexidade computacional.

Diversas técnicas e métodos são necessários para tornar possível o desenvolvimento e implantação de uma arquitetura de sistema. Decompor um sistema complexo e monolítico em camadas permite a criação de um ambiente facilitado e componentes com melhores definições. Abaixo serão especificadas as camadas pertencentes a arquitetura apresentada nesse trabalho.

3.1.1 Camada de Middleware

Essa é a camada responsável por realizar as principais funções do sistema e opera em modo bidirecional. Ela age como uma interface entre a camada de *hardware* e a camada das aplicações, além de ser responsável por funções essenciais, tais como, gerenciamento de dispositivos, gerenciamento de informações, filtragem dos dados, agregação de valores, análise de semântica, controle de acesso e da abstração dos protocolos de acesso dos diversos dispositivos da camada das coisas. É necessário que se tenha conhecimento de cada dispositivo conectado a sua rede tendo a possibilidade de contextualizar esse dispositivo e seu ambiente, além de ter o controle de acesso, conexão e desconexão, solicitação e recebimento de dados de cada dispositivo. É necessário também discernir um dispositivo *fog* de um *edge*, ter conhecimento sobre cada *fog* e os dispositivos conectados a ele e suas funções. Mais ainda, é necessário que disponibilize de forma eficiente as informações para o consumidor, provendo uma interface de alto nível para o desenvolvedor de aplicações.

3.1.2 Camada de Fog Computing

A camada de *fog computing* é o principal elemento do trabalho e tem como objetivo criar uma camada federada, ou seja, virtual, tendo como principal função aproximar de forma eficiente as camadas de aplicações e as camadas de dispositivos. Na arquitetura proposta, essa camada virtual será alocada em ambiente de *System-on-a-Chip* (SoC), um sistema computacional completo num único chip. Tendo como justificativa processamento eficiente desde que não exija alta complexidade computacional, bem como a variedade de tipos de conexão, tais como, WiFi, *Bluetooth, Ethernet*, serial, entre outras, cada SoC atuará como um *edge-gateway*, que consiste em um concentrador de processamento de um ou mais dispositivos. Por exemplo, um *edge-gateway* de uma *SmartHome*, que pode estar conectado com o sistema de iluminação, eletroeletrônicos, carros, entre outros. Os dispositivos por sua vez, não mais se comunicam com o *middleware* central, mas sim com o *edge-gateway*, criando esse ambiente virtual de processamento na camada de *fog computing*. A Figura 3.1 mostra a camada de *fog computing* e seus componentes.

A camada de *fog computing*, assim como a camada de *middleware*, também opera em modo bidirecional e pelo fato de atuar como um *middleware* mais próximo dos dispositivos possui algumas de suas funções essenciais, tais como: abstração de diversos protocolos técnicos, incluindo, protocolos de rede e comunicação, portas de conexão, entre outros; módulo de controle dos ciclos de execução e operação e etc. Para o melhor entendimento os módulos que compõem a camada de *fog computing* estão listados abaixo:

- **Abstração de protocolos:** Tendo em vista que os dispositivos IoT conectados à rede são heterogêneos, é esperado que cada um tenha características próprias, como protocolos de conexões, tipos de dados, ou linguagem de comunicação. É necessário que exista uma abstração desses protocolos para tornar o acesso aos dispositivos mais harmônico, além de abstrair do programador a necessidade de conhecimento de todos os dispositivos e suas características.
- **Broker Server:** Na arquitetura proposta, quando um elemento da rede deseja receber uma determinada informação, ele realiza uma assinatura, fazendo uma requisição para outro elemento responsável por gerir as publicações e as assinaturas. O *broker server* é o elemento responsável por essa função e atua como intermediário no processo de comunicação. Partindo dos dispositivos para a camada de *fog*, em um primeiro momento são recolhidos os dados dos sensores, os quais são publicados no *broker server*, a partir desse momento, consumidores interessados podem fazer a assinatura do *broker* a fim de se obter os dados produzidos nos dispositivos. Nesse caso, a própria camada de *fog computing* assina para a utilização dos dados, processando e enviando as informações pertinentes a camada de *middleware* que fica responsável por entregar os resultados as aplicações requisitantes.
- **Gerenciador de dispositivos:** Módulo responsável por prover suporte à conexão de novos dispositivos através de informações de protocolos de comunicação e conexão de dispositivos já conhecidos.

- **Gerenciador de comunicação:** Módulo responsável por administrar as comunicações entre as camadas consumidoras e provedoras de serviços. Responsável também por definir o melhor tipo de protocolo de aplicação para ser usado. Exemplos de protocolos são: *Socket* (TCP), WS-SOAP (API), WS-REST e o MQTT.

- **Gerenciador de contexto:** O contexto é qualquer informação que possa ser extraída de uma entidade a fim de caracterizá-la. Essa entidade pode ser um usuário, um dispositivo ou um nodo *fog*. O contexto pode ser usado para prover serviços de informações relacionados a entidades. O módulo gerenciador de contexto é responsável por armazenar informações de entidades e também realizar procedimentos com essas informações (raciocínio de contexto), e assim obter informações contextualizadas.

- **Segurança e privacidade:** É essencial a presença de mecanismos de segurança em ambientes IoT. Muitas vezes as informações que trafegam em redes IoT são sigilosas, como por exemplo informações médicas ou informações pessoais. A camada de *fog computing* para IoT precisa ter funções de segurança que protejam os dados trafegados e também que não permita a alteração dos mesmos.

- **Processamento de Eventos Complexos:** O módulo para processamento de eventos complexos (CEP) opera através da avaliação da confluência de eventos, e consequentemente, da execução de uma ação. Os eventos podem ocorrer por um longo período de tempo. A correlação de eventos pode ser ocasional, temporal ou espacial. CEP requer o emprego de sofisticados intérpretes de eventos, detecção de padrões de acontecimento, e técnicas de correlação. CEP é comumente usado para detectar e responder a anomalias de negócios, ameaças e oportunidades. Geralmente, motores de ontologias e regras são usados para fornecer vocabulários comuns de um domínio e conhecimento comportamental, expressando restrições e reação a eventos [25].

Além disso, o sistema ainda deve lidar com o armazenamento dos dados produzidos pelos dispositivos de sua rede local. Para isso podem ser utilizados bancos de dados *lightweight*, que podem ser executados até mesmo em dispositivos de pouco poder computacional, pois ocupam pouco espaço em memória. No momento que é registrado os dados da camada de *fog* na camada de *middleware*, então é feita a limpeza no banco de dados do *fog*, liberando mais espaço para os novos dados que serão recebidos. Quando é necessária a consulta, a camada de *fog computing* pode solicitar alguma informação já excluída diretamente para a camada de *middleware* que realiza a consulta e repassa a informação para utilização do *fog*.

3.1.3 Camada de Comunicação

A camada de comunicação conecta diferentes tipos de fontes de dados e serviços localizados em ambiente altamente distribuído. O primeiro estágio de tratamento dos dados coletados

acontece nessa camada. Além disso, também é responsável pelas mensagens de roteamento, serviços *Publish/Subscribe*, e também por realizar a comunicação entre diferentes plataformas, quando necessário.

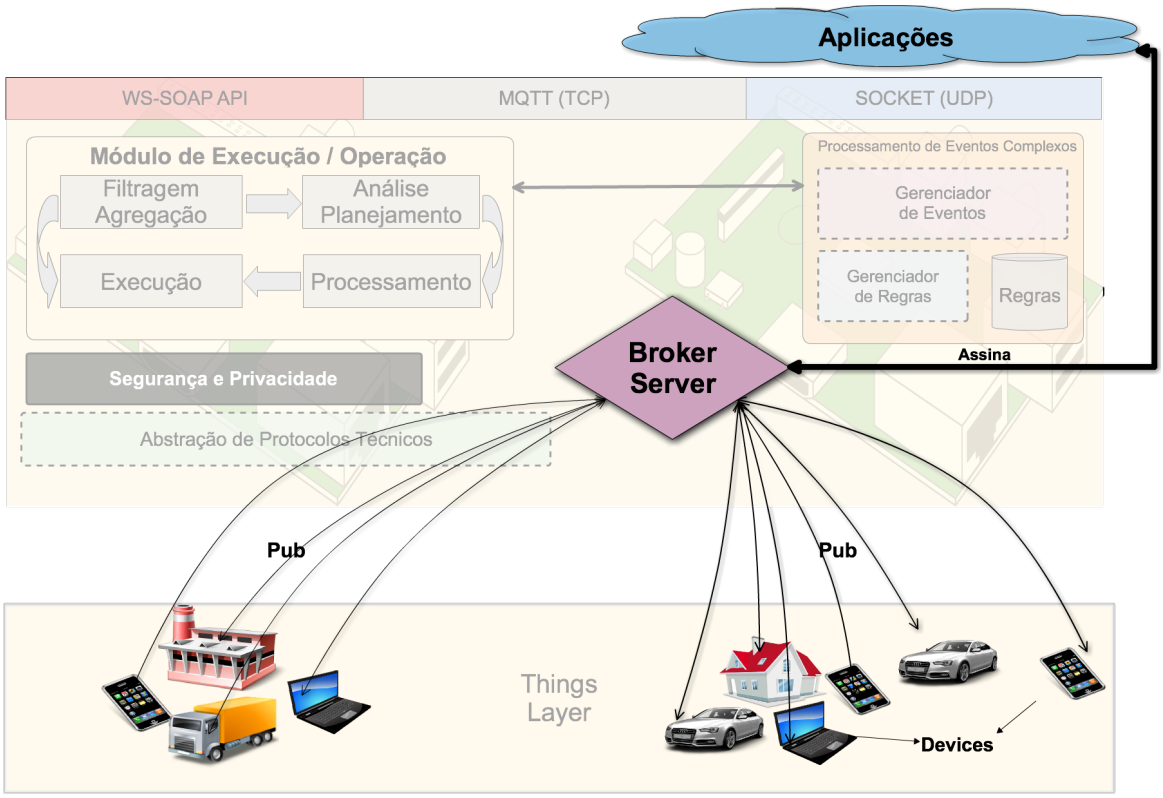

Figura 3.2 – Comunicação Tradicional e Real Time via Broker

O *Broker* é classificado como o elemento principal em qualquer protocolo *Publish/Subscribe* [3]. Além disso, é responsável por primeiro, receber todas as mensagens, filtrar e classificar de acordo com o tipo de prioridade do dado. O *broker* também é responsável por receber e administrar as assinaturas de consumidores interessados, para os quais envia as mensagens com os dados de interesse.

Na Figura 3.2 é possível identificar o *broker server* como um elemento da camada de *fog computing*, porém com atuação conjunta com a camada de comunicação. O *broker* opera através do recebimento de publicações de mensagens com dados dos dispositivos. As mensagens recebidas são filtradas e classificadas para então serem processadas e armazenadas no banco de dados da camada de *fog computing*. Se já houverem clientes que assinaram as mensagens de um determinado dispositivo, o *broker* juntamente com o *fog* realiza o envio para os consumidores interessados, caso não existirem assinaturas, é mantida a mensagem armazenada até que seja realizada uma consulta ou que seja feito o registro na plataforma de *middleware*.

3.1.4 Camada de Edge Computing

Essa é a camada que é executada nos extremos das bordas da rede, sendo na maior parte das vezes embarcada nos próprios dispositivos e responsável por receber dados dos sensores e operar nos atuadores conectados. Nessa camada são implementados os serviços de comunicação e segurança, seguindo os mesmos padrões da camada de *fog*, para trafegar de forma confiável os dados produzidos até a camada de *fog computing*. Além disso, essa camada também deve ser responsável por executar algum tipo de processamento, mesmo que seja de pouca complexidade computacional. Por exemplo: Um módulo com um sensor de temperatura e o controle sobre um exaustor operando dentro de um intervalo de temperatura no qual é ativado e desativado de acordo com o sensoreamento de temperatura. Esse tipo de processamento pode ser feito no próprio dispositivo, sem a necessidade de intervenção da camada de *fog computing* ou da camada de *middleware*.

3.2 FaaS4IoT

O sistema *Fog-as-a-Service for Internet of Things* foi planejado para ser executado em dispositivos distribuídos nas bordas da rede, geralmente dispositivos sem grande poder computacional. Dessa maneira, foram buscadas apenas soluções que cumprissem sua tarefa, seja de comunicação ou segurança, da forma mais eficiente e com o menor armazenamento possível, para ser portável em dispositivo SoC sem um grande poder de processamento como o de um servidor. A Figura 3.3 ilustra a arquitetura desenvolvida utilizando o COMPaaS como controlador principal de *middleware* e seus canais de comunicação, além do FaaS4IoT e suas principais características, MQTT, TLS e HSQLDB.

Esses SoCs, sejam eles embarcados em roteadores, caixas de postes de luz, *smart devices* dentro de uma *smart home*, entre outros, sempre estão em ambiente altamente distribuído o que implica na utilização de dispositivos de computação de tamanho reduzido, consequentemente sem grande quantidade de memória para armazenamento e sem um processador poderoso para executar grandes cálculos computacionais. Para o FaaS4IoT fazer a comunicação com os *edge devices* foi utilizado o padrão de comunicação MQTT [31], que é um protocolo *Machine-to-Machine* (M2M) trafegado pela rede TCP. Sobre a rede TCP foi aplicado o protocolo de segurança TLS, que exige um certificado no primeiro registro do dispositivo, garantindo que apenas dispositivos cadastrados sejam conectados.

MQTT significa *Message Queuing Telemetry Transport*. É um protocolo de mensagens *publish/subscribe*, em português publicar/subscrever, extremamente leve e de implementação facilitada, projetado para dispositivos distribuídos e de baixa largura de banda, alta latência ou redes não confiáveis [31]. Os princípios do projeto são minimizar a largura de banda da rede e o tanto de recursos que requisita do dispositivo garantindo a confiabilidade e algum grau de garantia de entrega com métodos de confirmação de recebimento. Esses princípios tornam o protocolo ideal para

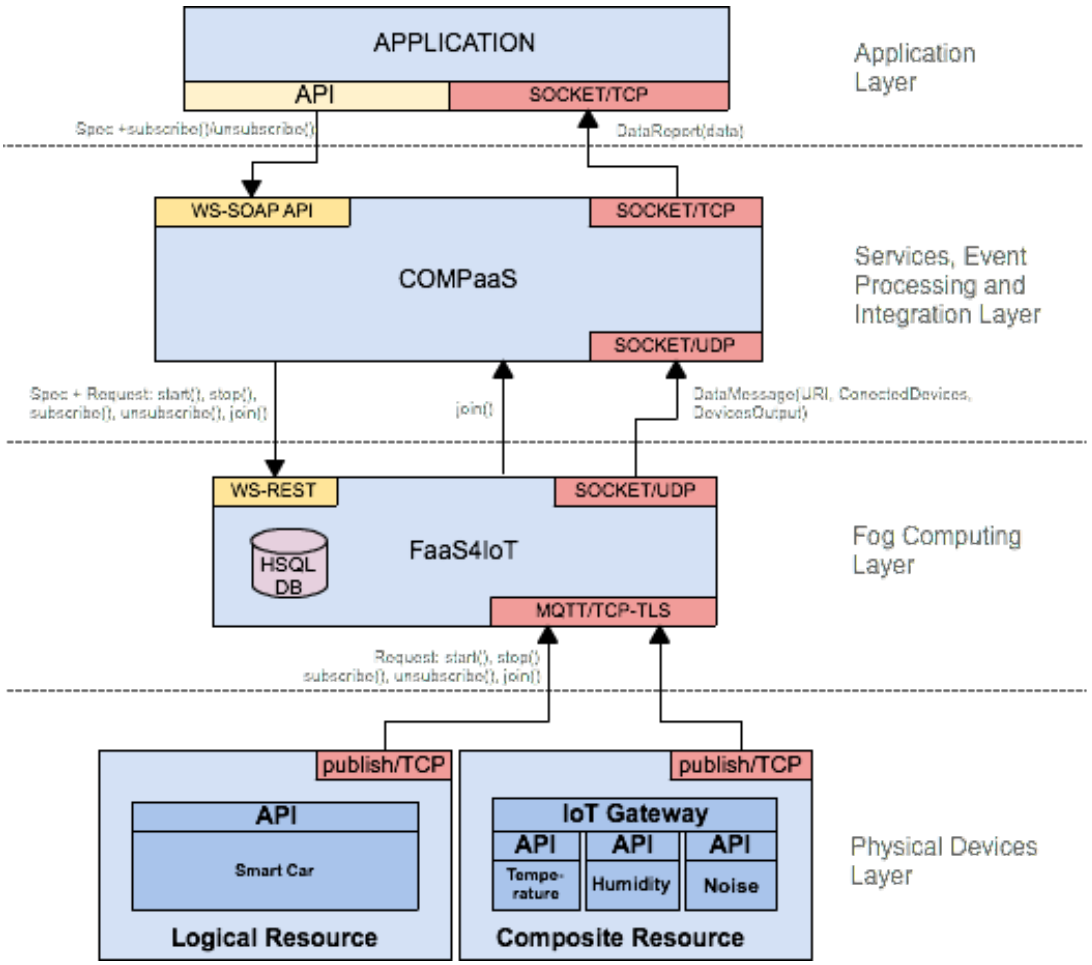

Figura 3.3 – Faas4IoT

sistemas de Internet das Coisas. O MQTT opera em dois elementos principais, o *broker server*, que estabelece todas as conexões com os dispositivos e também recebe todas as mensagens publicadas pelos clientes MQTT. Os clientes são o segundo elemento do MQTT e são configurados nos próprios dispositivos de borda, consumindo apenas 3 KB de memória para fazer o registro do endereço do *broker server*. O cliente estabelece um canal direto de comunicação com o *broker server*, por onde vai publicar seus dados produzidos pelos sensores, por meio de pacotes trafegados pela rede TCP. O cliente também tem a possibilidade de receber pacotes de mensagem vindos do *broker*, o que é uma vantagem pois possibilita a criação de aplicações com sensoreamento e atuação em tempo real.

A comunicação do FaaS4IoT com o COMPaaS ocorre por meio da utilização de *Web Services* REST e *Socket*/UDP. Para se estabelecer a comunicação, o FaaS4IoT faz uma solicitação ao COMPaaS através do método *join()*. O COMPaaS responde através do WS REST com os métodos *start(), stop(), subscribe(), unsubscribe()*. Para enviar os dados e informações de processamento de seus dispositivos o FaaS4IoT utiliza o protocolo de comunicação *Socket*/UDP.

Para cumprir os requisitos de segurança foi utilizado o protocolo de segurança TLS, que utiliza mecanismos de criptografia para proteger o canal de comunicação TCP. No servidor *broker* existe um certificado que pode ser interpretado usando a chave criptográfica que é de conhecimento dos dispositivos autorizados para acesso. Isso cria um canal de comunicação sobre o TCP que é

seguro e trafega os pacotes de mensagem de todos os dispositivos localizados em ambiente distribuído. Esse mecanismo garante a segurança entre os dispositivos *edge* e *fog computing*. Entre *fog* e COMPaaS a comunicação ocorre sobre a rede UDP, sendo assim o protocolo de segurança que é utilizado é o DTLS, que é uma versão do TLS para comunicações UDP. Dessa forma é garantida a segurança em todos os níveis de comunicação.

O banco de dados que precisa ser executado em um dispositivo de borda precisa ser eficiente em termos de armazenamento e velocidade na consulta de dados. Para cumprir esses requisitos foi utilizado o gerenciador de banco de dados HSQLDB, que é um *software* de banco de dados relacional SQL escrito em Java, sendo assim, portável em multi-plataformas. O HSQLDB permite o armazenamento de dados de duas formas, a primeira delas em memória volátil que permite o acesso a memória de forma muito mais rápida, porém os dados são perdidos quando o dispositivo é desligado. A outra forma de armazenamento é em arquivos, mantendo o armazenamento para consulta sempre que requisitado. Levando em consideração que as memórias utilizadas para armazenamento em SoC geralmente são memórias *flash*, é mantida a garantia de acesso em tempo que não compromete o desempenho final do sistema.

A linguagem de programação escolhida para o desenvolvimento do FaaS4IoT foi o Java [37], por diversos fatores. O primeiro deles é a portabilidade, tendo em vista que o Java é ideal para ser executado em dispositivos com diferentes plataformas e sistemas operacionais. Também possui uma extensa biblioteca de rotinas que facilitam a cooperação com protocolos TCP/IP, entre outros, com a possibilidade de implementação de métodos de segurança nesses canais de comunicação. Além disso também é eficiente na desalocação de memória automática por utilizar processos de coletas de lixo, ou em outras palavras, recuperar áreas inutilizadas de memória.

Para um melhor entendimento das fases desde a produção do dado no dispositivo, até a entrega do mesmo na aplicação requisitante, a subseção 3.2.1 apresenta o ciclo de eventos da utilização do FaaS4IoT com o COMPaaS.

3.2.1 Ciclo de Eventos

O ciclo de eventos da arquitetura implementada orquestra as três camadas do sistema, *edge*, *fog computing* e *middleware*. Começando a descrição do ciclo de eventos na camada mais inferior da arquitetura, a camada de *edge computing*, o dispositivo tem implementado em seu *software/firmware* um comando que solicita a conexão MQTT com o *broker server*, através do seu endereço IP e porta de comunicação, esse comando é executado toda vez que o dispositivo é conectado a energia. Essa solicitação é feita apenas na primeira conexão, se o *broker server* aceitar a conexão é então estabelecido um canal de comunicação permanente e aberto por onde podem-se enviar comandos e receber dados. Os comandos que o FaaS4IoT pode enviar para os dispositivos são:

- *disconnect()*: encerra-se a comunicação com esse dispositivo;

- *restart()*: solicita a reinicialização do dispositivo;
- *update()*: solicita a nova leitura dos sensores e novo envio desses dados;
- *stop()*: solicita a parada no recolhimento dos dados dos sensores, mas mantém o canal de comunicação aberto.

O sistema de *fog* também comunica o *middleware* principal toda vez que recebe uma nova conexão, informando o perfil do novo dispositivo em um arquivo XML que contem diversas informações sobre o dispositivo, como por exemplo, seu endereço URI, Id, nome, tipo de dado produzido, entre outros, como pode ser visto na Figura 3.4.

```
<profileInformation xsi:type="deviceProfile"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <uri>
        http://192.168.0.10:8080/LogicalResource/resources
    </uri>
    <name>Termometer 15</name>
    <location>
        <name>Silo 4 - level 1 - pendulum 2</name>
        <latitude>-27.817244 S</latitude>
        <longitude>-51.697457 W</longitude>
    </location>
    <purpose>Temperature monitor</purpose>
    <owner>GSE</owner>
    <keywords>
        <keyword>Temperature</keyword>
        <keyword>Degrees</keyword>
        <keyword>Celsius</keyword>
    </keywords>
    <services>
        <service>
            <responseType>Double</responseType>
            <generationTime>1000</generationTime>
            <availableService>
                Temperature generation
            </availableService>
        </service>
    </services>
    <model>
        <deviceClass>Termometer</deviceClass>
        <number>1903</number>
        <name>Digital Termometer</name>
    </model>
    <information>
        <status>ONLINE</status>
    </information>
</profileInformation>
```

Figura 3.4 – XML de Registro do Dispositivo Enviado Pelo FaaS4IoT.

O FaaS4IoT recebe os dados dos sensores através do *broker server* da comunicação MQTT. Ao receber um novo dado, primeiro é feito um registro do evento, arquivando o dado recebido, a URI do dispositivo que produziu o dado, tipo de dado recebido, tempo de recebimento, entre outros. O *fog* por ser o elemento principal nesse cenário de rede local, se comunica com todos os dispositivos da região, assim consegue efetuar o processamento de todos, por exemplo, determinando a temperatura de uma sala através da média dos valores dos quatro sensores de temperatura do local, ou em um ambiente de cidades inteligentes um *fog* controlando o sistema de iluminação publica de todo o quarteirão. Com as diversas informações do ambiente, o *fog* pode verificar os acontecimentos em

comparação com as regras. As regras são padrões de acontecimento previamente estabelecidos, como por exemplo: para se maximizar a produção de morangos em ambiente de estufa, sabe-se que a temperatura do ambiente deve estar entre 21ºC e 23ºC, além da umidade relativa do solo sempre entre 60% e 75%. Esses valores são utilizados como regras para referência no método de verificação da ocorrência de eventos, na identificação de um evento, é gerado um alerta que é imediatamente comunicado ao COMPaaS. Além disso, pode existir alguma atividade pré-programada na ocorrência do evento, por exemplo, o FaaS4IoT determinou que a temperatura esta acima do ideal, é então enviado um comando acionando o sistema de exaustão. Em outras palavras, a união dos mecanismos e técnicas para o monitoramento de eventos, juntamente com as regras de padrões pré-determinados cria o processamento de eventos complexos, que completa o ciclo de processamento que acontece dentro do sistema de *fog computing* que pode ser visto na camada central da Figura 3.5. Os objetivos da utilização de regras em conjunto com o registro de eventos dos dispositivos, é que seja possível detectar a confluência de eventos específicos, auxiliando na prevenção de erros e no auxílio na tomada de decisões, colaborando para garantir a qualidade e confiabilidade do sistema.

Dessa forma, o FaaS4IoT e os dispositivos *edge* são responsáveis apenas pela produção, processamento e gerenciamento dos dados que serão posteriormente disponibilizados como serviços pelo middleware. O *edge* produz dados através da leitura de seus sensores e entrega os mesmos para a camada de *fog computing*. O FaaS4IoT além de realizar o processamento e poder controlar os atuadores dos dispositivos, realiza o processamento também com o objetivo de produzir novos dados mais complexos para serem utilizados até mesmo pelo próprio FaaS4IoT e os dispositivos conectados a ele. Ao processar mais de um tipo de dado vindos de diversas fontes, pode se produzir dados mais complexos, contextualizados e confiáveis, para serem utilizados auxiliando em serviços para cidades inteligentes, automação industrial, entre outras aplicações.

O COMPaaS por sua vez, tem o controle de todos os nodos FaaS4IoT conectados e os demais dispositivos, sendo assim pode produzir informações mais complexas e precisas sobre um ambiente bem mais amplo e distribuído. Além disso, o COMPaaS também é responsável por gerenciar e administrar os consumidores interessados que assinaram o recebimento dos serviços disponibilizados pela camada de *fog*. Para o consumidor interessado o COMPaaS provê uma interface de conexão, por onde a aplicação pode enviar uma especificação, repassando as informações as quais deseja receber, o endereço para receber as informações assinadas, em qual protocolo quer que a mensagem seja codificado, XML ou JSON, por exemplo, e por fim, o tempo que deseja manter a assinatura para recebimento das informações.

Essa maneira de organização do sistema possibilita que o COMPaaS consiga gerenciar os dados e as requisições através do ciclo de eventos, onde determina quais dados e informações satisfazem determinados consumidores requisitantes. Também é possível que dados sejam produzidos mesmo sem nenhuma assinatura para recebê-lo, nesse tipo de situação, o mesmo é armazenado para o caso da possibilidade de consulta futura.

Quando o COMPaaS encerra um ciclo de eventos, ou seja, entregou para o consumidor requisitante a informação que ele solicitou, durante o tempo que foi assinado, é então enviado para

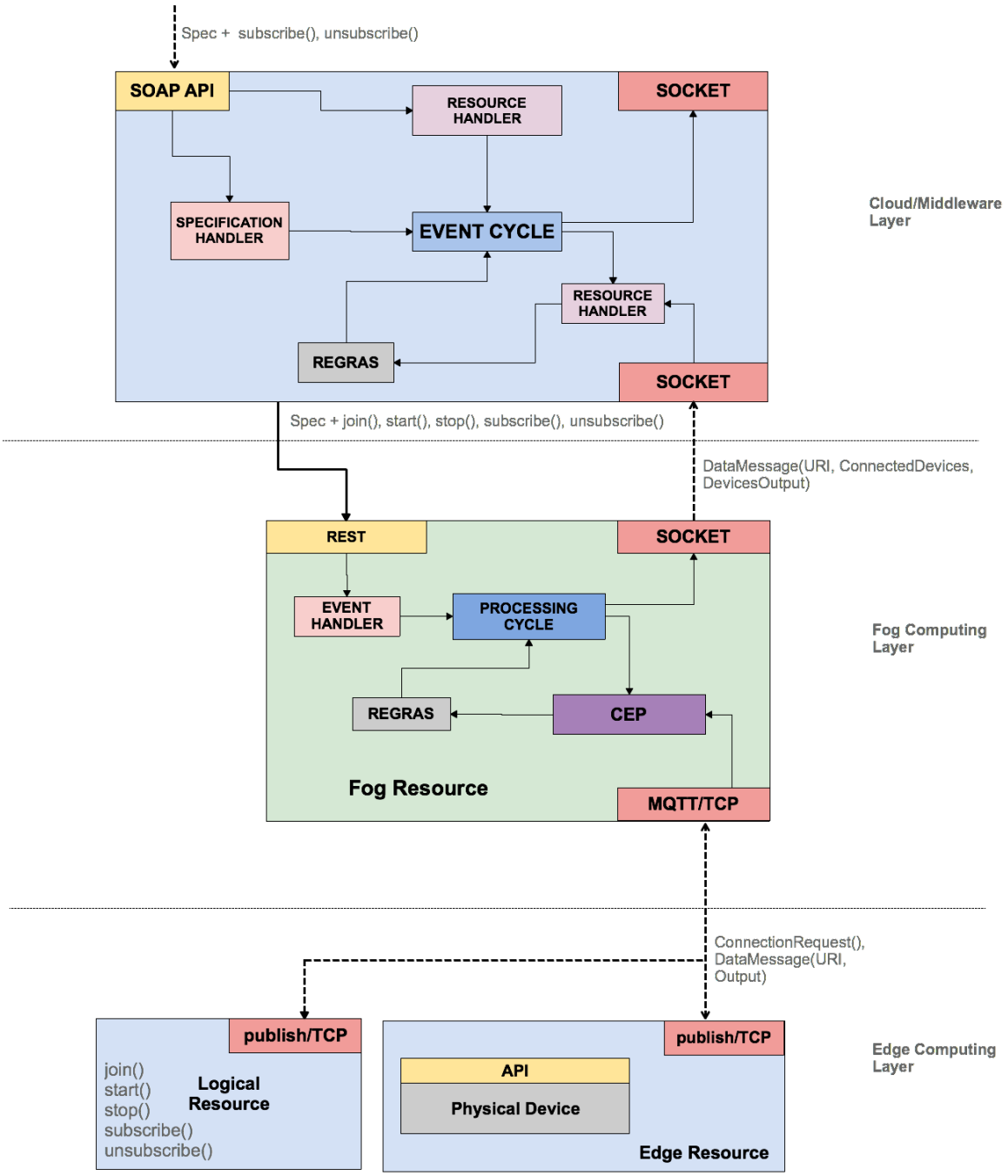

Figura 3.5 – Ciclo de Eventos COMPaaS e FaaS4IoT

o FaaS4IoT o comando *stop()*. O FaaS4IoT, por sua vez, interpreta esse comando e verifica qual dispositivo estava vinculado ao evento que esta sendo encerrado e repassa o comando *stop()* para esse dispositivo, inativando a coleta e envio dos dados, mas mantendo o canal de comunicação ativo. No caso de o COMPaaS receber nova assinatura para recebimento dos dados, o processo é o mesmo, mas com o comando *start()*, que é interpretado e repassado pelo FaaS4IoT ativando a coleta e envio de dados nos dispositivos de borda necessários para atender a requisição.

3.2.2 Transição de Estados do FaaS4IoT

O diagrama de transição de estados apresentado na Figura 3.6 é uma representação dos estados do que se pode encontrar no decorrer da execução de processos do sistema. O FaaS4IoT foi desenvolvido para atender aos diversos requisitos expostos anteriormente e uma série de etapas devem ser executadas para tornar isso possível. Assumindo que o FaaS4IoT, bem como o COMPaaS já estejam inicializados e comunicando-se entre si através do protocolo de comunicação *Web Service* REST, ao conectar o dispositivo *edge* a energia, o mesmo possui em sua memória um script o qual executa o software desenvolvido para esse dispositivo. Independente do tipo de sensor conectado, ou se o dispositivo *edge* é responsável por um atuador, o mesmo ao ser inicializado solicita um canal de comunicação seguro com o FaaS4IoT, realizando o protocolo de *Handshake* entre as duas plataformas. Se o protocolo for bem sucedido então é criado um canal de comunicação seguro sobre a rede TCP/IP.

O dispositivo com o canal seguro estabelecido, primeiro verifica se possui um identificador (ID) em sua memória, se existir, apenas solicita o canal de comunicação MQTT com o *broker server* do FaaS4IoT, senão, faz a requisição de um ID. Essa etapa é realizada para o caso desse dispositivo já ter sido conectado anteriormente e se por ventura ocorrer um período de falta de energia elétrica, ao ser normalizado o serviço, pode acontecer dos dispositivos *edge* receberem novos endereços IP da rede. Com a execução dessa etapa é possível executar um método no FaaS4IoT capaz de verificar o ID com o registro do banco de dados, no caso de ser um novo dispositivo, é então gerado um ID, criado um registro do mesmo com o endereço IP do dispositivo e arquiva no banco de dados, se já for um ID existente, é então atualizado o endereço IP no registro do *edge* para manter a sincronia dos dados produzidos e recebidos dos clientes IP com os respectivos dispositivos ID criadores. Esse ID é então respondido para o dispositivo *edge* que grava o código identificador em sua memória, ao ser realizada essa etapa, é então solicitada a comunicação MQTT, além de iniciar o recolhimento dos dados dos sensores.

O FaaS4IoT faz o registro da conexão MQTT com o endereço IP e ID, além de outras diversas informações sobre o dispositivo, tais como, nome, modelo, fabricante, entre outras, o registro é então feito também no COMPaaS para que dessa forma tenha o conhecimento de todos os dispositivos conectados e suas funções. Além disso o FaaS4IoT também inicia o ciclo de eventos ficando preparado para iniciar o recebimento de dados do novo dispositivo, essa inicialização do ciclo também é notificada para o COMPaaS.

O dispositivo ao ser inicializado, está preparado para comunicar os dados produzidos nos sensores, assim como está apto para receber comandos, como por exemplo, *start(), stop(), update()*, entre outros, do FaaS4IoT de acordo com suas necessidades. Os dados produzidos são sempre publicados no *broker server* do FaaS4IoT, canal de comunicação constantemente aberto com todos os dispositivos e assinantes. O FaaS4IoT, cada vez que recebe um novo dado, realiza o registro de um evento, arquivando, IP do dispositivo que produziu o dado, ID, tipo de dado recebido, qual dado foi recebido, o dado em si, além do horário no qual o dado foi produzido, entre outros. Sobre essa

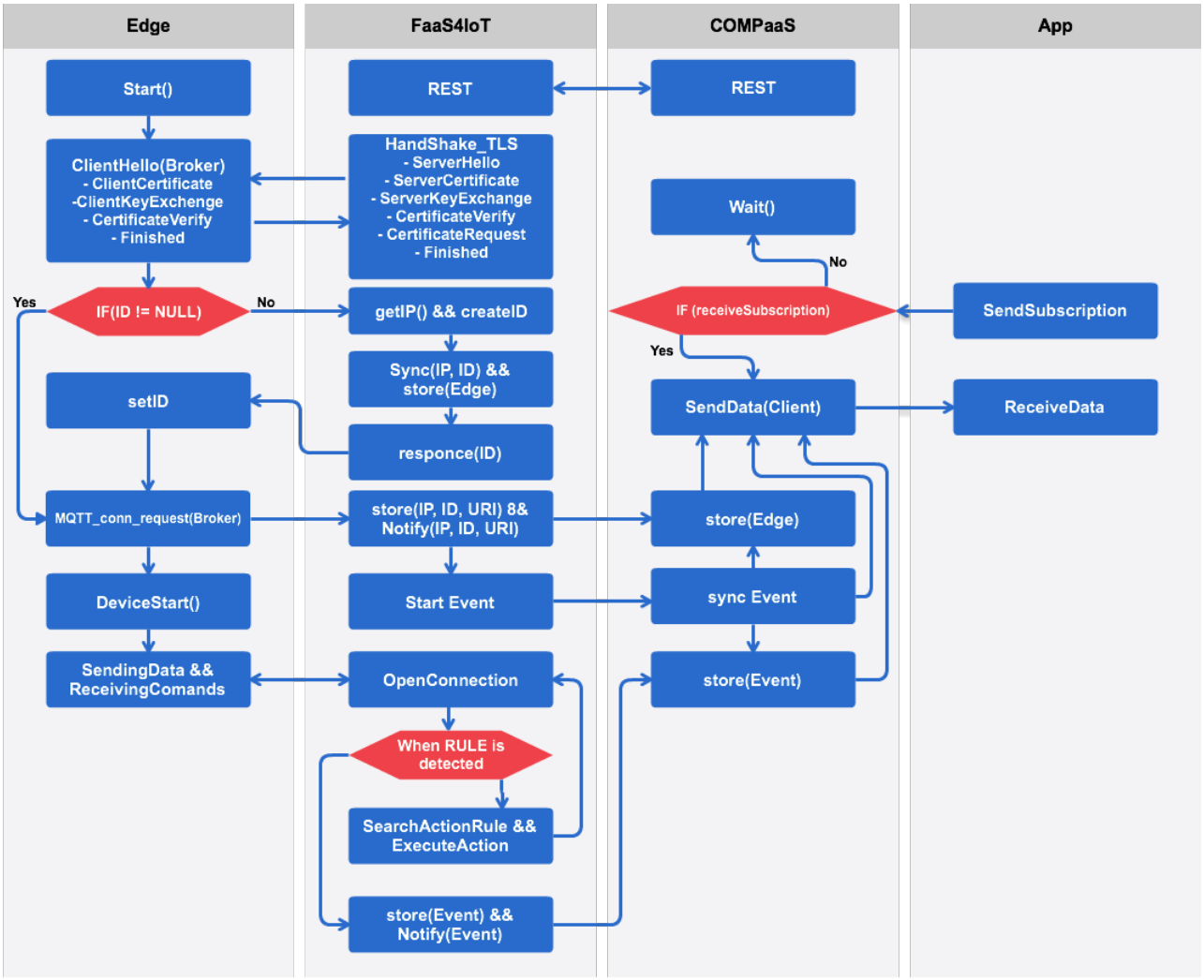

Figura 3.6 – Diagrama de Transição de Estados do FaaS4IoT.

lista de registro de eventos existem métodos para a verificação da ocorrência de eventos complexos, baseando-se em regras estabelecidas previamente estabelecidas, como por exemplo: "Regra temperatura, ao ser detectada a temperatura maior que 24 graus, ativar o sistema de exaustão, até que a temperatura volte ao ideal".

O FaaS4IoT ao detectar um evento, realiza uma consulta para verificar qual a ação recomendada na ocorrência desse evento, executa a ação nos dispositivos ou no FaaS4IoT, em paralelo realiza o registro do evento de detecção, evento o qual também pode ser detectado por outra regra, que geraria outra ação e registro. Também é realizada a notificação dos eventos ocorridos no COMPaaS, que está monitorando as assinaturas dos consumidores interessados, na ocorrência de uma assinatura, é iniciado o processo de envio das informações produzidas com os dispositivos, eventos e ocorrência de eventos, para as aplicações consumidoras. Pode acontecer a situação de o COMPaaS receber uma assinatura de um dispositivo que esta conectado a rede mas não está inicializado, nesse caso, o COMPaaS envia um comando para o FaaS4IoT, solicitando a determinada operação sobre o dispositivo, informando o endereço IP da rede e o ID, o que possibilita que o FaaS4IoT encontre o dispositivo na rede e inicialize o mesmo.

A Figura 3.6 representa a transição de estados de apenas um dispositivo, mas é importante destacar que o FaaS4IoT está preparado para administrar diversas conexões e comunicações de dispositivos em paralelo, monitorando a confluência de eventos de todos os dispositivos. Os detalhes de implementação e métodos utilizados serão apresentados no próximo capítulo do trabalho.

3.3 Considerações Finais

A arquitetura proposta atende aos diversos requisitos explanados anteriormente. Provê segurança e conformidade na transição dos dados entre todas as camadas do sistema. Também oferece suporte ao armazenamento eficiente de dados, através da utilização de bancos de dados *lightweight* próprios para dispositivos de pouco poder computacional (SoC). Além disso, o processamento em dois níveis de distribuição faz com que o volume de transferência de mensagens e o tempo no tratamento das mesmas seja reduzido drasticamente, entregando a informação para as aplicações IoT requisitantes em tempo satisfatório.

O processamento local diminui a latência do sistema e possibilita até mesmo a execução de aplicações em tempo real, tendo em vista que o *broker server* pode receber assinaturas diretamente de aplicações e a camada de *fog computing* pode realizar o processamento dos dados sem necessidade de comunicação com a camada de middleware. A camada de *fog computing* posteriormente apenas realiza o registro das operações executas, além dos dados e informações processadas, mantendo assim o nível hierárquico característico em arquiteturas de *middleware* SOA.

4. AVALIAÇÃO E TESTES

Para validação, a arquitetura proposta foi desenvolvida utilizando a plataforma de *middleware Cooperative Middleware Plataform-as-a-Service* (COMPaaS) como referência para criar o *Fog-as-a-Service for Internet of Things* (FaaS4IoT). FaaS4IoT é um sistema de *fog computing* para ser executado em dispositivos próximos das bordas da rede, na maior parte das vezes dispositivos *System-on-a-Chip* (SoC) [50] são os utilizados. Para tornar isso possível utilizou-se o protocolo de comunicação MQTT [31] por não necessitar de muita memória para ser executado, garantir até três níveis de QoS, o nível mínimo é zero e garante a entrega do pacote, mas sem confirmação de recebimento, também chamado de *"fire and forget"*. O nível um garante a entrega do pacote ao servidor com o recebimento de uma confirmação da entrega. E por fim, o QoS mais alto é o dois, ele garante que cada mensagem seja recebida pelo servidor pelo menos uma vez. É o nível mais seguro e confiável, entretanto, também é o nível que leva mais tempo para garantir o QoS, visto que a garantia é fornecida por dois fluxos de troca de mensagens entre o cliente e servidor. Também foi aplicado o protocolo de segurança TLS [19] sobre a comunicação TCP [34], protocolo de rede utilizado pelo MQTT. Da mesma forma, é utilizado o HyperSQL (HSQLDB) [23], um banco de dados *lightweight* que armazena as informações de forma eficaz, utilizando pouco espaço para armazenamento em memória.

Para testar o sistema FaaS4IoT trabalhando em paralelo ao COMPaaS, juntamente com os dispositivos *edge* produzindo dados e se comunicando, foram criados dois cenários para serem aplicados os testes. No primeiro cenário foi simulado um ambiente de iluminação pública com diversos postes de luz se comunicando com um poste controlador principal, onde o FaaS4IoT esta embarcado. Já o segundo cenário está voltado para uma área que recebe constantes investimentos no Brasil, o agronegócio, onde o FaaS4IoT foi aplicado para coordenar e operar uma estufa agrícola. Os testes realizados dentro desses cenários são apresentados mostrando um comparativo entre o sistema tradicional de middleware, operando sem *fog* e *edge computing*, e a arquitetura implementada neste trabalho.

O restante do capítulo apresenta a descrição do estudo de caso na seção 4.1 e as configurações do ambiente na seção 4.2. Por fim, apresenta o cenário de iluminação pública na seção 4.3 e o teste no cenário do setor agrícola na seção 4.4.

4.1 Descrição do Estudo de Caso

Para serem realizados os testes utilizando a arquitetura implementada foram simulados dois cenários reais de aplicação que se beneficiariam com a utilização dos conceitos e técnicas de *fog* e *edge computing*.

O primeiro cenário simula um ambiente de cidade inteligente com um sistema de iluminação pública, onde é possível controlar cada poste separadamente, ligando e desligando a lâmpada,

controlando a intensidade da luz (dimerizador) e obtendo informações de um sensor de temperatura. Nesse cenário cada poste de luz pode estar conectado à Internet, podendo se comunicar com um poste principal, onde o FaaS4IoT esta embarcado. O FaaS4IoT consegue determinar a temperatura da região coberta pelos postes, além de disponibilizar todos os dados de todos os postes. Também foram desenvolvidas aplicações que assinam o recebimento de informações do FaaS4IoT.

O segundo cenário, também é dentro da área de cidades inteligentes, mas no domínio do agronegócio inteligente. É simulado o ambiente de uma estufa hortifrúti, com diversos módulos espalhados pelo ambiente. Cada módulo possui um sensor de temperatura e um sensor de umidade relativa do solo. Alguns módulos são responsáveis pelos atuadores que acionam as válvulas de irrigação e o sistema de exaustão da estufa. O sistema consegue monitorar os dados do ambiente e operar sobre os atuadores, dessa forma mantém a estufa dentro de condições favoráveis, a fim de maximizar a produção.

Os testes executados em ambos os cenários de aplicação têm dois objetivos principais:

1. Validação dos mecanismos utilizados no desenvolvimento do sistema, que são: comunicação, segurança e banco de dados eficiente em termos de armazenamento.

2. Verificar se o sistema atende aos principais requisitos funcionais de arquiteturas híbridas de *fog* e *edge computing*, que são: protocolos de segurança confiáveis nos canais de comunicação, comunicação eficiente entre as camadas da arquitetura com o menor numero de comunicações possíveis com o middleware principal, prover autonomia aos dispositivos, dessa forma, auxiliando nas tomadas de decisões.

Para se comprovar que uma arquitetura híbrida de *fog* e *edge computing* pode ser mais eficiente que as arquiteturas tradicionais, é necessário se comparar os tempos de execuções nos dois tipos de sistemas. Para serem realizados os testes sem a camada de *fog computing*, foi utilizado apenas o COMPaaS com os dispositivos conectados a ele. Também foi testada a utilização do TLS no FaaS4IoT, aplicando diferentes testes com o protocolo ativo e inativo. Por fim, os tempos de execuções e demais resultados obtidos nos dois tipos de implementações do sistemas foram comparados e serão apresentados em gráficos e tabelas no decorrer das próximas seções to trabalho.

É importante mencionar que arquiteturas tradicionais IoT, também são muito eficientes e confiáveis em diversos ambientes, principalmente em ambientes com maiores capacidades de rede. O objetivo dos testes realizados não é mostrar que o COMPaaS ou plataformas de *middleware* tradicionais IoT são ineficientes, mas sim, comprovar que a utilização de um camada de processamento mais próxima dos dispositivos trabalhando em paralelo com o *middleware* principal, melhora a eficiência e o desempenho da arquitetura como um todo.

4.2 Configuração do Ambiente

Para tornar possível esse ambiente de simulação utilizando o sistema de *middleware* COMPaaS e o sistema de *fog computing* FaaS4IoT, foi necessária a criação de uma infraestrutura de máquinas virtuais (VM) e emulação de processadores de microcontroladores.

Para emular o processamento e criar as máquinas virtuais foi utilizado a plataforma QEMU, que usa o método de compilação dinâmica dividindo uma tarefa em diversos blocos menores de tarefas, tornando a execução muito mais eficiente. QEMU pode ser executado em múltiplas plataformas e sistemas operacionais, e é capaz de emular processadores PowerPC, ARM, Sparc32-64, MIPS, entre outros, suportando a criação de até 255 máquinas virtuais [12].

A Tabela 4.1 mostra os dispositivos de hardware utilizados para a execução dos testes, suas características, sistema operacional (SO), a utilização do QEMU, entre outros.

Tabela 4.1 – Características de hardware.

Descrição	**Fog**	**Edge**	**Middleware**
Máquina	Dell Inspiron I5	MacBook Pro i5	Dell All-in-one I5
Processador	Intel Core i5-2.50 GHz	Intel Core i5-2.3 GHz	Intel Core i5-3.2 GHz
Memória RAM	8 Gb	8 Gb	8 Gb
HD	1 Tb	256 Gb	1 Tb
SO	Ubuntu 16.04 LTS	MacOS X El Capitan	Ubuntu 16.04 LTS
Máquina Virtual	QEMU	QEMU	-
Processador VM	ARM 64 bit 1.2 GHz	RISC 32 bit 80 MHz	-
Memória RAM VM	1 Gb	512 Kb	-

Cada máquina virtual recebe um endereço único que será a porta de comunicação entre as plataformas. Da mesma forma o COMPaaS e o FaaS4IoT recebem endereços únicos. Na máquina virtual da camada de *fog* foi utilizado o sistema operacional Lubuntu 14.04 [55]. Lubuntu é uma versão *lightweight* do sistema operacional Ubuntu, ideal para dispositivos sem muitos recursos de *hardware*.

Os dois cenários para aplicação dos testes serão apresentados nas próximas seções do trabalho com demais detalhes de implementação e de infraestrutura de dispositivos. Também são apresentadas as discussões sobre os resultados obtidos fazendo uma análise crítica e comparativa.

4.3 Cenário 1 - Iluminação Pública em Cidades Inteligentes

O constante investimento, bem como as diversas vantagens obtidas ao serem utilizadas aplicações reais de IoT em ambientes de cidades inteligentes faz com que empresas e universidades, cada vez mais dediquem tempo no desenvolvimento de serviços que funcionem em benefício das pessoas. As cidades inteligentes, principalmente em países desenvolvidos, já são uma realidade e diversos sistemas já foram aplicados e estão em funcionamento em todo o mundo. A cidade de Nova

York, por exemplo, possui um sistema de coletas de lixo inteligente, onde as lixeiras das ruas da cidade avisam quando estão cheias e precisam ser limpas. Dessa forma, o custo com o abastecimento de combustível dos caminhões de lixo é reduzido, a partir da criação de rotas de coletas mais eficientes, onde o caminhão de lixo passa apenas nas lixeiras que necessitam ser limpas [13]. Outro exemplo é a cidade de Santander na Espanha que investiu na infraestrutura de mais de 20000 sensores de diversos tipos de medições, além de toda a infraestrutura de comunicação entre esses sensores. Além disso também foi criada uma infraestrutura de *cloud computing*, *fog computing* em dispositivos *gateway* espalhados pela cidade, além dos módulos receptores de sensores espalhados por toda a cidade. Para o usuário final existe um aplicativo multi-plataforma sobre a cidade, onde é possível verificar as vagas de estacionamento nos principais bairros, ruas com o trânsito mais intenso com a possibilidade de criação de rotas mais rápidas com base no sensoriamento das vias, nível de ruído e qualidade do ar nas ruas da cidade, entre outras diversas informações. Além disso, também é possível informar um defeito de iluminação pública ou rede hidráulica, entre diversos tipos de serviços [18].

Com a intenção de criar um sistema de iluminação pública para tornar mais eficiente o consumo de energia elétrica nos postes de luz, foi desenvolvida a simulação de um cenário onde existem diversos postes de luz, cada um embarcando um sensor de movimento e um sensor de temperatura e umidade. Em cada poste existe um microcontrolador responsável por receber os dados desses sensores, além de estabelecer a comunicação com o FaaS4IoT, a camada de *software* onde são publicados os dados produzidos pelos sensores dos dispositivos. Cada microcontrolador possui em sua memória um código que solicita a criação de um canal de comunicação MQTT com a camada virtual de *fog computing* através do endereço URI do dispositivo FaaS4IoT, além da porta de rede para realizar a comunicação e também os métodos responsáveis por realizar o recolhimento dos dados dos sensores. Na camada de *edge computing* ainda no dispositivo, sem intervenção da camada de *fog*, é executado um processamento de baixa complexidade computacional, onde é alterada a luminosidade da lâmpada ao ser detectado um movimento no sensor. Cada vez que o sensor detecta um movimento, é então acionada a rotina que aumenta a luminosidade da lâmpada por padrão sempre em 75%, para 100% e mantém nesse estado de luminosidade máxima pelo período de 2 minutos, caso não forem detectados mais movimentos. Assim cada poste de luz consegue controlar a sua luminosidade de acordo com a necessidade de acionamento, a luminosidade que anteriormente estaria sempre em 100%, mesmo nas situações onde nenhuma pessoa estivesse próxima ao poste, agora permanece por padrão em capacidade reduzida, como consequência proporciona uma economia no custo de energia nos períodos que o poste está com a luminosidade baixa.

O FaaS4IoT com o monitoramento dos dados dos sensores de movimento e luminosidade consegue produzir informações através do processamento desses dados e assim produzir informações consistentes sobre a real economia de energia. O FaaS4IoT para monitorar a temperatura de sua área de cobertura, solicita esse dado uma vez por hora para seus sensores, tendo em vista que as alterações de temperatura não ocorrem com tanta frequência. Se for detectada uma alteração de temperatura maior do que 1ºC, o dispositivo por padrão publica essa alteração no *broker server* do FaaS4IoT pelo canal de comunicação constantemente aberto MQTT.

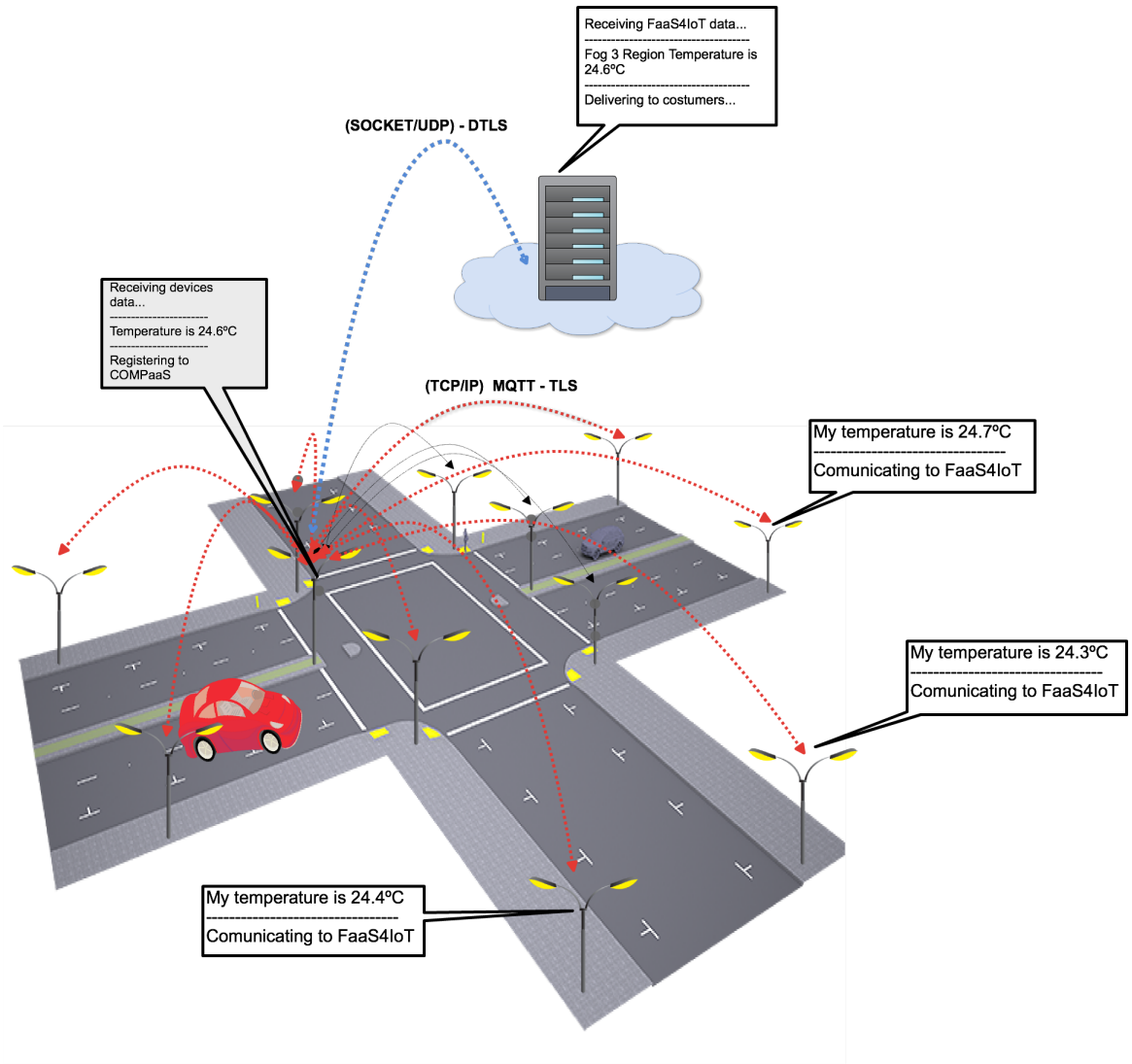

Figura 4.1 – COMPaaS, FaaS4IoT e edge em um cenário de iluminação pública.

O FaaS4IoT ao receber as temperaturas de todos os postes de luz, calcula a média dos valores, estabelecendo a temperatura do ambiente de cobertura, informação a qual é registrada diretamente no COMPaaS, como mostra Figura 4.1.

Todas essas informações são enviadas para o COMPaaS através do canal de comunicação UDP estabelecido. O COMPaaS tem a capacidade de gerenciar diversos FaaS4IoT que estejam conectados a ele. Além disso também gerencia as assinaturas de aplicações e consumidores interessados. Os consumidores podem assinar o recebimento de informações de um FaaS4IoT específico, de diversos, ou de todos, sendo responsabilidade do COMPaaS interpretar e encaminhar os comandos aos respectivos dispositivos *fog*. Para tornar isso possível, o FaaS4IoT cada vez que recebe uma nova conexão de um dispositivo, repassa o perfil do mesmo para o COMPaaS, que assim tem o conhecimento dos endereços de comunicação de todos os dispositivos, bem como os dados que produzem, o dispositivo *fog* com quem se comunicam, protocolos de comunicação, entre outros. Mesmo que o consumidor requisite a temperatura de apenas um poste, o COMPaaS realiza a consulta em seus repositórios de dados e dispositivos, verificando o status do dispositivo e o endereço de

comunicação do FaaS4IoT responsável pelo recebimento de seus dados, tornando possível a entrega dessa informação ao consumidor.

Tabela 4.2 – Testes realizados sem o uso de TLS.

Teste	**FaaS4IoT**	**Edge**	**COMPaaS**
1	1	50	1
2	1	100	1
3	2	100	1
4	2	200	1
5	3	200	1

Em um cenário de cidades inteligentes, o cidadão poderia ter um aplicativo em seu celular no qual monitoraria o consumo de energia dos postes de luz de sua rua, podendo visualizar gráficos da economia nos períodos de redução da luminosidade. Além disso também seria possível consultar a temperatura da rua com a obtenção do valor em tempo real.

Tabela 4.3 – Testes realizados com o uso de TLS.

Teste	**FaaS4IoT**	**Edge**	**COMPaaS**
1	1	50	1
2	1	100	1
3	2	100	1
4	2	200	1
5	3	200	1

A primeira fase de testes concentrou-se em avaliar a eficiência dos métodos utilizados na implementação do FaaS4IoT, onde foram realizados testes com o MQTT, responsável pela comunicação, e o protocolo de segurança TLS aplicado na rede TCP. Diversos testes foram realizados com diferentes números de dispositivos, sem a utilização do protocolo TLS, como mostra a Tabela 4.2. Já a Tabela 4.3 mostra os testes realizados com o uso do protocolo TLS. As tabelas dos testes executados com e sem o uso do protocolo de segurança TLS, apresentam o número de dispositivos simulados em cada camada da arquitetura. A utilização do MQTT possibilita a execução do protocolo de *handshake* entre cliente e *broker server*, o qual se concluído com sucesso, abre uma sessão com segurança TLS entre as duas plataformas. Além disso, foram realizados testes a fim de se validar a eficiência na consulta de valores no banco de dados utilizado para armazenar os dados no FaaS4IoT. Os testes foram realizados controlando o tempo de requisição de informações pela aplicação, até a entrega da informação pelo COMPaaS. A Tabela 4.4 mostra os testes realizados e o número de valores consultados em cada respectivo teste. Com a realização desses testes é possível determinar se o sistema atende ao primeiro objetivo dos testes da arquitetura implementada, que tem a intenção de validar se mesmo com a utilização dos métodos de comunicação, segurança e armazenamento na camada de *fog computing*, a eficiência do sistema é melhorada.

Uma das etapas dos testes foi realizada sem a utilização do FaaS4IoT, pois teve como objetivo verificar os tempos de conexão, comunicação, processamento e entrega da informação a aplicação, com a execução dentro de uma arquitetura tradicional dos sistemas IoT. Para isso foram

Tabela 4.4 – Consulta de dados no banco

Teste	**Numero de dados consultados**
1	1
2	10
3	100
4	250
5	500
6	1000

conectados os variados números de dispositivos dos respectivos testes com o COMPaaS. Os tempos obtidos nas execuções dos diversos testes serão discutidos posteriormente com os demais resultados dos testes de validação dos métodos.

Outra etapa dos testes teve a intenção de validar os conceitos de arquitetura híbrida de *fog* e *edge computing*, como segurança e confiabilidade, baixo tempo de comunicação entre as camadas da arquitetura, além do menor volume de comunicação com o *middleware* principal possível. Dessa forma, para se validar a segurança foram realizados testes tentando conectar dispositivos sem o certificado de segurança TLS e outros com a chave criptográfica errada, no qual o FaaS4IoT deve impedir esse tipo de conexão, pois pode ser um ataque que tiraria a confiabilidade do sistema. Também foram realizados testes monitorando os tempos de comunicação entre as camadas do sistema, os resultados dos tempos serão discutidos na próxima subseção do capítulo 4. Isso finaliza as etapas dos testes sendo possível validar os dois objetivos principais e estabelecer o real ganho de eficiência e desempenho do sistema através da utilização da arquitetura híbrida.

4.3.1 Discussão

O COMPaaS diferente do FaaS4IoT, não utiliza o MQTT para estabelecer a comunicação entre a camada de dispositivos e o seu *middleware* principal de processamento, pois utiliza dois protocolos de comunicação, no primeiro deles, o COMPaaS disponibiliza um *web service* REST no qual pode executar os comandos *join(), subscribe(), unsubscribe(), start()* e *stop()* nos dispositivos, e o segundo canal é um *socket* UDP no qual os dispositivos entregam seus dados produzidos.

Para serem executados os primeiros testes de comparação entre os dois modelos de arquitetura de IoT, foram criados os respectivos números de dispositivos, cada um recebendo um endereço URI único, bem como a porta de comunicação também exclusiva para cada dispositivo. Posteriormente foram realizadas as conexões, primeiro apenas com o COMPaaS, recolhendo os tempos de conexão com os dispositivos, bem como a realização de testes de produção de dados nos sensores dos dispositivos, entrega dos dados no canal *socket* UDP do COMPaaS, processamento, até por fim a entrega das informações em uma aplicação requisitante. Nesse primeiro momento onde apenas os tempos de conexão, produção, transporte, processamento e entrega importavam, o dispositivo simulava um sensor de temperatura obtendo o valor de graus Celsius e comunicando o mesmo a cada 10 segundos com o COMPaaS. Segundo, foram realizadas as conexões dos mesmos

números de dispositivos *edge*, produzindo os mesmos dados de temperatura e publicando os dados produzidos no *broker server* do FaaS4IoT, que com os dados de todos os dispositivos determina a temperatura média do ambiente e entrega essa informação para a aplicação requisitante.

Tabela 4.5 – Tempo de conexão dos dispositivos ao COMPaaS e ao FaaS4IoT

teste	**Dispositivos**	**COMPaaS**	**FaaS4IoT**
1	1	8235	78
2	10	11037	136
3	50	17638	831
4	100	23457	4060
5	200	29503	15733

Na Tabela 4.5 são apresentados os tempos de conexão de diferentes números de dispositivos com o COMPaaS e com o FaaS4IoT. Os valores são apresentados em milissegundos e foram determinados com a medição do tempo entre a inicialização dos dispositivos tanto no COMPaaS como no FaaS4IoT, mais o envio de um comando dessas plataformas para os dispositivos solicitando o recolhimento do valor do sensor de temperatura, tempo de sensoriamento no dispositivo, transporte, seja por canal UDP ou MQTT, e por fim a entrega do dado na plataforma, sendo então calculado e determinado o tempo da execução dessas tarefas nas diferentes plataformas.

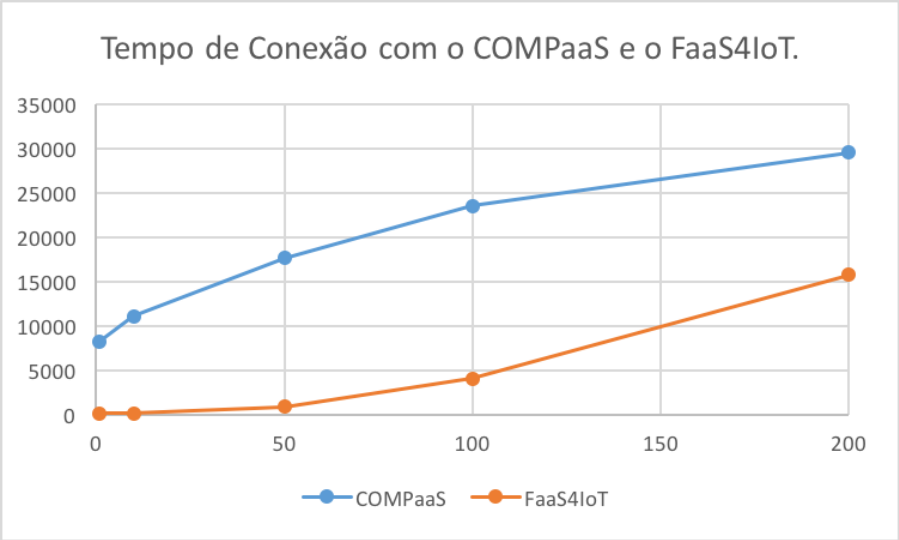

Figura 4.2 – Tempo de Conexão com COMPaaS e FaaS4IoT.

O gráfico criado com esses resultados pode ser visto na Figura 4.2, onde, ao analisarmos podemos visualizar que o tempo de conexão de diferentes números de dispositivos com o protocolo de comunicação MQTT tende a aumentar cada vez que um novo dispositivo é conectado a rede. Entretanto, considerando que o FaaS4IoT é responsável apenas pela camada de *fog computing* em redes locais de sensores, dificilmente com mais de 500 sensores, o tempo de conexão, inicialização, seguido de todas as etapas seguintes até a entrega da informação na aplicação, foi consideravelmente mais eficiente com a utilização do MQTT, em comparação com um protocolo de comunicação também muito eficiente e confiável amplamente utilizado em sistemas tradicionais IoT. O MQTT

tem vantagem na comunicação entre dispositivos de ambiente distribuído, pois é um protocolo de comunicação especificamente desenvolvido para esses tipos de dispositivos IoT, espalhados em um ambiente distribuído. O MQTT comprova a vantagem de sua utilização nesse ambiente também no requisito de armazenamento, visto que o código e biblioteca utilizados para a criação do *MQTT Client* no *software* do dispositivo, acresce entre 3 e 7 KB no tamanho final do arquivo, exatamente como era informado na documentação do MQTT [31].

O MQTT utiliza a porta de comunicação TCP/IP como padrão para a publicação e assinatura dos dados. Sobre o canal de comunicação TCP/IP pode ser aplicado o protocolo de segurança com o objetivo de garantir a privacidade e a integridade dos dados em uma comunicação entre duas camadas da arquitetura, *fog edge computing*. Aplicando o método de segurança é necessário que o protocolo de *handshake* seja executado na primeira conexão entre o dispositivo e a plataforma. O protocolo de *handshake* tem o objetivo de fazer a troca de informações entre as camadas a fim de se entrar em um acordo criptográfico, já tendo sido executada a verificação dos certificados, bem como as chaves privadas utilizadas para a confirmação, sendo possível concluir que é uma conexão segura e assim habilitar a troca de mensagens. A Tabela 4.6 apresenta os tempos dos testes executados a fim de se comparar a utilização do protocolo de segurança e o real impacto dessa utilização no desempenho final do sistema. Os testes foram realizados conectando os diversos números de dispositivos ao FaaS4IoT. Em virtude da limitação de emulação de processadores da plataforma QEMU, foi determinado o valor máximo de dispositivos conectados ao FaaS4IoT em 200. Para a realização dos testes com mais de 200 sensores, foram utilizados mais de um dispositivo embarcando o FaaS4IoT, com suas inicializações ocorrendo em execução paralela. No caso de 600 dispositivos foram utilizadas três instâncias FaaS4IoT, cada uma comunicando-se com no máximo 200 dispositivos. O tempo contabilizado no teste foi obtido entre a conexão e o recebimento do primeiro dado, concluindo um ciclo de recebimento.

Tabela 4.6 – Tempo de comunicação com o TLS

teste	**Dispositivos**	**Com TLS**	**Sem TLS**
1	50	14065	831
2	100	53581	4060
3	200	135013	15733
4	400	138945	16244
5	600	135781	15927

A análise dos resultados pode ser feita com o gráfico da Figura 4.3 onde é apresentado um comparativo entre os tempos de conexão com diferentes números de dispositivos, utilizando e não utilizando o protocolo de segurança TLS. É possível verificar que o tempo de conexão utilizando o protocolo de segurança TLS é realmente maior quando não utilizado. Esse tempo maior de conexão é justificado pelas diversas etapas de processamento e trocas de informações realizadas com o protocolo de *handshake* na primeira comunicação no canal de segurança.

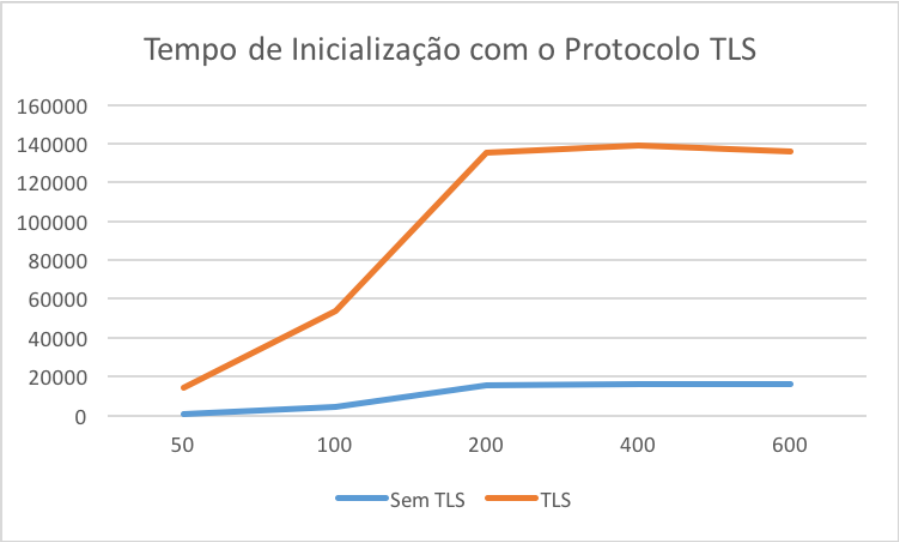

Figura 4.3 – Tempo de inicialização com o uso do TLS.

Por outro lado, o gráfico da Figura 4.4, apresenta os tempos obtidos monitorando o tráfego de 5 pacotes de mensagens entre o Faas4IoT e o dispositivo, incluindo o protocolo de *handshake*, solicitação e recebimento de dados utilizando o TLS.

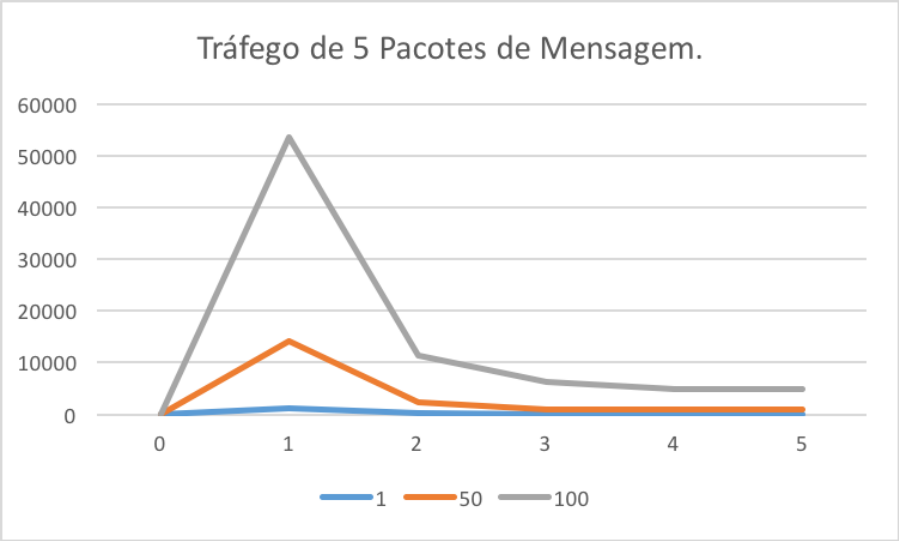

Figura 4.4 – Tempo de comunicação de 5 Pacotes de Mensagens .

Analisando os gráficos é possível se concluir que a utilização do protocolo de segurança TLS não compromete o desempenho e eficiência do sistema, tendo em vista que o protocolo de *handshake* é realizado apenas na primeira conexão que estabelece o canal seguro, dessa forma o tempo de comunicação com a segurança TLS é comprometido apenas no tráfego do primeiro pacote de mensagem. Do tráfego do segundo pacote em diante o tempo de comunicação é mantido praticamente o mesmo das execuções sem o uso do TLS independente da quantidade de dispositivos conectados.

Por fim, com o objetivo de validar o outro requisito determinado para a criação de uma arquitetura de *fog* e *edge computing* eficiente, foi realizado um teste no banco de dados *lightweight*

utilizado para armazenar os dados na camada de *fog computing*. A Tabela 4.7 apresenta os testes realizados a partir de uma requisição enviada pela aplicação diretamente para o FaaS4IoT. Considerando que o sistema esteja sendo executado e armazenando os dados de pelo menos 1000 leituras para atender a todas as solicitações, foi desenvolvida uma aplicação que repassa um XML para o FaaS4IoT, no qual informa o número de valores os quais deseja ter acesso, o tipo de dado, que pode ser, temperatura ou umidade e o formato no qual deseja receber as informações, podendo ser XML ou JSON. O FaaS4IoT por sua vez interpreta esse XML e realiza as consultas no banco de dados para atender as requisições. Os valores encontrados são então convertidos para o formato solicitado pela aplicação e então é enviado o pacote com a mensagem de retorno, concluindo a obtenção do tempo de consulta.

Tabela 4.7 – Tempo de Consulta de dados no banco de dados HSQLDB

numero	**tempo (ms)**
1	43
10	59
100	77
250	114
500	187
1000	290

Analisando os tempos da Tabela 4.7 é possível verificar que mesmo quando o número de valores consultados é dobrado, o tempo de consulta não segue o mesmo padrão, ou seja, o HSQLDB consegue gerenciar, administrar e entregar os dados na camada de *fog computing* de forma que não compromete o desempenho e a eficiência final do sistema atendendo ao último requisito imposto para a criação de uma arquitetura híbrida de *fog* e *edge computing* capaz de melhorar ainda mais os sistemas tradicionais IoT.

4.4 Cenário 2 - Estufa Agrícola Autônoma

A necessidade de produzir mais em períodos climáticos desfavoráveis fez com que agricultores e pesquisadores procurassem meios de proteger as plantas dos danos causados pelas intempéries. Uma grande parte do território brasileiro possui períodos favoráveis para o plantio de hortaliças, mas há épocas do ano que as condições climáticas não favorecem o seu desenvolvimento. Com o emprego da estufa agrícola, é possível produzir durante todo o ano, independente das condições climáticas externas. A estufa agrícola reproduz um cenário em constante inovação, além de ser um grande ambiente de testes com a utilização de novas tecnologias e métodos de plantio. A agricultura em ambiente fechado é considerada a revolução da agricultura para o futuro, pois em ambiente fechado é possível manter e controlar as condições climáticas ideais para o crescimento de diversas culturas. A utilização de sensores climáticos espalhados pelo ambiente em comunicação com o sistema de exaustão e hidráulico, possibilita que a estufa esteja sempre na temperatura programada, além do solo estar sempre com a umidade relativa ideal para o melhor crescimento das plantas [42]. A estufa

por ser coberta ainda previne pragas e doenças nas plantas, além de chuvas de granizo, tempestades, entre outros.

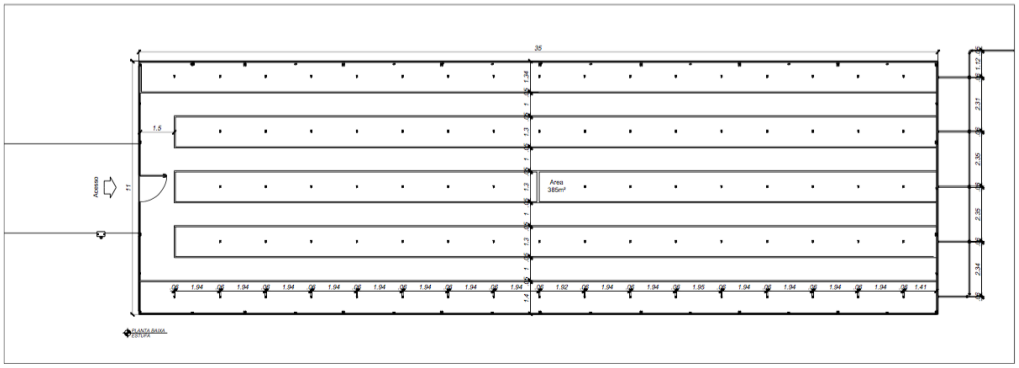

Figura 4.5 – Planta baixa.

O segundo cenário de simulação reproduz exatamente uma estufa agrícola com diversos sensores espalhados pelo ambiente, representados nos pontos pretos dentro das caixas de terra, como mostra a planta baixa da Figura 4.5, monitorando a temperatura e umidade, além da umidade relativa do solo. Os dados produzidos nesse sensoriamento possibilitam que a plataforma FaaS4IoT atue no sistema de exaustão e sistema de irrigação. Utilizando padrões de plantio definidos em pesquisas agrícolas é possível que se crie um ambiente muito mais favorável para as plantas independente das condições climáticas externas, além disso, o crescimento das plantas seguindo as condições ideais de temperatura e umidade relativa do solo, maximiza a produção de diversas culturas.

A Figura 4.7 ilustra a estufa agrícola simulada nesse estudo de caso, com dimensões de 35 metros de comprimento, por 11 metros de largura. Nesse espaço existem diversas caixas de terra com as plantas, também é onde se encontram os módulos de sensores, cada um equipado com um sensor de temperatura e umidade e outro de umidade relativa do solo, como mostra a Figura 4.8. Além disso, nas portas da estufa existem sensores de movimento, que ao detectarem movimento acionam o sistema de iluminação e os módulos de atuadores conectados ao sistema de exaustão e hidráulico.

Esses sensores e atuadores espalhados pelo ambiente criam a camada de *edge computing*. Na Figura 4.9 o dispositivo SoC está com a plataforma de *fog computing* FaaS4IoT embarcada, com a qual os dispositivos *edge* comunicam os seus dados. O recebimento dos dados ocorre através do uso do protocolo de comunicações MQTT, no qual é criado o *broker server* no dispositivo SoC utilizado. Os dispositivos *edge*, aqui atuando como clientes MQTT, publicam seus dados produzidos pelos sensores diretamente no *broker server*, através do seu endereço URI, considerando que o mesmo já esteja previamente registrado no dispositivo *edge*. A Figura 4.6 mostra o método de conexão configurado no dispositivo para ser executado toda vez que o mesmo for conectado à energia. O método *MQTT Connect* quando executado estabelece a comunicação entre dispositivo e o *broker server* de forma facilitada com a possibilidade de já iniciar com o protocolo de segurança sendo executado em paralelo. Isso é possível pois o protocolo de comunicação MQTT opera utilizando o canal de rede TCP/IP no qual também pode ser aplicado o protocolo de segurança TLS, tornando

```
 8  void MQTT::connect(const char* server_host, uint32_t port, boolean security)
 9  {
10    uint16_t crc;
11    crc = client->request(CMD_MQTT_CONNECT, 0, 0, 4);
12    crc = client->request(crc,(uint8_t*)&remote_instance, 4);
13    crc = client->request(crc,(uint8_t*)server_host, strlen(server_host));
14    crc = client->request(crc,(uint8_t*)&port, 4);
15    crc = client->request(crc,(uint8_t*)&security, 1);
16    client->request(crc);
17  }
```

Figura 4.6 – Sensores de temperatura e umidade relativa do solo.

a comunicação entre essas camadas *edge* e *fog* da arquitetura segura e privada. Também para ser realizado o gerenciamento e armazenamento de todos os dados produzidos nos sensores e recebidos no *broker server* do FaaS4IoT, está o gerenciador de banco de dados HSQLDB. Também serão realizados testes com a intenção de se validar a consulta de informações, verificando se o tempo de execução não compromete o desempenho do sistema. Além do tempo de resposta na execução de uma consulta, o gerenciador de bancos de dados também deve ser eficiente no armazenamento de dados, tendo em vista que o FaaS4IoT foi desenvolvido para ser executado em dispositivos SoC, muitas vezes com limitações de capacidade de memória.

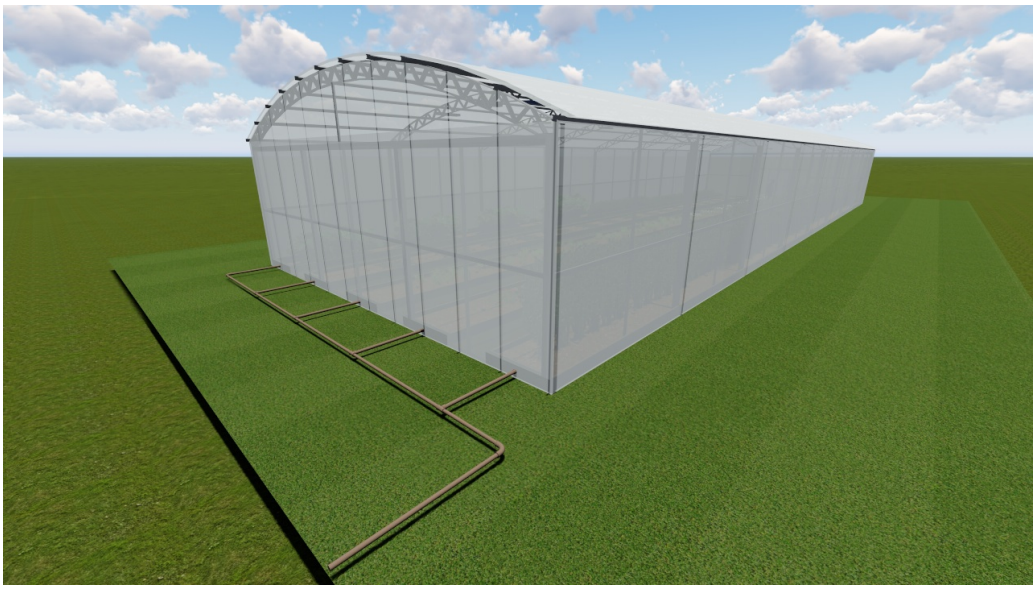

Figura 4.7 – Estufa

Alguns dos testes realizados nesse cenário do agronegócio, são os mesmos testes executados no ambiente de cidade inteligente. O objetivo da realização dos mesmos testes em diferentes tipos de aplicação do sistema FaaS4IoT, é de validar a utilização da arquitetura híbrida de *fog* e *edge computing* em diferentes cenários, seguindo diferentes padrões de controle do ambiente, além de diferentes dispositivos produzindo dados e se comunicando. A análise dos resultados desses testes também torna possível a validação da plataforma no sentido da heterogeneidade dos dispositivos em cenários de aplicação de sistemas de IoT. Os testes realizados a fim de se validar a eficiência e o real impacto de tempo com a utilização do protocolo de segurança TLS foram realizados incluindo os métodos de *handshake* tanto no *broker server*, quanto nos dispositivos *edge*, atuando como clientes

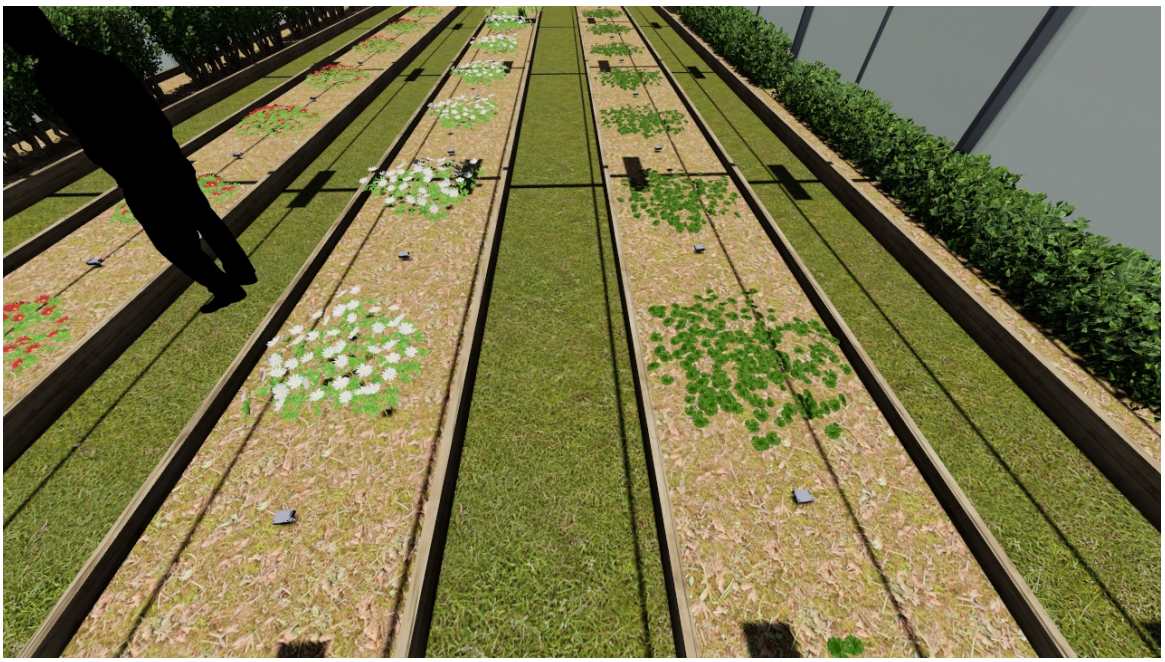

Figura 4.8 – Sensores de temperatura e umidade relativa do solo.

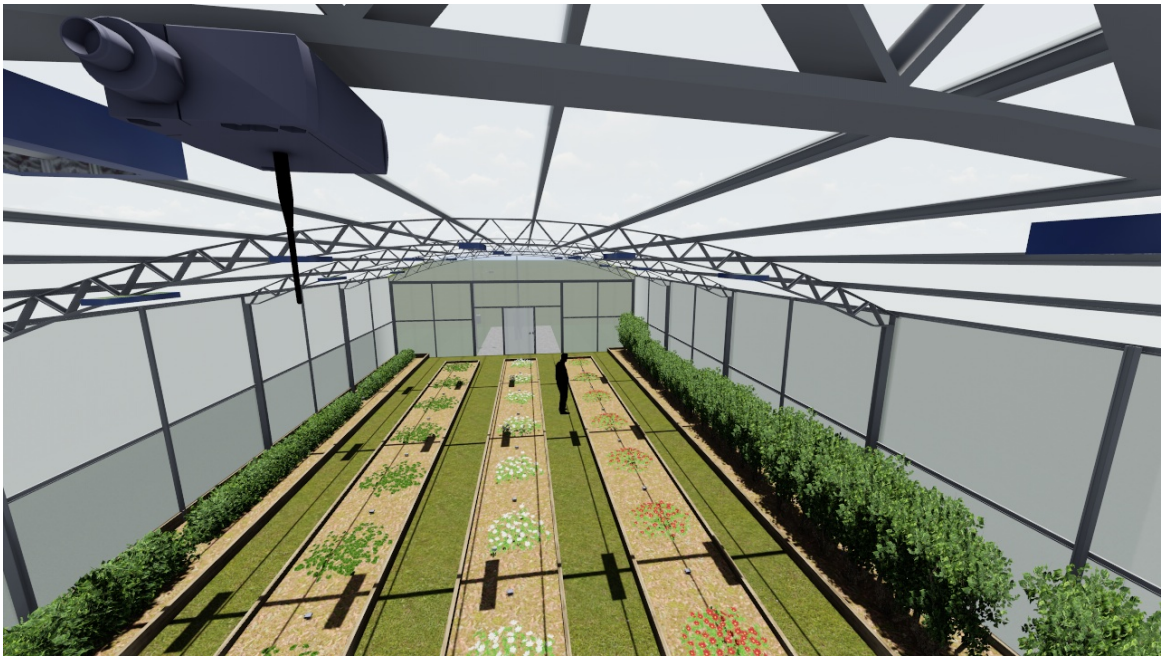

Figura 4.9 – Dispositivo SoC embarcando o FaaS4IoT.

nesse processo de comunicação. O protocolo de *handshake* é executado apenas na primeira comunicação entre as duas camadas da arquitetura, após a confirmação da verificação dos certificados, o canal é validado e mantido aberto e protegido. A Figura 4.10 apresenta o protocolo de *handshake*, que consiste na troca de mensagens e informações a fim de se confirmar o acordo criptográfico entre as duas partes, que caso não encontre os parâmetros programados previamente antes da sessão a ser executada, todo o processo de *handshake* descrito acima terá de ser invalidado e desfeito.

O protocolo de *handshake* é iniciado com a troca de mensagens de *"hello"* entre cliente e servidor para chegar a um acordo se o mesmo algoritmo criptográfico esta sendo usado em ambas

as camadas, no caso de ser um algoritmo diferente, o processo é interrompido. Posterior a essa etapa, acontece a troca de certificados para permitir que o cliente e o servidor se autentiquem. Essa informação é então passada para a camada de registro, informando os parâmetros de segurança negociados. A última etapa do processo é permitir que cliente e servidor possam verificar se ambos estão usando os mesmos parâmetros de segurança e verificar se a negociação não foi interceptada por um atacante. Com isso é então estabelecido um canal TCP/IP seguro para trafegar os pacotes dos métodos *publish/subscribe* da comunicação MQTT.

Source	Destination	Protocol	Info
Client	Server	TLS1.2	Client Hello
Server	Client	TLS1.2	Hello verify request
Client	Server	TLS1.2	Client Hello
Server	Client	TLS1.2	Server Hello
Server	Client	TLS1.2	Certificado (fraguimento)
Server	Client	TLS1.2	Certificado (remontado), troca de chave do servidor, requisição do certificado, Server Hello concluído
Client	Server	TLS1.2	Certificado (fraguimento)
Client	Server	TLS1.2	Certificado (remontado), troca de chave do cliente, certificado verificado
Client	Server	TLS1.2	Dados
Server	Client	TLS1.2	Dados
Client	Server	TLS1.2	Dados

Figura 4.10 – Protocolo de handshake entre cliente e servidor.

Os testes executados com e sem o uso de TLS podem ser vistos nas Tabelas 4.8 e 4.9, que apresentam os testes realizados e o número de dispositivos simulados em cada camada da arquitetura, a fim de se validar a utilização de mecanismos de segurança no canal de comunicação TCP entre a camada de *fog* e *edge computing*. A comunicação entre FaaS4IoT e COMPaaS ocorre da mesma forma que no cenário de iluminação pública, através do uso do protocolo de comunicação UDP, protegido pelo protocolo de segurança DTLS já executado por padrão no COMPaaS.

Tabela 4.8 – Testes realizados sem o uso de TLS.

teste	**FaaS4IoT**	**Edge**	**COMPaaS**
1	1	50	1
2	1	100	1
3	2	100	1
4	2	200	1
5	3	200	1

Tabela 4.9 – Testes realizados com o uso de TLS.

teste	**FaaS4IoT**	**Edge**	**COMPaaS**
1	1	50	1
2	1	100	1
3	2	100	1
4	2	200	1
5	3	200	1

Da mesma forma que os testes executados dentro do cenário de iluminação pública, a primeira etapa dos testes foi realizada sem a utilização do FaaS4IoT, pois teve como objetivo verificar os tempos de conexão, comunicação, processamento e entrega da informação a aplicação que realiza a requisição, seguindo os padrões de uma arquitetura de *middleware* SOA. Para isso foram realizadas as conexões dos variados números de dispositivos dos respectivos testes com o COMPaaS. Os tempos obtidos nas execuções dos diversos testes serão discutidos posteriormente com os demais resultados dos testes de validação dos métodos.

Outra etapa dos testes que será apresentada posteriormente em uma análise comparativa do sistema nos dois cenários de aplicação, teve a intenção de validar os conceitos de arquitetura híbrida de *fog* e *edge computing*, que são: segurança e confiabilidade, baixo tempo de comunicação entre as camadas da arquitetura, além do menor volume de comunicação com o *middleware* principal possível. Por fim, é realizada uma comparação entre o volume de mensagens trafegadas pela rede nas execuções dos testes, utilizando a arquitetura tradicional IoT e utilizando a arquitetura híbrida com *fog* e *edge computing*. Isso finaliza as etapas dos testes sendo possível validar os dois objetivos principais e estabelecer o real ganho de eficiência e desempenho do sistema através da utilização da arquitetura híbrida.

4.4.1 Discussão

Seguindo os mesmos padrões identificados nas execuções dos testes em ambiente de cidades inteligentes, a comparação dos tempos de conexão de diferentes números de dispositivos utilizando as arquiteturas tradicionais de middleware IoT e arquitetura híbrida unindo os conceitos de *edge* e *fog computing* mostrou que a utilização de uma camada virtual de processamento mais próxima dos dispositivos minimiza o volume de comunicação entre dispositivos e *core middleware*. Como a carga principal de comunicação ocorre dentro de rede local, trafegando dados em rede *wireless*, o tempo de transporte dos dados entre as diferentes camadas acontece de maneira muito eficiente, maximizando o desempenho da arquitetura como um todo. Nesse tipo de arquitetura a comunicação com o *core middleware* de processamento ocorre na maior parte das vezes com finalidade de registro, entregando as informações para a plataforma, de forma que o controle e conhecimento de toda a rede seja possível, tendo em vista que o camada de *middleware* tem conhecimento de todos os nodos FaaS4IoT e assumindo que essa plataforma esteja sendo executada em um servidor, com grandes capacidades de armazenamento e poder para a execução de cálculos de maior complexidade computacional. Nesse nível da arquitetura, o uso de mecanismos de *Data Mining* com os diversos tipos de dados disponibilizados, torna possível a produção de informações mais complexas e contextualizadas.

Os resultados obtidos comparando os tempos nos dois tipos de arquiteturas pode ser visto no gráfico apresentado na Figura 4.11 e mostram que o MQTT é mais eficiente em ambiente distribuído em comparação com o protocolo de comunicação *web service* REST executado para trafegar dados com os dispositivos no COMPaaS.

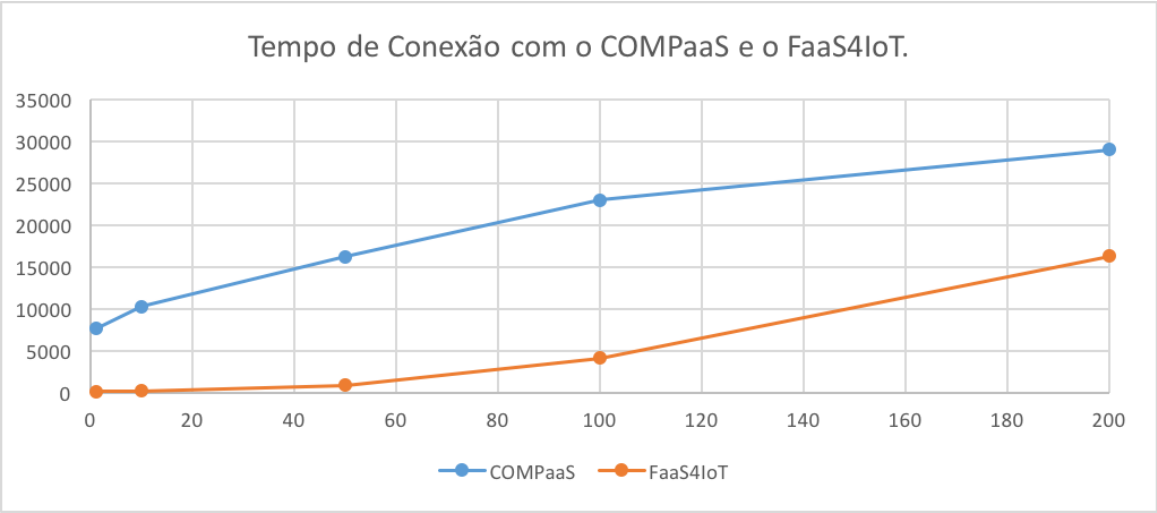

Figura 4.11 – Tempos em comparação com os dois tipos de arquitetura de sistemas IoT

A variação mínima nos tempos de execução desse teste acontecem apenas pela diferença nos códigos fontes que estavam sendo executados, tando na camada de *fog*, como na camada de *edge computing* em virtude das adaptações necessárias para simular esses diferentes cenários. A Tabela 4.10 mostra os tempos em milissegundos das execuções dos testes.

Tabela 4.10 – Tempo de conexão dos dispositivos ao COMPaaS e ao FaaS4IoT

teste	**Dispositivos**	**COMPaaS**	**FaaS4IoT**
1	1	7587	83
2	10	10233	147
3	50	16221	855
4	100	22971	4122
5	200	28973	16261

Diferente dos testes realizados no primeiro cenário, foram executados testes na intenção de validar os mecanismos de regras e processamento de eventos complexos utilizados para prover autonomia na tomada de decisões na camada de *fog* e *edge computing*. Para serem desenvolvidos os métodos de controle de eventos complexos foi utilizado o *Drools Fusion* [24], que é o módulo responsável pela adição de capacidades de processamento de eventos à plataforma. Evento, é um registro de uma alteração significativa de estado no domínio de aplicativo em um determinado ponto no tempo. Suportar Processamento de Eventos Complexos, porém, é muito mais do que simplesmente entender o que é um evento. Os cenários do CEP compartilham várias características comuns e distintivas, entre elas, normalmente é utilizado para processar enormes volumes de eventos, mas apenas uma pequena porcentagem dos eventos são de real interesse. Os eventos são geralmente imutáveis, uma vez que são um registro de mudança de estado, além disso, as regras e consultas sobre eventos devem ser executadas em modos reativos, isto é, reagem à detecção de padrões de eventos, onde geralmente há relações temporais fortes entre eventos relacionados. Com base nestas características gerais comuns, o *Drools Fusion* atende um conjunto de metas para dar suporte ao processamento de eventos complexos de forma apropriada: permitir detecção, correlação, agregação

e composição de eventos; Suportar processamento de fluxos de eventos; Suportar restrições temporais para modelar as relações temporais entre eventos, entre outros. O biblioteca *Drools* implica na utilização de uma série de regras a fim de se monitorar o ambiente e reagir de forma apropriada na ocorrência do evento. A Figura 4.12 apresenta como exemplo, uma das regras utilizadas para realizar os testes de monitoramento da estufa. Na regra programada, ao ser detectado que a temperatura média do ambiente é maior do que 24° Celsius, é então disparado um evento ativando o sistema de exaustão, para assim controlar a temperatura.

```
29  rule "Max_Temp"
30  when
31      $event: EventType( (getEventType.getAttributeName(Temp) >= maxTemp("24") )
32      $event: EventType( (getEventType.getStatus().getAttributeName(ExstSys) != ExstSys_ONLINE) )
33  then
34      System.out.print("Evento ativado: " + event.getType());
35      exstSys = ex.online();
36      retract($event);
37  end
```

Figura 4.12 – Tempos em comparação com os dois tipos de arquitetura de sistemas IoT

Com o sistema monitorando a ocorrência de eventos, a tomada de decisões é autônoma e independente de comunicação com a plataforma de *middleware*. A Figura 4.13 apresenta a captura de tela do gerenciador de dispositivos no momento da execução do teste de desconexão, no qual foi simulado algum momento que a comunicação entre COMPaaS e FaaS4IoT não é viável. Observando a execução do teste, é possível identificar que o FaaS4IoT continuou em funcionamento e monitorando os dispositivos *edge* independente da conectividade com a plataforma de controle. Os dados e operações executadas no FaaS4IoT nos períodos sem conexão, são armazenados e no momento que o canal de comunicação é novamente ativado, é executada a operação de registro no COMPaaS, que compara o conteúdo do banco de dados para verificar quais os dados transmitidos estão desatualizados.

Id	ProfileName	Status	URI	Port	Info
1	COMPaaS	Offline	192.168.0.10	4554	Disconnected;
2	FaaS4IoT	Online	192.168.0.13	8883	Receiving data from sources/ Event monitoring started;
3	Dev01Temp	Online	192.168.0.14	8883	Generating active;
4	Dev02Temp	Online	192.168.0.15	8883	Generating active;
5	Dev03Atu	Online	192.168.0.16	8883	Generating active;
6	Dev04TempAtu	Online	192.168.0.17	8883	Waiting to communicate;
7	Dev05Exst	Online	192.168.0.18	8883	Generating active;
8	Dev06Temp	Online	192.168.0.19	8883	Transmitted;
9	Dev07Temp	Stopped	192.168.0.20	8883	Warning: Device stopped by user;

Figura 4.13 – Gerenciador de dispositivos no momento em que o COMPaaS esta offline.

O HSQLDB possui diferentes formas de armazenamento, na primeira delas, os dados são armazenados temporariamente em memória RAM, esse tipo de armazenamento é utilizado como *buffer* para valores usados mais frequentemente, tendo em vista que o acesso a memória RAM acontece de forma mais rápida do que em disco. A segunda forma de armazenamento é em disco, os os dados são guardados de forma permanente, porém com o acesso mais lento, relacionado as capacidades do *Hard Drive* (HD) utilizado. Diferente do teste executado no primeiro cenário, a

intenção dos testes no banco de dados foi de se comparar os tempos de consulta utilizando as duas formas de armazenamento. O teste de comparação no tempo de consulta de valores no banco de dados foi realizado e os resultados são apresentados no gráfico da Figura 4.14, a unidade de medida dos valores da tabela é milissegundo (ms).

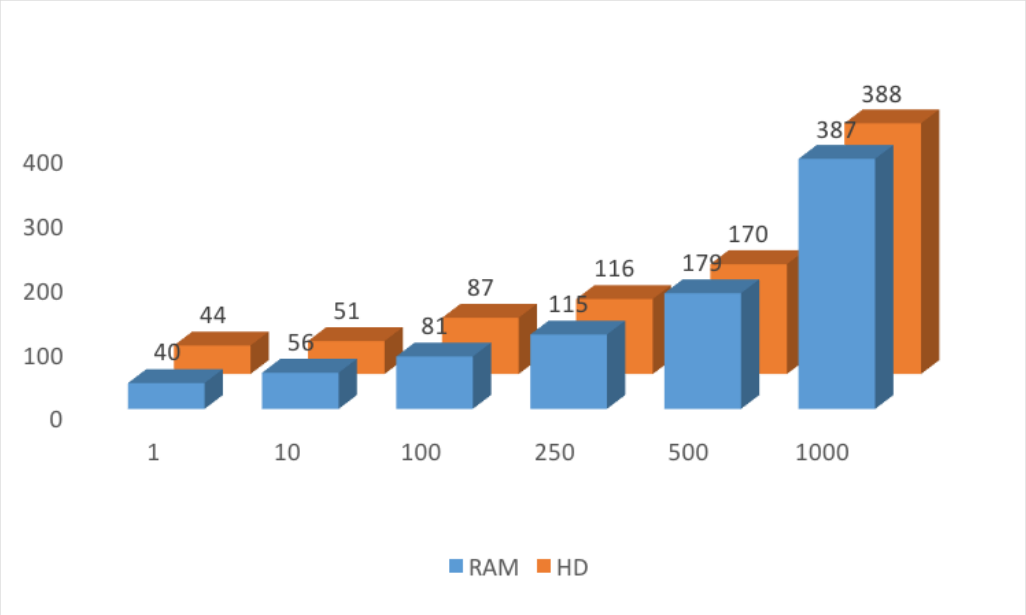

Figura 4.14 – Tempo de consulta de valores no HSQLDB.

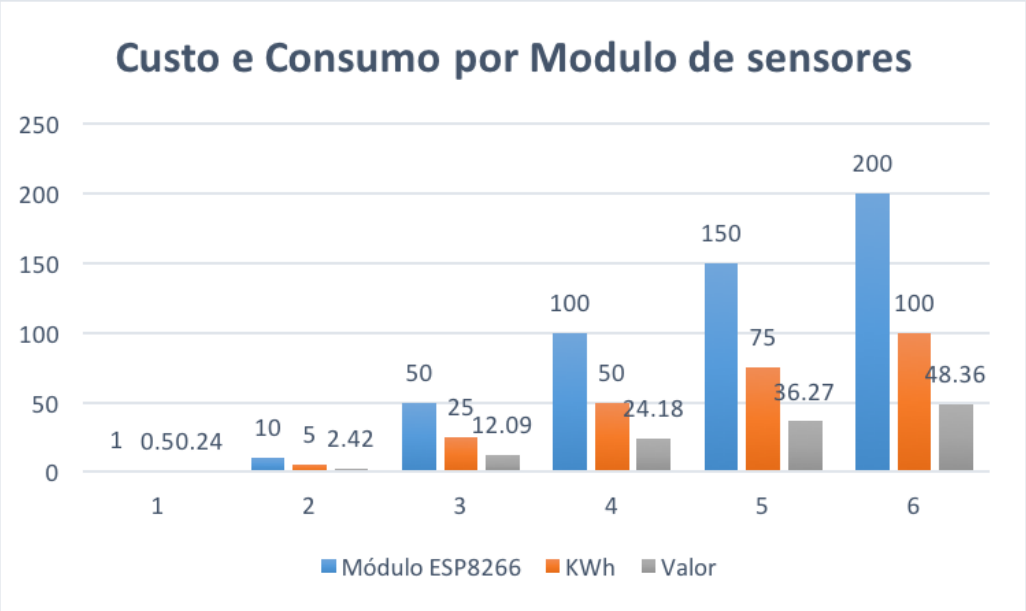

Figura 4.15 – Custo e Consumo com a Utilização de Módulos de Sensores

Diversos dispositivos estão disponíveis no mercado de produtos eletrônicos, entre eles, o ESP8266 [16], um microcontrolador que opera em 80MHz, trabalha com a arquitetura RISC de 32 bits, além disso possui 32 KBytes de RAM para instruções, 96 KBytes de RAM para dados 64 KBytes de ROM para *boot* e por fim, uma memória Flash de 512 KBytes. O módulo ESP8266 trabalha

com sinal em 3,3 Volts, utilizando um wattímetro Nissei RS-50 [36], foi possível detectar que um módulo com um sensor de temperatura, umidade e um atuador, no teste um módulo relé de 10A, chega ao pico máximo de 0,5 kWh, ou seja, se existissem 200 módulos ESP em uma estufa, o custo mensal seria de aproximadamente R$48,36. Estudos mostram que apenas a utilização de sensores monitorando a rede hidráulica, já é possível se economizar mais de 43% no consumo de água e aumentar o ganhos em mais de 20% com a melhora na produção [29]. A Figura 4.15 apresenta um gráfico com diversas simulações do custo mensal de variados números de módulos.

A realização dos testes no segundo cenário comprovou que é possível utilizar de forma eficiente o FaaS4IoT em mais de um domínio de aplicação, além de validar a autonomia na tomada de decisões ainda na camada local de processamento e dispositivos.

4.5 Considerações Finais

A realização dos testes no segundo cenário comprovou que é possível utilizar de forma eficiente o FaaS4IoT em mais de um domínio de aplicação, além de validar a autonomia na tomada de decisões ainda na camada local de processamento e dispositivos.

Um testes realizado nos dois cenários validou a confiabilidade no protocolo de segurança TLS utilizado. Foram realizados testes simulando diferentes situações de risco que poderiam acontecer em uma aplicação real da arquitetura. No teste realizados foram configurados quatro dispositivos para solicitarem conexão com o FaaS4IoT, um dos dispositivos, possuindo o certificado e as chaves corretas, portanto estabelecendo a conexão com sucesso, um segundo dispositivo tenta a conexão, porém não possui o certificado e nem as chaves de segurança, o FaaS4IoT portanto, deve recusar a conexão, bem como nos dispositivos três e quatro que possuem, certificado fora dos padrões do FaaS4IoT e chaves de segurança inválidas, respectivamente.

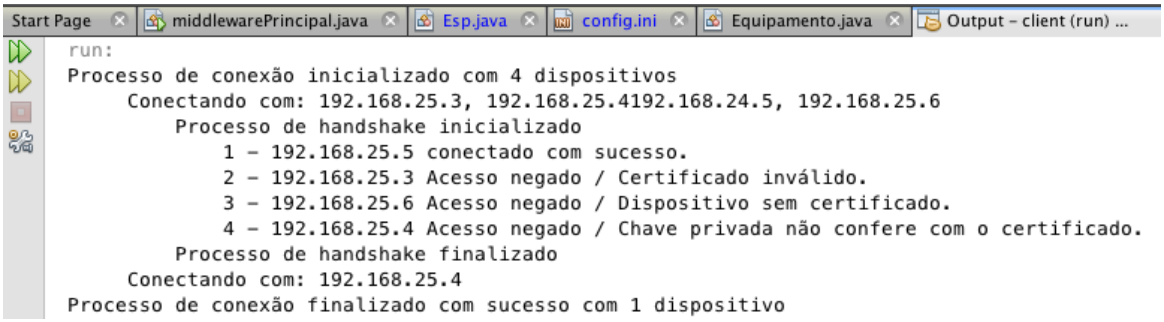

Figura 4.16 – FaaS4IoT Rejeitando a Conexão com Dispositivos Fora dos Padrões.

A Figura 4.16 apresenta as saídas mostradas no sistema, ao detectarem os diferentes tipos de dispositivos e requisitos para a conexão. Nos dois cenários foram realizados diversos testes conforme os descritos, alterando certificados e chaves cinco vezes, a fim de realmente comprovar que o uso do protocolo TLS, mesmo com lento protocolo de *handshake* na primeira comunicação, garante a segurança no canal de comunicação das camadas de *edge* e *fog computing*, sem prejudicar

o desempenho e eficiência do sistema. As informações sobre as solicitações de conexão que aparecem na Figura 4.16 representam o mesmo resultado obtido nos outros diversos testes dentro dos dois cenários, exceto os endereço IP que variam de acordo com o endereço atribuído pelo roteador de rede. Na execução de todos os testes o FaaS4IoT permitiu a conexão de apenas o dispositivo dentro dos padrões de certificado e chaves de segurança e recusou a conexão dos demais dispositivos em virtude de serem uma ameaça ao sistema, que podem comprometer a integridade e confiabilidade da plataforma de forma geral.

5. CONCLUSÃO E PUBLICAÇÕES

Nesse trabalho foi detalhada toda a evolução no planejamento e desenvolvimento de uma arquitetura de *fog computing* e *edge computing* para sistemas IoT, tendo como principal objetivo aproximar de maneira eficiente as camadas consumidoras e provedoras de serviços.

O FaaS4IoT foi planejado e desenvolvido com a intenção de tornar possível algum tipo de processamento já na camada dos dispositivos, ou próximo dos mesmos, ou seja, planejado para ser embarcado em um dispositivo SoC para ser utilizado como *gateway* na rede. Diversos requisitos foram listados a fim de serem cumpridos, entre eles podemos listar:

- Autonomia na tomada de decisões.
- Segurança e confiabilidade no transporte de mensagens entre as camadas da arquitetura.
- Protocolo de comunicação eficiente para dispositivos IoT.
- Armazenamento eficiente de dados na camada de *fog computing*.

A execução dos testes mostrou que o FaaS4IoT é eficiente em diversos domínios de aplicação, além de cumprir aos requisitos elicitados na fase de planejamento. O MQTT atende a eficiência, confiabilidade e praticidade no comunicação entre as camadas de *fog* e *edge computing*. O uso do TLS para manter a segurança no canal de comunicação se mostrou eficaz, mantendo a segurança sem prejudicar o desempenho final da arquitetura. O uso do HSQLDB também atende aos requisitos de eficiência no tempo de consultas ao banco de dados. *Fog computing* e *edge computing* também exigem que sejam cumpridos diversos requisitos, entre eles, estar preparado para trabalhar na casa dos bilhões de dispositivos se comunicando, o qual o FaaS4IoT cumpre, pois cada *broker server* MQTT aceita até cerca de 65000 clientes, e o COMPaaS por sua vez seria responsável por milhares de FaaS4IoT. Outro requisito é arquitetura orientada a eventos, o qual o FaaS4IoT também cumpre através da utilização dos mecanismos do CEP.

A Tabela 5.1 apresenta algumas das principais características de sistemas tradicionais de *middleware* e arquiteturas de *fog computing*, aqui sendo representado pelo uso do FaaS4IoT. As características listadas para comparação são: localização, que diz respeito ao local onde é realizado o processamento dos dados. O tamanho, que diz respeito as capacidades e utilização de espaço físico dos *clusters*, servidores e dispositivos SoC. Execução, representando as configurações e requisitos que precisam ser atendidos para o funcionamento da plataforma. Operação, representando o ambiente de instalação dos dispositivos, bem como a necessidade de intervenção humana para o funcionamento do sistema. A possibilidade de desenvolvimento de aplicações em tempo real, também é listada como uma característica, bem como a conectividade com a Internet e necessidade de largura de banda, que é a característica que representa as necessidades para o funcionamento constante em alta disponibilidade do sistema e as necessidades de largura de banda.

Tabela 5.1 – Comparação entre sistemas tradicionais de *middleware* e o FaaS4IoT.

	Middleware	**FaaS4IoT**
Localização	Centralizado em um pequeno número de servidores.	Distribuído em diversas localizações em amplo espaço geográfico. Os dispositivos FaaS4IoT distribuídos podem ser gerenciados por servidores de *middleware*
Tamanho	*Clusters* e servidores são grandes em tamanho e necessitam de uma infraestrutura sofisticada para funcionamento.	Aplicado em dispositivos SoC que são consideravelmente menores se observados de forma individual, entretanto a utilização de diversos dispositivos SoC, possibilita um processamento mais eficiente que o de um servidor.
Execução	Geralmente requer de métodos sofisticados para funcionamento, além de diversos requisitos de software que precisam ser cumpridos para o correto funcionamento.	Fácil inicialização, o dispositivo SoC possui em sua memória um *script* que coloca em funcionamento o FaaS4IoT e todos os seus recursos sempre que o mesmo é conectado a energia.
Operação	Opera em instalações e ambientes selecionados e totalmente controlados por operadores. Necessita de especialistas para serem realizados eventuais manutenções e reparos.	Opera em ambientes onde tem sua implantação adaptada para o melhor funcionamento nesses locais. O FaaS4IoT precisa de pouca ou nenhuma intervenção humana para o seu correto funcionamento. Pode ser adaptado para pequenas ou grandes aplicações.
Aplicação em tempo real	Requer da utilização de diversos recursos e tecnologias de infraestrutura para que seja possível a execução de aplicações com a necessidade de um retorno imediato em ambiente real.	Ao ser desenvolvida uma aplicação que realiza a assinatura diretamente no *broker server* do FaaS4IoT é possível a execução de aplicações de tempo real, ou pelo menos de aplicações próximas de tempo real, visto que o processamento ocorre no momento do recebimento do dado, já sendo possível a criação de uma informação ou atuação de acordo com alguma requisição da aplicação.
Conectividade com a Internet e necessidade de largura de banda	Requer que os dispositivos e consumidores possuam uma conectividade com a Internet constante durante a utilização dos serviços. Necessidade de largura de banda cada vez maior com a conexão de novos dispositivos se comunicando e produzindo dados.	Pode operar de forma autônoma mesmo que a comunicação com a plataforma de *middleware* não esteja disponível no momento. A carga máxima de trocas de mensagens acontece em ambiente de rede local, o FaaS4IoT repassa ao *middleware* apenas os registros necessários para que o *middleware* tenha o total controle sobre os dispositivos e sistema.

Com a análise da Tabela 2.1 ficam claras todas as vantagens da utilização dos conceitos de *fog* e *edge computing*, além de validar a utilização do FaaS4IoT potencializando o desempenho de toda a arquitetura de forma geral. É importante mencionar que *fog computing* e *edge computing* não são substitutas de *cloud computing*, ou de *middleware*, mas sim, que essas tecnologias complementam uma a outra. São diversas as funções complementares que *fog*, e *edge* são capazes de proporcionar, as quais permitem ao usuário experimentar uma nova geração da computação, e também servem como requisito para que aplicações de tempo real e de baixa latência possam ser executadas nas bordas da rede. Além disso, essa combinação também permite suporte para análises completas de uma grande quantidade de dados na camada principal da rede.

5.1 Publicações

Abaixo estão listados os artigos publicados, escritos durante o período do curso de mestrado:

- **Schenfeld, M. C.; Amaral, L.; Matos, E. D.; Hessel, F. "Arquitetura para Fog Computing em Sistemas de Middleware para Internet das Coisas". Em: Congresso da Sociedade Brasileira de Computação, 2016, pp. 1819–1829.** No qual foi submetido todo o projeto da camada de *fog computing*, além de uma ampla pesquisa sobre trabalhos relacionados tanto na área acadêmica quanto no meio da indústria.
- **Matos, E.; Amaral, L. A.; Tiburski, R. T.; Schenfeld, M. C.; Hessel, F.; de Azevedo, D. "A Sensing-as-a-Service Context-Aware system for internet of things environments". Em: 2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC), 2017.**

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Abdelwahab, S.; Hamdaoui, B.; Guizani, M.; Rayes, A. "Enabling smart cloud services through remote sensing: An internet of everything enabler", *IEEE Internet of Things Journal*, vol. 1–3, June 2014, pp. 276–288.

[2] Al Faruque, M.; Vatanparvar, K. "Energy Management-as-a-Service Over Fog Computing Platform", *IEEE Internet of Things Journal*, vol. PP–99, 2015, pp. 1–1.

[3] Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. "Internet of Things: A Survey on Enabling Technologies, Protocols and Applications", *IEEE Communications Surveys & Tutorials*, vol. PP–99, 2015, pp. 1–1.

[4] Amaral, L. A.; Tiburski, R. T.; de Matos, E.; Hessel, F. "Cooperative middleware platform as a service for internet of things applications". In: ACM Symposium on Applied Computing (SAC), 2015, pp. 488–493.

[5] Atzori, L.; Iera, A.; Morabito, G. "The internet of things: A survey", *Computer Networks*, vol. 54–15, 2010, pp. 2787 – 2805.

[6] Bjelica, M. Z.; Golan, G.; Radovanovic, S.; Papp, I.; Velikic, G. "Adaptive device cloud for internet of things applications". In: Consumer Electronics-China (ICCE), 2014, pp. 1–3.

[7] Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. "Fog Computing and Its Role in the Internet of Things". In: Mobile cloud computing (MCC), 2012, pp. 13–16.

[8] Carboni, D.; Pintus, A.; Piras, A.; Serra, A.; Badii, A.; Tiemann, M. "Scripting a Smart City: The CityScripts Experiment in Santander". In: International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2013, pp. 1265–1270.

[9] Chang, H.; Hari, A.; Mukherjee, S.; Lakshman, T. V. "Bringing the cloud to the edge". In: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2014, pp. 346–351.

[10] Chappell, D. "Enterprise service bus". "O'Reilly Media, Inc.", 2004, 33-49p.

[11] Chiang, M.; Zhang, T. "Fog and IoT: An Overview of Research Opportunities", *IEEE Internet of Things Journal*, vol. 3–6, dec 2016, pp. 854–864.

[12] Conservancy, S. F. "Qemu documentation". Captured in: http://www.qemu-project.org/, fev 2017.

[13] Dastjerdi, A. V.; Buyya, R. "Fog Computing: Helping the Internet of Things Realize Its Potential", *Computer*, vol. 49–8, 2016, pp. 112–116.

[14] de Matos, E.; Amaral, L. A.; Tiburski, R. T.; Schenfeld, M. C.; Hessel, F.; de Azevedo, D. "A Sensing-as-a-Service Context-Aware system for internet of things environments". In: IEEE Annual Consumer Communications & Networking Conference (CCNC), 2017, pp. 1–4.

[15] Doukas, C. "Building Internet of Things with the ARDUINO". CreateSpace Independent Publishing Platform, 2012, 77-83p.

[16] ESP8266. "Esp8266". Captured in: http://www.esp8266.com, jan 2016.

[17] Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. "Hypertext transfer protocol–http/1.1", Relatório Técnico, W3C, 1999, 77-181p.

[18] Filograna, A.; Smiraglia, P.; Gilsanz, C.; Krco, S.; Medela, A.; Su, T. "Cloudification of public services in smart cities the clips project". In: IEEE Symposium on Computers and Communication (ISCC), 2016, pp. 153–158.

[19] Force, I. E. T. "The transport layer security (tls) protocol". Captured in: https://www.ietf.org/rfc/rfc5246.txt, jan 2017.

[20] Garcia Lopez, P.; Montresor, A.; Epema, D.; Datta, A.; Higashino, T.; Iamnitchi, A.; Barcellos, M.; Felber, P.; Riviere, E. "Edge-centric Computing: Vision and Challenges", *ACM SIGCOMM Computer Communication Review*, vol. 45–5, 2015, pp. 37–42.

[21] Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. "Internet of Things (IoT): A vision, architectural elements, and future directions", *Future Generation Computer Systems*, vol. 29–7, 2013, pp. 1645–1660, 1207.0203.

[22] Hamdaqa, M.; Tahvildari, L. "Cloud computing uncovered: a research landscape", *Advances in Computers*, vol. 86, 2012, pp. 41–85.

[23] HyperSQL. "Hypersql". Captured in: http://hsqldb.org, jan 2016.

[24] jBOSS. "Drools fusion documentation". Captured in: https://docs.jboss.org/drools/release/6.2.0.CR3/drools-docs/html/DroolsComplexEventProcessingChapter.html, fev 2017.

[25] Laliwala, Z.; Chaudhary, S. "Event-driven service-oriented architecture". In: Service Systems and Service Management, 2008, pp. 1–6.

[26] Lazaroiu, G. C.; Roscia, M. "Definition methodology for the smart cities model", *Energy*, vol. 47–1, 2012, pp. 326–332.

[27] Lewis, G.; Echeverria, S.; Simanta, S.; Bradshaw, B.; Root, J. "Tactical cloudlets: Moving cloud computing to the edge". In: Military Communications Conference (MILCOM), 2014, pp. 1440–1446.

[28] Li, H.; Wu, Z. "Research on distributed architecture based on soa". In: Communication Software and Networks (ICCSN), 2009, pp. 670–674.

[29] Lichtenberg, E.; Majsztrik, J.; Saavoss, M. "Profitability of sensor-based irrigation in greenhouse and nursery crops", *HortTechnology*, vol. 23–6, 2013, pp. 770–774.

[30] Masip-Bruin, X.; Marín-Tordera, E.; Gómez, A.; Barbosa, V.; Alonso, A. "Will it be cloud or will it be fog? f2c, a novel flagship computing paradigm for highly demanding services". In: Future Technologies Conference (FTC), 2016, pp. 1129–1136.

[31] MQTT. "Mqtt". Captured in: https://mqtt.org, Nov 2016.

[32] Nakamura, Y.; Suwa, H.; Arakawa, Y.; Yamaguchi, H.; Yasumoto, K. "Design and Implementation of Middleware for IoT Devices toward Real-Time Flow Processing". In: Distributed Computing Systems Workshops (ICDCSW), 2016, pp. 162–167.

[33] Nakamura, Y.; Suwa, H.; Arakawa, Y.; Yamaguchi, H.; Yasumoto, K. "Middleware for Proximity Distributed Real-Time Processing of IoT Data Flows". In: International Conference on Distributed Computing Systems (ICDCS), 2016, pp. 771–772.

[34] NetworkSorcery. "Transport layer connection oriented byte stream protocol". Captured in: http://www.networksorcery.com/enp/protocol/tcp.htm, fev 2017.

[35] Nikoloudakis, Y.; Panagiotakis, S.; Markakis, E.; Pallis, E.; Mastorakis, G.; Mavromoustakis, C. X.; Dobre, C. "A Fog-Based Emergency System for Smart Enhanced Living Environments", *IEEE Cloud Computing*, vol. 3–6, nov 2016, pp. 54–62.

[36] Nissei. "Wattímetro nissei rs-50". Captured in: http://www.meterexpert.com/e/SWT_meter_11.htm, mar 2017.

[37] Oracle. "Java se development kit 8 documentation". Captured in: http://www.oracle.com/technetwork/java/javase/documentation/jdk8-doc-downloads-2133158.html, jan 2017.

[38] Patel, H. M.; Hu, Y.; Hédé, P.; Joubert, I. B. M. J.; Thornton, C.; Naughton, B.; Julian, I.; Ramos, R.; Chan, C.; Young, V.; Tan, S. J.; Lynch, D. "Mobile-Edge Computing", *White Paper, Mobile-edge Computing (MEC) industry initiative*, vol. 1, 2014, pp. 1–36.

[39] Patidar, S.; Rane, D.; Jain, P. "A survey paper on cloud computing". In: International Conference on Advanced Computing and Communication Technologies (ACCT), 2011, pp. 394–398.

[40] Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. "Context aware computing for the internet of things: A survey", *Communications Surveys Tutorials, IEEE*, vol. 16–1, First 2014, pp. 414–454.

[41] Perera, C.; Zaslavsky, A.; Liu, C. H.; Compton, M.; Christen, P.; Georgakopoulos, D. "Sensor search techniques for sensing as a service architecture for the internet of things", *IEEE Sensors Journal*, vol. 14–2, 2014, pp. 406–420.

[42] Rajalakshmi, P.; Mahalakshmi, S. D. "Iot based crop-field monitoring and irrigation automation". In: International Conference on Intelligent Systems and Control (ISCO), 2016, pp. 1–6.

[43] Riazul Islam, S. M.; Daehan Kwak; Humaun Kabir, M.; Hossain, M.; Kyung-Sup Kwak. "The Internet of Things for Health Care: A Comprehensive Survey", *IEEE Access*, vol. 3, 2015, pp. 678–708.

[44] Sanseverino, E. R.; Scaccianoce, G.; Vaccaro, V.; Zizzo, G.; Pennisi, S. "Smart city and public lighting". In: IEEE International Conference on Environment and Electrical Engineering (EEEIC), 2015, pp. 665–670.

[45] Schaffers, H.; Komninos, N.; Pallot, M.; Trousse, B.; Nilsson, M.; Oliveira, A. "Smart cities and the future internet: Towards cooperation frameworks for open innovation". In: The Future Internet Assembly, 2011, pp. 431–446.

[46] Schenfeld, M. C.; Amaral, L.; Matos, E. D.; Hessel, F. "Arquitetura para Fog Computing em Sistemas de Middleware para Internet das Coisas". In: Congresso da Sociedade Brasileira de Computação (SEMISH), 2016, pp. 1819–1829.

[47] Schenfeld, M. C.; Vargas, F. D.; Rebonatto, M. T.; Paixão, O. "Middleware para equipamentos médicos em System on a Chip". In: Brazilian Congress on Biomedical Engeneering (CBEB), 2014, pp. 2608–2611.

[48] Services, W. "Web services". Captured in: http://www.w3.org/TR/sawsdl/, Nov 2015.

[49] Skarlat, O.; Schulte, S.; Borkowski, M.; Leitner, P. "Resource Provisioning for IoT Services in the Fog". In: IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 2016, pp. 32–39.

[50] Specout. "System on a chip specification". Captured in: http://system-on-a-chip.specout.com, jan 2017.

[51] Stankovic, J. a. "Research Directions for the Internet of Things", *Internet of Things Journal, IEEE*, vol. 1–1, 2014, pp. 3–9.

[52] Sundmaeker, H.; Guillemin, P.; Friess, P.; Woelfflé, S. "Vision and challenges for realising the internet of things", *Cluster of European Research Projects on the Internet of Things, European Commision*, vol. 1–March, 2010.

[53] Tan, L. "Future internet: The Internet of Things". In: International Conference on Advanced Computer Theory and Engineering (ICACTE), 2010, pp. V5–376–V5–380.

[54] Tiburski, R. T.; Amaral, L. A.; Matos, E. D.; Hessel, F. "The importance of a standard securit y archit ecture for SOA-based iot middleware", *IEEE Communications Magazine*, vol. 53–12, dec 2015, pp. 20–26.

[55] Ubuntu. "Lubuntu documentation". Captured in: https://help.ubuntu.com/community/Lubuntu/Documentation, fev 2017.

[56] XMPP. "Xmpp". Captured in: https://xmpp.org, Dez 2016.

[57] Yang, K.-p.; Alkadi, G.; Gautam, B.; Sharma, A.; Amatya, D.; Charchut, S.; Jones, M. "Park-a-lot: An automated parking management system", *Computer Science and Information Technology*, vol. 1–4, 2013, pp. 276–279.

[58] Yangui, S.; Ravindran, P.; Bibani, O.; Glitho, R. H.; Ben Hadj-Alouane, N.; Morrow, M. J.; Polakos, P. A. "A platform as-a-service for hybrid cloud/fog environments". In: IEEE Workshop on Local and Metropolitan Area Networks (LANMAN), 2016, pp. 1–7.

[59] Yi, S.; Li, C.; Li, Q. "A Survey of Fog Computing". In: Workshop on Mobile Big Data (Mobidata), 2015, pp. 37–42.

[60] Yue, K.; Wang, X.-L.; Zhou, A.-Y.; et al.. "Underlying techniques for web services: A survey", *Journal of software*, vol. 3, 2004, pp. 428–442.

[61] Zao, J. K.; Gan, T. T.; You, C. K.; Méndez, S. J. R.; Chung, C. E.; Wang, Y. T.; Mullen, T.; Jung, T. P. "Augmented brain computer interaction based on fog computing and linked data". In: International Conference on Intelligent Environments (IE), 2014, pp. 374–377.

[62] Zhang, S.-K.; Zhang, J.-W.; Li, W. "Design of M2M Platform Based on J2EE and SOA". In: International Conference on E-Business and E-Government (ICEE), 2010, pp. 2029–2032.

[63] Zhou, M.; Zhang, R.; Zeng, D.; Qian, W. "Services in the cloud computing era: A survey". In: International Universal Communication Symposium, (IUCS), 2010, pp. 40–46.

APÊNDICE A – TESTES REAIS

Além das simulações apresentadas na execução dos testes dos dois cenários de aplicação, o FaaS4IoT também foi aplicado em um teste real em comunicação com dispositivos reais, porém, em virtude do número de equipamentos disponíveis, um cenário de menor escala foi aplicado.

O FaaS4IoT, ainda sendo executado em uma emulação de processamento SoC, com um sistema operacional com porte para processadores ARM, foi conectado com módulos de sensores, conforme a Figura APÊNDICE A.1. Ela mostra a utilização de diversos componentes para tornar o teste possível. Foram utilizados um módulo ESP8266 e um módulo NodeMCU, no ESP estavam conectados os sensores de temperatura e umidade do ar, DHT11, e o sensor de umidade relativa do solo modelo YI-69. No NodeMCU foi conectado também um sensor de temperatura e umidade DHT11, e um módulo relé para até 10A onde seria conectado o sistema de exaustão ou hidráulico, portanto o NodeMCU representa o atuador do sistema.

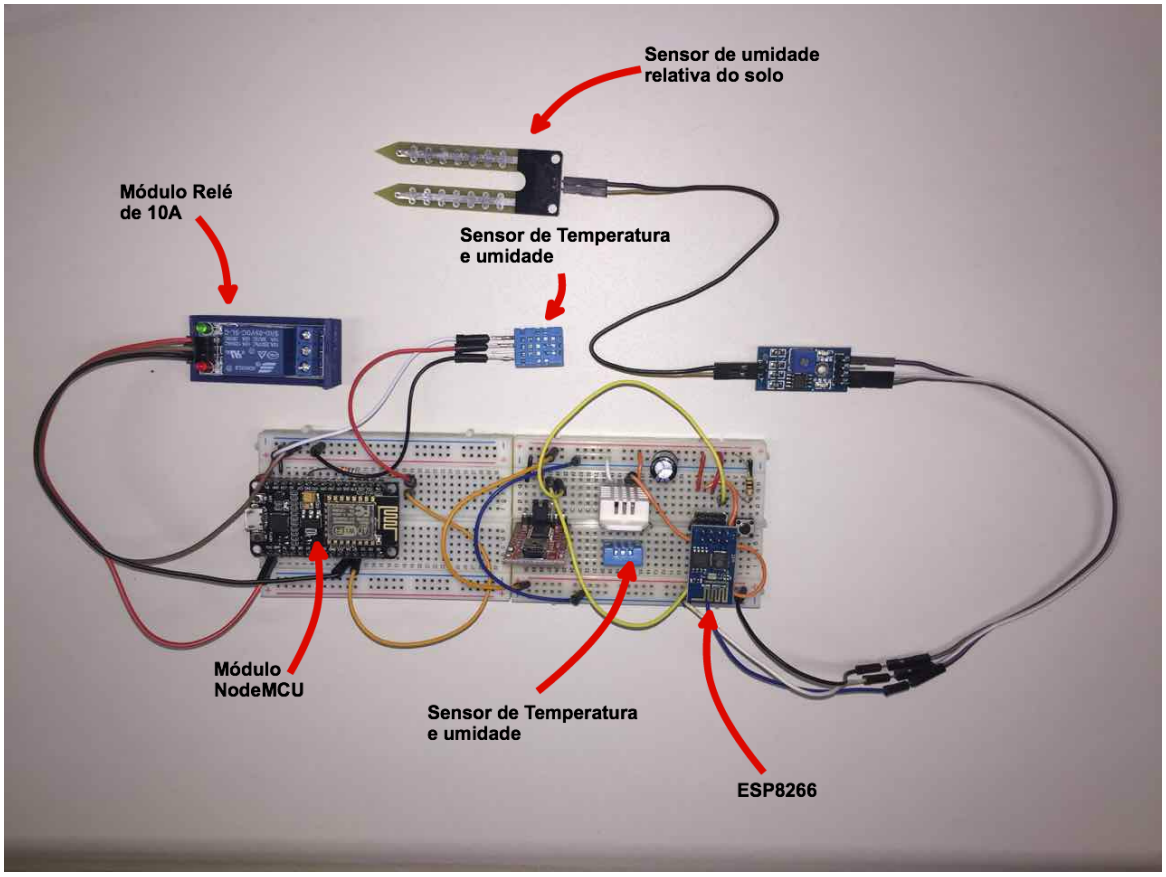

Figura APÊNDICE A.1 – Dispositivos com sensores e atuadores

A execução dos testes reais mostrou que a utilização dos mecanismos de verificação de eventos obtidos pelo sensoreamento do ambiente funciona, sendo possível detectar eventos e responder de imediato, seja gerando um alerta ou ativando o relé, que habilita a passagem de corrente elétrica.