Ampliando a Segurança de Aplicações para a Internet das Coisas Baseadas na Plataforma FIWARE

Caio Thomás Oliveira

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2017

Caio Thomás Oliveira

Ampliando a Segurança de Aplicações para a Internet das Coisas Baseadas na Plataforma FIWARE

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Flávio de Oliveira Silva, Ph.D.
Coorientador: Prof. Pedro Frosi Rosa, Ph.D.

Uberlândia
2017

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

O48a
2017

Oliveira, Caio Thomás, 1991
Ampliando a segurança de aplicações para a internet das coisas baseadas na plataforma FIWARE / Caio Thomás Oliveira. - 2017.
98 p. : il.

Orientador: Flávio de Oliveira Silva.
Coorientador: Pedro Frosi Rosa.
Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Disponível em: http://dx.doi.org/10.14393/ufu.di.2017.68
Inclui bibliografia.

1. Computação - Teses. 2. Internet - Medidas de segurança - Teses. 3. Redes de computadores - Protocolos - Teses. 4. Internet - Medidas de segurança - Teses. I. Silva, Flávio de Oliveira. II. Rosa, Pedro Frosi. III. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

CDU: 681.3

Dedico a todos que me ajudaram nessa caminhada.

Agradecimentos

Agradeço primeiramente a dedicação e incentivo dados a mim pelos meus pais que me auxiliaram a superar obstáculos e o apoio incondicional. Também agradeço o carinho da minha irmã e familiares. Estendo o reconhecimento aos meus amigos, aos companheiros de laboratório, aos professores, a equipe da VM9 e do FIWARE pela paciência e o estímulo para desenvolver este trabalho. Agradeço também a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela concessão da bolsa durante o período deste trabalho.

"Nem tudo na vida se resolve com dois cliques."
(Wendel Henrique Ferreira)

Resumo

O uso de dispositivos conectados ao ambiente tem sido crescente nos últimos anos. Utilizando diferentes tecnologias de comunicação esses dispositivos permitem o monitoramento do ambiente e também uma atuação remota. Neste cenário, surge o conceito Internet das Coisas ou Internet of Things (IoT) que visa em interligar tais dispositivos por meio da Internet à uma gama de aplicações. A IoT está cada vez mais presente em ambientes empresariais e domésticos e o potencial desta tecnologia está sendo aplicado em diferentes negócios. Entretanto, isso significa uma maior exposição de informações restritas que podem ser alvo de ataques. Desta forma, a segurança torna-se um requisito fundamental para a proteção e prevenção contra ataques virtuais às aplicações e aos dispositivos.

A plataforma FIWARE possui uma arquitetura de componentes responsáveis por interligar os dispositivos às aplicações de IoT, diminuindo a complexidade e fornecendo um conjunto de serviços comuns aos desenvolvedores. O IoT Agent é o componente que permite a integração de dispositivos e protocolos de comunicação com o restante da plataforma. Na versão IDAS 5 existem lacunas quanto a segurança, então, este trabalho visa incorporar à arquitetura serviços de segurança fim-a-fim através do uso de cifragem e controle de acesso em todas requisições NGSI. A principal contribuição foi a implementação em NodeJs do suporte ao DTLS 1.2 no protocolo LWM2M/CoAP e sua incorporação ao IoT Agent. As intervenções permitiram ao IoT Agent suportar o TLS na comunicação com o MQTT e o NGSI. As modificações permitem que outros protocolos possam se beneficiar dos resultados obtidos visto que todo o código está aberto no GitHub. Através de uma avaliação experimental foi possível validar a implementação realizada. Pode-se observar que as requisições realizadas tiveram um pequeno acréscimo médio em termos de latência, mas esse custo é compensado através da ampliação da segurança na plataforma.

Palavras-chave: Internet do Futuro, Internet das Coisas, Segurança, Controle de Acesso, Protocolos.

Abstract

The use of devices connected to the environment has been increasing in recent years. Using different communication technologies these devices allow to monitor and actuate in the environment. The Internet of Things (IoT) uses the Internet to interconnect such devices through to a wide range of applications. IoT is increasing its presence in business and home environments and the potential of this technology is being applied in different fields. However, this means a greater exposure of restricted information rising potential to threats its exploitation by attacks. Thus, security becomes a key requirement for the protection and prevention of cyberattacks to applications and devices.

FIWARE Platform has an architecture of components responsible for interconnecting devices to IoT applications, decreasing complexity and providing a common set of services to developers. The IoT Agent is the component that allows integration of devices and its communication protocols with the rest of the platform. The version IDAS 5 of the platform presents some security gaps, so this work aims to incorporate end-to-end security services using encryption and access control in all NGSI requests. The main contribution was the implementation in NodeJs of DTLS 1.2 to support in the LWM2M/CoAP protocol and its addition to the IoT Agent. The work also allowed the IoT Agent support TLS in the communication of MQTT and NGSI. The source code of this work is open in the GitHub and can be used to support security services in other IoT communication protocols. Through an experimental evaluation it was possible to validate the implementation. It can be observed that the requests have had a small increase in terms of latency, but this cost is compensated by the increase of security in FIWARE based IoT Applications.

Keywords: Future Internet, Internet of Things, Security, Access control, Protocols.

Lista de ilustrações

Lista de tabelas

Lista de siglas

API *Application Programming Interface*

AMQP *Advanced Message Queuing Protocol*

ARM *Architecture Reference Model*

CoT *Cloud of Things*

CoAP *Constrained Application Protocol*

DTLS *Datagram Transport Layer Security*

FI Future Internet

GEs *Generic Enablers*

HTTP *Hypertext Transfer Protocol*

HTTPS *Hypertext Transfer Protocol Secure*

IDM *Identity Manager*

IETF *Internet Engineering Task Force*

IP *Internet Protocol*

IoT *Internet of Things*

ISO *International Organization for Standardization*

JSON *JavaScript Object Notation*

LWM2M *Lightweight M2M*

M2M *Machine-to-Machine*

MIT *Massachute Institute of Technology*

MQTT *Message Queuing Telemetry Transport*

NFC *Near Field Communication*

NGSI *Next Generation Service Interface*

NPM Narwhals Poke Mammals

OSI *Open Systems Interconnection*

OMA *Open Mobile Alliance*

PANs *Personal Area Networks*

PaaS *Platform as a Service*

PSK *Pre-Shared Key*

QoS Qualidade de Serviço

RDF *Resource Description Framework*

REST *REpresentational State Transfer*

RFID *Radio Frequency Identification*

SaaS *Software as a Service*

SOA *Service Oriented Architecture*

SSL *Secure Sockets Layer*

TCP *Transmission Control Protocol*

TLS *Transport Layer Security*

URIs *Uniform Resource Identifiers*

XML *eXtensible Markup Language*

XMPP *Extensible Messaging and Presence Protocol*

Sumário

Capítulo 1

Introdução

A informática e a Internet tornaram-se naturais em ambientes de negócios e domésticos. Esse fator possibilitou o paradigma promissor que é a integração de várias tecnologias e soluções de comunicação a fim de conectar objetos. A Internet das Coisas ou *Internet of Things* (IoT) é uma proposta que consiste em integrar as "coisas" que se interligam entre si e são complementares para viabilizar objetos do mundo real para um contexto virtual. Este conceito foi atribuído pela primeira vez por Kevin Ashton enquanto trabalhava no projeto Auto-ID no *Massachute Institute of Technology* (MIT), mais tarde foi realizado pesquisa no domínio de *Radio-Frequency Identification* (RFID) (MADAKAM; RAMASWAMY; TRIPATHI, 2015; SUNDMAEKER et al., 2010; TAN; WANG, 2010).

Dessa forma, um grande número de objetos heterogêneos são conectados e capazes de interagir a fim de oferecer um objetivo. Tais objetos podem consistir em sensores e atuadores no qual são considerados "coisas" capazes de fornecer informações dos ambientes e controlar remotamente. O objetivo em IoT é prover um modo avançado de comunicação entre diferentes sistemas e dispositivos facilitando a interação com aplicativos que serão utilizados por usuários nas diversas atividades do cotidiano. Isto foi possível com o avanço de várias tecnologias que incluem virtualização de rede, computação em nuvem (*cloud computing*) e o desenvolvimento de pequenos dispositivos de baixo custo conectados em rede sem-fio (*wireless*) amplamente difundida pela indústria.

De fato, a IoT é uma das notáveis paradigmas emergentes que visam contribuir em diferentes tipos de domínios de aplicação, devido a isso, o seu impacto é expressivo no cenário industrial e acadêmico (RAZZAQUE et al., 2016). A variedade de dispositivos cresce na medida que as aplicações são desenvolvidas principalmente nas áreas como: saúde, cidades, agricultura, transporte, militar, comercial, doméstico, entre outros (TAN; WANG, 2010; AL-FUQAHA et al.,). Entretanto, estas aplicações introduzem maiores desafios no cenário de ambientes inteligentes, já que é necessário que os dispositivos sejam de baixo custo e pequenos em escala de tamanho por consequência os recursos são limitados em termos de computação, armazenamento e comunicação. Com o avanço da tecnologia o contexto da IoT tem melhorado no desenvolvimento de protocolos, redes,

dispositivos, sensores e sistemas (XU; HE; LI, 2014).

Diante da grande quantidade de dispositivos e softwares envolvidos em projetos IoT, é necessário estabelecer um planejamento para desenvolver sistemas que conversem entre si por meio de uma interface definida. Para isto, a solução de Plataformas IoT visam oferecer um conjunto de serviços para o desenvolvimento de aplicações IoT que utilizam a computação em nuvem. Dessa forma, o *middleware* é a uma camada responsável por processar um grande número de eventos de elementos heterogêneos e adaptar os comportamentos dos ambientes de modo a simplificar a complexidade das aplicações IoT. Tais arquiteturas têm atraído atenção da indústria e de pesquisadores já que oferecem soluções para lidar com um processo complexo e crítico através da escalabilidade de recursos (RAZZAQUE et al., 2016).

Desse modo, muitas pesquisas e empresas estão em expansão no desenvolvimento de aplicações IoT e em busca de respostas quanto a questões desafiadoras relativas as tecnologias de hardware e software utilizados em seus projetos (SCHAFFERS et al., 2011; KHAN et al., 2012). Pode-se notar que ao implantar sistemas reais de IoT existe uma grande quantidade de desafios, dentre estes problemas estão relacionados a vulnerabilidades do uso da rede aberta em dispositivos no qual levanta questões relativas à privacidade, segurança, autenticação e autorização (RAZZAQUE et al., 2016; AL-FUQAHA et al.,).

1.1 Motivação

O avanço de tecnologias de computação em nuvem, dispositivos embarcados, protocolos e redes possibilitou a materialização do paradigma Internet das Coisas. Tendo em vista a heterogeneidade de tecnologias deve-se considerar a necessidade de proteger as informações colhidas pelos dispositivos até à aplicação. É essencial observar que um dos desafios para o sucesso da IoT situa-se na segurança da informação. Nota-se que a comunicação das "coisas" ou objetos mantêm-se por meio de interfaces de rede, no qual, abre-se a possibilidade para diferentes ataques de segurança. Os ataques buscam explorar vulnerabilidades e podem comprometer seriamente a privacidade por conter informações com dados sigilosos ou influenciar o comportamento ou obter o controle do sistema IoT.

A segurança em IoT torna-se vital principalmente por possuir diferentes tipos de tecnologias de *hardware*, *software* e grande quantidade de dispositivos conectados na rede. Tendo em vista uma aplicação IoT, deve-se conciliar as prioridades dos requisitos da aplicação equilibrando com a segurança das informações trafegadas na rede por meio da cifragem/decifragem e autenticação. A proteção e a privacidade dos dados contra agentes não autorizadas são de extrema importância para evitar ataques e espionagem, assim, a segurança é um requisito chave para os sistemas de informação.

O FIWARE (FIWARE, 2017c) é uma plataforma da Internet do Futuro que permite trabalhar no contexto IoT por meio de um um conjunto de *middleware* e serviços que visam

integrar dispositivos, protocolos e informações. Apesar de ser uma plataforma robusta, alguns componentes não possuem aspectos de segurança desenvolvidos e acoplados ao projeto. Portanto, este trabalho tem como objetivo principal elaborar e implementar mecanismo de segurança e controle de acesso para arquitetura FIWARE de modo que garanta a integridade e a autenticidade da informação desde o protocolo utilizado no dispositivo até a camada de aplicação.

1.2 Objetivos e Desafios da Pesquisa

O principal objetivo desta pesquisa é ampliar a segurança de aplicações de IoT baseadas na plataforma FIWARE através da proposição de mecanismos de segurança através de cifragem e decifragem e controle de acesso. Para se alcançar o objetivo citado, os seguintes objetivos específicos devem ser alcançados:

- Implementar um IoT Broker-FI que permita a agregação de dados de diferentes módulos IoT Agents de forma segura e possibilite integrar *plugins* para aplicações externas.
- Desenvolver autenticação e autorização do IoT Agent Node Lib, oferecendo integração com IoT Broker-FI e os IoT Agents responsáveis pelos protocolos *Lightweight Machine-to-Machine/Constrained Application Protocol* (LWM2M/CoAP) e *Message Queuing Telemetry Transpor* (MQTT).
- Permitir que requisições *Next Generation Service Interface* (NGSI) sejam realizadas de forma cifrada utilizando *Hypertext Transfer Protocol Secure* (HTTPS) e seja integrado com IoT Broker-FI e o IoT Agent.
- Permitir que o IoT Agent MQTT ofereça suporte para *Transport Layer Security*(TLS).
- Permitir que o IoT Agent LWM2M/CoAP ofereça suporte ao Datagram Transport Layer Security (DTLS) 1.2 e validar esse suporte com um cliente e servidor compatível com DTLS 1.2 como LESHAN Lightweight M2M. Verificar o impacto da segurança utilizando testes de integração entre o IoT Broker-FI, IoT Agent e os componentes da plataforma FIWARE, Orion Context Broker e PEP Proxy.
- Verificar o impacto em uma implementação real utilizando testes de integração providos entre os componentes da plataforma IoT e os módulos implementados.

Os desafios do presente trabalho estão relacionados em coletar, analisar, elaborar e implementar recursos no âmbito da tecnologia FIWARE, e considerar a existência de peculiaridades não tratadas na plataforma. Por conseguinte, o maior desafio é lidar com uma grande quantidade de protocolos, cenários e arquiteturas IoT.

1.3 Contribuições

Os componentes propostos e desenvolvidos ao longo deste trabalho foram incorporados a um produto produto real, chamado FI-GUARDIAN que é parte de uma plataforma para Cidades Inteligentes da empresa VM9 (VM9IT, 2017). Esse trabalho conjunto ocorreu no âmbito do programa CNPq RHAE.

Além disso, as demais contribuições do trabalho estão relacionadas à plataforma FIWARE e o desenvolvimento de mecanismos de segurança. Foram incorporados criptografia nos agentes para os protocolos MQTT e LWM2M/CoAP e chamadas NGSI. Uma importante contribuição foi a implementação do DTLS 1.2 no LWM2M/CoAP para ser utilizado no agente, já que não existia em NodeJs. Outra contribuição é a implementação do IoT Broker-FI que permite a manipulação de contextos e agentes na versão IDAS 5 (CATALOGUE, 20017).

Os mecanismos desenvolvidos estão disponibilizados para todos os pesquisadores envolvidos com FIWARE no GitHub (THOMAS, 2017) e toda comunidade científica e desenvolvedores. Desta maneira, o presente trabalho amplia o nível de segurança da arquitetura, possibilitando seu aprimoramento.

1.4 Organização do Trabalho

Esta dissertação está organizada em cinco capítulos. O Capítulo 2 apresenta os aspectos conceituais sobre a Internet das Coisas, bem como, os elementos principais que compõe a arquitetura dos dispositivos. Também apresenta as principais os protocolos de mensagens utilizadas pelos dispositivos, aspectos de segurança e detalha o conceito de *middlewares* e plataformas de IoT. Além disso, os aspectos conceituais e arquiteturais do FIWARE são explorados minuciosamente, e é apresentada a situação atual da arquitetura.

O Capítulo 3 detalha as principais necessidades encontradas no FIWARE e apresenta uma proposta na arquitetura para implementar a confidencialidade, controle de acesso e o componente IoT Broker-FI para realizar associações e manipulação de vários IoT Agents. O Capítulo 4 apresenta os experimentos, resultados e a avaliação da implementação do mecanismo de segurança na plataforma utilizando o Wireshark para fazer a análise dos pacotes trafegados. O Capítulo 5 faz uma conclusão, apresentando as contribuições do trabalho, os objetivos alcançados e os trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo se destina a apresentar a fundamentação teórica deste trabalho, considerando os aspectos da Internet das Coisas. Primeiro será detalhado o conceito e os elementos que possibilitam este paradigma e também é abordado a arquitetura básica dos dispositivos e a arquitetura em IoT. Para compreender e contextualizar a contribuição do trabalho são apresentados os aspectos teóricos dos protocolos de mensagens LWM2M/CoAP e o MQTT. Além disso, é abordado a segurança da informação juntamente com os protocolos TLS e DTLS que implementam a criptografia nas mensagens permitindo a comunicação segura. Também apresenta uma visão geral das plataformas em IoT disponíveis no mercado em que oferecem *middleware* que visa simplificar o desenvolvimento de novos serviços em IoT. A plataforma FIWARE é detalhada com seus principais componentes, e por fim, é apresentado o cenário atual da arquitetura para o desenvolvimento de uma aplicação real em IoT.

2.1 Internet das Coisas

O conceito de Internet das Coisas pode ser visto como um conjunto de objetos interconectados que coleta e transmite informações para uma ampla gama de aplicações a fim de monitorar e atuar no ambiente (AL-FUQAHA et al.,). A subseção 2.1.1 descreve de forma detalhada os elementos que compõe a IoT. Já a subseção 2.1.2 e 2.1.3 apresentam respectivamente a arquitetura básica dos dispositivos e a arquitetura em IoT.

2.1.1 Elementos do IoT

A história e o conceito da IoT se funde com o avanço das tecnologias de dispositivos, protocolos, redes e a virtualização de computadores. Desse modo, os vários elementos IoT tornam-se um sistema que interliga nós de processamento visando um consumo de dados em tempo real. Assim, segundo Al-Fuqaha et al. () são apresentado os elementos principais que compõe uma IoT e a Figura 1 ilustra uma visão geral.

Figura 1 – Segundo Al-Fuqaha et al. () estes são os elementos IoT.

Fonte: Al-Fuqaha et al. ()

- **Identificação:** é crucial para identificar os objetos de forma única para conectá-los à internet. Muitos métodos são utilizados para identificar, como: RFID, Near Field Communication (NFC), endereço IP.

- **Dispositivos:** possui sensores e atuadores que coletam informações do ambiente ou podem manipular ou reagir de acordo com um comando pré-definido.

- **Comunicação:** em IoT diferentes objetos distribuídos são conectados e oferecem serviços específicos. Os diversos dispositivos devem operar com pouca energia e falhas de rede podem ser comuns. A partir desses problemas, a escolha da tecnologia de comunicação deve ser considerada relevante ao projeto.

- **Computação:** consiste no processamento de micro controladores, FPGAs, e aplicações de softwares que representam a habilidade de computar como as plataformas *cloud*.

- **Serviços:** os serviços podem ser classificados de diversas formas. Serviços de identidade pode ser considerados um dos mais importantes e é utilizado em outros tipos de serviços. Toda aplicação precisa trazer objetos do mundo real para o virtual e é necessário identificar essas entidades. Agregação de informação consiste em serviço que coleta e ordena os dados recebidos pelos dispositivos para que sejam processados e reportado na aplicação IoT. Os Serviços de Colaboração e Inteligência consiste em agir sobre os serviços de agregação de informações usando os dados obtidos para tomar decisões para assim reagir de modo adequado. Os Serviços de Ubiquidade tem como objetivo prover Serviços de colaboração e Inteligência em qualquer momento e qualquer lugar em que eles sejam necessários.

- **Semântica:** refere-se a habilidade de extrair conhecimento de diferentes provedores de serviços. Tal conhecimento, incluem descobrimento, uso de recursos e modelamento de informação. Também inclui o reconhecimento, análise do dado para fazer com que as decisões sejam corretas. Vale lembrar que a semântica pode ser vista como a parte importante para enviar demandas à determinado recurso.

No aspecto da implementação em um ambiente real exitem várias empresas que integram seus dispositivos com plataformas em IoT. A Figura 2 ilustra o esquema dos dispositivos e *gateway* da Libelium (LIBELIUM, 2017b).

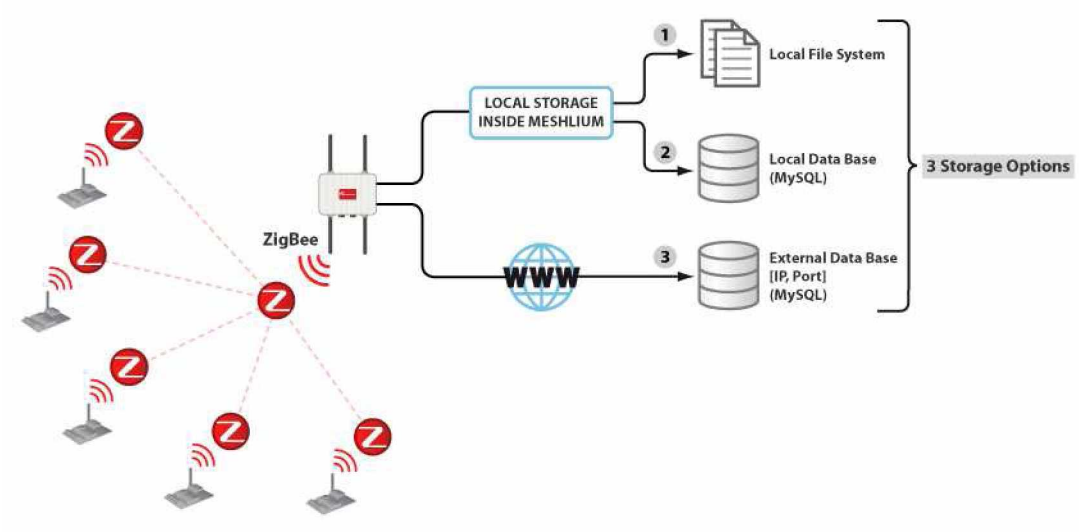

Figura 2 – Visão da arquitetura IoT da Libelium.

Fonte: Libelium (2017b)

Esse processo permite entender as reais necessidades dos requisitos para implementar sistemas IoT. Desse modo, são utilizados dispositivos configurados de forma manual que conectam com a infraestrutura de rede local utilizada no ambiente. Todas as informações colhidas são enviadas à um *gateway* que envia para uma plataforma IoT, como: IBM Bluemix, Telefónica (FIWARE) e Microsoft Azure (LIBELIUM, 2017a). Note que o *gateway* pode armazenar os dados em caso de falhas ou enviar para a Internet.

2.1.2 Arquitetura Básica dos Dispositivos

Nesta seção são abordados os conceitos básicos da arquitetura dos dispositivos. O bloco de construção dos dispositivos foca em quatro unidades: processamento/memória, comunicação, energia e sensores/atuadores.

- **Processamento e memória:** consiste no armazenamento de dados e softwares, um microcontrolador e um conversor analógico-digital para receber sinais dos sensores. As CPUs utilizadas nos dispositivos são as mesmas que em sistemas embarcados e podem apresentar alto poder computacional. Geralmente existe memória externa, como flash que visa ser considerada a memória secundária. Vale lembrar que os requisitos necessários são baixo consumo de energia e menor espaço ocupado na placa.

- **Comunicação:** a unidade de comunicação que é sem fio ou com. Geralmente algumas plataformas utilizam rádio de baixa potência, entretanto, a comunicação é de curto alcance e apresenta perdas frequentes.

- **Fonte de Energia:** é responsável por alimentar o dispositivo. Existem diversas fontes de energias utilizáveis, como: elétrica, baterias, solar e outras.

- **Sensores/Atuadores:** os sensores são unidades que realizam monitoramento do ambiente, como: temperatura, umidade, presença e outros. Atuadores são dispositivos que produzem movimento, atendendo a comandos que podem ser manuais, elétricos ou mecânicos.

2.1.3 Arquitetura para IoT

Ao conectar uma grande quantidade de dispositivos em uma rede pode ser um desafio prover a escalabilidade, interoperabilidade, confiabilidade e a qualidade de serviço. Desta forma, esta subseção apresenta duas proposta de arquiteturas que são utilizadas para projetar uma aplicação em IoT considerando estes requisitos.

A Figura 3 ilustra o modelo básico de arquitetura base proposta por (WU et al., 2010) que apresenta três camadas. A camada de percepção consiste em objetos físicos, que, por meio de sensores coletam e passam informações para camada de rede. Esta camada basicamente lida com a identificação e coleções de objetos de informações específicas por meio de sensores. A camada de rede ou camada de transmissão visa transmitir a informação obtida pela camada para a superior. Além disso, procura abstrair as tecnologias de comunicações, protocolos e serviços de gerenciamento relacionados à camada de rede. A camada de aplicação é responsável por processar as informações recebidas e fornecer serviços para os clientes ou outras aplicações.

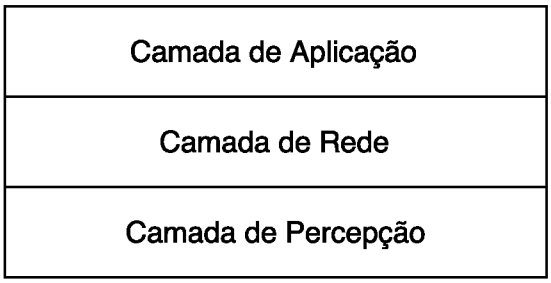

Figura 3 – Arquitetura de 3 camadas para IoT.

Fonte: Wu et al. (2010)

Segundo Khan et al. (2012) e Tan e Wang (2010), a arquitetura base pode ser dividida em cinco camadas como mostrado na Figura 4. A camada de percepção e de rede possuem as mesmas características descritas anteriormente, já as camadas diferentes são a *middleware* e a camada de negócios. A camada *Middleware* é responsável por implementar diferentes tipos de serviços relacionados com IoT. Cada dispositivo conecta e comunica com apenas outros diapositivos que possuem o mesmo tipo de serviço. Esta camada têm várias funções, como: oferecer serviços para aplicações de forma simples, lidar com grande quantidade de eventos, integrar sistemas e comunicação heterogênea. A camada de aplicação recupera os dados que foram processados pelo *middleware* para ser utilizado em determinada aplicação. A camada de negócios é responsável pelo gerenciamento de todo sistema IoT que incluem aplicações e serviços. Com isso, é produzido um modelo de negócio, gráficos, digramas baseados nos dados recebidos da camada de aplicação. Desta forma, baseado nas análises dos resultados esta camada irá ajudar a determinar as ações futuras e traçar as estratégias. Em Tan e Wang (2010) ressalta a dificuldade de se conectar uma grande quantidade de dispositivos, assim, entre a camada de percepção e a rede é necessário a camada *gateway* que visa ser um componente que proporciona o ponto de ligação de comunicação.

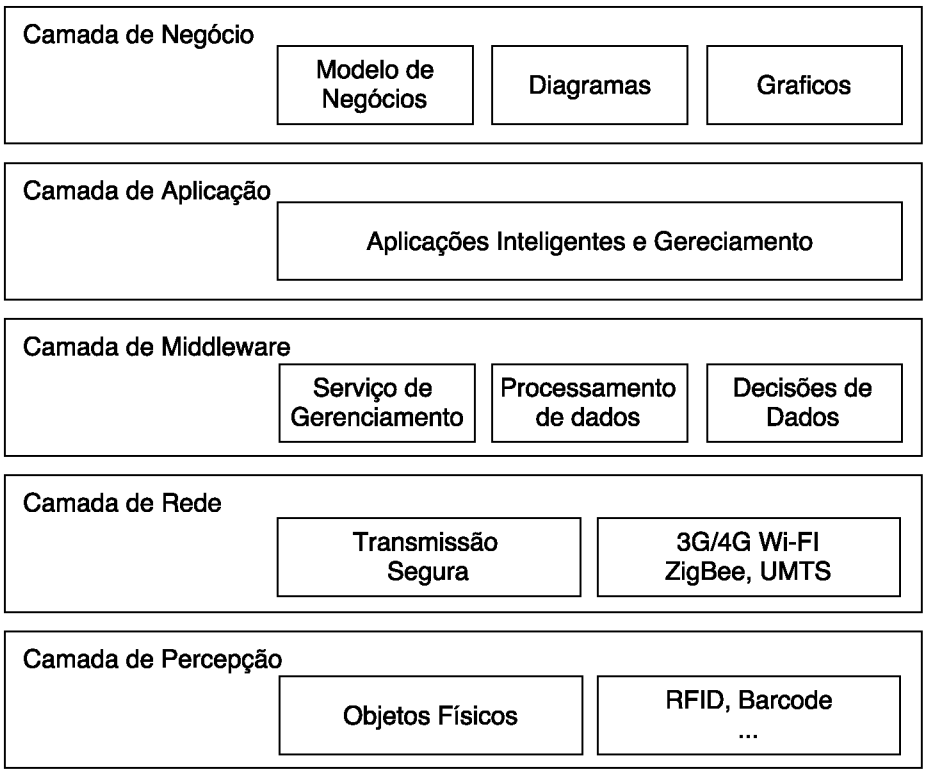

Figura 4 – Arquitetura de cinco camadas para IoT.

Fonte: Khan et al. (2012), Tan e Wang (2010)

2.2 Protocolos

Esta seção visa detalhar os protocolos MQTT, CoAP e o LWM2M/CoAP utilizados neste trabalho. Estes protocolos de mensagens são utilizados a partir do dispositivo até um *gateway* de IoT.

2.2.1 Constrained Application Protocol

O Constrained Application Protocol (CoAP) é um protocolo que está na camada de aplicação padronizado pela IETF Constrained RESTful environments (CoRE) Working Group (GAZIS et al., ; SHELBY; HARTKE; BORMANN, 2014). Seu objetivo é emular os comandos do REST, entretanto a sua diferença está no protocolo UDP o que possibilita o cenário mais adequado para aplicações em IoT. Este protocolo encapsula no corpo da mensagem um *payload* que consiste na carga de uma transmissão de dados. O CoAP define o cabeçalho da mensagem, códigos *request/response*, opções de mensagens e mecanismo de retransmissão. O protocolo tem como principal objetivo ser uma alternativa HTTP para RESTful API em dispositivos de recursos limitados e suporta os métodos básicos para GET, POST, PUT, DELETE em que devem ser mapeados para aqueles métodos HTTP.

O CoAP pode facilmente traduzir comandos HTTP e simplificar a integração com a web. Ao contrário HTTP, as mensagens CoAP são trocadas de forma assíncrona entre CoAP *end-point* sobre o UDP. O CoAP permite o *broadcast* e *multicast* para serem usados por endereçamento e troca de mensagem assíncrona (LAINE,). As requisições e respostas de mensagens podem ser marcadas como "confirmable" (recebem um *ack*) ou "nonconfirmable" (esquecidas). O CoAP utiliza quatro tipos de mensagens: *confirmable, non-confirmable, reset e acknowledgement.* A confiabilidade do protocolo CoAP é implementada ao combinar as mensagens *confirmable* e *non-confirmable* sobre o UDP (PERERA et al., 2014).

2.2.2 OMA Lightweight M2M

O protocolo OMA Lightweight M2M (LWM2M) é um protocolo da Open Mobile Alliance para o gerenciamento do dispositivo IoT (M2M, 2017). O Lightweight M2M define o protocolo de comunicação da camada de aplicação entre um servidor e um cliente que está localizado em um dispositivo LWM2M. Os dispositivos LWM2M são geralmente aqueles que possuem recursos limitados e deve utilizar um protocolo compacto e eficiente do modelo de dados.

A especificação Lightweight provê API para configuração do dispositivo, conectividade, monitoramento, estatísticas, segurança, provisionamento de servidor. O LWM2M Enabler define o protocolo de comunicação da camada de aplicação entre o servidor e o

cliente. O LWM2M Server é tipicamente localizado em um *datacenter* privado ou público e pode hospedar M2M Service Provider, Network Service Provider ou Application Service Provider. O LWM2M Client reside no dispositivo e é integrado com a biblioteca ou uma função do dispositivo.

São utilizados quatros interfaces lógicas que são definidas entre o cliente o servidor, e a Figura 5 ilustra as principais ações. O *Bootstrap* consiste no pré-configuração e inicialização do servidor. O *Device Discovery/Registration* ocorre quando o cliente registra no servidor. O *Device Management/Service* Enablement acontece quando o servidor executa no cliente funções, como: *read, write, execute, create, delete, write attribute, discover.* O *Information Reporting* é quando o servidor deseja observar um determinado recurso no cliente. Quando esse dado é alterado no cliente, este notifica o servidor.

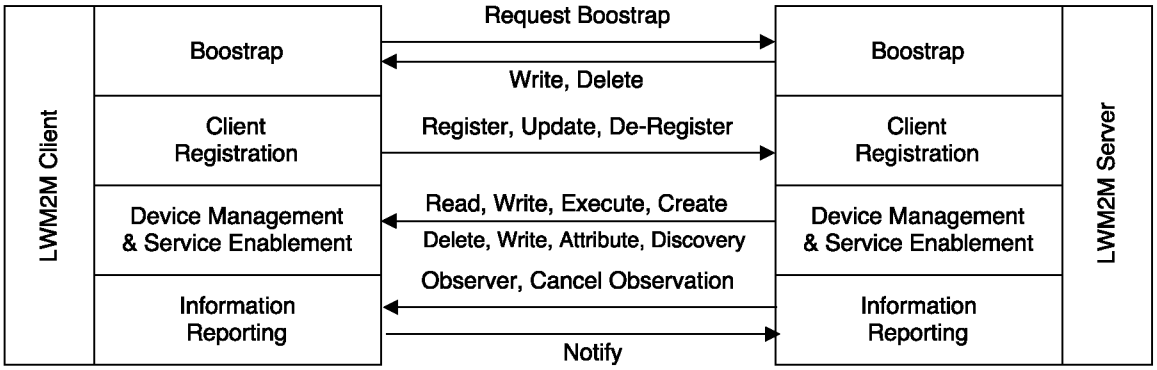

Figura 5 – Interfaces de execução entre cliente e servidor no protocolo Lightweight M2M.

Fonte: Rao et al. (2015)

A pilha do protocolo LWM2M utiliza o IETF CoAP como forma de transferir sobre os protocolos UDP e SMS. Ao contrário de HTTP, CoAP foi concebido para as necessidades dos dispositivos restritos. O CoAP é um protocolo *one-to-one* para transferir a informação de estado entre cliente e o servidor, sendo adequado para o modelo LWM2M. A interação do cliente e servidor pode pode acontecer sobre SMS ou UDP.

Cada LWM2M Client possui um modelo simples de recursos organizados em objetos onde cada peça de informação é disponível para o servidor LWM2M. Este modelo de dados é definido pela OMA por meio de nomenclatura de objetos com a mesma semântica proporcionando um padrão para o gerenciamento de dispositivos IoT.

O protocolo LWM2M cria uma abstração do protocolo CoAP em que os comandos GET, PUT, POST, DELETE são traduzidos por funções (*read, write, execute, create e etc.*) que manipulam tais objetos definidos no cliente. Além disso, qualquer organização poderá criar seus objetos compatíveis com o escopo endereçado em dispositivos M2M e suas aplicações. A Figura 6 ilustra a arquitetura Lightweight M2M com LwM2M Server e LwM2M Client.

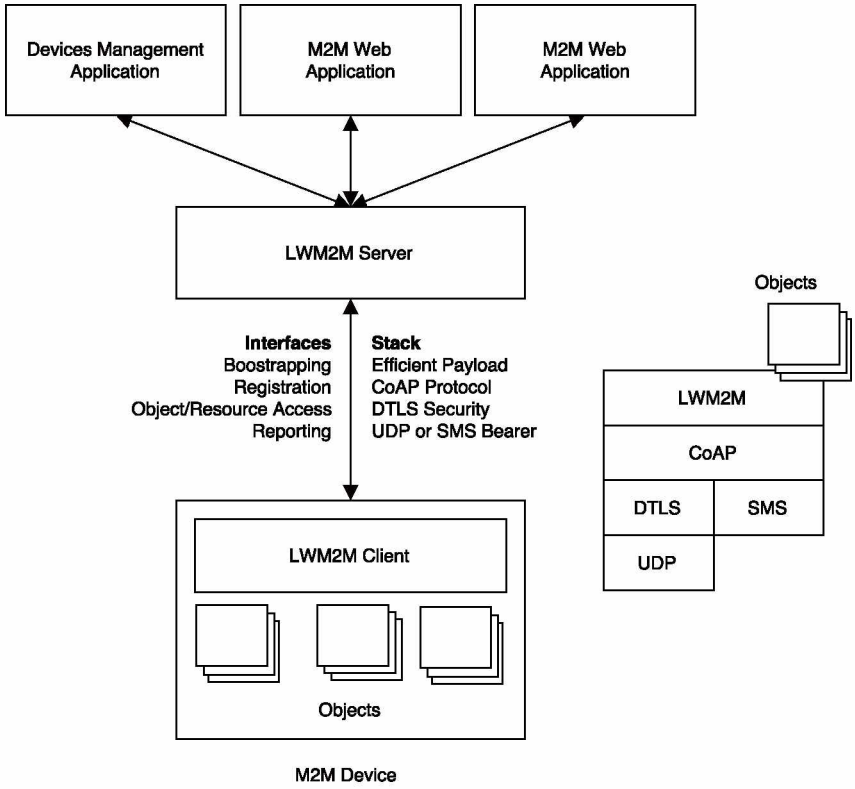

Figura 6 – Arquitetura Lightweight M2M com LWM2M Server e LWM2M Client.

Fonte: Rao et al. (2015)

2.2.3 Message Queue Telemetry Transport

O protocolo Message Queue Telemetry Transport (MQTT) foi desenvolvido pela IBM e Eurotech em 1999, e sua especificação mais recente é a versão 3.1.1 (COHN, 2015). Consiste em um protocolo de aplicação orientado à mensagens que provê controle de comunicação controlada por fluxo com garantias de entrega. É projetado para ter baixo consumo de banda de rede e requisitos de hardware, sendo extremamente leve, simples, também conhecido como M2M. O MQTT pode ser utilizado em vários tipos de dispositivos em diferentes áreas de negócios, como: saúde, energia, aplicativo de mensagem do Facebook (GAZIS et al.,). Um exemplo de uma implementação em código aberto do MQTT é o Mosquitto (MOSQUITTO, 2017).

O MQTT utiliza o paradigma *publish/subscribe* (pub/sub) para a troca de mensagens. Este paradigma implementa um *middleware* chamado de MQTT *Broker*. O MQTT *Broker* é responsável por receber, enfileirar e retransmitir as mensagens recebidas dos *publishers* para os *subscribers* (PERERA et al., 2014). O *publisher* é responsável por se conectar ao MQTT *Broker* e publicar mensagens. Já o *subscriber* é responsável por se conectar ao MQTT *Broker* e receber as mensagens que ele tiver interesse. Na Figura 7 ilustra a arquitetura do paradigma pub/sub.

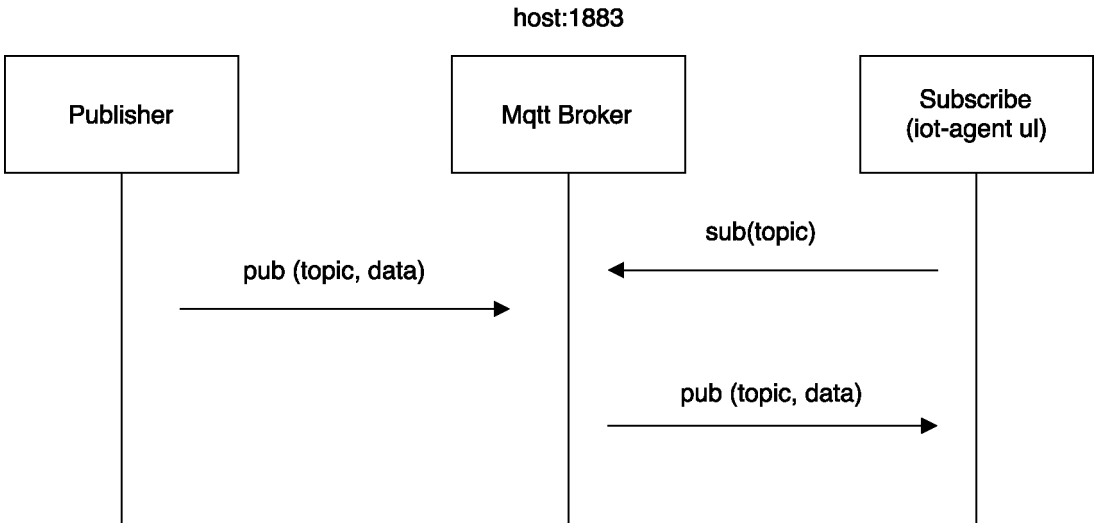

Figura 7 – Visão do modelo *publish/subscribe* do MQTT.

Fonte: Perera et al. (2014)

O paradigma pub/sub utiliza o conceito de tópicos para processar as mensagens, em que cada mensagem é enviada para um determinado tópico. Diferente de outros protocolos de mensagem, o *publisher* não envia a mensagem diretamente ao *subscriber*, mas sim ao *broker*. O *publisher* envia a mensagem para o MQTT *Broker* em um determinado tópico. O *Broker* é responsável por receber a mensagem do *publisher* e fazer uma pré-filtragem das mensagens e enviá-las para os *subscribers* que estivem registrados em um determinado tópico.

2.3 Segurança da Informação

A segurança da informação é o conceito relacionado com a proteção de um conjunto de informações com o objetivo de preservar o valor que possuem para uma organização. A X.800 da ITU-T *Security Architecture for OSI* define serviços de segurança utilizados na transferências de dados por meio de protocolos de comunicação (ITU-T STUDY GROUP 17, 1991).

A especificação divide os serviços em cinco categorias: serviço de autenticação refere-se à garantia de que uma comunicação entre duas entidade são autênticas, ou seja, o serviço precisa assegurar que a conexão não sofra a interferência de um terceiro fingindo ser uma das partes legítimas. O controle de acesso consiste na prevenção do uso não autorizado de um recurso pela capacidade de limitar o acesso. A confidencialidade dos dados é proteção das mensagens de usuários transmitidos em uma comunicação, assim, um atacante não consegue observar o conteúdo e as características da informação. A integridade é a garantia de que a informação seja mantida em seu estado original, ou seja, tendo em vista em protegê-la na transmissão contra alterações indevidas. O não-repúdio

é proteção contra a negação de uma das partes em uma comunicação, ou seja, ambas as partes podem provar que a mensagem foi enviada ou recebida.

Os mecanismos de segurança definidos na recomendação X.800 são utilizados para detectar, prevenir ou recuperar de um ataque sobre o sistema. Alguns mecanismos de segurança são: cifragem (*encipherment*), assinatura digital (*digital signature*), controle de acesso (*access control*), integridade de dados (*data integrity*), permuta de credenciais (*authentication exchange*), inserção de bits (*traffic padding*), controle de rotas (*routing control*) e terceiros confiáveis (*notarization*).

A Cifragem/Decifragem consiste em utilizar algoritmos de criptografia que permite que os dados sejam encriptados e, então, decriptados. A assinatura digital é garantir que o não repúdio de uma mensagem enviada por um emissor. O controle de acesso uma série de mecanismos que impõem direitos de acesso aos recursos. A integridade de dados consiste na garantia da precisão e consistência do dado. A permuta de credenciais é o mecanismo que garante a identidade de uma entidade pela troca de informações. A inserção de bits visa inserir bits nas lacunas de um fluxo de dados para evitar análise da mensagem. Já o mecanismo de roteamento permite a escolha e a alteração de rotas em caso de suspeita de quebra da segurança. Por fim, para garantir reconhecimento e validade da informações na troca de dados pode-se usar terceiros confiáveis.

A utilização de sistemas conectados e o tráfego por meio de redes de computadores permite que as informações fiquem vulneráveis e sujeitas à ameaças de elementos externos o que pode comprometer uma aplicação (WHITMORE; AGARWAL; XU, 2015). No contexto de IoT é um desafio a segurança e a privacidade, já que existe uma grande quantidade de dispositivos distribuídos (KHAN et al., 2012; AL-FUQAHA et al.,). Com isso, torna-se essencial a proteção e o controle de acesso em uma arquitetura IoT, uma vez, que as "coisas" se comunicam por meio das interfaces de rede o que abre possibilidade de ataques em aplicações e dispositivos (SADEGHI; WACHSMANN; WAIDNER, 2015). Os ataques ativos e passivos são exemplo de atos que se baseiam na exploração de ameaças existes nos sistemas (SICARI et al., 2015; MAHMOUD et al., 2015). Um ataque passivo é por meio de monitoramento e análise da rede verificando os pacotes trafegados; um ataque ativo é através da interferência de agentes externos utilizando os protocolos existentes para manipular a aplicação ou dispositivos.

2.3.1 Protocolos TLS e DTLS

Para prover a segurança em um sistema de informação deve-se utilizar um conjunto de técnicas e ferramentas. Para realizar a troca de informações de forma segura entre os sistemas em uma rede é necessário utilizar os protocolos de segurança para garantir que pacotes não possam ser visualizados por terceiros. Desta forma, eles permitem a comunicação segura entre os lados cliente e servidor. Os protocolos cumamente utilizados são: TLS (DIERKS, 2008) e o DTLS (SEGGELMANN, 2011).

O TLS são protocolos que implementam a criptografia que foram desenvolvidos pela necessidade de se ter um mecanismo que possibilitasse o sigilo dos dados e a garantia de autenticidade nas transações entre diferentes aplicações. Estes protocolos são implementados de modo a atuar como um camada de aplicação sobre a camada de transporte TCP/IP (DIERKS, 2008). O protocolo foi projetado possibilitando suportar diversos algoritmos de criptografia e assinatura digital. O TLS utiliza estes certificados digitais para estabelecer a autenticação mútua entre cliente e servidor parar garantir a integridade dos dados pelo uso de criptografia.

O DTLS permite a comunicação segura entre aplicações similar ao TLS, mas utiliza o UDP na camada de transporte. Entretanto, o TLS foi projetado sobre a camada de transporte TCP em que depende de algumas características, como: confiabilidade, ordem na entrega dos pacotes e detecção de repetição. Como o UDP não possui garantia que os pacotes irão chegar ou não, então, o protocolo utiliza uma abordagem de tempos de retransmissão e números de mensagens para estabelecer o canal seguro (RESCORLA; MODADUGU, 2012; MODADUGU; RESCORLA, 2004). O DTLS não oferece nenhuma melhora sobre o modelo TLS e todas suas características são para executar protocolos que utilizam a camada UDP, como o CoAP (KOTHMAYR et al., 2013).

Em ambos protocolos são necessários estabelecer o canal seguro com o cliente e o servidor. Para estabelecer a comunicação são utilizados algoritmos de criptografia. Este procedimento é chamado de *handshaking*, no qual o cliente e o servidor definem vários parâmetros usados para estabelecer a conexão segura. A Figura 8 mostra as trocas de mensagens (*handshake*) para estabelecer a conexão segura no TLS.

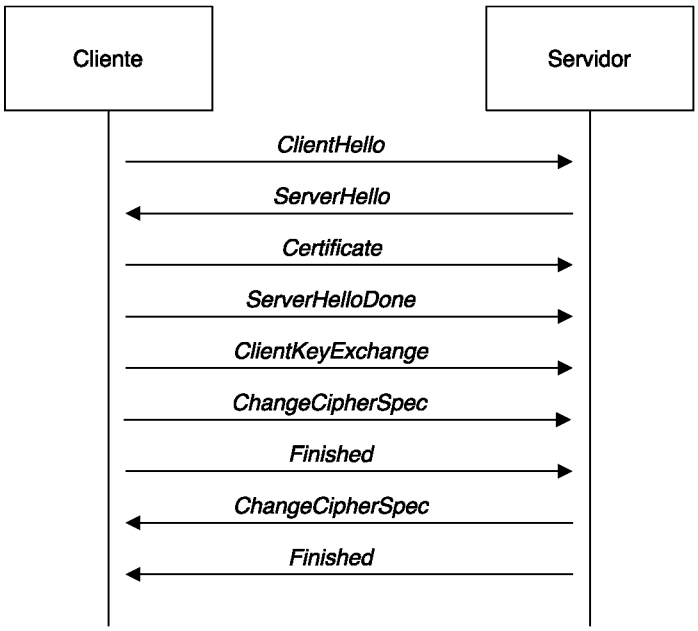

Figura 8 – Estabelecimento de uma conexão segura TLS.

Fonte: Dierks (2008)

O *handshake* é constituído por duas fases. Na primeira, é definido a chave entre o cliente e o servidor, a autenticação do servidor e a troca da chave *Master*. Já na segunda parte, são feitos a autenticação do cliente (se requerida) e o fim do *handshake*. Quando o processo do *handshake* estar concluído, a transferência de dados entre aplicações pode ser iniciada de forma segura. O cliente e o servidor usam as chaves de sessão para cifragem e decifgragem os dados que enviam para o outro e para validar a sua integridade. Caso alguma etapa durante o *handshake* falhar, a conexão segura não é estabelecida.

O DTLS suporta certificados públicos ou chave pré-compartilhada a *Pre-Shared Key* (PSK) entre as duas partes para estabelecer o canal seguro (GARCIA-MORCHON et al., 2013). Este sistema utiliza algoritmos de criptográficos de chave simétrica, ou seja, a chave é a mesma utilizada para encriptação e decrepitação das mensagens. A desvantagem está em não fornecer não-repúdio em um cenário que não se pode confirmar o grupo específico que envia a chave, já que a chave compartilhada entre os clientes. Já o TLS pode utilizar o modelo de criptografia simétrica ou assimétrica. Ela é um contra ponto da chave simétrica que utiliza protocolos de criptografia que requerem duas chaves matematicamente ligadas sendo uma delas privada e a outra pública. Esse modelo permite serviços de segurança, como: cifragem/decifragem, assinatura digital, integridade de dados e a troca de chaves.

Apesar de ser amplamente utilizado, as técnicas de segurança possuem algumas falhas e vulnerabilidades. Pode-se destacar algumas no contexto do TLS (SATAPATHY; M., 2016): *Man-in-the-middle* é um ataque que pode ocorrer no estabelecimento do *handshake*, no qual o atacante se coloca entre a comunicação com a finalidade de roubar a sessão. Isto ocorre por meio de roubo de chaves do servidor e falsos certificados digitais induzindo o cliente não distinguir a autenticidade do servidor (MEYER; SCHWENK, 2013). Outro tipo de ataque é o *Compression Ratio Info-Leak Mass Exploitation* (CRIME) em que possibilita o invasor o acesso aos conteúdos guardados em *cookies* de sessão quando utiliza o TLS. Com esses *coockies* é possível acessar informações protegidas comprometendo integridade do acesso do usuário (SARKAR; FITZGERALD, 2013). O *Renegotiation Flaw* permite um atacante injetar texto simples nas requisições feitas pelo usuário sem destruir a conexão segura (MEYER; SCHWENK, 2013). O BEAST explora uma vulnerabilidade no *cipher block chaining* (CBC) utilizada para criptografar os dados no protocolo TLS v1.0, assim, é possível descriptografar partes do pacote e *cookies* do HTTP (SARKAR; FITZGERALD, 2013).

O DTLS utiliza a camada UDP e seu *handshake* possui uma abordagem diferente do TLS, com o uso de *cookies* para evitar o *Denial of Service* (DOS) no servidor, a fragmentação e a reconstrução de mensagens (SEGGELMANN, 2011; MODADUGU; RESCORLA, 2004; MEYER; SCHWENK, 2013). O ataque de negação de serviço pode ocorrer quando um cliente deseja estabelecer um canal seguro com o servidor consumindo os seus recursos. Para evitar o problema o DTLS utiliza uma mensagem adicional chamada de *HelloVerifyRequest*. O cliente recebe o *cookie* e envia assinados. O servidor verifica a

assinatura do *cookie* para saber a autenticidade do cliente antes de alocar os recursos necessários. O DTLS possui vulnerabilidades similar ao TLS e permite a realização de ataques do tipo: man-in-the-middle (MiTM), CRIME e BEATS (FARDAN; PATERSON, 2013; SHEFFER; HOLZ; SAINT-ANDRE, 2015).

A Figura 9 mostra o estabelecimento do *handshake* DTLS. O cliente envia para o servidor uma chave *Pre-Shared Key* (PSK). Se a chave for válida no servidor é enviado um certificado para o cliente utilizado para realizar a cifragem e decifragem das mensagens. Deste modo, é finalizado o processo de *handshake* e estabelecido o canal seguro entre cliente e servidor para enviar as mensagens.

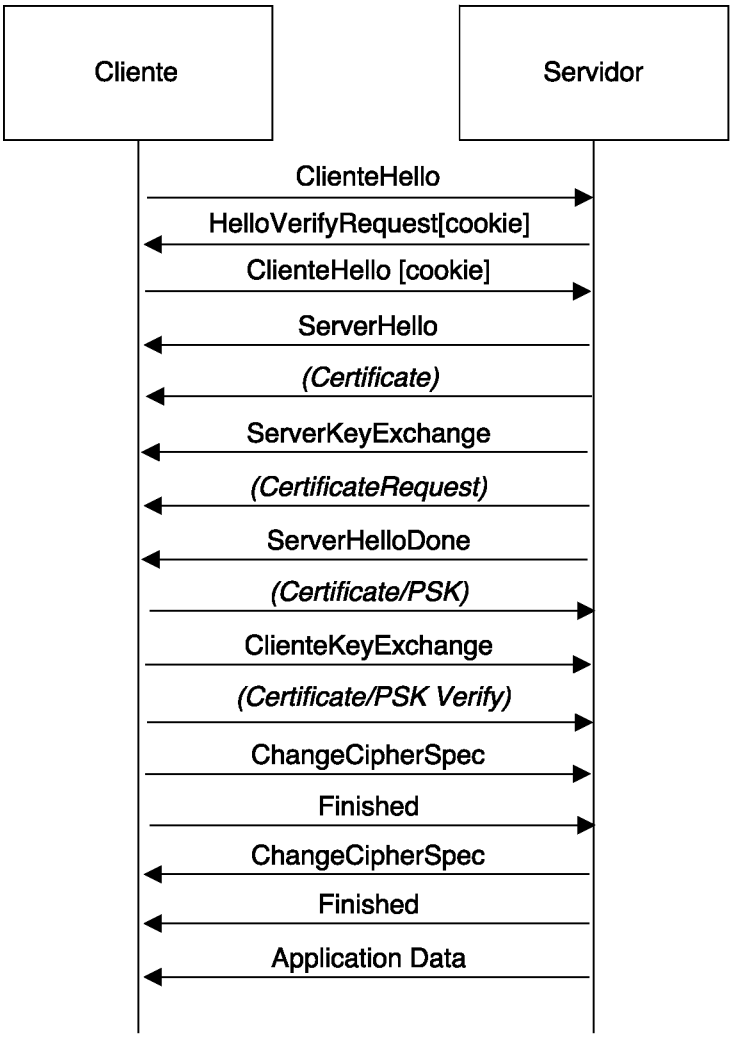

Figura 9 – Estabelecimento de uma conexão segura DTLS.

Fonte: Seggelmann (2011)

2.4 *Middleware* em IoT

Middleware consiste em uma camada de software que serve como uma interface entre os componentes IoT gerando comunicação entre os dispositivos e aplicações. Desta forma, têm ganhado destaque devido ao papel importante na simplificação do desenvolvimento de novos serviços e a integração de tecnologias (ATZORI; IERA; MORABITO, 2010). O principal objetivo é "esconder" detalhes técnicos fundamental para infraestruturas IoT. Com isso, torna-se fundamental para aplicações externas utilizarem sua API sem a necessidade de ter conhecimento de protocolos e definições semânticas (RAZZAQUE et al., 2016).

As arquiteturas de *middleware* possuem abordagens do *Service Oriented Architecture* (SOA) em que permite a criação de serviços interoperáveis que podem ser reutilizados e compartilhados entre aplicações e empresas. Assim, o uso de interfaces comuns e protocolos permitem fornecer uma visão horizontal de um sistema corporativo.

Além disso, facilita a integração entre diferentes componentes e reduz o tempo para adaptar as mudanças impostas pelo desenvolvimento do mercado. De acordo com Khan et al. (2012) e Tan e Wang (2010), a camada de *middleware* está entre a camada de rede e a camada de aplicação, no qual oferece cooperação de processos entre elas. Em IoT há muita heterogeneidade entre as tecnologias utilizadas e esta camada deve suportar diferentes requisitos de comunicação com a rede e oferecer uma visão padronizada para aplicação.

O *middleware* possui vários requisitos e funcionalidades que devem ser satisfeitos em uma aplicação IoT. Um dos desafios é deixa-los autônomos para monitorar, alocar e provisionar de forma flexível por meio de virtualização os recursos computacionais. O sistema deve estar disponível a todo tempo principalmente em aplicações críticas e deve ser resiliente para garantir a recuperação em caso de falhas. A confiabilidade deve alcançar o sistema distribuído garantindo as camadas de aplicação dos dispositivos e o processamento de dados. Desta forma, o controle de acesso, autenticação e autorização são utilizados para proteger contra ataques passivo e ativo. Os dados colhidos são importantes para as aplicações, com isso, a comunicação entre os serviços heterogêneos deve ser feita por meio de uma interface padrão (RAZZAQUE et al., 2016).

2.5 Plataformas IoT

As plataformas IoT tornaram-se uma tendência e oferecem um conjunto de *middleware* que são utilizados para o desenvolvimento de uma aplicação em IoT sendo executada na computação em nuvem. Diversas propostas são implementadas pela indústria, nesta seção, é apresentada uma comparação de seus recursos.

O AWS IoT (AMAZON, 2017a) é uma plataforma que permite conectar os dispositi-

vos aos serviços de computação em nuvem da Amazon de forma que processe informações colhidas do ambiente. A plataforma oferece uma API que permite que os dispositivos sejam autenticados e estabeleçam conexão com o AWS IoT utilizando os protocolos MQTT, HTTP ou WebSockets. O *gateway* do dispositivo do AWS IoT permite que os dispositivos se comuniquem com o AWS IoT com segurança usando um modelo de publicação. O AWS IoT disponibiliza controle de acesso e confidencialidade utilizando a camada de transporte TCP em todos os pontos de conexão do sistema distribuído. O protocolo de segurança é o TLS utilizando certificados X.509 e o método de autenticação da computação em nuvem da Amazon chamado SigV4 (AMAZON, 2017b).

Kaa (KAAIOT, 2017) é uma plataforma de *middleware* que permite a construção de soluções completas de IoT. Desta forma, fornece um conjunto de serviços e ferramentas reduzindo o custo, risco e tempo associados na implementação. Possui um modelo de esquema que armazena as informações colhidas e uma API é utilizada no dispositivo para enviar os dados à aplicação IoT Gateways que realiza a ponte pra a aplicação em nuvem KaaServer. A plataforma suporta cifragem a partir dos dispositivos por meio do TLS.

A plataforma IBM Watson IoT (IBM, 2017) consiste em um conjunto de ferramentas e componentes por meio da computação em nuvem. Possui suporte apenas dos protocolos MQTT e HTTP para comunicar o *gateway* e os dispositivos. Para aplicações são oferecidos API seguras que possibilitam utilizar ferramentas de análise de dados.

O Hub IoT do Azure (MICROSOFT, 2017) da Microsoft oferece aplicativos em computação em nuvem para IoT possibilitando os dispositivos utilizarem o MQTT, HTTP e *Advanced Message Queuing Protocol* (AMQP) como protocolo. Além disso, permite realizar a autenticação dos dispositivos para enviar os dados. Dentre as funcionalidades de suas aplicações está em habilitar recursos de inteligência artificial e análises avançadas em computação em nuvem. E também oferece a possibilidade de operar os dispositivos em modo offline ou com conectividade intermitente.

A Tabela 1 oferece uma visão das funcionalidades que são implementadas pelas plataformas apresentadas. Pode-se observar que todas as plataformas são executadas em computação em nuvem e permitem gerenciar os dados por meio de APIs próprias. Além disso, oferecem documentação para implementar nos dispositivos a comunicação dos protocolos de mensagens disponíveis. Os protocolos utilizados comumente é o MQTT e o HTTP. Mas FIWARE oferece mais protocolos, como o LWM2M/CoAP que pode ser utilizado em dispositivos com baixo recursos de rede, e também oferece o SigFox que possibilita utilizar sua própria rede de comunicação. Entretanto, o controle de acesso não é implementado em seu *gateway* de forma satisfatória e não apresenta confidencialidade de forma fim-a-fim entre seus componentes. Neste trabalho é utilizada a plataforma FIWARE e a seção 2.6 descreve detalhadamente os principais componentes utilizados para desenvolver uma aplicação real em IoT. Além disso, mostra o estado atual e evidencia questões relacionados aos mecanismos de segurança aplicado no controle de acesso e na confidencialidade.

Tabela 1 – Comparação das funcionalidades das plataformas IoT.

Requisitos	AWS IoT	Kaa	IBM Watson IoT	Hub IoT do Azure	FIWARE
Virtualização (Computação em Nuvem)	Sim	Sim	Sim	Sim	Sim
Protocolos dos Dispositivos	MQTT, HTTP, WebSockets	MQTT, HTTP	MQTT, HTTP	AMQP, MQTT, HTTP	MQTT, HTTP, LWM2M/ CoAP, SigFox
Gateway	Sim	Sim	Sim	Sim	Sim
Controle de acesso	Sim	Sim	Sim	Sim	Não implementa em todas chamadas NGSI no gateway (IoT Agent).
Confidencialidade	TLS	TLS	TLS	TLS	Não
Gerenciamento de dados por API	Sim	Sim	Sim	Sim	Sim
Documentação para Implementar	Sim	Sim	Sim	Sim	Sim
Código Aberto	Não	Sim	Não	Não	Sim

2.6 Plataforma FIWARE

A virtualização da infraestrutura e a grande quantidade de dispositivos distribuídos permitiram que pesquisas fossem direcionadas no aspecto da Internet do Futuro (FI) por empresas e governos. Tal conceito possibilitou o desenvolvimento de uma nova geração de ferramentas e serviços que poderão ser utilizados por outras aplicações. Um exemplo é o programa EU FP7 que desenvolve projetos de casos de uso para novas aplicações de FI (STRAVOSKOUFOS; SOTIRIADIS; PETRAKIS, 2016), como o FIWARE.

A plataforma FIWARE é de código aberto e disponível publicamente na Internet. Seu sucesso está na inovação e experimentação de diferentes tecnologias por meio de um conjunto interfaces padronizadas para ser utilizada no desenvolvimento de aplicações (FIWARE, 2017c). Também, o modelo de referência para cada componente FIWARE disponibilizado publicamente para ser acessado por desenvolvedores de aplicações. Outra iniciativa é o *FIWARE Acceleration Programme* que busca promover soluções integradas e aplicações para a plataforma, com foco especial para *start-up* desenvolverem inovações. O programa FIWARE Mundus foi desenvolvido para contemplar domínios e *stakeholders* de diferentes países.

A plataforma é baseada na arquitetura de referência IoT *Architecture Reference Model* (ARM) e *Cloud based* IoT (STRAVOSKOUFOS; SOTIRIADIS; PETRAKIS, 2016). O modelo de referência consiste em um conjunto de conceitos, relações para formular um *framework* de entidade em IoT. Isso possibilita uma arquitetura padrão entre diferentes tipos de *middleware*. A Figura 10 ilustra a arquitetura de referência ARM que provê um conjunto de características para o modelo *Cloud based*. Desta forma, o FIWARE são implementados os conceitos e os requisitos essenciais definidos nesta arquitetura de referência possibilitando a validação deste modelo.

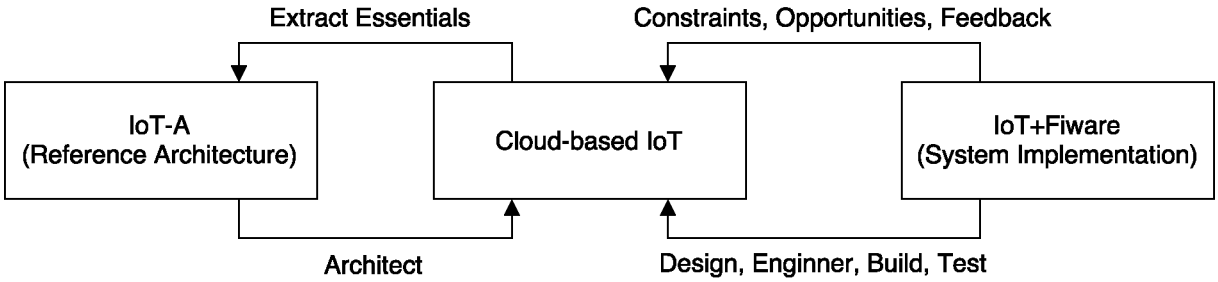

Figura 10 – Relação entre os modelos de referências de arquitetura.

Fonte: Stravoskoufos, Sotiriadis e Petrakis (2016)

Ainda é um desafio a padronização do domínio de IoT em que muitas vezes resulta em abordagens fragmentadas de sistemas (STRAVOSKOUFOS; SOTIRIADIS; PETRAKIS, 2016). Com isso, a arquitetura IoT ainda tem muitas lacunas em requisitos que são considerados importantes, como: escalabilidade, interoperabilidade e independência de aplicações. Para isso, o projeto IoT-A da EU (BASSI et al., 2013) propõe um modelo de referência para arquiteturas IOT provendo como os dispositivos interagem e são integrados com os serviços.

O modelo de arquitetura IoT que está sendo desenvolvido pela FIWARE é modular e reutilizável baseado na computação em nuvem. O projeto é baseado na plataforma aberta de infraestrutura OpenStack que visa ser um aprimoramento com funcionalidades IoT. Desta forma, implantar uma instância FIWARE é o mesmo que implantar uma instância OpenStack com um conjunto de serviços adicionais que são operação, monitoramento, verificação de nós, além de imagens que oferecem serviços do FIWARE. Esta plataforma é a principal fornecedora da UE que oferece especificações abertas para serviços que podem ser difundidos em diferentes locais geográficos por meio da internet (STRAVOSKOUFOS; SOTIRIADIS; PETRAKIS, 2016). O modelo FIWARE *cloud* define três papéis denominado como serviço a ser consumido (desenvolvedor), provedor de serviço (especificação aberta do FIWARE) e criação de serviço (implementação GE).

Uma das principais inovações é uma infraestrutura de *cloud computing* aberta em que provê aplicações e serviços por meio dos *Generic Enablers* (GEs) que podem ser

instanciados para serem executados. A Figura 11 ilustra a arquitetura da Plataforma FIWARE baseada no conceito *Cloud of Things (CoT)* que busca prover serviços abstratos de virtualização de computação em nuvem para sistemas em IoT. Com isso, é utilizando a virtualização de rede e máquinas virtuais para instanciar arquiteturas IoT com diferentes GEs.

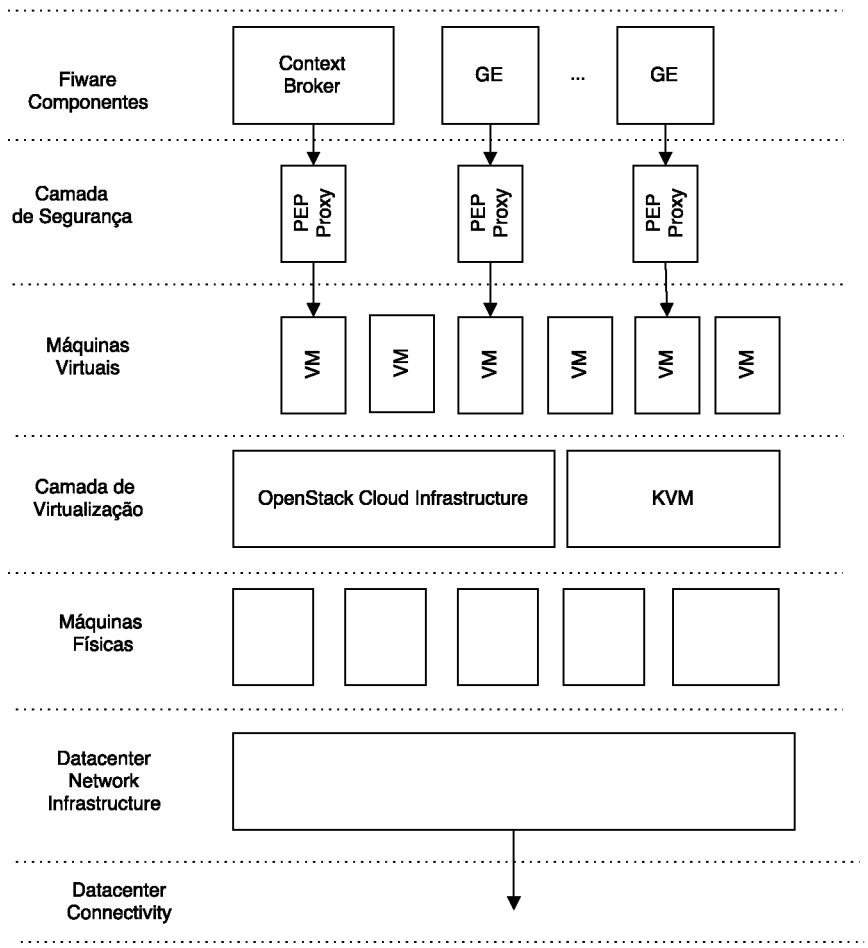

Figura 11 – Arquitetura CoT da FIWARE Plataform.

Fonte: Fiware (2017a)

A Mirantis e o FIWARE estão desenvolvendo a plataforma que possa gerenciar os componentes de múltiplas infraestruturas virtualizadas. Isso possibilita instalar o OpenStack para gerenciar os elementos de computação em nuvem (máquinas virtuais, redes, armazenamento, autenticação). As modificações realizadas visa agregar GEs de forma simplificada sem necessidade de instalar elementos nas máquinas virtuais. Suas principais características estão em escalabilidade sendo implantado por diversas locais de forma distribuída e massivamente escalável os objetos armazenados (FIWARE, 2017).

O FIWARE Lab é um serviço que permite experimentar as funcionalidades dos GEs apenas executando as instâncias disponibilizadas pelo FIWARE. Essa infraestrutura permite a comunidade de desenvolvedores e empreendedores a implantação dos GEs e a

sua exploração por meio de um catálogo. Assim, os desenvolvedores precisam selecionar de forma apropriada o GE que irá utilizar, também podem ser utilizadas em um ambiente local ou remotamente e serem ativadas de acordo com as necessidades da aplicação. O FIWARE Lab está sendo implantando em uma rede geograficamente distribuída por FIWARE Labs federados em diferentes cidades sendo operado por determinada organização específica.

2.6.1 Arquitetura

A arquitetura representa toda plataforma FIWARE com os seus principais *Generic Enablers* que utilizam a interface *Next Generation Service Interface* (NGSI-09/NGSI-10) como padrão para comunicação entre as aplicações. O padrão NGSI foi desenvolvido pela *Open Mobile Alliance* (OMA) que provê uma interface rica para ser utilizado em vários serviços web por meio do *Representational State Transfer* (REST) (OMA, 2017). Portanto, os desenvolvedores não precisam lidar com a complexidade e a fragmentação das tecnologias de IoT e cenários de implantação. Os GEs são *open source* tem ajuda de vários parceiros da plataforma FIWARE e é uma alternativa para soluções proprietárias. A Figura 12 mostra a visão geral da arquitetura FIWARE e seus módulos, uma vantagem deste modelo é que os módulos podem ser utilizados de forma independente.

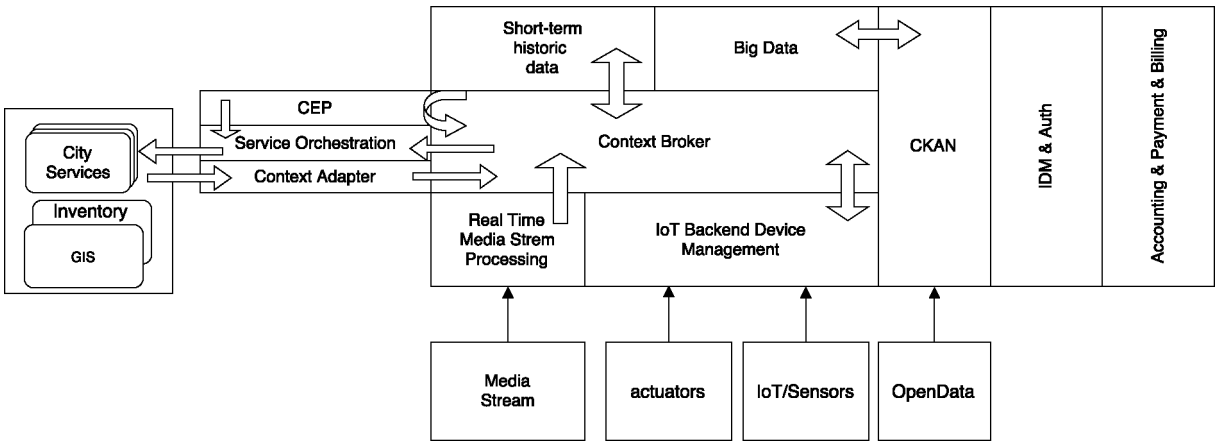

Figura 12 – Componentes da plataforma Fiware.

Fonte: Fiware (2017c)

O principal componente é o *Context Broker* que gerencia a entrada e saída de informações por meio do NGSI API. O IoT Backend Mangement consiste em gerenciar as interações e traduzir informações dos dispositivos ou *gateways* para o Context Broker por meio de chamadas NGSI. O módulo *Identity Manager* é responsável por gerenciar e realizar autenticação e autorização dos outros componentes. O *Orchestrating* é todo sis-

tema feito pelo modelo *multitenant* onde é definido acesso a determinado contexto entre os componentes.

2.6.1.1 Interfaces NGSI-9/NGSI-10

O FIWARE utiliza o padrão NGSI (FIWARE, 2017b; FIWARE, 2017a) para comunicação entre seus componentes como forma de padronizar as entidades de contexto. Deste modo, é possível produzir, publicar, consumir informações do contexto e exportar para aplicações, assim, é necessário transformar as informações providas pelos objetos em "coisas". Para conhecer o NGSI é necessário estabelecer terminologias específicas de conceitos, como: entidade, atributo e elementos de contexto *(Context Elements)*. A Figura 13 ilustra o modelo de informação definido na especificação NGSI. Esse modelo é uma estrutura de dados com informações contextuais que podem ser associadas à entidades de contexto em determinado domínio.

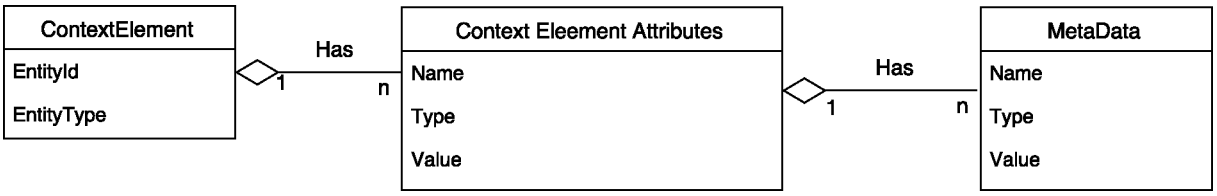

Figura 13 – Modelo de informação definido na especificação NGSI.

Fonte: Fiware (2017c)

As informações contextuais são organizadas por meio de elementos de contexto e devem possuir um identificador *(EntityId)* e um tipo *(EntityType)*. Assim, esta tupla $< EntityId, EntityType >$ identifica de forma única uma entidade. Uma entidade constitui de objetos ou partes envolvidas em um domínio, ou seja, providos de acordo sua existência no mundo real, como uma pessoa, um lugar, um objeto. Desta forma, cada dispositivo pode ser mapeado como uma Entidade associada à um contexto. No modelo do NGSI, as entidades virtuais possuem um identificador e um tipo, exemplo: entidade chamada "João" e um tipo "Pessoa". As informações que caracterizam uma entidade são representadas através de atributos. Um atributo possui $< nome, tipo, valor >$, deste modo, cada valor obtido pelo dispositivo deve ser mapeado a diferentes atributos de uma entidade como: temperatura, pressão, localização. É possível colocar atributo de geolocalização por meio de valor com as coordenadas separadas por vírgula, além de permitir consultas com restrição de áreas. O atributo metadados é composto por $< nome, tipo, valor >$ e pode ser adicionado opcionalmente. Seu objetivo é trazer informações relacionadas a um determinado atributo. A Figura 14 mostra a representação de uma entidade de contexto em formato JSON.

```
{
  "id": "Room1",
  "type": "Room",
  "temperature": {
    "value": 23,
    "type": "Float"
  },
  "position": {
    "value": 40.418889, -3.691944,
    "type": "geo:point"
  }
}
```

Figura 14 – Exemplo de uma entidade NGSI em formato JSON.

A interface NGSI possui com conjunto de chamadas padronizadas para gerenciar as entidades. Com isso, a NGSIv1 possui a especificação para NGSI-09/10 com operações distintas. Já a NGSIv2 está em desenvolvimento e atualmente apenas o Orion Context Broker suporta algumas chamadas (FIWARE, 2017d). O padrão NGSI permite que as informações sejam recuperadas de duas formas: chamadas síncronas ocorre quando produtor e consumidor trocam dados entre si formando um fluxo contínuo entre os dois elementos; chamadas assíncrona não possui uma sequência de informação transmitida, ou seja, uma entidade tem subscrição para ser notificada quando algum dado é alterado.

O NGSI-10 realiza funções que opera o contexto acerca das entidades de contexto. A Tabela 2 representa as operações que são disponibilizadas para sistemas que utilizam esta interface.

- ❑ Cadastrar, atualizar e deletar informações sobre entidades de contexto.
- ❑ Recurar informações de contexto com suas entidades e atributos.

O NGSI-9 visa gerenciar o registro e a disponibilidade de contexto. A Tabela 3 representa as operações que são disponibilizadas para sistemas que utilizam esta interface. As suas principais funções são:

- ❑ Registro e atualização de entidades de contexto.
- ❑ Consulta de disponibilidade de contexto e informações de atributos associadas à elas.
- ❑ Subscrição com o interesse de ser notificado sobre a disponibilidade de entidades de contexto e informações associadas à elas.

Tabela 2 – Operações definidas na interface NGSI-10 (FIWARE, 2017a).

Operação	**Descrição**
Update Context	Permite registar, atualizar e remover informações de uma entidade de contexto. Para cada requisição associado com o atributo *updateAction* com seguintes valores: *update*, atualiza um atributo existente na entidade ou cadastra uma nova; *append*, adiciona novos atributos em uma entidade de contexto; *delete*, remove a entidade.
Query Context	Utilizado para recuperar a informação armazenada de foma síncrona.
Subscribe Context	Permite notificar uma informação quando ocorre atualização, com isso, uma aplicação externa não necessita realizar requisições de *queryContext* para verificar a mudança no contexto.
Update Context Subscription	Permite atualizar uma subscrição de notificação.
Unsubscribe Context Subscription	Dada a identificação de uma subscrição esta função cancela para não receber notificações.
Notify context resource	Operação que deve ser implementada pelo usuário que busca executar subscrições para receber notificações da sobre as entidades do contexto. Para isso, na chamada da requisição é realizada uma consulta *{notificationURI}/QueryContext*.

Tabela 3 – Operações definidas na interface NGSI-9 (FIWARE, 2017b).

Operação	**Descrição**
Register Context	Permite registrar e atualizar informações referentes a uma entidade de contexto. A mensagem deve ter informação sobre entidade (nome, atributos, metadatos). É possível determinar um provedor de dados que pode ser um dispositivo ou uma aplicação.
Discover Context Availability	Verifica se uma entidade existe no contexto disponíveis.
Availability Context Subscription	Funcionalidade para permitir realizar subscrição para ser notificado sobre a disponibilidade de entidades de contexto.
Update Availability Context Subscription	Permite atualizar as entidades do contexto e seus atributos de interesse de uma subscrição que foi cadastrada.
Delete Availability Context Subscription	Cancela o interesse de uma subscrição que foi cadastrada.
Notify context resource	Operação que deve ser implementada pelo usuário que busca executar subscrições para receber notificações da disponibilidade de entidades do contexto.

Apesar do NGSI-9 e NGSI-10 oferecerem chamadas padronizadas e utilizando o protocolo HTTP, as chamadas não utilizam fielmente todas abstrações do estilo arquitetural REST. Para isso, o FIWARE utiliza apenas algumas conformidades definidas por meio de operações mais comuns, como: POST, GET, PUT, DELETE.

2.6.2 Orion Context Broker

O Orion Context Broker GE é um componente de *middleware* responsável por gerenciar todo o ciclo de vida da informação do contexto. Com isso, suas principais operações incluem atualizações, consultas, registros e inscrições de elementos. A principal característica do Orion Context Broker está em fornecer acesso de diferentes fontes de informação de forma independente e padronizada. Assim, se um dispositivo deixa de funcionar a sua última informação de contexto pode ser recuperada.

A Figura 15 mostra a arquitetura lógica do modelo *Publish/Subscribe* do Orion Context Broker GE, com seus principais componentes e interações. As aplicações produtoras enviam atualizações no Orion por meio da operação *update*, sendo armazenado no banco de dados. A aplicações consumidores podem consumir os dados por meio da *queryContext* ou são notificadas por meio de subscrições sobre as mudanças de determinados atributos de uma entidade de contexto.

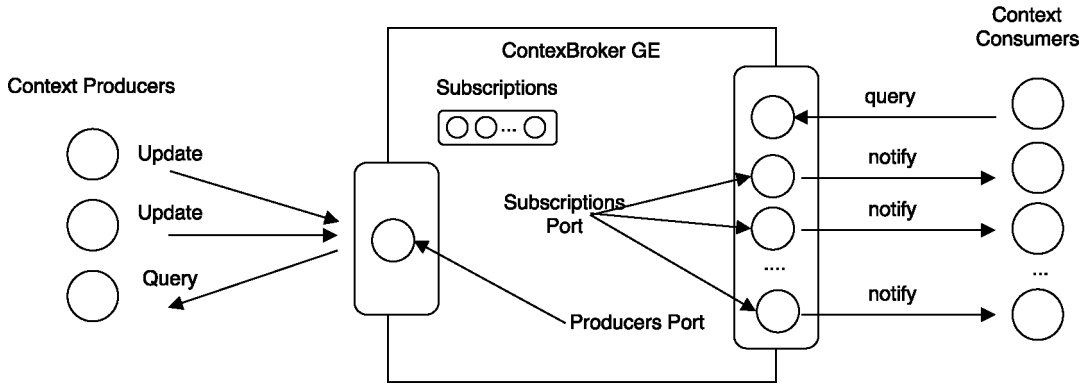

Figura 15 – Visão geral da arquitetura lógica do modelo *Publish/Subscribe* do Orion Context Broker.

Fonte: Fiware (2017c), Fiware (2017d)

O Orion Context Broker também implementa um mecanismo em que as informações do contexto não é gerenciada diretamente pelo Context Broker, mas por um *Context Providers*. Com isso, *Context Provider* é o elemento que provê os dados para o Orion Context Broker e geralmente pode realizar atualizações no contexto do Orion. Desta forma, o Orion Context Broker torna-se apenas um ponto de acesso para os dados do contexto, independente da localização do provedor. Esta abordagem tem várias vantagens: possibilita que as aplicações interagirem com diferentes pontos de acesso; se o contexto for alterado,

esta mudança será realizada de forma transparente para os clientes; a escalabilidade pode ser aumentada se for utilizado vários Orion Context Brokers. O *Context Consumer* é o elemento que consome o contexto em que geralmente são as aplicações. Com isso, o Orion pode ter diferentes papéis, em um momento trabalha como um fornecedor de informações e em outros como consumidor de dados.

O Orion Context Broker utiliza o MongoDB para armazenamento dos dados de contexto. Este banco é de código aberto, de alta performance, sem esquemas, orientado a documentos JSON. A sua escolha se deve ao fato de que as informações estruturadas no contexto podem ser convertidas neste formato. Uma vantagem deste banco para o Orion é que é mais fácil modelar as informações de modo natural reduzindo a atividade de entrada e saída. Para o funcionamento de geolocalização no Orion é necessário a versão 2.4 ou maior do MongoDB.

2.6.3 IoT Agents

As aplicações reais ligadas à Internet das Coisas possuem diferentes necessidades e requisitos, desta forma, a camada de comunicação torna-se um desafio para o modelo. Com isso, diferentes dispositivos de hardware são interligados e devem comunicar com o contexto. Entretanto, deve-se considerar que tais dispositivos podem ser de diferentes marcas e muitas vezes utilizam protocolos, codificações de dados e tecnologias distintas. Dada essa importância é essencial criar uma camada que interliga os diferentes dispositivos com seus respectivos protocolos e transforme em dados NGSI.

Desta forma, o IoT Agents consiste em software modulares que trabalham com protocolos de interação NGSI10/9, e visam ser uma camada que faz intermediação com os dispositivos e o Orion Context Broker. Este módulo destina-se a ser um *gateway* para os dispositivos disponíveis para comunicar com os restos dos componentes, baseado na arquitetura de agentes devido a modularidade para integrar dispositivos na plataforma. Pode ser considerado como um tradutor entre os dispositivos e os protocolos que enviam e recebem as informações, e possui uma linguagem comum e modelo de dados NGSI para a plataforma FIWARE. O IoT é distribuído em diferentes repositório, deste modo, significa que vários agentes com outras opções de transporte podem ser adicionados no futuro.

Um dispositivo contém sensores e atuadores e estes são mapeados a diferentes protocolos de acordo com suas funcionalidades. Assim, o IoT Agent faz o mapeamento destes recursos em uma entidade que será utilizada no contexto. Vale ressaltar que o Orion Context Broker possui informações de seu provedor que no caso é o endereço do IoT Agent que possui o dispositivo. Isso possibilita que requisições realizadas pelo NGSI possam ser redirecionadas para o IoT Agent e por fim ser executado por um atenuador no protocolo.

O IoT Agent pode pré-cadastrar um conjunto de tipos que possuem seus respectivos atributos que podem ser utilizados para associar ao dispositivo, facilitando o cadastro. Com isso, as configurações podem ser utilizadas quando um grupos de dispositivos simila-

res são conectados para um IoT Agent. Quando um dispositivo é provisionado é atribuído uma configuração se casar com o *type*, o *service* ou *subservice*. Neste caso, todas as informações de configuração é mapeadas com a informação do dispositivo para criar um *Device Object* que será armazenada no sistema. Além do mais, dependendo do protocolo utilizado, é possível realizar o registro do dispositivo sem a necessidade de intervenção humana, apenas utilizando tipo. Esta funcionalidade é importante, pois mostra como os dispositivos IoT podem se comportar de forma autônoma e o IoT Agent garante a integridade e comunicação com o contexto.

O FIWARE desenvolveu diferentes tipos de cenários no qual possibilita comportamentos diferentes realizando interações com o contexto e o protocolo do dispositivo. Estes cenários podem ser ilustrados no diagrama de sequência da Figura 16.

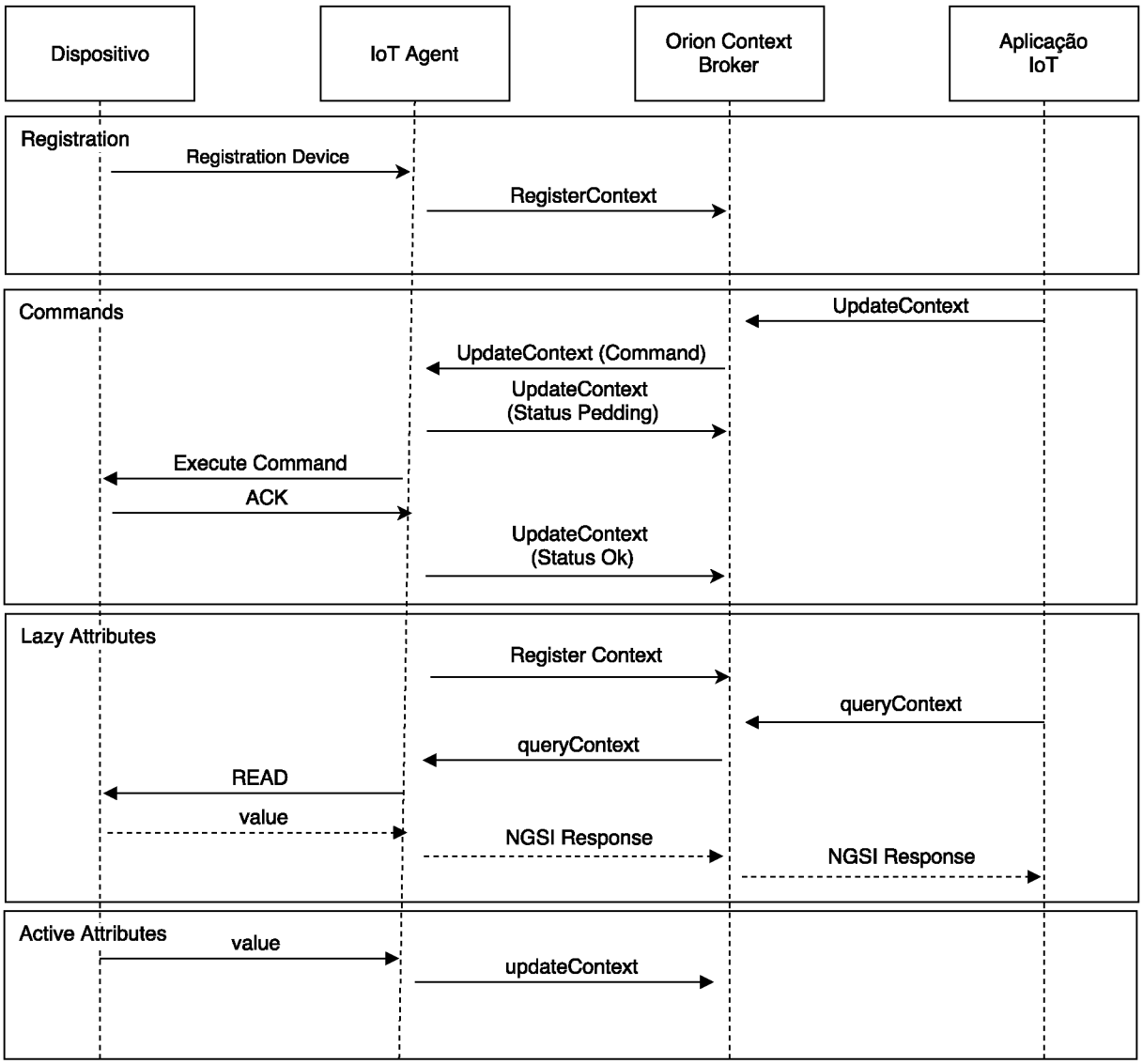

Figura 16 – Diagrama de Sequência para o comportamento dos dispositivos.

Fonte: Fiware (2017b)

Os atributos ativos são aqueles dados que são colhidos em intervalos de tempos e enviados para o contexto, com isso, os dados se mantêm sempre atualizados. Os atributos *lazy* são sensores passivos, ou seja, diferentes dos atributos ativos, os dados não são enviados para o contexto. Desta forma, cabe ao contexto enviar uma requisição para o IoT Agent que irá traduzir requisitando os dados necessários. Esta funcionalidade evita que o dispositivo fique sempre disponível enviando informações para o contexto. Os atributos *Command* são utilizados para enviar comandos ao dispositivo, sendo uma operação ou ação em que o dispositivo pode realizar. Quando uma atualização em um atributo de comando é feita no Orion, o *command* é executado no IoT Agent. Por sua vez, é enviado para o Orion informando o "status" de espera para que o resultado seja retornado ao contexto. Quando o dispositivo executa o comando uma nova atualização é realizada no Orion informando o resultado.

O "status" de informação que pode estar associado ao atributo *command* são definidos como: O *pending* ocorre quando o comando foi enviado para o dispositivo, mas ainda não respondeu e o IoT Agent está esperando a resposta para enviar ao Orion. O *expired read* é quando o comando foi expirado, o dispositivo não irá perguntar para ele. O *delivered* é quando o dispositivo recebeu o comando, mas não irá enviar o resultado. O *delivered but no respond* é quando o comando foi entregue, mas sem resultado comando a partir do dispositivo. O *OK* é quando o comando está completo, ou seja, já foi executado no dispositivo e é possível verificar no atributo "info" o resultado.

O IoT Agent desenvolvido pela FIWARE possibilita duas maneiras de executar os comandos o *Pooling* e o *Push*.

- **Pooling commands:** consiste em uma fila de comandos que será executados de acordo com um intervalo de tempo. O IoT Agent armazena uma fila de comandos pendentes para serem executados. Esse modelo é utilizado quando o dispositivo não fica online todo tempo, com isso, funções de armazenamento são utilizadas para tais comandos.

- **Push commands:** são enviados de forma síncrona, ou seja, a informação será enviada ao dispositivo e sua resposta retornada ao IoT Agent.

Portanto, o IoT Agent lida com a criação do *NGSI Contex Entity* no Orion Contex Broker por cada um dos dispositivos conectados. Também, age como um provedor *(Contex Producer)* para os atributos relacionados e provê uma administração e configuração dos dispositivos. Por ser modularizado em diferentes protocolos, o IoT Agent torna-se maior tolerante a falhas já que o Orion desempenha o papel principal do contexto e os agentes apenas traduzem informações NGSI para o protocolo. Assim, desempenhando um papel de mapeamento de conjunto de dispositivos que pode ser traduzido para o Orion. Em caso de falhas pode ser substituído por outro que possui as mesmas configurações dos dispositivos.

O FIWARE preocupou em desenvolver versões para os diferentes protocolos para serem utilizados no dispositivo com o objetivo de facilitar o desenvolvimento de aplicações em IoT, os principais são: MQTT, OMA Lightweigh M2M, JSON, HTTP, Sigfox, OneM2M e Telefonica Thinking Things. A versão disponível do IoT Agent é a IDAS 5 implementado na linguagem NodeJs.

2.6.3.1 Principais Interações

O IoT Agent conecta os dispositivos e realiza as traduções dos protocolos para o contexto. Para que esta tradução ocorra é necessário executar algumas operações.

Criação de Serviço: Para os casos onde os dispositivos não são especificados individualmente os serviços podem ser provisionados seguindo um padrão em que contém o mesmo tipo de informação formando um grupo. Um serviço é identificado por um tipo, recurso, serviço e subserviço. Os seus atributos podem compreender, como:*active, lazy e command.* Além isso, dependendo do protocolo pode-se utilizar de uma chave de *ApiKey* no qual determina quais sãos os dispositivos habilitados para enviar medições e receber comandos do IoT Agent.

Registro de Dispositivo: O registro do dispositivo pode ser feito pelo comando REST API ou automaticamente quando o dispositivo passar a enviar informações para o IoT Agent, assim, neste modelo depende de cada protocolo. Para o protocolo Lightway M2M/CoAP o registro é feito sempre que o dispositivo é ligado, já o MQTT quando irá enviar uma medição para o agente. Ao realizar um registro as informações do serviço podem ser utilizadas para mapear os atributos por meio da ApiKey ou tipo.

Atributos Ativos: O protocolo MQTT o Iot Agent faz uma subscrição no Broker para que ao chegar uma mensagem do cliente, seja enviado para o IoT Agent. Já para o LWM2M/CoAP ao fazer um registro no IoT Agent, o protocolo envia para o cliente que deseja receber notificações do atributo, com isso, é utilizado a função *observe* do protocolo CoAP. A Figura 17 mostra a sequência de operações para o registro de dispositivo utilizando o protocolo LWM2M/CoAP. Vale notar que após o registro, o cliente é notificado pelo IoT Agent quais serão os atributos que são observáveis e vão ser enviados ao sofrer alguma modificação no objeto.

Atributos Lazy: Para os atributos dos dispositivos que marcaram como *lazy*, as atualizações e leitura do contexto são mapeados para ler e escrever operações a partir da interface de gerenciamento de dispositivos. Esta funcionalidade depende do protocolo, pode notar que o MQTT não permite já é utilizado o método de *publish* e *subscribe.* Neste caso, não é possível executar já que ao realizar a subscrição o dado é notificado para o IoT Agent, desta forma, o dado não pode ser enviado por demanda ou por uma requisição do IoT Agent. O protocolo Lightway/CoAP permite que o IoT Agent faça requisições por demanda ao cliente.

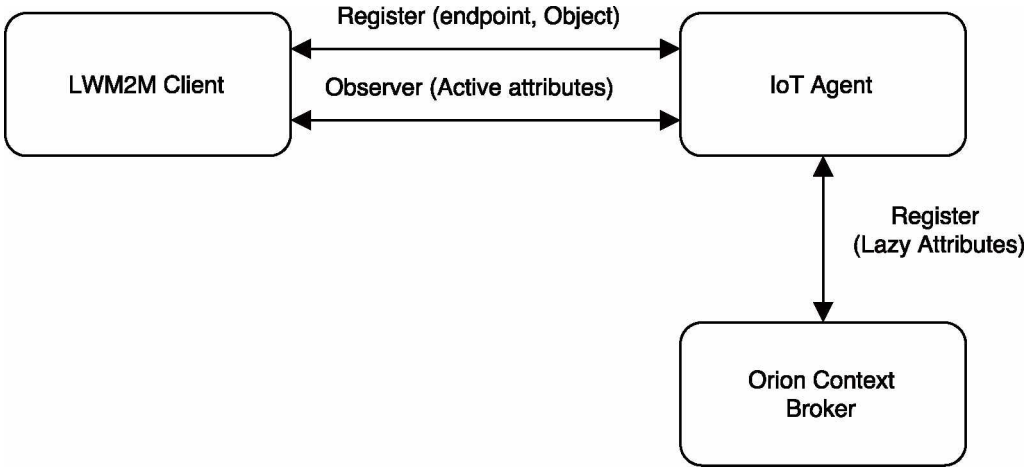

Figura 17 – Registro de Device no protocolo LWM2M/CoAP.

Fonte: Fiware (2017b)

Comandos: Os comandos são emitidos quando há a necessidade de executar alguma funcionalidade. Com isso, o protocolo Lightway/CoAP já estabelece em sua definição a função *execute* sendo utilizada para esta finalidade. O protocolo MQTT utiliza uma subscrição feita pelo IoT Agent para executar. A Figura 18 mostra o diagrama de fluxo de requisições.

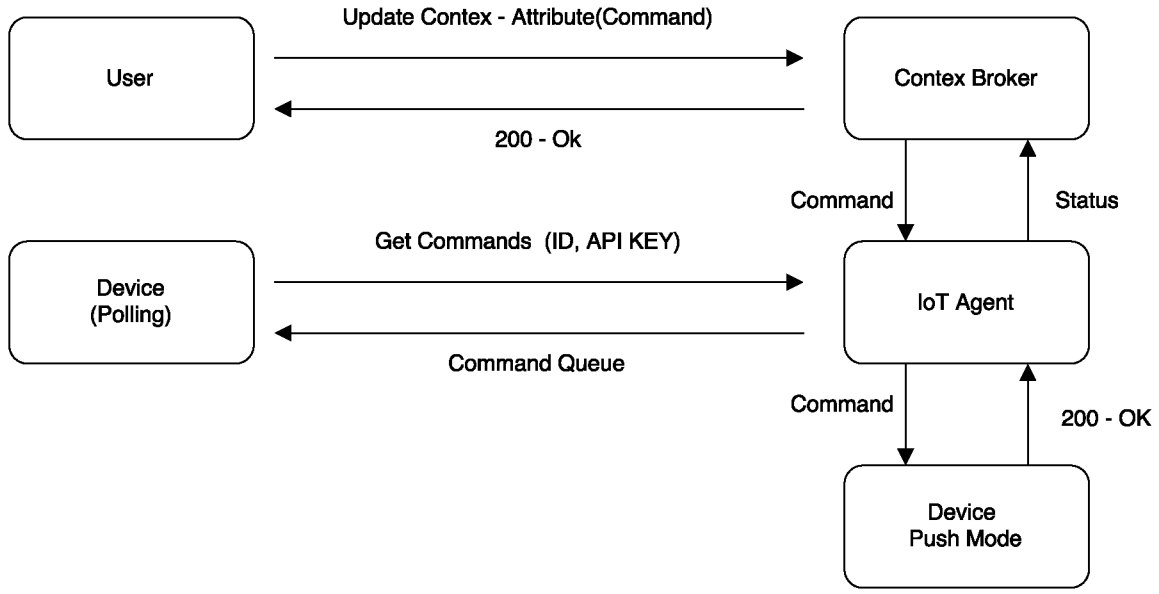

Figura 18 – Digrama para Device Commands do Ultralight 2.0 (MQTT).

Fonte: Fiware (2017b)

2.6.4 NEC IoT Broker

Consiste em um componente que permite acesso a dados de IoT com o objetivo de manipular as informações necessárias entre os IoT Agents e o Orion Context Broker. O IoT Broker é um componente sem estado já que não armazena informações detalhadas do contexto, apenas verifica a disponibilidade. Seu objetivo é executar consultas de informações a favor das aplicações, encontrar recursos provedores de informação (dispositivos) e coletar, transferir as informações entre as aplicações. Assim, o IoT Broker possibilita que vários dispositivos de diferentes protocolos estejam associados à uma entidade de contexto.

A Figura 19 ilustra os componentes IoT Broker e IoT Discovery GE interagindo junto com aplicações que pode ser Orion Context Broker ou outras softwares que possuem interface NGSI. Além disso, mostra o Context Provider que é os dispositivos que alimenta dados para entidades. Note que o Discovery GE busca associar as entidades e dispositivos, com isso, armazena as informações de configurações para serem utilizadas pelo IoT Broker. Vale lembrar que as interfaces NGSI-9 é utilizada para comunicação entre o IoT Broker e Discovery e as interfaces NGSI-10 são utilizadas entre o Orion e os IoT Agents.

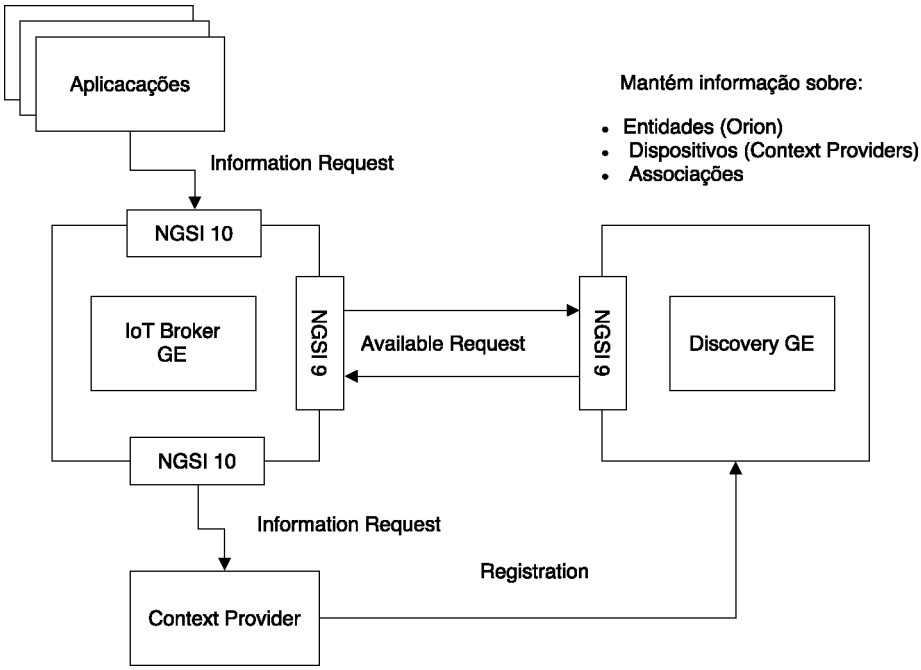

Figura 19 – Componentes do IoT Broker.

Fonte: Catalogue (2017)

A NEC Configuration Management é a implementação da especificação do IoT Discovery. O IoT Discovery interage com o AeronBroker fazendo o papel do NEC IoT Broker. Ambas implementações são desenvolvidas pela NEC em Java e suas implementações estão disponível no GitHub ou no catálogo de GEs (CATALOGUE, 2017).

2.6.5 PEP Proxy

Este módulo consiste em um *proxy* que trabalha de forma independente do FIWARE, interceptando as requisições dos componentes e fazendo a validação de controle de acesso. Esse controle de acesso utiliza módulos que realiza a validação de acesso, sendo o *Keyrock Identity Manager* (IDM) ou Keystone do OpenStack. É importante observar a utilização de serviços da *Cloud of Things*. O IoT Agent deve também possuir as configurações de controle de acesso para requisitar o token relacionado ao dispositivo.

Do ponto de vista prático, todas as requisições no Orion Context Broker devem passar pelo *proxy* primeiro. Se na requisição NGSI o cabeçalho tiver o *token* válido, é redirecionado ao Orion. A Figura 20 ilustra um exemplo com o IoT Agent enviando informações ao Orion. Para isso, o IoT Agent deve fazer uma requisição para recuperar um *token* de autenticação. Então, quando uma chamada NGSI ocorrer do IoT Agent até o *proxy*, o cabeçalho da requisição terá o *token*. Desta forma, o PEP Proxy garante que as requisições sejam aceitas ou não no Orion Context Broker.

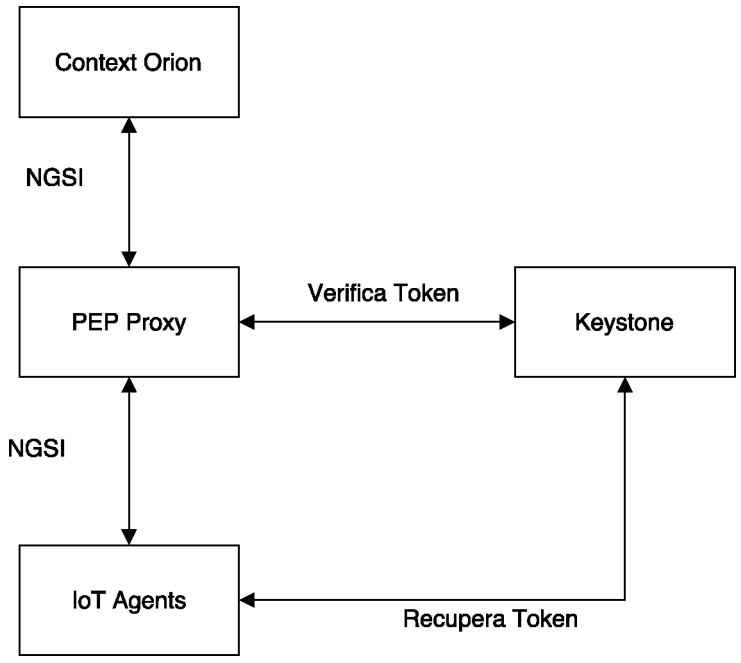

Figura 20 – Arquitetura PEP Proxy.

2.7 Cenário Atual Plataforma FIWARE em IoT

Nesta seção é apresentado a estrutura FIWARE de forma resumida que pode ser utilizada por aplicações terceiras. O FIWARE possui vários GEs que podem ser usados para as mais diversas aplicações em diferentes domínios. Para implementar uma aplicação padrão

pode ser utilizado os principais componentes: IoT Agent, NEC IoT Broker, PEP Proxy e Orion Context Broker. Em resumo, o IoT Agent consiste em um *gateway* que gerencia os dispositivos. O PEP Proxy redireciona apenas as requisições NGSI autorizadas. O NEC IoT Broker é utilizado para realizar a associação e composição dos IoT Agents, ou seja, é possível compor dados de diferentes sensores que podem se comunicar por diferentes protocolos associados a uma determinada entidade. O Orion Context Broker armazena os dados colhidos pelos dispositivos. A Figura 21 ilustra a visão da arquitetura FIWARE e os GEs necessários para serem utilizados em um cenário padrão.

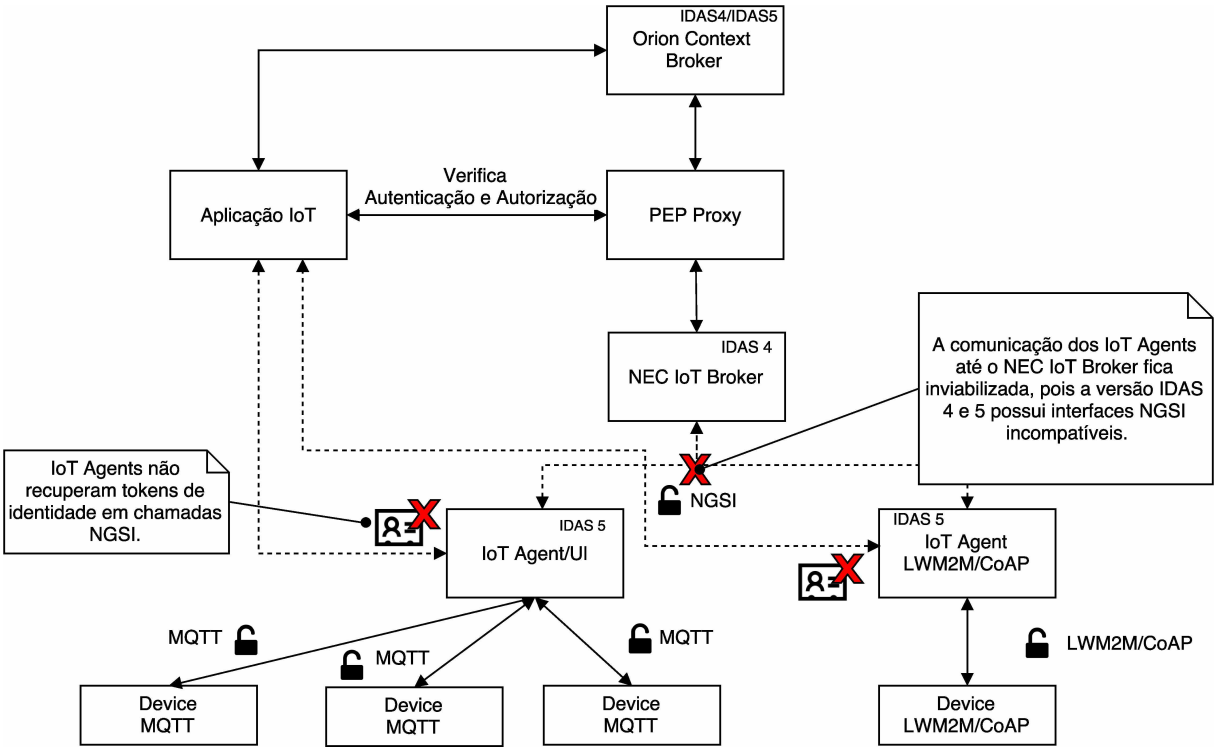

Figura 21 – Arquitetura FIWARE para um cenário padrão.

Ao executar tais ferramentas pode-se notar que haviam incompatibilidade de versões, uma vez, que a plataforma é de código aberto. Na Figura 21 observa-se que o IoT Agent está na versão IDAS 5, mas o NEC IoT Broker possui a versão IDAS 4. Como houve mudanças na interface NGSI de uma versão para outra, a comunicação entre os *middlewares* foi inviabilizada por questões de compatibilidade. Dentre as mudanças podemos citar a inclusão obrigatória de cabeçalhos na requisição para *service* e *subservice*. Assim, os IoT Agents obrigam utilizar as informações de serviços, mas o NEC IoT Broker não possui tal funcionalidade fazendo com que a associação e a composição de vários dispositivos seja inviável.

O FIWARE oferece o PEP Proxy para autorizar as requisições NGSI enviadas do IoT Agent até o Orion Context Broker. Entretanto, pode-se observar que no IoT Agent a implementação de recuperação de *token* para realizar a autenticação e autorização es-

tava incompleta nas requisições NGSI. As requisições que não implementam a capacidade de recuperar o token são *UpdateContext*, *RegisterContext*, *SubscribeContext* e *unsubscribeContext*. Podemos notar que as duas primeiras requisições são essenciais, pois são executadas sempre que um dispositivo é conectado no IoT Agent. Com isso, esta lacuna impossibilita o funcionamento correto do PEP Proxy. Assim, perde-se a capacidade de prover autenticação e autorização das requisições NSGI.

2.7.1 Segurança de Dados

Podemos considerar a implantação dos componentes FIWARE em duas visões sendo a centralizada ou distribuída. Em ambos cenários os componentes podem estar localizados em servidores para manter e processar as informações. O modelo centralizado considera que os dispositivos, IoT Agents, PEP Proxy, IoT Broker e Orion estejam no mesmo domínio de rede local. Já o modelo distribuído define-se que os dispositivos estejam em domínios de redes diferentes e que as mensagens são enviadas por meio da Internet. Esta visão leva em consideração que ao implementar um ambiente real a disponibilidade dos IoT Agents é necessária, pois pode ser utilizado réplicas de máquinas virtuais para garantir o serviço. A Figura 22 mostra as duas visões de uso da rede na plataforma FIWARE.

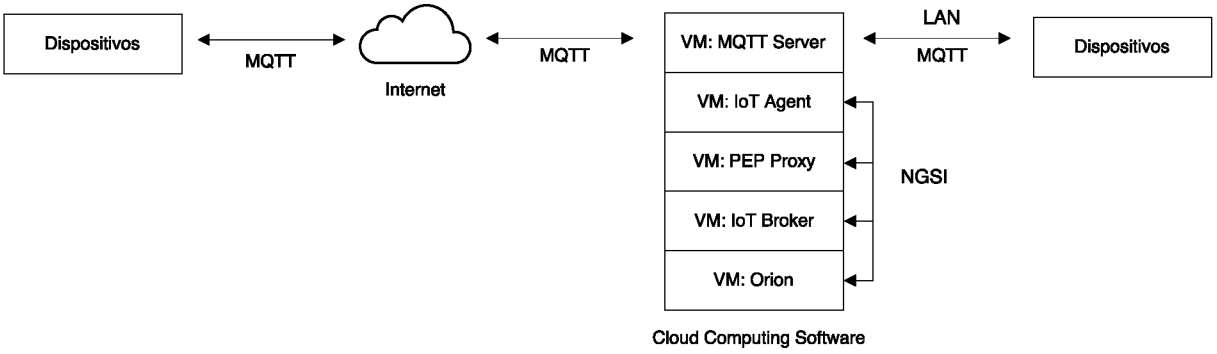

Figura 22 – Visão geral da arquitetura de rede na plataforma FIWARE utilizando o protocolo MQTT.

Se observamos a rede na plataforma FIWARE em domínios centralizados ou distribuídos, a segurança é um requisito fundamental. Uma vez que vários dispositivos estão conectados em uma rede e a garantia dos dados trafegados deve ser preservada. Nota-se que as informações podem sofrer modificações de terceiros a partir dos protocolos utilizados nos dispositivos, assim, possibilita ataques de dados na aplicação.

Foi realizado um experimento de envio de pacotes de cada protocolo de mensagem MQTT e LWM2M/CoAP utilizando os seus respectivos IoT Agents. Desse modo, o *software* Wireshark foi utilizado para monitorar os pacotes trafegados na rede. Com

isso, algumas objeções foram encontradas ao executar e ao analisar os pacotes de dados plataforma FIWARE.

Para o teste com o protocolo MQTT os mecanismo de segurança adotado no IoT Agent mostraram fracos. Já que apenas as mensagens enviadas dos dispositivos ao IoT Agent tinham autenticação de clientes no MQTT Broker, sendo possível realizar um ataque ativo por meio de inferências com este protocolo utilizando os dados de usuários e o tópico. A Figura 23 mostra o pacote interceptado.

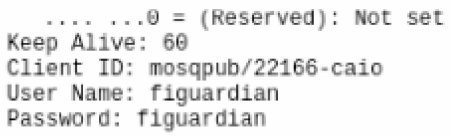

Figura 23 – Pacote MQTT interceptado informando o usuário e senha aceitos pelo MQTT Broker.

Outro problema é que o IoT Agent considera o controle de pacotes aceitos pelo MQTT Broker por meio de tópicos definidos. Assim, o MQTT Broker filtra os tópicos que estão incorretos, entretanto, quando o pacote não está cifrado um outro usuário pode utilizar o mesmo tópico para enviar dados alterados. Isso possibilitaria alterar alguma informação confidencial ou executar um comando no dispositivo sem grandes dificuldades. Com isso, a criptografia garante que elementos externos não visualizem o padrão de tópicos aceitos pelo protocolo MQTT.

Da mesma forma, o LWM2M/CoAP não possui criptografia em suas conexões implementadas em NodeJs para o IoT Agent. Também é possível visualizar as informações do pacote CoAP e executar comandos nos dispositivos. A Figura 24 mostra o pacote CoAP com o dado de uma informação requisitada pelo IoT Agent.

Foi possível notar que na comunicação NGSI realizada pelos IoT Agents as informações da entidade poderiam ser visualizadas. A Figura 25 mostra um pacote enviado do IoT Agent para outra aplicação sem criptografia. Com isso, os mecanismos de segurança quanto à criptografia das informações não foram implementadas nos IoT Agent. De fato, os IoT Agents não foram projetados pensando em segurança. Esta ausência pode causar problemas em aplicações de diversos tipos, principalmente quando se pode alterar ou executar comandos nos dispositivos, espionar informações confidenciais de várias entidades por meio da comunicação NGSI.

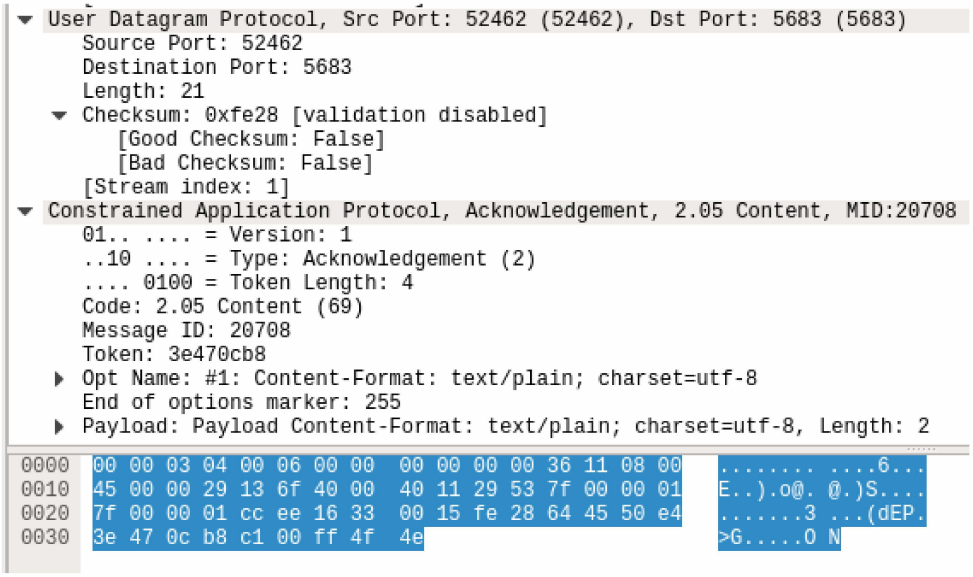

Figura 24 – Pacote CoAP interceptado com o valor ON.

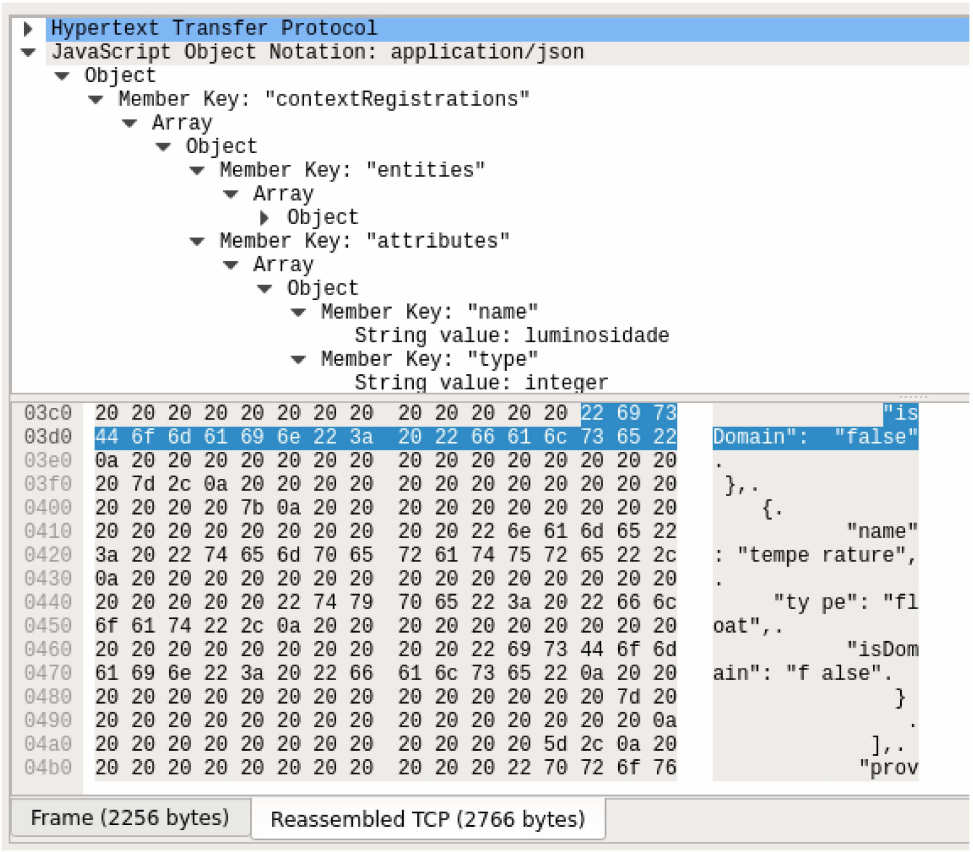

Figura 25 – Pacote NGSI interceptado com os dados de uma entidade.

Capítulo 3

Proposta e Desenvolvimento

Neste capítulo, serão expostos a proposta da arquitetura FIWARE e suas modificações para ser utilizada em uma aplicação de ambiente real. Será abordado os mecanismo de segurança de confidencialidade para os protocolos MQTT, LWM2M/CoAP e NGSI no IoT Agent. Também detalha a refatoração de controle de acesso no IoT Agent. Por fim, apresenta o novo componente IoT Broker-FI que visa ter uma nova abordagem do NEC IoT Broker para a versão IDAS 5 da plataforma FIWARE.

3.1 Proposta

A plataforma FIWARE têm a finalidade de simplificar a conectividade dos dispositivos para a Internet das Coisas. Com isso, visa encapsular as complexidades e a alta fragmentação das tecnologias direcionadas a este ambiente, ao mesmo tempo em que amplifica o potencial de sua utilização. Entretanto, o estado atual da plataforma inviabiliza a sua utilização para uma aplicação em um ambiente real em que a segurança é um requisito não funcional obrigatório. De forma resumida, a Figura 26 ilustra uma visão geral da proposta de arquitetura necessária para o uso dos componentes da Plataforma FIWARE com capacidade de suportar os aspectos de segurança envolvendo autenticação, autorização e confidencialidade de dados.

É possível observar a necessidade de implementar uma comunicação segura nos principais pontos da plataforma FIWARE. Pode-se notar que os dados trafegados dos dispositivos até o contexto podem ser visualizados por agentes externos. O IoT Agent MQTT trata a segurança de forma genérica utilizando apenas em nível de aplicação credenciais de autenticação do dispositivo por meio de usuário e senha. De fato, este recurso não garante os princípios de segurança, uma vez que os dados dos pacotes trafegados podem ser visualizados. Apesar da biblioteca MQTT em NodeJs possuir recursos de segurança em nível de transporte, o IoT Agent não incorpora tais funcionalidades. O LWM2M/CoAP desenvolvido pelo FIWARE em NodeJs não possui recursos de segurança, assim, nenhum aspecto de segurança para este protocolo é abordado em seu IoT Agent. As comunicações

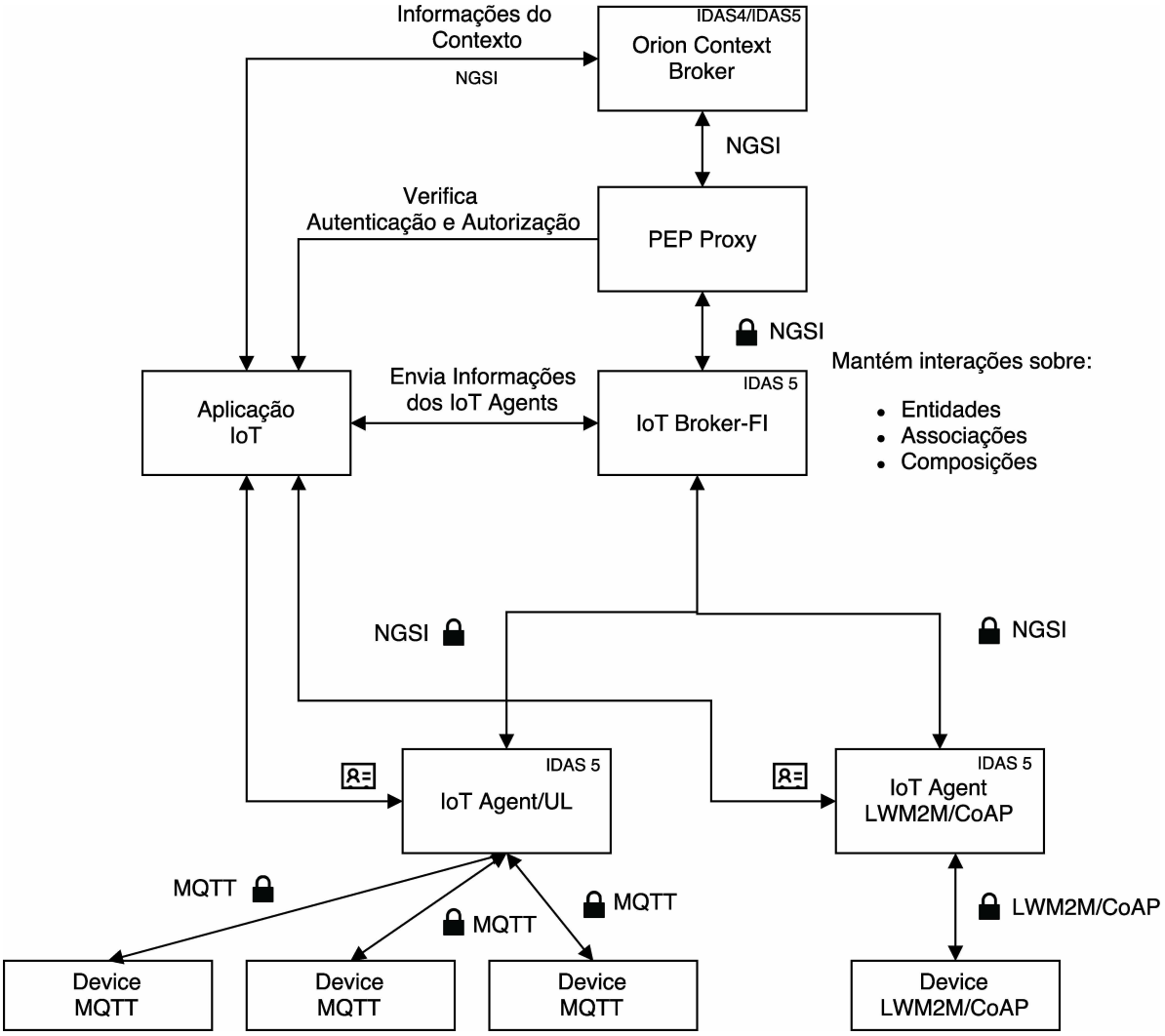

Figura 26 – Visão geral da arquitetura proposta.

NGSI podem trazer informações confidenciais dos dispositivos ou aplicações, neste caso, questões de criptografia de dados também devem ser consideradas e não são implementadas nos IoT Agents.

A autenticação e a autorização no IoT Agent estava com a implementação incompleta nas principais chamadas NGSI. Com isso, a implementação do recurso de controle de acesso torna-se importante para permitir acesso a determinadas entidades do contexto associados à um serviço. Deste modo, os IoT Agents devem realizar a autenticação e autorização para recuperar as informações de identidades que serão anexados no cabeçalho da requisição para serem validadas. Será utilizado o PEP Proxy para realizar a autorização e autenticação dos *tokens* enviados no cabeçalho pelo IoT Agent. Com isso, o PEP Proxy irá controlar o acesso das requisições NGSI enviadas para o Orion Context Broker. O NEC IoT Broker é incompatível com a versão IDAS 5 dos IoT Agents e não possui uma comunicação segura no NGSI. Dessa forma, há a necessidade de desenvolver um novo

middleware que tem como finalidade realizar associações e composições de IoT Agents de diferentes protocolos utilizando uma comunicação segura.

Este trabalho foca na segurança do ambiente FIWARE por meio da criptografia e mecanismo de segurança no IoT Agents. Outra parte do trabalho visa desenvolver um broker de comunicação para que realize associações de vários IoT Agents de forma autônoma. Na sequência, este capítulo visa detalhar os recursos de comunicação segura na seção 3.2, controle de acesso na seção 3.3 e os aspectos implementados no IoT Broker na seção 3.4.

3.2 Confidencialidade

A confidencialidade é um princípio da segurança da informação em que visa proteger os dados trocados entre emissor e destinatários. Com isso, seu objetivo é garantir que a informação será acessada por meio do controle de acesso e que a informação trocada não seja visualizada por terceiros. Para desenvolver a comunicação segura serão necessários utilizar protocolos que garantam que os pacotes trafegados compartilhem este conceito.

O IoT Agent está implementado na linguagem NodeJs e está na versão IDAS 5. O IoT Agent é composto por diversas bibliotecas e estas são utilizadas para oferecer um padrão em agentes de diferentes protocolos. O IoT Agent-NodeLib é uma biblioteca que realiza o cadastro de dispositivos, serviços, e executa funcionalidades NGSI independente do protocolo utilizado. Assim, os IoT Agents de cada protocolo importam esta biblioteca para implementar versões para diferentes protocolos. Neste trabalho será utilizado IoT Agent MQTT que possui comunicação MQTT e o IoT Agent LWM2M/CoAP que conecta os dispositivos com o protocolo OMA Lightweigh M2M.

A Figura 27 mostra a pilha de protocolos necessários para implementar a segurança de criptografia dos pacotes em um IoT Agent. Note que o dispositivo envia informações utilizando o MQTT ou LWM2M/CoAP, e tais dados são convertidos para NGSI utilizando o protocolo HTTP ou HTTPS.

Assim, o objetivo é implementar o mecanismo de segurança utilizado os protocolos existentes e adaptar para cada necessidade neste projeto. Para isto, é utilizado as tecnologias TLS para camada de transporte que utilizam *Transmission Control Protocol/Internet Protocol* (TCP/IP) e o DTLS sobre o *User Datagram Protocol* (UDP).

Utilizando as duas técnicas TLS e DTLS de criptografia de mensagens, será detalhado a implementação dos recursos de segurança na plataforma FIWARE de acordo com a arquitetura proposta da Figura 26 e a pilha de protocolos da Figura 27.

Primeiro, há a necessidade de cifrar as mensagens NGSI entre o IoT Agent, IoT Broker-FI e o PEP Proxy. Deste modo, é necessário que ambas aplicações conversem entre si por meio de servidores HTTPS que é a implementação do TLS sobre o protocolo HTTP. Com isso, as chamadas NGSI podem ser transmitidas por meio de uma conexão criptografada.

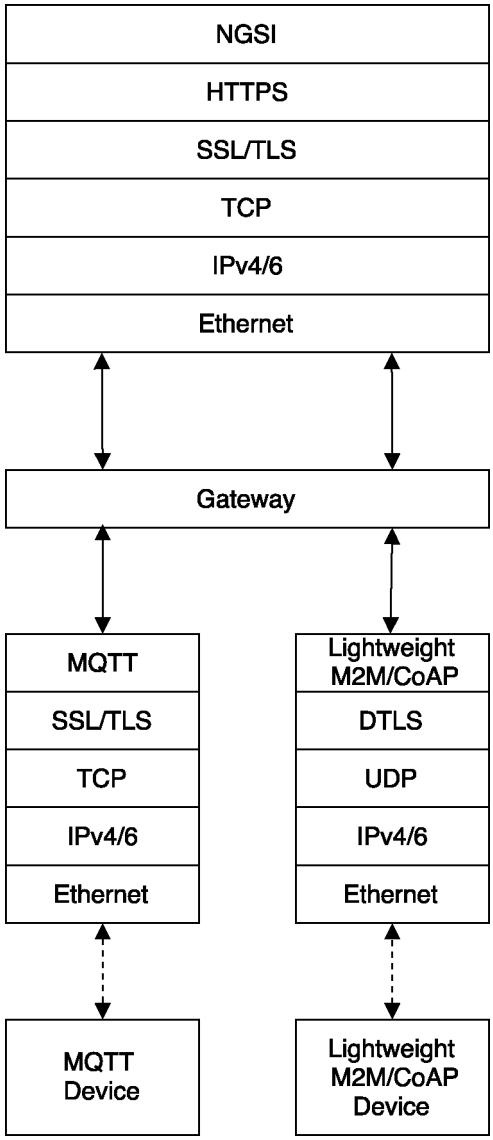

Figura 27 – Pilha de protocolos necessários para implementar a comunicação segura no IoT Agent.

A Figura 28 resume as portas usualmente utilizadas em cada componente. É necessário observar que o IoT Agent pode realizar requisições ou receber por meio de dos atributos *lazy* ou *command*, fazendo também o papel de servidor. Com isso, foi implementado no IoT Agent servidores HTTP e HTTPS para receber operações *command* e *lazy*, sendo a porta 4062 utiliza para o TLS e 4061 sem aspectos de segurança. O IoT Broker-FI possui a porta 4021 aberta para comunicação NGSI com HTTP e utilizando HTTPS na porta 4022. Essa abordagem é semelhante a servidor HTTP que usualmente utiliza a porta 80 para HTTP e 443 para suportar HTTPS. O PEP Proxy possui a possibilidade de configurar o TLS ou sem comunicação segura utilizando a porta é 1026 para fazer um controle de acesso das mensagens NGSI.

O protocolo MQTT têm como mecanismo de segurança o TLS, pois o protocolo utiliza a camada de transporte TCP/IP. O uso de criptografia se faz necessário, já que somente

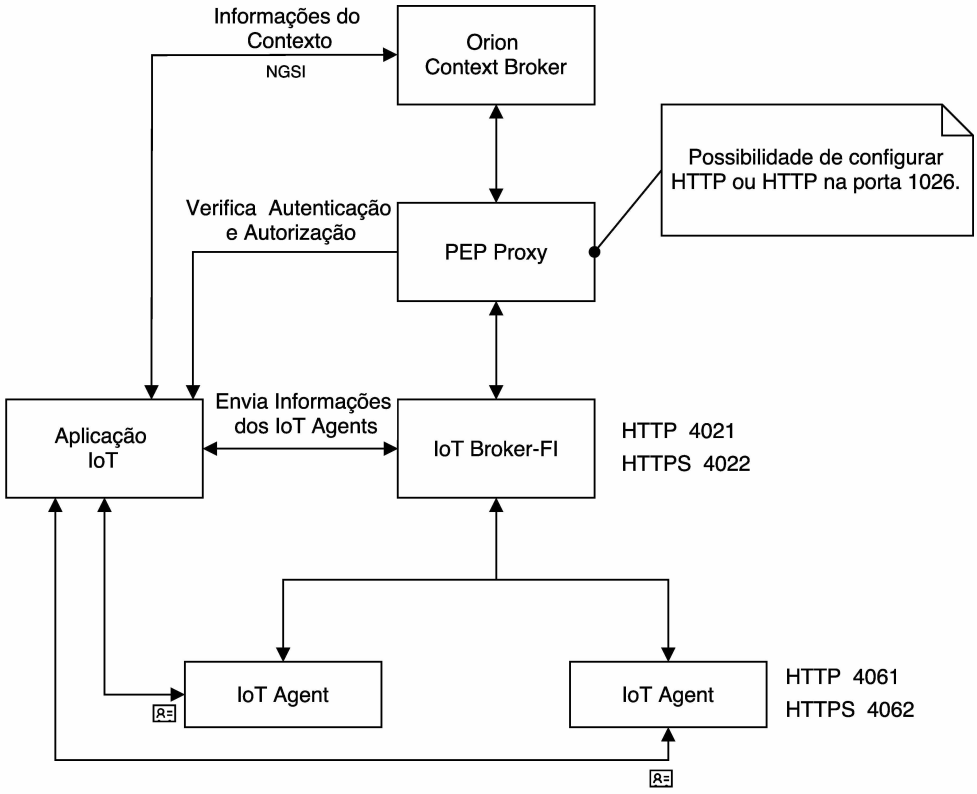

Figura 28 – Portas em uso para requisições NGSI que utilizam HTTP e HTTPS.

a autenticação do cliente ao MQTT Broker não é suficiente. Entretanto, o IoT Agent desenvolvido pelo FIWARE não incorporou tais funcionalidades que são implementadas neste trabalho.

Para implementar a segurança no IoT Agent foi necessário um estudo na biblioteca e no protocolo MQTT, já que a segurança deve estar em todas as instâncias, como: dispositivo, o servidor que hospeda o MQTT Broker e IoT Agent. A biblioteca MQTT utilizada no IoT Agent desenvolvida em NodeJs já implementa o encapsulamento dos recursos TLS e não possui o recurso PSK, apenas o uso de certificados. Para implementar no IoT Agent foi necessário mudanças na forma de conectar, estabelecer as definições de certificados nos arquivos de configuração do componente. O dispositivo MQTT pode configurar o tempo de conexão com o MQTT Broker para que o estabelecimento do *handshake* não seja em cada envio de mensagem. Este recurso é o *Keep Alive* que visa garantir o tempo em que a conexão está aberta com o MQTT Broker para um nova conexão. Além disso, é possível determinar o *connectTimeout* que é o tempo máximo de conexão do cliente com o MQTT Broker. Para que a confidencialidade deste protocolo se estabeleça é necessário que as configurações de certificados estejam definidas corretamente em cada instância no cliente, no MQTT Broker e no IoT Agent. A Figura 29 mostra os parâmetros necessários de configuração no IoT Agent onde são definidos o endereço do MQTT Broker, a possibilidade de definir autenticação e os caminhos dos certificados para utilizados para estabelecer a comunicação segura.

```
config.mqtt = {
    host: localhost,
    port: 8883,
    username: fiware,
    password: ufu,
    cert: /etc/mosquitto/easy/user3.client.crt,
    key: /etc/mosquitto/easy/user3.client.key
};
```

Figura 29 – Exemplo de configuração do MQTT no IoT Agent MQTT.

O Lightweight M2M/CoAP é um protocolo que faz mapeamento de coisas em objetos e realiza abstração do protocolo CoAP. O protocolo sobrejacente do CoAP é o UDP, desta forma, para prover a segurança não será utilizado o TLS. Para isto, o DTLS pode ser utilizado para ter as mesmas garantias como: TLS sobre o TCP. O DTLS é similar ao TLS e utiliza uma chave PSK no dispositivo. O DTLS não estava implementado no protocolo CoAP utilizado no protocolo Lightweight M2M desenvolvido pelo FIWARE em NodeJs, como definido na especificação OMA (OMA, 2017).

Para implementar as funcionalidades do DTLS no protocolo Lightweight M2M/CoAP, foi utilizado a biblioteca no repositório da Neustar no GitHub que implementa o encapsulamento do CoAP (NEUSTAR, 2017). O protocolo CoAP/DTLS desenvolvido pela Neustar utiliza o ARM mBed que é uma biblioteca código aberto para o desenvolvimento do SSL. Assim, o cliente e o servidor possui a chave PSK utilizada para manter o canal seguro. O servidor possui o certificado que será enviado para o cliente.

O protocolo Lightweight M2M é composto objetos e o gerenciamento dos recursos são feitas utilizando requisições do protocolo CoAP que possuem um significado definido na especificação do LWM2M/CoAP. Segundo a especificação Alliance (2017) ao utilizar o DTLS, o cliente e o servidor devem fazer o *handshake* e estabelecer o canal seguro antes de trocar qualquer informação. Depois de estabelecer a comunicação segura, as entidades LWM2M devem permanecer durante um longo período de tempo para evitar comprometer o contexto de segurança. Deste modo, a ideia da implementação é utilizar a primeira conexão e estabelecer um canal seguro entre o cliente e o servidor para o envio e o recebimento de dados, assim, evitando acordos desnecessários. Pode-se observar que o cliente e o servidor podem enviar e receber dados, como ilustrado na Figura 6 da seção 2.2. Esta abordagem permite que depois de estabelecer a conexão segura, apenas pacotes encapsulados CoAP serão trafegados, sendo útil no cenário *observe* em que são realizadas muitas notificações de dados no servidor.

Para implementar o DTLS desenvolvido pela Neustar no Lightweight M2M/CoAP do FIWARE foram necessários realizar algumas modificações. Já que a versão desenvolvida a conexão segura era finalizada ao realizar uma requisição CoAP. Dentre as modificações,

foram feitas mudanças no protocolo CoAP para que reutilizasse o *socket* para executar novas requisições e trabalhar como servidor de dados. Os *sockets* são utilizados para fornecer acesso aos serviços da camada de transporte que permite a troca de mensagens entre os processos no modelo cliente/servidor. Com isso, foi necessário criar um *AgentRequestSocket* e *ServerSocket* para utilizar o *socket* já aberto da conexão segura para enviar as mensagens CoAP. Assim, quando uma requisição é realizada é recuperado o *socket* do requisitante por meio do endereço e a porta para enviar as informações. O requisitante pode ser um servidor ou cliente, sendo que o servidor possui uma lista de clientes conectados. Já o cliente possui apenas um *socket* conectado com o servidor.

Para receber os dados, são utilizados a biblioteca de Eventos que visa disparar funções reativas para o *socket* de acordo com o tipo da mensagem. Um exemplo é para a funcionalidade *observe* que envia notificações ao servidor sempre que um objeto é alterado. Com isso, o *socket* trata a requisição sempre que um dado chegar. O uso de Eventos se justifica pelo fato de que foram necessários poucas modificações no protocolo, visando utilizar um único *socket* para enviar e receber dados em uma conexão segura. Com estas modificações realizadas no protocolo da Neustar, foi possível estabelecer apenas uma vez o canal seguro com o cliente LWM2M/CoAP. A Figura 30 ilustra o *handshake* e a troca de pacotes de registro entre cliente e servidor no LWM2M/CoAP.

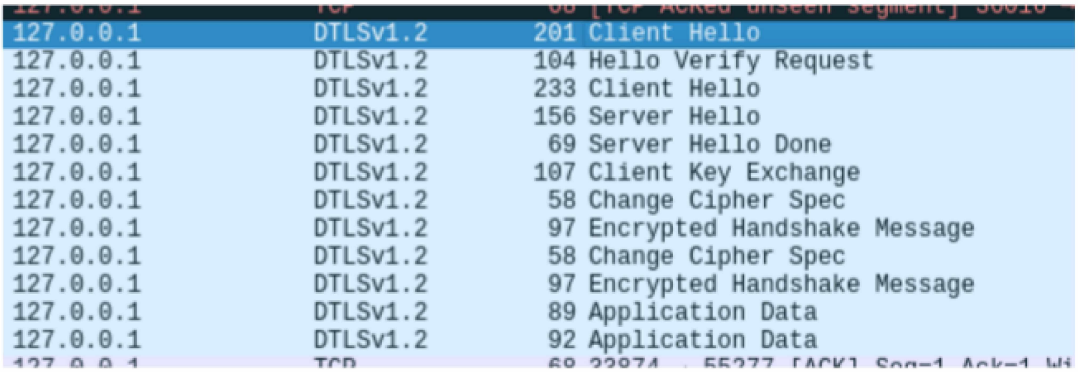

Figura 30 – Handshake do DTLS.

Por fim, para implementar o DTLS no Lightweight M2M/CoAP são necessárias as seguintes modificações e bibliotecas:

- A *Lwm2m-node-lib* implementa o protocolo OMA Lightweight M2M, este protocolo visa realizar requisições CoAP e criar um servidor no dispositivo possibilitando a manipulação de objetos. Para manipular os objetos são feitas requisições por meio da *AgentRequestSocket*, e o *ServerSocket* é utilizado para estabelecer a conexão do protocolo LWM2M/CoAP no cliente e no servidor.

- A *Node-coap-dtls* que faz o encapsulamento do CoAP e DTLS desenvolvido pela Neustar, nesta biblioteca foi realizados adaptações para reutilizar o *socket*.

- A *Node-mbed-dtls* é a camada DTLS 1.2 que compila os arquivos .cc e .h utilizando o Node-gyp. Esta biblioteca cria e manipula *socket*, servidor e cliente.

A Figura 31 mostra o diagrama de pacotes para implementar o LWM2M/CoAP no IoT Agent.

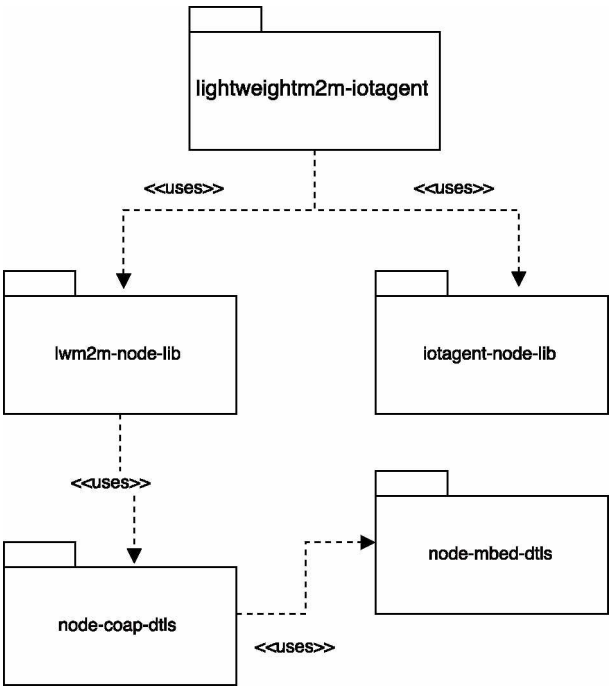

Figura 31 – Diagrama de pacotes do IoT Agent Lightweight M2M/CoAP.

3.3 Controle de Acesso

O controle de acesso aos recursos do Orion Context Broker é importante para impedir que agentes não autorizados façam alterações de dados. Esta funcionalidade de autorização e autenticação estava incompleta no IoT Agent nas chamadas *UpdateContext*, *RegisterContext*, *SubscribeContext* e *unsubscribeContext*. Isso faz com que controle de acesso não funcione corretamente no Orion Context Broker. Vale lembrar que as duas primeiras chamadas são essenciais quando um dispositivo inicia o processo de envio de dados para o contexto.

Para isto, foi necessário refatorar o código do IoT Agent para implementar os recursos necessários de controle de acesso. Uma parte deste trabalho visar implementar chamadas que recuperam o *token* no IoT Agent que permitem que as requisições NGSI sejam autenticadas e autorizadas no PEP Proxy. Assim, o PEP Proxy verifica se *token* é válido para rejeitar ou permitir a requisição para ser enviada ao Orion Context Broker.

Para que o controle de acesso funcione é necessário que o usuário cadastre as informações em uma aplicação de segurança. Esta aplicação pode ser o Keystone do OpenStack,

KeyRock ou uma aplicação personalizada desenvolvida para o projeto IoT. As informações que devem ser cadastradas são tipos de entidade que representam um grupo de dispositivos, serviços e subserviços. O sistema de autenticação deve fornecer uma chave específica para este tipo de entidade. O usuário deve cadastrar no IoT Agent informações mais detalhadas do tipo de entidade, como: chave de permissão, endereços do contexto, dados do protocolo, atributos separados em *lazy*, *active* e *command*. Quando o IoT Agent é acionado por algum dispositivo, é feito uma requisição de *token* na aplicação de segurança com os dados de usuário, senha e a chave da entidade. Este sistema verifica se os dados conferem, em caso afirmativo, um *token* é enviado para ser anexado no cabeçalho da requisição NGSI.

O *token* permite que os dispositivos acessem recursos privados das entidades durante um certo período. Cabe a aplicação de segurança que fornece o *token* definir este tempo de acesso. A Figura 32 mostra o digrama de sequência detalhando os passos necessários para realizar a recuperação do *token* no IoT Agent.

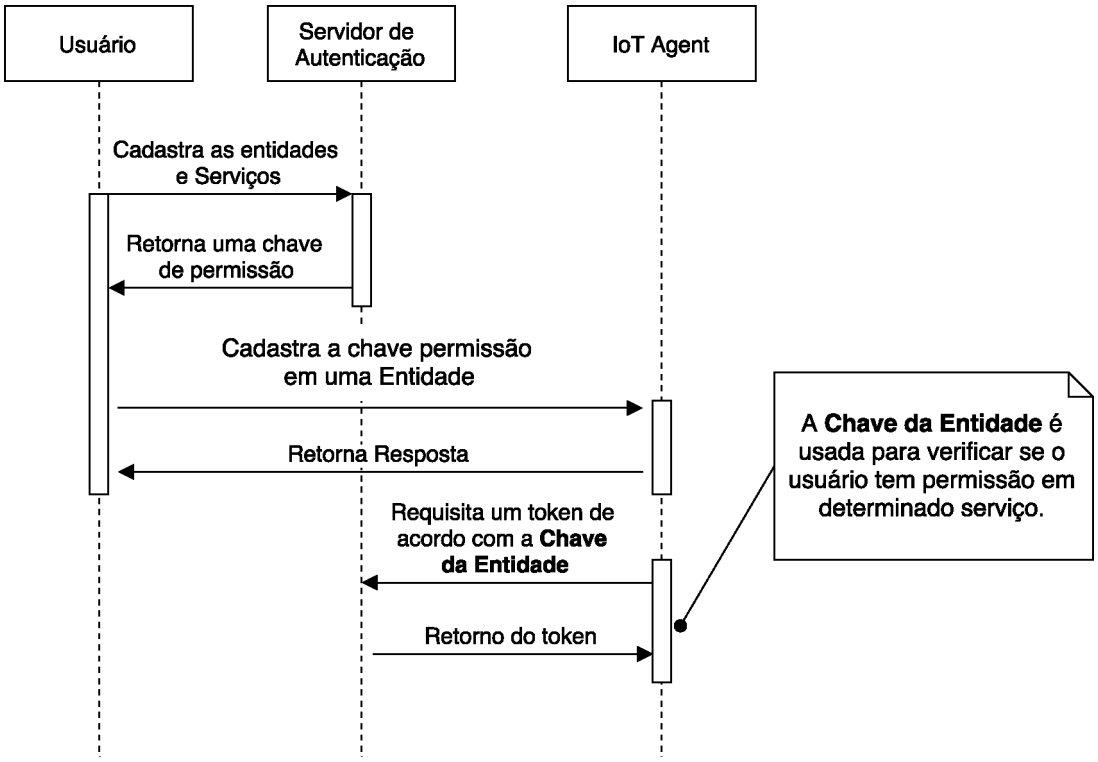

Figura 32 – Diagrama de sequência o IoT Agent recuperar o *token*.

Quanto a refatoração das chamadas houve mudanças significativas em várias funções, uma vez, que as interfaces NGSI estavam distribuídas no código-fonte. Um desafio encontrado foi fazer com que as chamadas *RegisterContext* e *UpdateContext* requisitasse apenas uma vez o *token* de acesso, já que são processadas em sequência quando um dispositivo é ativado. Para isto, foi armazenado os dados do *token* temporariamente em *cache* juntamente com os dados do dispositivo para que seja recuperado as informações de forma rápida e sem requisições desnecessárias.

Vale observar que o IoT Agent faz o papel de servidor e pode receber do Orion Context Broker requisições NGSI *command* e *lazy* para executar funções nos dispositivos. Entretanto, é necessário realizar o controle de acesso para que elementos externos não realizem alterações no IoT Agent em que pode realizar comandos e acessar dados no dispositivo. Neste caso, todas as requisições inclusive do Orion Context devem ter no cabeçalho *X-Auth-Token* válido. A Figura 33 mostra uma requisição NGSI provida de uma aplicação IoT com o *token* anexado.

```
{
    "url": "https://localhost:4062/v1/updateContext",
    "method": "POST",
    "headers": {
        "fiware-service": "universidade",
        "fiware-servicepath": "/facom",
        "X-Auth-Token": "gAAAAABYUm3xr2bBe1yA1GeD2V-2S_Ka3Q2c2Bzss44ntrUbw"
    },
    "json": {
        "contextElements": [
            {
                "type": "Room",
                "isPattern": "false",
                "id": "Room1",
                "attributes": [
                    {
                        "name": "turn",
                        "type": "string",
                        "value": "OFF",
                    }
                ]
            }
        ],
        "updateAction": "UPDATE"
    }
}
```

Figura 33 – Exemplo requisição com o *token* no cabeçalho *X-Auth-Token*.

3.4 IoT Broker-FI

O componente NEC IoT Broker do FIWARE visa executar no servidor como um *middleware* e seu objetivo é manipular as informações necessárias entre os IoT Agents e o Orion Context Broker. Assim, possibilita que uma entidade tenha informações providos de diferentes IoT Agents o que significa que as informações colhidas de diferentes dispositivos e protocolos. Entretanto, este *middleware* disponibilizado pelos desenvolvedores

não é compatível com o IoT Agent e Orion Context Broker por causa das interfaces de comunicação NGSI defasadas, fazendo com que suas funcionalidades deixem de existir na arquitetura proposta. O NEC IoT Broker obriga que toda entidade seja armazenada manualmente no IoT Discovery, neste caso, torna-se inviável o registro manual para aplicação real.

Como o NEC IoT Broker possui interfaces NGSI defasadas impossibilitando a comunicação com o IoT Agent, será necessário desenvolver um novo componente IoT Broker-FI que visa englobar as mesmas características, mas com outras funcionalidades e metodologias aplicadas. Este novo método visa ler a requisição NGSI e armazenar as informações colhidas sem intervenção humana, diferente do modelo proposto pela NEC IoT Broker em que era necessário registrar cada entidade.

Este novo broker possui a função de criar novas aplicativos por meio de *plugins* que podem ser acoplados no sistema de forma independente e trabalham em conjunto para realizar uma tarefa. Assim, este aplicativo permite que a troca de mensagens contenha apenas as informações necessárias dos IoT Agents ajudando os desenvolvedores sem seus projetos. Um exemplo implementado é a funcionalidade de enviar informações das entidades (identificação, tipo, endereço do IoT Agent, duração do objeto) para realizar integração de dados aplicações externas. Também, pode fazer o controle de acesso de acordo com a arquitetura dos componentes FIWARE a ser utilizada. De fato, este ponto de comunicação possibilita a criação de pequenas funcionalidades de forma rápida. Também, permite que as requisições sejam processadas por apenas uma camada de *middleware*, em vez de utilizar diferentes aplicações em máquinas virtuais distintas gerando tráfego desnecessário.

Portanto, o desenvolvimento IoT Broker-FI têm por finalidade em agregar funções providas do IoT Broker de forma que considere compatibilidade com os IoT Agents e permita interagir com grande número de provedores e consumidores de forma automática. Vale lembrar que o *middleware* deve possuir comunicação segura nas requisições NGSI utilizando o TLS, da mesma forma que a implementação do IoT Agent. Com isso, foi desenvolvida uma nova versão em NodeJs seguindo os mesmos padrões arquiteturais adotados pelo desenvolvedores FIWARE. Em síntese será detalhado as principais funcionalidades do IoT Broker-FI nas seções 3.4.1 e sua arquitetura de software na seção 3.4.2.

3.4.1 Interações NGSI entre Componentes

Nesta subseção são descritas as principais interações da comunicação NGSI providas pelos IoT Agents e Orion Context Broker. Em todos cenários apresentados o PEP Proxy faz controle de acesso no Orion Context Broker.

3.4.1.1 RegisterContext

Para que uma entidade exista no contexto Orion, o IoT Agent realiza o mapeamento de atributos do dispositivo e cria uma entidade. Assim, o IoT Agent envia para o IoT Broker-FI o *RegisterContext* que redireciona para o Orion a fim de criar o Registro de Contexto (*Context Registrations*) que tem uma combinação de entidades e seus atributos. A Figura 34 ilustra a arquitetura com as setas que representam o sentido das requisições e os números consiste na ordem.

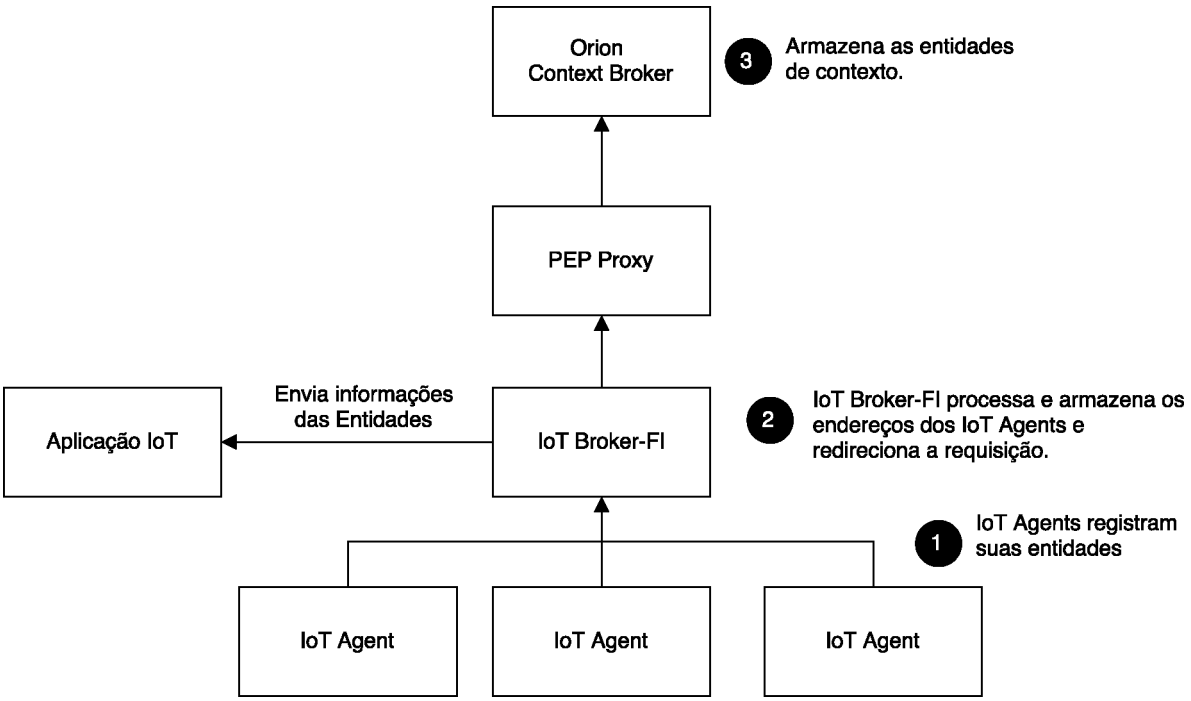

Figura 34 – Visão geral da arquitetura para a função *RegisterContext*.

Quando o IoT Broker-FI recebe este registro, é executado vários eventos: as entidades são enviadas para uma aplicação externa com seus respectivos endereços do IoT Agent denominada de *providingApplication*; as entidades enviadas para o Orion Context Broker são alterados os *providingApplication* com o endereço do IoT Broker-Fi. Isso ocorre, pois quando uma requisição de consulta (QueryContext) do Orion Context Broker for redirecionada para o IoT Agent, será enviado para o IoT Broker-FI que processará as requisições.

A Figura 35 ilustra a requisição JSON enviado pelo IoT Agent ao IoT Broker-FI. A Figura 36 mostra os dados que são armazenados no IoT Broker-FI e os informações que podem ser envidas para aplicações externas e armazenadas no IoT Broker-FI. Note que os atributos são desnecessários, pois o mesmo fica armazenado no Orion Context Broker que são acessado pelas aplicações externas utilizando as informações enviadas pelo IoT Broker-FI.

```
{
    "contextRegistrations": [
        {
            "entities": [
                {
                    "type": "Room",
                    "isPattern": "false",
                    "id": "Room1"
                }
            ],
            "attributes": [
                {
                    "name": "temperature",
                    "type": "float",
                    "isDomain": "false"
                }
            ],
            "providingApplication": "http://endereco:porta"
        }
    ],
    "duration": "P1M"
}
```

Figura 35 – Exemplo de uma requisição NGSI com *Context Registrations*.

```
[
    {
       "type": "Room",
       "isPattern": "false",
       "id": "Room1",
       "providingApplication: "http://endereço:porta",
       "duration": "P1M",
       "registrationId: "58a1ef0d3c1468c7942aa451",
       "service": "ufu",
       "subservice": "/facom",
       "expiration": "1487010094065"
    }
]
```

Figura 36 – Entidade armazenada no IoT Broker-FI e enviada para aplicações externas.

3.4.1.2 QueryContext

O Orion pode recuperar as informações que estão disponível nos dispositivos denominados de atributos *lazy*. Quando uma consulta é acionado no Orion Context Broker a requisição irá para IoT Broker-FI. De fato, o broker recupera os endereços dos IoT Agents para realizar uma série de requisições de cada elemento. Quando todas as requisições re-

tornam seus resultados, é concatenado ou realiza fusão dos atributos iguais de diferentes IoT Agents. Um diferencial do NEC IoT Broker é que IoT Broker-FI possibilita a escolha do método de fusão de atributos, e este recurso pode ser importante em determinada aplicação em IoT. A Figura 37 mostra o diagrama de sequência para uma consulta de contexto.

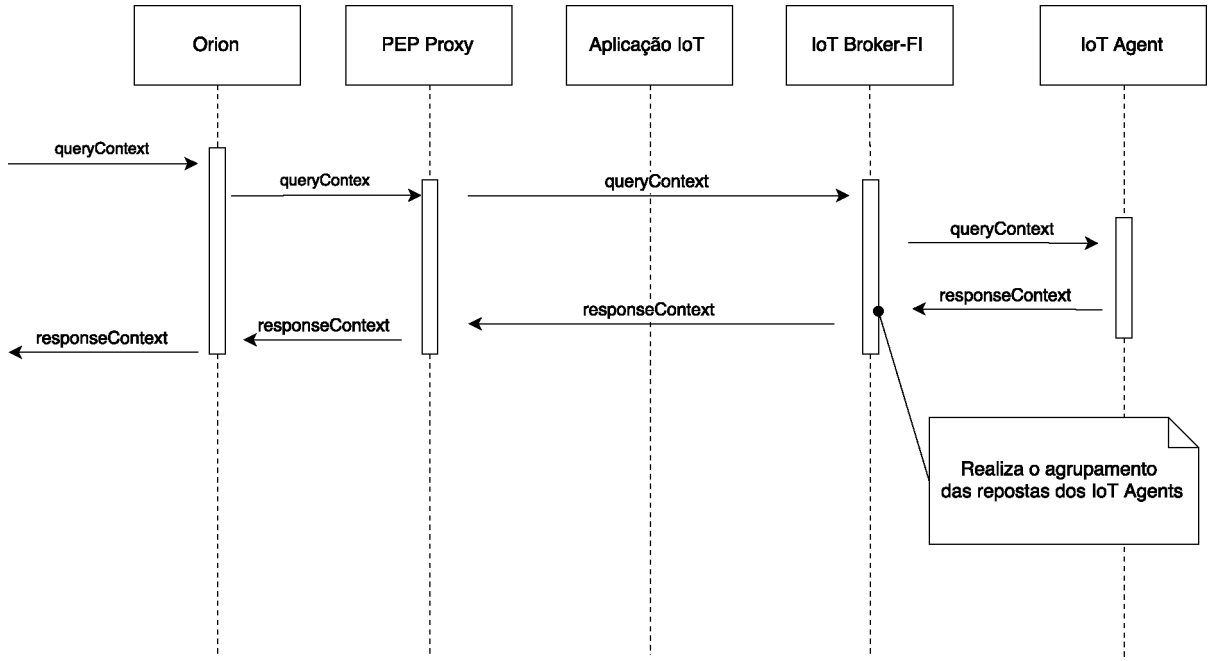

Figura 37 – Diagrama de sequência de uma consulta de contexto.

3.4.1.3 UpdateContext

Se a requisição for *UpdateContext* existem duas origens:

- IoT Agent: A requisição irá enviar para o IoT Broker-FI que redireciona para o Orion Context Broker.
- Orion Context Broker: Este evento ocorre no cenário do FIWARE para atributos *Command*. Quando ocorre uma requisição neste cenário, o IoT Broker-FI recupera todos os endereços dos IoT Agents associados na entidade e redireciona a requisição.

A Figura 38 mostra as requisições em duas origens Orion Context Broker e IoT Agent. Em (a) apenas redireciona a requisição de atualização no Orion Context Broker e não há alteração no estado IoT Broker-Fi. Em (b) mostra que ao receber uma requisição do Orion Context Broker, ela é redirecionada para os diferentes endereços do IoT Agent. Os números mostram a ordem de execução das chamadas NGSI.

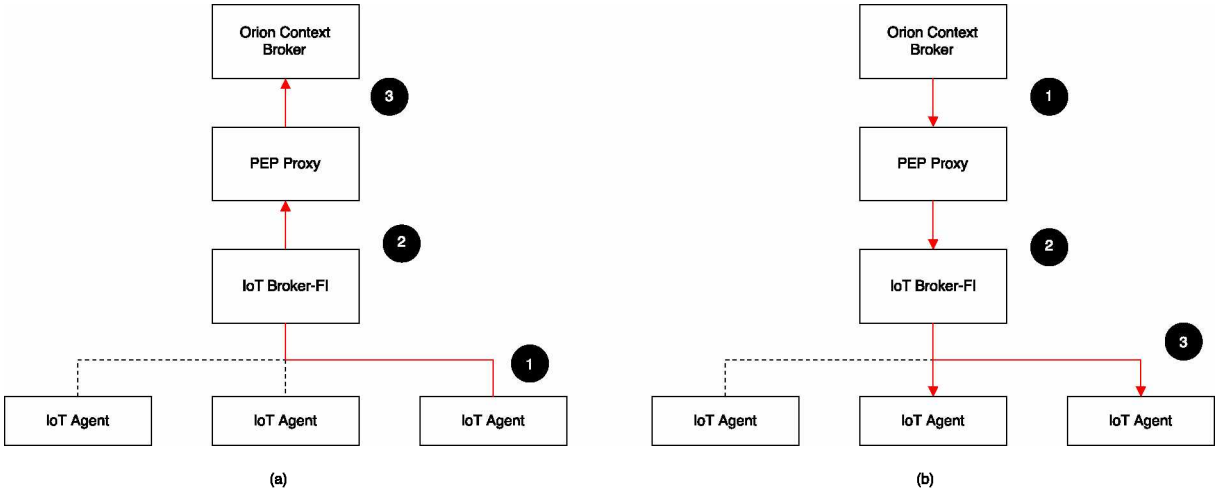

Figura 38 – Visão do *UpdateContext* na arquitetura.

3.4.2 Arquitetura

O desenvolvimento deste módulo é na linguagem NodeJs utilizando a estrutura web de roteamento da biblioteca do Express (NODE, 2017). A razão de desenvolver com estas características é que o IDAS 5 segue este mesmo padrão. A vantagem de utilizar esta linguagem é que o NodeJs utiliza o modelo entrada/saída (IO) não-bloqueante, assim, nenhuma tarefa pesada de IO irá travar a aplicação. Com isso, essas tarefas são executadas em *background* e as funções são finalizadas por meio do tratamento de funções de *callback* que podem retornar falhas ou sucesso de tarefas.

No Express são utilizados as funções de *middlewares* cujo o objetivo é ter acesso ao objeto da solicitação (*request*) e objeto de resposta (response) e a próxima função no ciclo de solicitação-resposta da aplicação. Quando chega uma requisição no IoT Broker-FI é verificado os parâmetros da URL se a operação é valida de acordo com o padrão de operação definidos pelo FIWARE. Depois é verificado se o conteúdo da mensagem está no padrão NGSI. Foi implementado uma função de adicionar um *plugin* personalizado para ser executado nesta pilha de tarefas. Este arquivo com *plugin* deve respeitar os parâmetros padrão do *middleware* do Express. Com isso, é possível desenvolver novas funções para uma aplicação. Depois, a requisição será processada de acordo com o tipo de interação detalhada na seção 3.4.1. Vale lembrar que para cada interação *register, query, update* é importante verificar quem é o requisitante, podendo ser IoT Agent ou Orion Context Broker.

O IoT Broker consiste em um *middleware* que envia e recebe requisições NGSI de elementos externos. Com isso, para que a comunicação NGSI seja segura foi implementado dois servidores sendo o HTTP e HTTPs para receber as requisições criptografadas.

Seguindo o modelo do FIWARE IDAS 5, suas aplicações permitem que as informações sejam utilizadas apenas em memória ou em bancos de dados (MongoDB). Assim, o IoT

Broker-FI também possui duas formas de armazenamento: a memória, ou seja, ao fechar o sistema todos os dados são perdidos; o MongoDB armazena os dados em JSON de maneira persistente. As informações armazenadas são os dados das entidades e do IoT Agent, ilustradas anteriormente na Figura 36. Tais informações podem ser alteradas no IoT Broker-FI utilizando recursos REST API, as principais funções de manipulação nas entidades são: adicionar, atualizar, listar, remover. Foi implementado controle de acesso utilizando *tokens* para não permitir que alterações indevidas dos dados dos IoT Agents armazenados.

O Orion Context Broker utiliza a duração para definir um ciclo de vida de uma entidade, assim, quando este tempo termina, as entidades são expiradas de forma automatica ou manual. Assim, o IoT Broker implementa o conceito de duração (*Durations*) que visa definir um período de tempo descrito de forma padrão. Este conceito foi desenvolvido pela International Organization for Standardization (ISO) e é utilizado a norma 8601 para representação de data e hora. As durações são representadas por um conjunto de letras pré-definidas na documentação. Estas letras possuem um significado de tempo que são associados com números definindo um intervalo de tempo. Este recurso mostra que os dispositivos podem entrar em funcionamento de forma arbitrária.

Por fim, a implementação da arquitetura do IoT Broker-FI é organizada nos seguintes pacotes: Em **Common** contém funcionalidades comuns relacionado ao *middleware* do Express.js. Em **Entity** relaciona à operações de armazenamento de objetos entidade. Em **Middleware** que possui funcionalidades do *proxy* e administração de *logs*. Já o **NGSI** realiza operações de tradução da camada NGSI do FIWARE. Em **Plugins** contém arquivos que contém funções personalizadas para serem desenvolvidas por aplicações externas. Em **Northbound** oferece funcionalidade REST à determinadas funções, como: manipulação das entidades armazenadas no IoT Broker-FI podendo registrar, atualizar, listar e remover. Na pasta **Templates** contém arquivos JSON estruturais que verifica se o corpo passado pela mensagem corresponde ao padrão NGSI. A Figura 39 mostra o diagrama de pacotes do IoT Broker-FI com os respectivos relacionamentos.

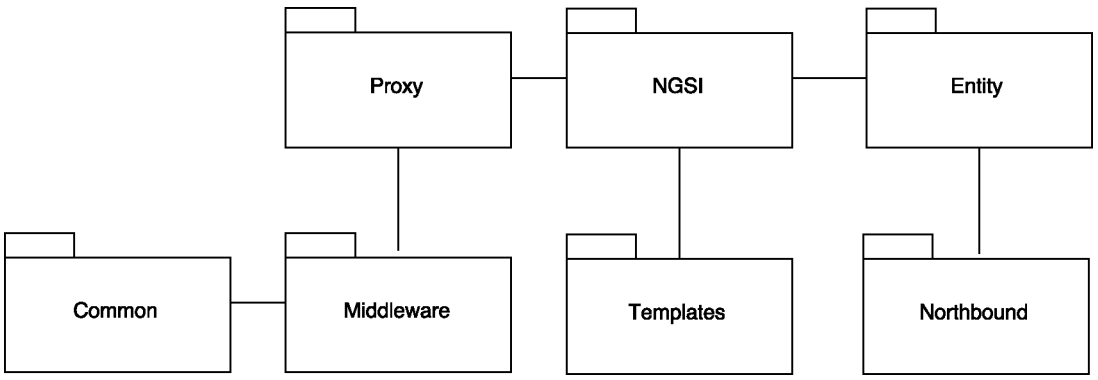

Figura 39 – Diagrama de Pacotes do IoT Broker-FI.

Capítulo 4

Experimentos e Análise dos Resultados

No Capítulo 3 foi detalhada a proposta e as intervenções realizadas nos componentes. O objetivo do presente capítulo é descrever os experimentos realizados para a validação do desenvolvimento. Primeiramente, será descrito o método utilizado para validar a proposta apresentada na seção 4.1. Na seção 4.2 serão descritos os experimentos realizados para o protocolo MQTT e LWM2M/CoAP, demostrando a execução em todos os componentes utilizados. Por fim, a seção 4.3 apresenta uma breve avaliação sobre os resultados obtidos nos experimentos, mostrando o custo-benefício, capacidades e as limitações deste trabalho.

4.1 Método para a Avaliação

Com o objetivo de validar a arquitetura proposta para a comunicação segura entre o dispositivo, o IoT Agent, o IoT Broker-FI e o Orion Context Broker, foram montados dois cenários de testes. O primeiro envolve a comunicação sem segurança e o segundo considera o uso de criptografia para as trocas de mensagens e ainda a autenticação e autorização, conforme proposto na arquitetura apresentada na Figura 26 da seção 3.1.

Tanto para o protocolo MQTT e LWM2M/CoAP inicialmente foi realizada a inspeção dos pacotes a fim de validar a sequência de mensagens necessárias para a comunicação entre os componentes envolvidos. Além disso, a inspeção permitiu observar o uso da criptografia ou não da criptografia em cada cenário.

A seguir foi realizada a medida de latência em cada cenário. A latência é o tempo de comunicação entre um dispositivo e o Orion Context Broker. Nesse caminho a mensagem sai, ou chega do dispositivo, passando através do IoT Agent e o IoT Broker.

Para o protocolo MQTT a medida de latência foi realizada para atributos *active* e *command*. No caso do *command* a avaliação levou em conta a influência do tempo do *handshake*. Para isso, a primeira abordagem considerou a realização do *handshake* no momento da conexão do dispositivo com o MQTT Broker. Já na segunda abordagem, a cada envio de mensagem pelo dispositivo um *handshake* é realizado. Para o protocolo

LWM2M/CoAP a medida de latência foi realizada levando em conta atributos *active, lazy* e *command.*

Por fim, para validar o suporte do DTLS 1.2 pelo IoT Agent LWM2M/CoAP foi utilizado um servidor compatível com o DTLS 1.2. O servidor utilizado para essa validação foi o LESHAN, que é escrito em Java e suporta o uso do DTLS 1.2 no protocolo LWM2M/CoAP.

4.2 Experimentos

Os experimentos foram executados em uma rede local utilizando um computador com Intel Core I7 com 4 núcleos de 2.00GHz com 8 GB de memória RAM e o sistema operacional Ubuntu. A ferramenta para analisar os pacotes e colher os dados de tempos foi o Wireshark. É importante destacar que para cada protocolo foi utilizado o seu respectivo IoT Agent. Os componentes IoT Agent e IoT Broker-FI são desenvolvido em NodeJs e foram executados utilizando o *Narwhals Poke Mammals* (NPM). A autenticação e autorização utilizou o Keystone do OpenStack por meio da porta 5000. A Tabela 4 mostra a versão dos sotwares utilizados neste experimento. A comunicação segura faz o uso de certificados. É importante observar que a maneira como este certificado é criado ou gerido foi abstraído, também são desconsideradas tempo de expiração do certificado, uma vez que não é o foco deste trabalho.

Tabela 4 – Versão dos *softwares* utilizados.

Software	Versão
Sistema Operacional Ubuntu	14.04 LTS
NodeJS	4.6.1
NPM	2.15.9
Keystone	9.1
Wireshark	2.0.5

4.2.1 IoT Agent-UL MQTT

No IoT Agent é possível armazenar as informações de grupos de dispositivos que possuem um *tipo*, *trust* e a *apikey*. Quando o dispositivo não possui seu cadastro e essa *apikey* existe no IoT Agent-UL MQTT, o mesmo passa a possuir as características deste grupo de dispositivos. Desta forma, o dispositivo é armazenado no IoT Agent e o *Register Context* é enviado por meio de uma requisição NGSI para armazenar a entidade no Orion Context Broker. A Figura 40 ilustra o diagrama de sequência em que o dispositivo realiza uma publicação no MQTT Broker e o IoT Agent faz um *subscribe*. Após realizar o registro de dispositivo no IoT Agent, é realizado o registro de contexto no Orion Context Broker. Depois que o *RegisterContext* é concluído, o dispostivo está apto para realizar a

atualização de dados (*Update Context*) no Orion Context Broker. Esse modo de operação faz parte do cenário *active* onde o dispositivo de forma ativa realiza atualizações periódicas de suas propriedades.

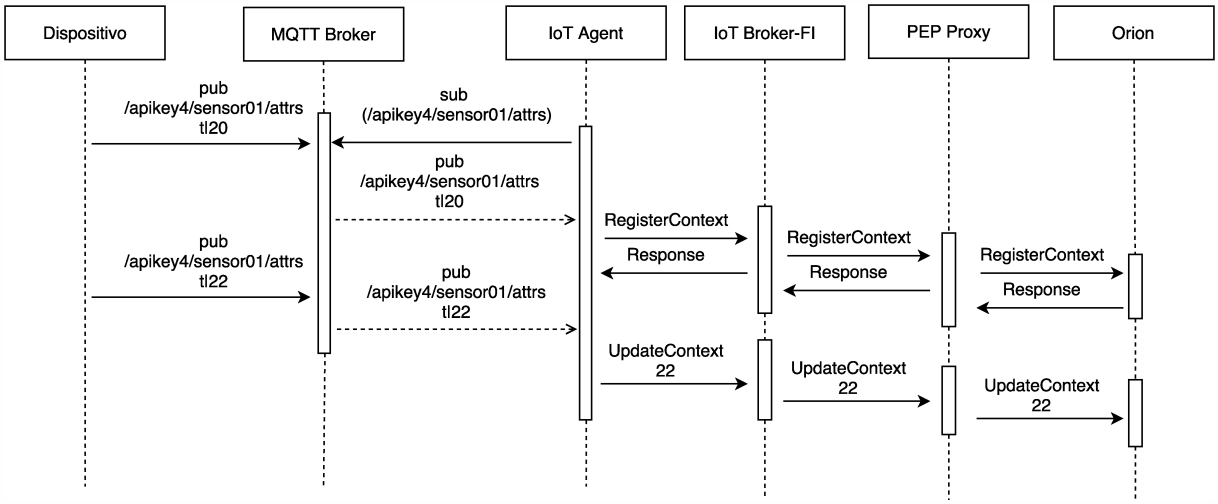

Figura 40 – Diagrama de sequência de um registro e publicações de mensagens MQTT.

A Figura 41 mostra o protocolo MQTT realizando um *publisher* do MQTT Broker até a porta que faz o *subscribe* do caminho especificado ilustrado nas linhas (1) e (2) com o protocolo MQTT e TCP, respectivamente. Note que a porta 5000 é utilizada para realizar a autenticação e autorização neste experimento e tem seu inicio na linha (3). No IoT Agent é armazenado o dispositivo de acordo com a *apikey* e é feito uma requisição na porta 4021 do IoT Broker-FI, indicada na linha (4).

Protocol	Info
MQTT (Linha 1)	Publish Message
TCP (Linha 2)	42680 → 1883 [ACK] Seq=147 Ack=62 Win=43776 Len=0 TSval=5266045 TSecr=5266045
TCP (Linha 3)	34842 → 5000 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=5266062 TSecr=0 WS=
TCP	5000 → 34842 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=5266062
TCP	34842 → 5000 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=5266062 TSecr=5266062
IPA	unknown 0x53
TCP	5000 → 34842 [ACK] Seq=1 Ack=354 Win=44800 Len=0 TSval=5266062 TSecr=5266062
IPA	unknown 0x54
TCP	34842 → 5000 [ACK] Seq=354 Ack=4097 Win=174720 Len=0 TSval=5266089 TSecr=5266089
IPA	unknown 0x33
TCP	34842 → 5000 [ACK] Seq=354 Ack=12289 Win=305664 Len=0 TSval=5266089 TSecr=5266089
IPA	unknown 0x2c
TCP	34842 → 5000 [ACK] Seq=354 Ack=16385 Win=436608 Len=0 TSval=5266089 TSecr=5266089
IPA	unknown 0x74
TCP	34842 → 5000 [ACK] Seq=354 Ack=19006 Win=465536 Len=0 TSval=5266089 TSecr=5266089
TCP	5000 → 34842 [FIN, ACK] Seq=19006 Ack=354 Win=44800 Len=0 TSval=5266089 TSecr=5266089
TCP	34842 → 5000 [FIN, ACK] Seq=354 Ack=19007 Win=465536 Len=0 TSval=5266090 TSecr=5266089
TCP	5000 → 34842 [ACK] Seq=19007 Ack=355 Win=44800 Len=0 TSval=5266090 TSecr=5266090
TCP	55803 → 4021 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=5266092 TSecr=0 WS=
TCP	4021 → 55803 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=5266092
TCP	55803 → 4021 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=5266092 TSecr=5266092
HTTP (Linha 4)	POST /NGSI9/registerContext HTTP/1.1 (application/json)
TCP	4021 → 55803 [ACK] Seq=1 Ack=666 Win=45056 Len=0 TSval=5266092 TSecr=5266092

Figura 41 – Envio de dado MQTT e o registro de uma entidade.

A Figura 42 mostra na linha (1) o registro da requisição no Orion Context Broker na porta 10026 e o retorno da resposta para o PEP Proxy que redirecionará ao IoT Agent mostrado na linha (2) e (3).

Protocol	Info
TCP	54385 → 10026 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=7904677 TSecr=7904677
HTTP (Linha 1)	POST /NGSI9/registerContext HTTP/1.1 (application/json)
TCP	10026 → 54385 [ACK] Seq=1 Ack=693 Win=45184 Len=0 TSval=7904677 TSecr=7904677
TCP	[TCP segment of a reassembled PDU]
TCP	54385 → 10026 [ACK] Seq=693 Ack=185 Win=44800 Len=0 TSval=7904677 TSecr=7904677
HTTP (Linha 2)	HTTP/1.1 200 OK (application/json)
TCP	54385 → 10026 [ACK] Seq=693 Ack=259 Win=44800 Len=0 TSval=7904677 TSecr=7904677
TCP	10026 → 54385 [FIN, ACK] Seq=259 Ack=693 Win=45184 Len=0 TSval=7904677 TSecr=7904677
TCP	[TCP segment of a reassembled PDU]
TCP	35047 → 1026 [ACK] Seq=666 Ack=208 Win=44800 Len=0 TSval=7904677 TSecr=7904677
HTTP (Linha 3)	HTTP/1.1 200 OK (application/json)
TCP	35047 → 1026 [ACK] Seq=666 Ack=282 Win=44800 Len=0 TSval=7904677 TSecr=7904677

Figura 42 – Registro no Orion Context Broker e a resposta da requisição.

Para ativar os recursos de segurança implementados é importante definir as configurações de portas no MQTT Broker, IoT Agent, IoT Broker-FI e no PEP Proxy. Desta forma, o IoT Broker irá responder pela porta 4022, o IoT Agent responde requisições do contexto por meio da porta 4062 e o MQTT Broker utiliza a 8883. A Figura 43 ilustra o pacote criptografado, diferente do apresentado na Figura 23 em que pode ser visualizado o conteúdo. A Figura 44 mostra uma publicação de mensagem MQTT e a requisição NGSI para registro de contexto. Pode-se comparar com a Figura 41 em que é realizado o mesmo experimento, entretanto, este possui recursos de segurança nas requisições MQTT e NGSI. A mensagem MQTT utiliza a porta 8883, o IoT Agent faz uma requisição NGSI pelo *socket* (45213) para IoT Broker-FI na porta 4022 indicado na linha (1).

```
▼ Data (57 bytes)
    Data: 17030300347700ff35ea04899c4585840f808b130719c033...
    [Length: 57]

0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ........ ........
0010  45 00 00 6d 6b 3b 40 00  40 06 d1 4d 7f 00 00 01   E..mk;@. @..M....
0020  7f 00 00 01 e4 bc 22 b3  5e 56 4f a4 28 8e 68 66   ......". ^VO.(.hf
0030  80 18 09 54 fe 61 00 00  01 01 08 0a 00 82 2c 67   ...T.a.. ......,g
0040  00 82 2c 66 17 03 03 00  34 77 00 ff 35 ea 04 89   ..,f.... 4w..5...
0050  9c 45 85 84 0f 80 8b 13  07 19 c0 33 3e 3d 32 29   .E...... ...3>=2)
0060  01 36 17 3e 20 32 3a a3  47 d8 6f c0 0e f5 d0 11   .6.> 2:. G.o.....
0070  a8 a3 a3 aa aa ce 53 73  ba 60 4a fe d3            ......Ss .`J..
```

Figura 43 – Pacote de dados MQTT cifrado.

Outro modo de operação possível para o dispositivo é o chamado *command*. Nesse caso o dispositivo pode receber comandos que por ele serão executados. A Figura 45 mostra o diagrama de sequência para este cenário. Em (1) mostra a ocorrência de suas

Protocol	Info
TCP	8883 → 33562 [PSH, ACK] Seq=4992 Ack=2289 Win=175744 Len=76 TSval=6433008 TSecr=6433008
TCP	33562 → 8883 [ACK] Seq=2289 Ack=5068 Win=305664 Len=0 TSval=6433008 TSecr=6433008
TCP	35739 → 5000 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=6433020 TSecr=0 WS=12
TCP	5000 → 35739 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=6433020 TS
TCP	35739 → 5000 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=6433020 TSecr=6433020
IPA	unknown 0x53
TCP	5000 → 35739 [ACK] Seq=1 Ack=354 Win=44800 Len=0 TSval=6433020 TSecr=6433020
IPA	unknown 0x54 unknown 0x38
TCP	35739 → 5000 [ACK] Seq=354 Ack=19006 Win=174720 Len=0 TSval=6433056 TSecr=6433056
TCP	5000 → 35739 [FIN, ACK] Seq=19006 Ack=354 Win=44800 Len=0 TSval=6433056 TSecr=6433056
TCP	35739 → 5000 [FIN, ACK] Seq=354 Ack=19007 Win=174720 Len=0 TSval=6433057 TSecr=6433056
TCP	5000 → 35739 [ACK] Seq=19007 Ack=355 Win=44800 Len=0 TSval=6433057 TSecr=6433057
TCP (Linha 1)	45213 → 4022 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=6433061 TSecr=0 WS=12
TCP	4022 → 45213 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=6433061 TS
TCP	45213 → 4022 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=6433061 TSecr=6433061
TCP	45213 → 4022 [PSH, ACK] Seq=1 Ack=1 Win=43776 Len=262 TSval=6433061 TSecr=6433061
TCP	4022 → 45213 [ACK] Seq=1 Ack=263 Win=44800 Len=0 TSval=6433061 TSecr=6433061
TCP	4022 → 45213 [PSH, ACK] Seq=1 Ack=263 Win=44800 Len=1355 TSval=6433063 TSecr=6433061
TCP	45213 → 4022 [ACK] Seq=263 Ack=1356 Win=174720 Len=0 TSval=6433063 TSecr=6433063
TCP	45213 → 4022 [PSH, ACK] Seq=263 Ack=1356 Win=174720 Len=191 TSval=6433063 TSecr=6433063
TCP	4022 → 45213 [PSH, ACK] Seq=1356 Ack=454 Win=45952 Len=274 TSval=6433064 TSecr=6433063
TCP	45213 → 4022 [PSH, ACK] Seq=454 Ack=1630 Win=177408 Len=694 TSval=6433064 TSecr=6433064

Figura 44 – Requisição NGSI cifrado de um registro de uma entidade.

requisições *subscribe*. O *subscribe* que possui final *cmd* significa que o dispositivo irá receber informações do IoT Agent por meio de uma *publish*, sendo o comando necessário para ser executado. O IoT Agent faz um *subscribe* com final *cmdexe* para que o dispositivo envie uma mensagem de resposta do comando. Este comando pode ser interpretado como "comando executado". Em (2) o Orion Context Broker é acionado para executar um atributo *command* no IoT Agent. Em (3) o IoT Agent converte a requisição NGSI para uma requisição MQTT, e faz uma publicação no dispositivo. Em (4) o dispositivo recebe a publicação e executa o comando pré-programado, então sua resposta é enviada por meio da publicação de mensagem *cmdexe*. O IoT Agent converte a mensagem recebida em NGSI e envia para o Orion Context Broker. Por fim, a cifragem demonstrada no cenário *active*, apresentado na Figura 40, também se aplica ao cenário *command*, mostrado na Figura 45.

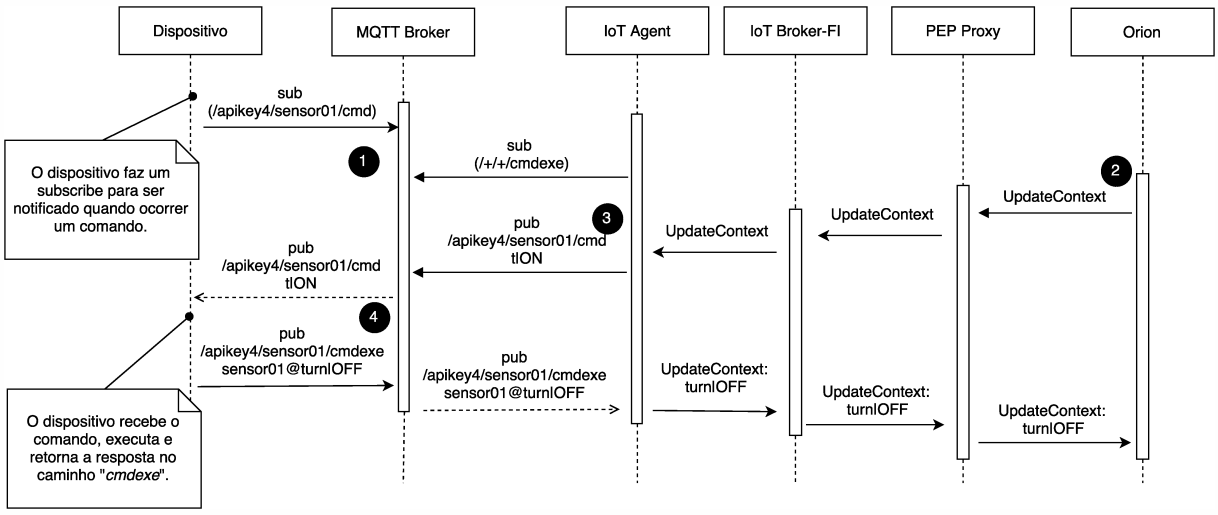

Figura 45 – Comando executado no dispositivo a partir de uma requisição NGSI.

4.2.2 LWM2M/CoAP

O protocolo OMA Lightweight M2M possui um conjunto de interfaces que são utilizadas pelo cliente e o servidor (IoT Agent). A Figura 5 ilustra estas operações de forma resumida. Quando um dispositivo conecta ao servidor, primeiro o cliente envia a requisição de registro. A Figura 46 mostra o diagrama de sequência para o registro. Vale notar que o IoT Agent faz o *RegisterContext* e depois uma requisição CoAP informando que deseja receber notificações de quando o dado é alterado através do método *observe*. Com isso, quando ocorrer alguma atualização do objeto no dispositivo, o dado é enviado para o contexto. De fato, isso representa o cenário de atributo *active* proposto pelo FIWARE. A Figura 47 mostra a requisição CoAP realizando o método POST e o pacote com as informações.

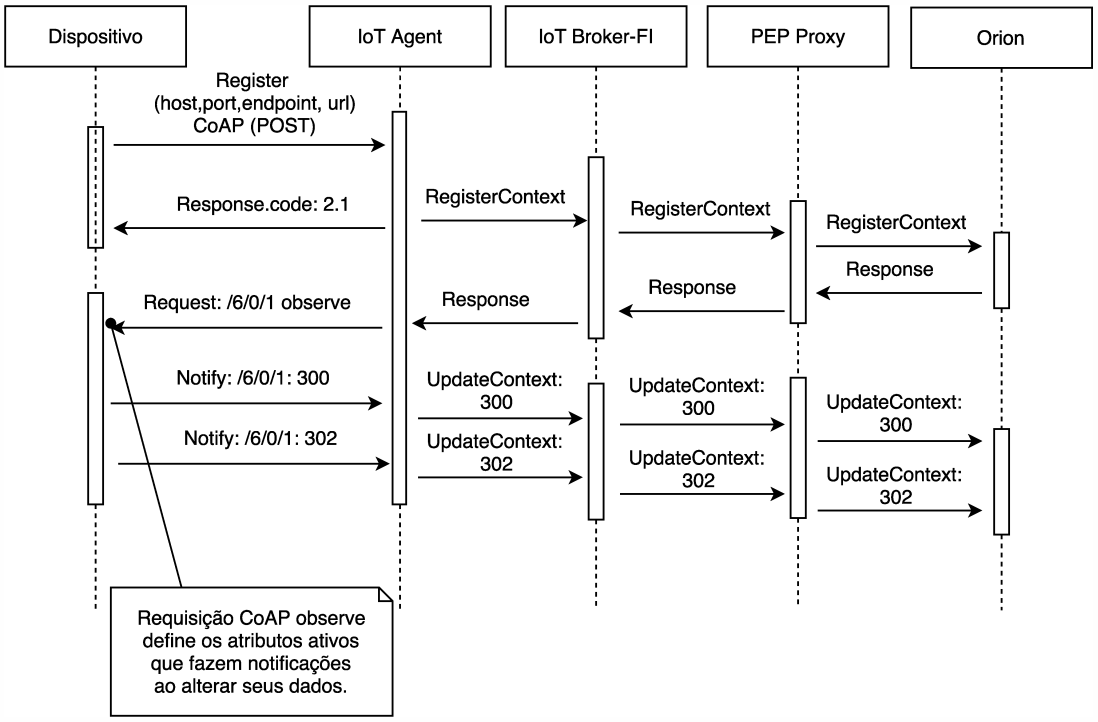

Figura 46 – Mensagem de registro e notificação de atributo enviada pelo protocolo CoAP.

Protocol	Length	Info
CoAP	111	CON, MID:54602, POST, TKN:bf 56 3c 14, /elemento/Room/rd?ep=sensor01&lt=85671&lwm2m=1.0&b=U
CoAP	57	ACK, MID:54602, 2.01 Created, TKN:bf 56 3c 14

Figura 47 – Registro de dispositivo no LWM2M/CoAP.

A implementação da camada DTLS no protocolo Lightweight/CoAP foi realizada satisfatoriamente, uma vez que não possuía o recurso no IoT Agent disponibilizado pela plataforma FIWARE. De fato, utilizando o DTLS 1.2 primeiro é feito o *handshake* e o envio do pacote de registro criptografado, como mostrado na Figura 30 na seção 3.2.

Este protocolo também possibilita executar o cenário de atributos *lazy* que são armazenados no dispositivo e pode ser lido pelo contexto Orion Context Broker por meio de uma *QueryContext*. A Figura 48 mostra o diagrama de sequência, em que o IoT Agent LWM2M envia as requisições CoAP READ para o dispositivo. Quando o Orion Context Broker é acionado envia a informação para o IoT Broker até o IoT Agent que processará as informações. Assim, o IoT Agent dispara uma grande quantidade de chamadas READ CoAP para retornar uma entidade para Orion Context Broker, como pode ser visto na Figura 49.

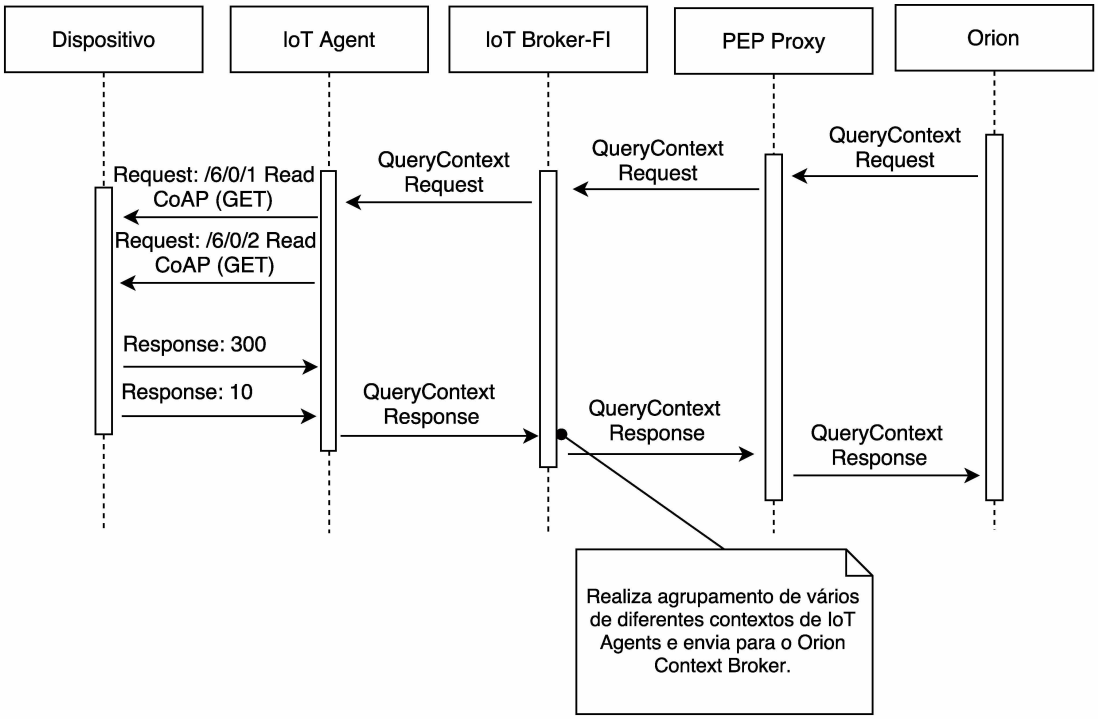

Figura 48 – Requisições READ no dispositivo para recuperar atributos *Lazy* no Contexto.

```
192.168.1.9     TCP          68 48084 → 4062 [ACK] Seq=505
192.168.1.9     TCP         892 48084 → 4062 [PSH, ACK] Seq
127.0.0.1       DTLSv1.2     87 Application Data
127.0.0.1       DTLSv1.2     87 Application Data
127.0.0.1       DTLSv1.2     87 Application Data
127.0.0.1       DTLSv1.2     87 Application Data
127.0.0.1       DTLSv1.2     87 Application Data
127.0.0.1       DTLSv1.2     87 Application Data
172.17.0.2      TCP          68 39804 → 27017 [ACK] Seq=364
```

Figura 49 – Requisições READ encapsulada com DTLS 1.2.

A Figura 50 ilustra o cenário em que um atributo *command* é executado no dispositivo por meio da requisição *UpdateContext* no Orion Context Broker. O IoT Agent recebe e faz uma requisição para o dispositivo com a função *execute* e método POST. O dispositivo

atualiza o valor do objeto e envia a resposta do *command* para o IoT Agent que retorna ao contexto. E a Figura 51 ilustra um pacote criptografado.

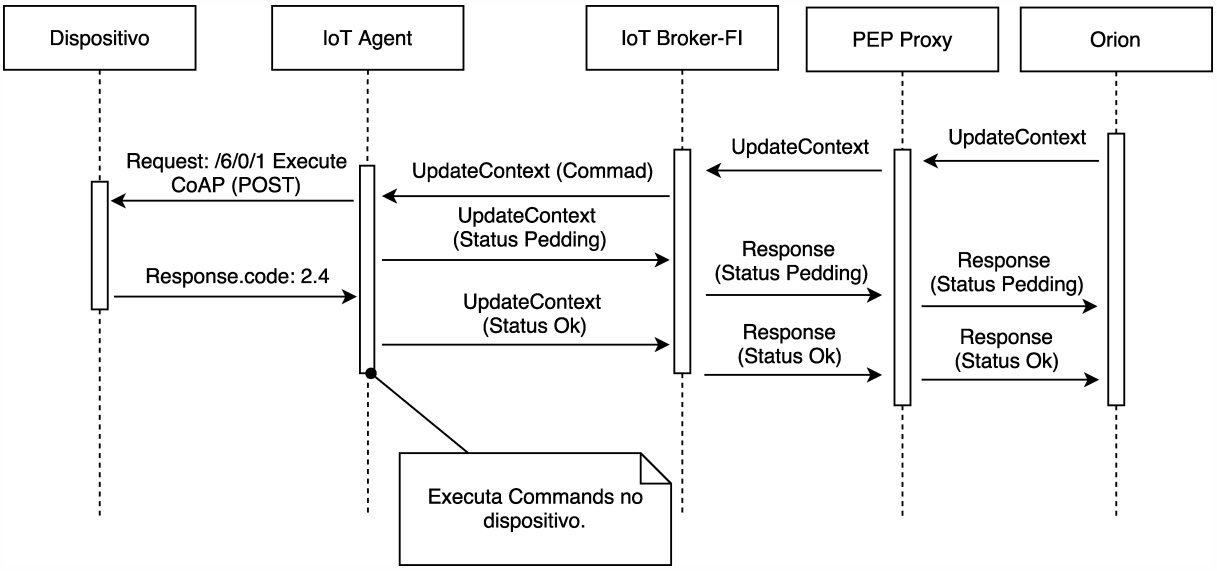

Figura 50 – Diagrama de sequência para o atributo *command* utilizando o LWM2M.

```
▸ User Datagram Protocol, Src Port: 45538 (45538), Dst Port: 5684 (5684)
▾ Datagram Transport Layer Security
  ▾ DTLSv1.2 Record Layer: Application Data Protocol: Application Data
      Content Type: Application Data (23)
      Version: DTLS 1.2 (0xfefd)
      Epoch: 1
      Sequence Number: 30
      Length: 24
      Encrypted Application Data: 000100000000001ea837d6acc6d6310bc97608be4dd58c7c
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ........ ........
0010  45 00 00 41 53 50 40 00  40 11 e9 59 7f 00 00 01   E..ASP@. @..Y....
0020  7f 00 00 01 b1 e2 16 34  00 2d fe 40 17 fe fd 00   .......4 .-.@....
0030  01 00 00 00 00 00 1e 00  18 00 01 00 00 00 00 00   ........ ........
0040  1e a8 37 d6 ac c6 d6 31  0b c9 76 08 be 4d d5 8c   ..7....1 ..v..M..
0050  7c                                                |
```

Figura 51 – Pacote CoAP criptografado utilizando o DTLS 1.2.

O contexto também pode atualizar algum valor de atributo no dispositivo utilizando a operação *UpdateContext*, com isso, o IoT Agent envia para o dispositivo uma requisição CoAP com a função *write* e o método PUT. Note que não é executado nenhuma operação de comando, apenas atualização de valor. A Figura 52 mostra o diagrama de sequência para este cenário.

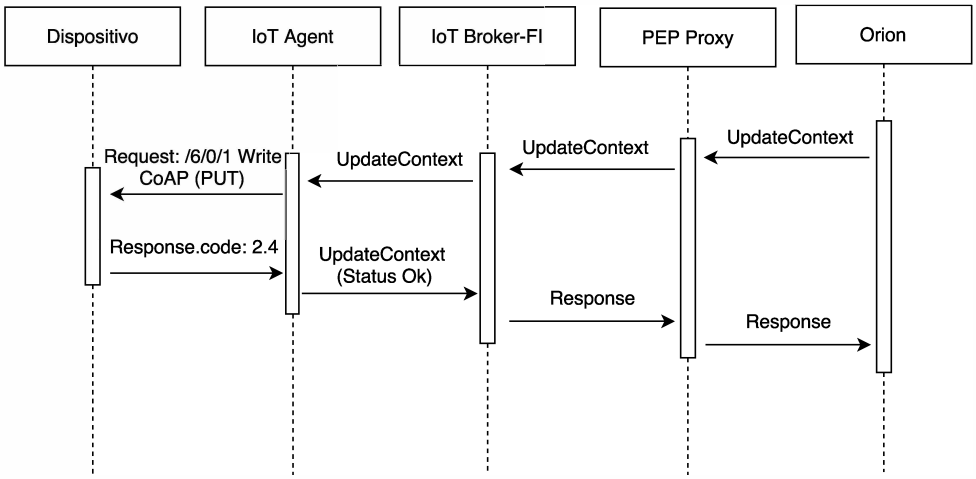

Figura 52 – Diagrama de sequência para o atributo *write* utilizando o LWM2M/CoAP.

Outro método de validação da implementação do suporte ao DTLS 1.2 pelo IoT Agent LWM2M/COAP proposto por este trabalho, foi permitir a comunicação do IoT Agent, desenvolvido em NodeJS com o LESHAN. O LESHAN é um cliente e servidor compatível com Lightweigth M2M2 escrito em Java. Deste modo, foi realizado um experimento utilizando o servidor LESHAN e o cliente desenvolvido em NodeJs. Para isto, foi necessário definir a chave PSK no cliente e no servidor. O servidor LESHAN possui uma interface web que pode ser utilizada para configuração de chaves e visualização de dispositivos e dados. A Figura 53 mostra a comunicação do cliente LWM2M/CoAP desenvolvido em NodeJs comunicando com o servidor LESHAN. O LESHAN realiza apenas um *handshake* para estabelecer o canal seguro, assim, foi possível validar o experimento de compatibilidade do DTLS 1.2.

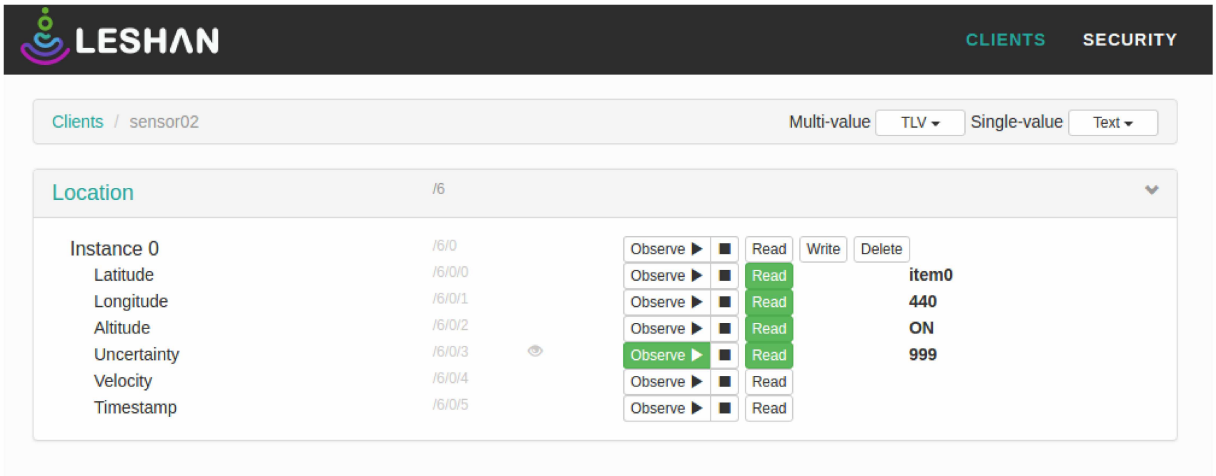

Figura 53 – Comunicação de um cliente em Node e o servidor LESHAN em Java.

4.3 Avaliação dos Resultados

A seção 4.2 mostra os resultados dos experimentos alcançados, os mecanismos de criptografia nos protocolos e as requisições NGSI com o TLS. Além do mais, as requisições de autenticação e autorização foram concretizadas satisfatoriamente nestes testes utilizando o Keystone.

Desse modo, essa seção apresenta uma avaliação comparativa da latência no cenário com e sem o uso de segurança, conforme o método apresentado na seção 4.1. As medições de tempos foi realizado pelo Wireshark e estão apresentados em segundos (s). O eixo Y dos gráficos são os tempos de latência e o eixo X. Foram executados 24 experimentos em rede local de cada cenário dos protocolos MQTT e LWM2M/CoAP.

Na aferição estatística dos dados amostrados, fora conduzido avaliação do intervalo de confiança para as médias dos tempos. Para isso, foi considerado um índice de confiança de 0.95 e os dados amostrados se comportaram na forma de distribuição normal. Desta forma, avaliar a potencialidade do critério de segurança devemos considerar o intervalo de confiança.

4.3.1 IoT Agent MQTT

Para o IoT Agent MQTT apenas dois cenários são suportados: atributos *active* ou *command*. Em ambos cenários *active* e *command* o *handshake* do protocolo MQTT é realizado anteriormente, e a conexão segura mantém aberta por um certo período de tempo. Para cada requisição NGSI é feito um *handshake* separadamente devido um questão de arquitetura do FIWARE.

Na Figura 54 é apresentado o gráfico de latência para os experimentos de atributos ativos. Vale lembrar que estes dados estão em segundos. A média dos tempos colhidos em segundos e o intervalo de confiança com segurança é $0,07354 \pm 0,00851$ e sem segurança é $0,048199 \pm 0,003693$. Para este caso, estatisticamente é razoável concluir que o tempo de latência com segurança possui um acréscimo de 52,58% na média, uma vez que são realizados o *handshake* do NGSI.

Já na Figura 55 mostra o tempo para o cenário em que os atributos *command* são executados no dispositivo. Neste caso, a média de tempo com segurança é $0,08815 \pm 0,01311$ e sem TLS $0,06119 \pm 0,00621$ em segundos. Se compararmos a execução com/sem segurança temos um aumento de 44% com o uso de cifragem e controle de acesso. Note que o tempo é considerado maior se compararmos os atributos *active*, já que são realizadas mais chamadas MQTT e também requisições NGSI para informar os *status* dos atributos sendo um acréscimo na média de 27% em teste sem segurança.

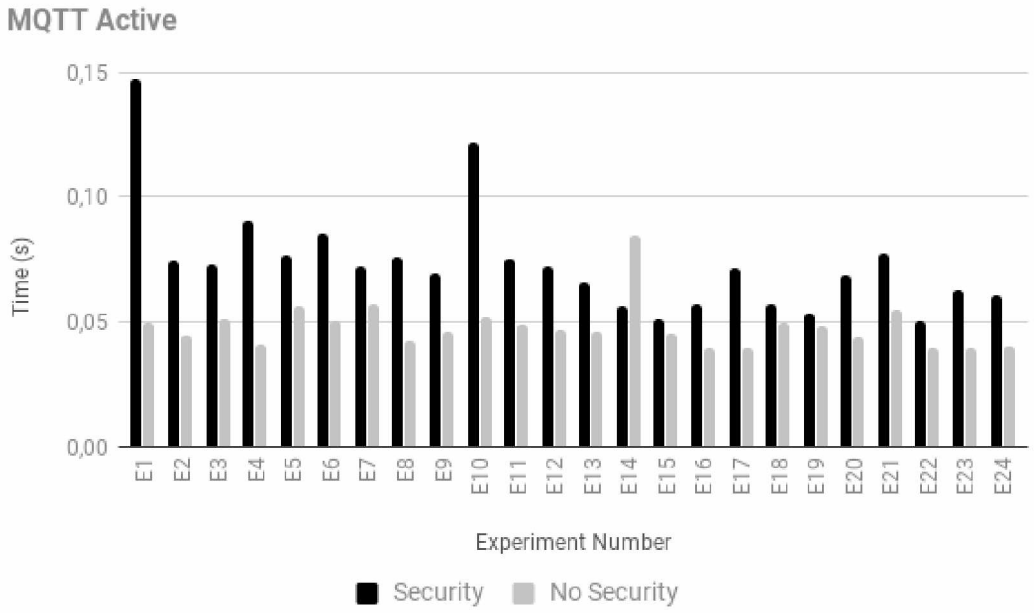

Figura 54 – Avaliação comparativa da latência para os atributos *active* do protocolo MQTT.

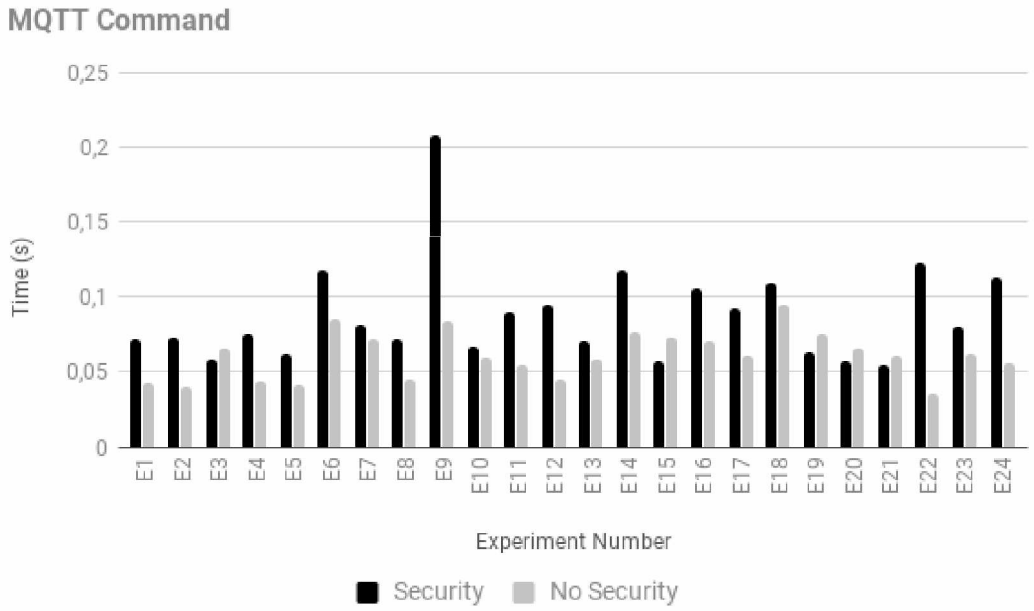

Figura 55 – Avaliação comparativa da latência para os atributos *Command* do protocolo MQTT.

4.3.2 IoT Agent LWM2M/CoAP

Foi realizada a avaliação do IoT Agent LWM2M/CoAP nos cenários *active*, *lazy* e *command*. Vale lembrar que todas as mensagens no protocolo CoAP possuem confirmação de recebimento. Desse modo, é possível gerar em qualquer cenário retransmissão de pacotes

se não houver a confirmação de recebimento, principalmente na versão com segurança que utiliza um *socket*. Diferente da versão sem segurança em que deve considerar o custo de abrir o *socket* sempre quando houver um envio de mensagem. E na versão com segurança é estabelecido o *handshake* entre o dispositivo e o IoT Agent, e utiliza o mesmo *socket* para enviar as mensagens.

A Figura 56 mostra o gráfico com os tempos para o cenário *active* em que explora os atributos que sofrem modificações e enviam atualizações para o IoT Agent. O tempo da latência pode-se ser considerado maior no cenário com DTLS em 31% se compararmos com a versão sem segurança. A média e o intervalo de confiança com segurança no MQTT é $0,07354 \pm 0,00851$ e no LWM2M/CoAP é $0,02804 \pm 0,00364$. Já para o experimento sem DTLS a média e o intervalo de confiança é considerado menor em 0.02130 ± 0.00447. Pode-se observar que neste cenário o protocolo CoAP possui tempos menores que o MQTT, mas cabe observar que foi realizado o teste com um atributo de contexto. Note que neste protocolo para cada mensagem é enviado um único atributo. No modelo MQTT é possível atualizar vários atributos utilizando uma única mensagem, com isso, a quantidade de atributos pode fazer diferença em um determinado projeto. Se aumentarmos o número de atributos é possível gerar re-envios de pacotes por causa do tráfego entre o dispositivo e o IoT Agent. Além disso, devemos observar que o MQTT utiliza a camada de transporte TCP, enquanto o CoAP utiliza o UDP.

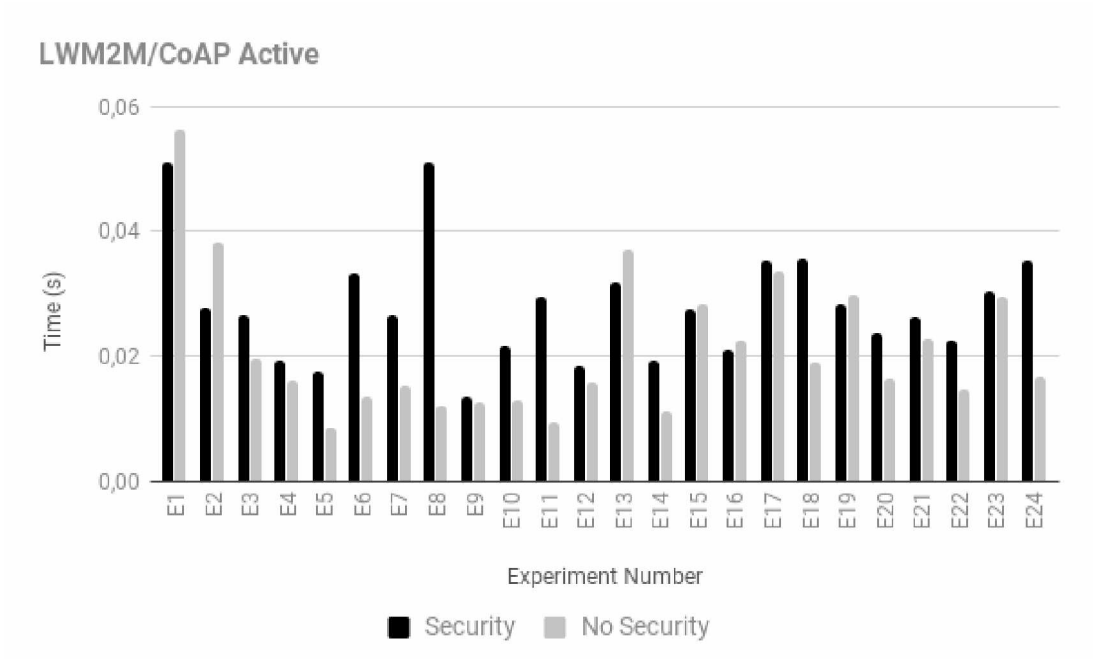

Figura 56 – Avaliação comparativa da latência para os atributos *active* do protocolo LWM2M/CoAP.

A Figura 57 mostra o tempo gasto para os atributos *lazy* em que são disparadas mensagens por meio de uma requisição NGSI (*QueryContext*). Com isso, o IoT Agent faz para cada atributo requisições CoAP utilizando o método GET para obter informação do atributo atual. Neste caso não podemos comparar com o MQTT, já que, o paradigma do protocolo não permite este cenário. A média de tempo utilizando o DTLS é 0.06006 ± 0.00551 em segundos e sem segurança é 0.05107 ± 0.00263 em segundos. Vale observar que o tempos com DTLS possui um pequeno acréscimo médio de 17% se comparar com a versão sem segurança.

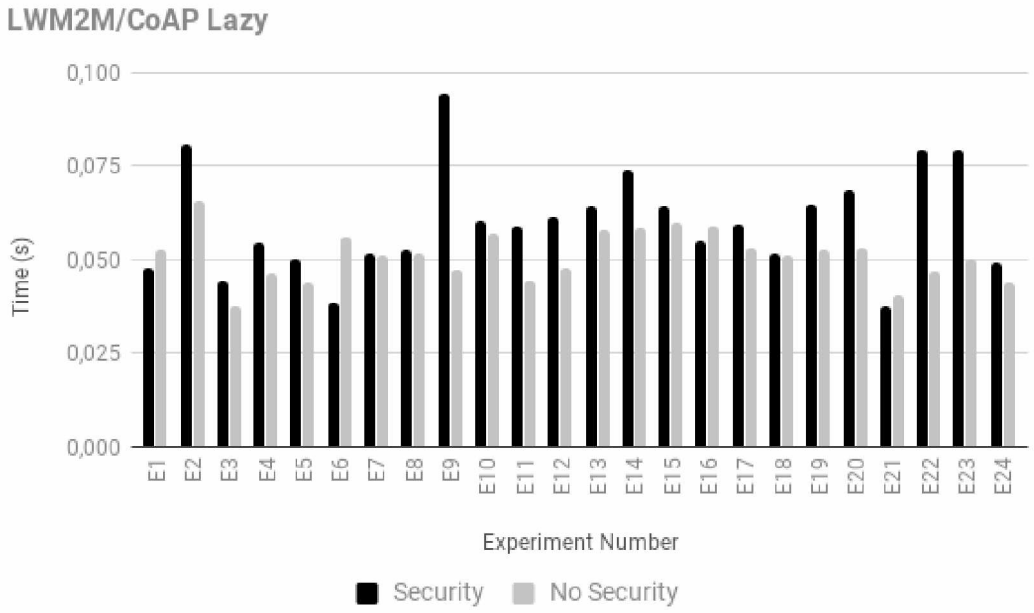

Figura 57 – Avaliação comparativa da latência para os atributos *Lazy* do protocolo LWM2M/CoAP.

O LWM2M/CoAP também possibilita realizar comandos (*command*) em determinado objeto no dispositivo. A Figura 58 ilustra o gráfico de tempos colhidos. No experimento sem segurança possui uma média e o intervalo de confiança de $0,02845 \pm 0,00315$ e com DTLS a média é $0,034444 \pm 0,00221$ em segundos. O tempo com segurança é maior em 21% em média quando se compara o teste sem segurança, isso ocorre por causa que no cenário *command* em que são realizadas mais requisições NGSI utilizando o HTTPS.

Observe que no MQTT sem e com segurança possuem respectivamente uma média e intervalo de confiança $0,08815 \pm 0,01311$ e sem TLS $0,06119 \pm 0,00621$ de tempos em segundos. Note que os tempos do LWM2M/CoAP são menores quando se compara com o MQTT, entretanto, se aumentarmos o números de atributos é possível aumentar o tempo para realizar este cenário no dispositivo. Pode-se observar que no cenário com DTLS é possível gerar retransmissão e, com isso, o IoT Agent pode esperar a resposta do dispositivo para enviar chamadas NGSI.

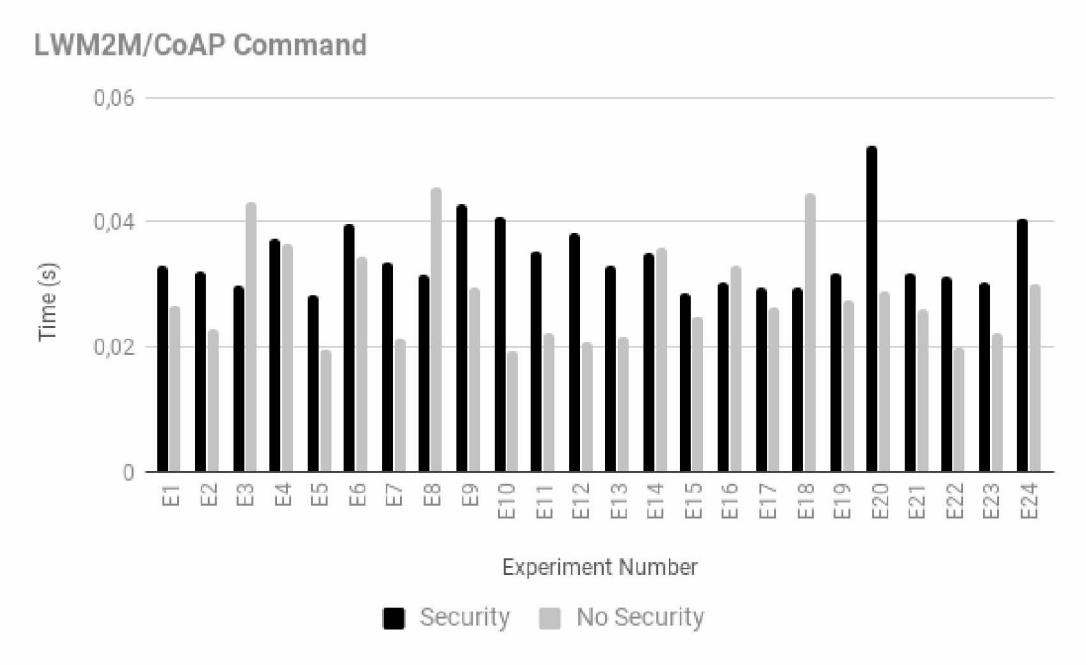

Figura 58 – Avaliação comparativa da latência para os atributos *Command* do protocolo LWM2M/CoAP.

A camada de segurança em toda arquitetura proposta possui vantagens e desvantagens. O principal benefício é garantir que as trocas de informações entre os elementos sejam cifrados e não possam ser acessadas por terceiros. Entretanto, cabe observar que a camada de segurança pode ser um desafio quando se trata em ambiente real. Se for considerado uma grande quantidade de dispositivos, será necessário um mecanismo para administrar os certificados para garantir a consistência entre os diversos componentes da proposta. Além disso, deve-se levar em conta a capacidade dos dispositivos, já que apresentam recursos de processamento próximo a sistemas embarcados e possuem baixo consumo de energia. Outro aspecto é a comunicação que muitas vezes podem apresentar falhas de rede. Isso pode ser um entrave, visto que antes de enviar o dado é realizado a negociação do algoritmo de criptografia para estabelecer o canal seguro. É importante ressaltar que o tratamento dessas questões não foi objetivo deste trabalho.

Capítulo 5

Conclusão

Este capítulo finaliza este trabalho, detalhando as principais conclusões obtidas ao propor a autenticação, autorização e cifragem no IoT Agent, e o desenvolvimento da aplicação IoT Broker-FI.

O Capítulo 2 faz uma revisão bibliográfica com as informações sobre IoT. São detalhados o histórico, os elementos que compõe os dispositivos para que seja feita a tradução do mundo real para o virtual. Além do mais, apresenta a arquitetura utilizada em IoT, os protocolos e os mecanismos de segurança utilizados neste trabalho. Deste modo, são detalhados em especial os protocolos MQTT, CoAP, Lightweight M2M/CoAP. As plataformas de *middleware* são apresentados os aspectos conceituais, os serviços providos a comparação com outras arquiteturas. Por fim, a plataforma FIWARE é detalhada com seus principais componentes que são utilizados neste trabalho.

Em geral, foi possível observar algumas lacunas no FIWARE, como: cifragem nos protocolos MQTT, Lightweight M2M/CoAP e nas requisições NGSI. De fato, esta é a meta mais relevante para o trabalho já que ampliar os recursos de segurança fim-a-fim na plataforma FIWARE é fundamental, uma vez, que este requisito desempenha um papel chave em uma aplicação de ambiente real. Já que os dados podem ser violados causando sérios danos para aplicações consumidores e dispositivos.

No Capítulo 3 são detalhados as necessidades e o desenvolvimento da confidencialidade, controle de acesso, e a aplicação IoT Broker-FI para agrupar e manipular informações de diferentes IoT Agents de forma segura. Deste modo, os objetivos introduzidos na seção 1.2 foram detalhados e demonstrados com mudanças ocorridas principalmente no IoT Agent, protocolos LWM2M/CoAP. O Capítulo 4 apresentou as demonstrações em busca a atingir as metas estabelecidas. Com isso, foi demonstrado por meio de análise de pacotes no Wireshark como prova de conceito das implementações realizadas neste trabalho. Também, ressaltou que a comunicação cifrada é importante, entretanto, existem ainda vários desafios como gerenciar os certificados dos componentes em um ambiente real, dispositivos embarcados com pouco recurso e que falhas na rede podem dificultar estabelecer o canal seguro.

Portanto, os objetivos específicos foram alcançados e os experimentos realizadas demonstram que o objetivo geral foi concluído satisfatoriamente. As seções 5.1 e 5.2 buscam destacar as principais contribuições deste trabalho e apresentar os trabalhos futuros para melhorar o cenário atual.

5.1 Principais Contribuições

A segurança dos pacotes trafegados foi uma premissa que deveria estar em todas camadas utilizadas pelo projeto. De fato, a principal contribuição foi implementar em NodeJs o DTLS 1.2 no LWM2M/CoAP, uma vez que foi necessário realizar modificações no protocolo CoAP para permitir a comunicação segura entre o cliente e servidor. Vale lembrar que na versão disponibilizada pelo FIWARE, a comunicação segura não estava implementada. Outra contribuição, foi garantir que esta versão do LWM2M/CoAP em NodeJs seja validada no LESHAN que é uma versão em Java disponibilizada pelo Eclipse.

Além disso, podem ser observadas a implementação de requisitos de segurança no IoT Agent MQTT permitindo que o dispositivo envie mensagens cifradas. Ademais, foi implementado a autenticação e autorização nos IoT Agents para que as requisições NGSI tenham permissão de acesso a determinado recurso no Orion Context Broker.

Foi desenvolvido um IoT Broker-FI que visa ser um ponto de comunicação entre os IoT Agents e o Orion Context Broker. Esta aplicação realiza o mapeamento de entidades com informações e permite o desenvolvimento de *plugins* que podem ser utilizados para aplicações externas. Outra função é permitir que o contexto de determinado tipo possa ser manipulado por diferentes IoT Agents com protocolos diferentes. Havia também uma lacuna de segurança entre os IoT Agent e o IoT Broker-FI, uma vez que as requisições NGSI não eram cifradas. Para isso, foi implementado servidores HTTPS e requisições com certificados.

Por fim, os mecanismos desenvolvidos foram disponibilizados para todos os pesquisadores envolvidos com FIWARE no GitHub (THOMAS, 2017) e toda comunidade científica e desenvolvedores. Desta forma, o trabalho permite que outros desenvolvedores ampliem e aprimorem a segurança em toda arquitetura FIWARE.

5.2 Trabalhos Futuros

Este trabalho permitiu ampliar a segurança para aplicações de IoT baseadas na plataforma FIWARE. Entretanto há um conjunto de áreas onde existe a perspectiva de trabalhos futuros. Essas áreas são descritas a seguir:

- **Suporte de protocolos de comunicação:**
  Do ponto de vista da infraestrutura, os dispositivos possuem tecnologia heterogê-

nea que incluem conexão entre diferentes tipos de rede e protocolos. O papel do *middleware* é possibilitar maior transparência na comunicação, na media que novos protocolos vão surgindo, cabe ao FIWARE realizar o suporte desta tecnologia em seus IoT Agents. O software foi desenvolvido seguindo o mesmo padrão adotado que possibilitam que novos protocolos sejam inseridos. Assim, como continuação deste projeto pode-se ampliar o IoT Agent com suporte à segurança para suportar outros tipos de protocolos, como: Extensible Messaging and Presence Protocol (XMPP) (SAINT-ANDRE, 2011) e Advanced Message Queuing Protocol (AMQP) (KARAGIANNIS et al., 2015).

- **Compatibilidade e Padronização:**
  O rápido crescimento da IoT dificultou a padronização, assim, muitos dispositivos e *middlewares* utilizam sua própria tecnologia juntamente com seu protocolo oferecendo serviços que não podem ser acessíveis à outros componentes. Desta forma, é importante que a compatibilidade e a padronização dos elementos da IoT para prover a interoperabilidade para todos os objetos, sendo um importante recurso que possibilita o desenvolvimento de novas aplicações. O FIWARE desenvolveu a padronização NGSI, entretanto, houve mudanças de versões fazendo com que os componentes não tivessem compatibilidade na comunicação entre os diferentes GEs FIWARE. Para isso, é necessário um esforço para desenvolver uma semântica mais genérica que possa ser compatível para as aplicações antigas. Além do mais, diferentes *middlewares* têm adotado diferentes padrões, uma solução poderia coletar diferentes padrões e desenvolver um modelo completo podendo ser adaptável para cada domínio.

- **Segurança e Privacidade:**
  A proteção de dados devem ser consideradas no contexto IoT, uma vez, que são questões difíceis por causa da sua implementação, mobilidade e complexidade. Com o modelo de *middlewares* orientados à computação em nuvem é possível que as informações dos dispositivos possam ser acessadas, coletadas, arquivadas e compartilhadas com facilidade. De fato, uma arquitetura IoT possui vários componentes, e estes devem ter mecanismos que garantem a criptografia para a integridade dos dados, além de serviços que determinam a autorização e o nível de acesso da informação de agentes externos. Um exemplo, é gerenciar e configurar os certificados TLS entre os diferentes componentes IoT Agents e gerenciar o acesso aos dispositivos.

- **Desenvolver IoT Agents mais autônomos:**
  Embora as arquitetura de *middleware* possuem características autônomas, ainda sim há a necessidade de intervenção humana para realizar as configurações das aplicações. Além disso, a autonomia possibilita que seja desenvolvida regras de inferência que podem evoluir e mudar automaticamente de acordo com o ambiente.

Pode-se considerar as técnicas potenciais relacionadas a dados, como linguagem de aprendizagem e algoritmos de mineração. Cada componente do *middleware* deve ser tolerante à falhas, assim, o sistema deve recuperar de forma resiliente para garantir a disponibilidade de suas funções. Considerando que a IoT utiliza uma infraestrutura de *cloud computing* é natural que o *middleware* ofereça níveis de flexibilidade dos componentes e tenha escalabilidade para aumentar e diminuir a capacidade de processamento de forma automática.

5.3 Submissão de Artigos Científicos

O artigo *"Improving Security on IoT Applications based on the FIWARE Platform"* foi submetido à 33rd ACM/SIGAPP Symposium On Applied Computing (ACM SAC) na trilha de IoT.

Referências

AL-FUQAHA, A. et al. Internet of things: A survey on enabling technologies, protocols, and applications. **IEEE Communications Surveys & Tutorials**.

ALLIANCE, O. M. **Lightweight Machine to Machine Technical Specification**. 2017. Disponível em: <http://openmobilealliance.org/release/LightweightM2M/V1_0-20151214-C/OMA-TS-LightweightM2M-V1_0-20151214-C.pdf>.

AMAZON. **AWS IoT**. 2017. Disponível em: <https://aws.amazon.com/pt/iot-platform/how-it-works/>.

_____. **Security and Identity for AWS IoT**. 2017. Disponível em: <http://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html>.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. **Computer networks**, Elsevier, v. 54, n. 15, p. 2787–2805, 2010.

BASSI, A. et al. Enabling things to talk. **Designing IoT Solutions With the IoT Architectural Reference Model**, Springer, p. 163–211, 2013.

CATALOGUE, F. **Backend Device Management - IDAS**. 20017. Disponível em: <https://catalogue.fiware.org/enablers/backend-device-management-idas>.

_____. **IoT Broker NEC**. 2017. Disponível em: <https://catalogue.fiware.org/enablers/iot-broker>.

COHN, R. J. C. R. J. **MQTT Version 3.1.1**. [S.l.], 2015.

DIERKS, T. The transport layer security (tls) protocol version 1.2. 2008.

FARDAN, N. J. A.; PATERSON, K. G. Lucky thirteen: Breaking the tls and dtls record protocols. In: IEEE. **Security and Privacy (SP), 2013 IEEE Symposium on**. [S.l.], 2013. p. 526–540.

FIWARE. **Developing IPv6-Enabled Apps with Fiware**. 2017. Disponível em: <https://www.fiware.org/tag/ipv6/>.

_____. **Documentation IoT Agent Framework**. 2017. Disponível em: <https://github.com/telefonicaid/iotagent-node-lib>.

FIWARE. 2017. Disponível em: <https://www.fiware.org/>.

_____. **FIWARE contribution to the OpenStack**. 2017. Disponível em: <https://www.fiware.org/2016/04/05/fiware-contribution-to-openstack/>.

_____. **NGSI-10 Open RESTful API Specification**. 2017. Disponível em: <https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-10_Open_RESTful_API_Specification>.

_____. **NGSI-9 Open RESTful API Specification**. 2017. Disponível em: <https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-9_Open_RESTful_API_Specification>.

_____. **OpenSpecification Data ContextBroker**. 2017. Disponível em: <https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Data.ContextBroker>.

_____. **Orion Context Broker**. 2017. Disponível em: <https://fiware-orion.readthedocs.io/en/master/user/walkthrough_apiv2/index.html>.

GARCIA-MORCHON, O. et al. Securing the ip-based internet of things with hip and dtls. In: ACM. **Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks**. [S.l.], 2013. p. 119–124.

GAZIS, V. et al. A survey of technologies for the internet of things. In: **2015 International Wireless Communications and Mobile Computing Conference (IWCMC)**. [S.l.: s.n.]. p. 1090–1095.

IBM. **IBM Watson IoT Platform**. 2017. Disponível em: <https://www.ibm.com/cloud-computing/bluemix/pt/internet-of-things>.

ITU-T STUDY GROUP 17. **X.800 Security architecture for Open Systems Interconnection for CCITT applications**. 1991. Disponível em: <http://www.itu.int/itu-t/recommendations/rec.aspx?rec=3102>.

KAAIOT. **Kaa Project**. 2017. Disponível em: <www.kaaproject.org>.

KARAGIANNIS, V. et al. A survey on application layer protocols for the internet of things. **Transaction on IoT and Cloud Computing**, v. 3, n. 1, p. 11–17, 2015.

KHAN, R. et al. Future internet: The internet of things architecture, possible applications and key challenges. In: **2012 10th International Conference on Frontiers of Information Technology**. [S.l.: s.n.], 2012. p. 257–260.

KOTHMAYR, T. et al. Dtls based security and two-way authentication for the internet of things. **Ad Hoc Networks**, Elsevier, v. 11, n. 8, p. 2710–2723, 2013.

LAINE, M. Restful web services for the internet of things.

LIBELIUM. **Meshlium Technical Guite**. 2017. Disponível em: <http://www.libelium.com/downloads/documentation/meshlium_technical_guide.pdf>.

_____. **Plug and Sense**. 2017. Disponível em: <http://www.libelium.com/products/plug-sense/models/>.

M2M, O. L. **Lightweight Machine to Machine Requirements**. 2017. Disponível em: <http://www.openmobilealliance.org/release/LightweightM2M/>.

MADAKAM, S.; RAMASWAMY, R.; TRIPATHI, S. Internet of things (iot): A literature review. **Journal of Computer and Communications**, Scientific Research Publishing, v. 3, n. 05, p. 164, 2015.

MAHMOUD, R. et al. Internet of things (iot) security: Current status, challenges and prospective measures. In: IEEE. **Internet Technology and Secured Transactions (ICITST), 2015 10th International Conference for**. [S.l.], 2015. p. 336–341.

MEYER, C.; SCHWENK, J. Sok: Lessons learned from ssl/tls attacks. In: SPRINGER. **International Workshop on Information Security Applications**. [S.l.], 2013. p. 189–209.

MICROSOFT. **Azure IoT Hub**. 2017. Disponível em: <https://azure.microsoft.com/pt-br/services/iot-hub/>.

MODADUGU, N.; RESCORLA, E. The design and implementation of datagram tls. In: **NDSS**. [S.l.: s.n.], 2004.

MOSQUITTO. **MQTT Mosquitto**. 2017. Disponível em: <https://mosquitto.org/documentation/>.

NEUSTAR. **Implementation CoAP extended to use DTLS**. 2017. Disponível em: <https://github.com/neustar/node-coap-dtls>.

NODE, F. **Express - framework de aplicativo da web NodeJs**. 2017. Disponível em: <http://expressjs.com/pt-br/>.

OMA. **OMA Specifications**. 2017. Disponível em: <http://www.openmobilealliance.org/wp/>.

PERERA, C. et al. Sensing as a service model for smart cities supported by internet of things. **Transactions on Emerging Telecommunications Technologies**, Wiley Online Library, v. 25, n. 1, p. 81–93, 2014.

RAO, S. et al. Implementing lwm2m in constrained iot devices. In: IEEE. **Wireless Sensors (ICWiSe), 2015 IEEE Conference on**. [S.l.], 2015. p. 52–57.

RAZZAQUE, M. A. et al. Middleware for internet of things: a survey. **IEEE Internet of Things Journal**, IEEE, v. 3, n. 1, p. 70–95, 2016.

RESCORLA, E.; MODADUGU, N. Datagram transport layer security version 1.2. 2012.

SADEGHI, A.-R.; WACHSMANN, C.; WAIDNER, M. Security and privacy challenges in industrial internet of things. In: IEEE. **Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE**. [S.l.], 2015. p. 1–6.

SAINT-ANDRE, P. Extensible messaging and presence protocol (xmpp): Core. 2011.

SARKAR, P. G.; FITZGERALD, S. Attacks on ssl a comprehensive study of beast, crime, time, breach, lucky 13 & rc4 biases. **iSEC Partners**, 2013.

SATAPATHY, A.; M., J. L. L. A comprehensive survey on ssl/ tls and their vulnerabilities. **International Journal of Computer Applications**, Foundation of Computer Science (FCS), NY, USA, New York, USA, v. 153, n. 5, p. 31–38, Nov 2016. ISSN 0975-8887.

SCHAFFERS, H. et al. Smart cities and the future internet: Towards cooperation frameworks for open innovation. In: SPRINGER. **The Future Internet Assembly**. [S.l.], 2011. p. 431–446.

SEGGELMANN, R. **Datagram Transport Layer Security**. [S.l.], 2011.

SHEFFER, Y.; HOLZ, R.; SAINT-ANDRE, P. **Summarizing known attacks on transport layer security (tls) and datagram tls (dtls)**. [S.l.], 2015.

SHELBY, Z.; HARTKE, K.; BORMANN, C. The constrained application protocol (coap). 2014.

SICARI, S. et al. Security, privacy and trust in internet of things: The road ahead. **Computer Networks**, Elsevier, v. 76, p. 146–164, 2015.

STRAVOSKOUFOS, K.; SOTIRIADIS, S.; PETRAKIS, E. G. Iot-a and fiware: bridging the barriers between the cloud and iot systems design and implementation. p. 146–153, 01 2016.

SUNDMAEKER, H. et al. Vision and challenges for realising the internet of things. 2010.

TAN, L.; WANG, N. Future internet: The internet of things. In: **2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)**. [S.l.: s.n.], 2010. v. 5, p. V5–376–V5–380. ISSN 2154-7491.

THOMAS, C. **GitHub Projects**. 2017. Disponível em: <https://github.com/caiothomas>.

VM9IT. **Smart City**. 2017. Disponível em: <https://www.vm9it.com>.

WHITMORE, A.; AGARWAL, A.; XU, L. D. The internet of things—a survey of topics and trends. **Information Systems Frontiers**, Springer, v. 17, n. 2, p. 261–274, 2015.

WU, M. et al. Research on the architecture of internet of things. In: **2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)**. [S.l.: s.n.], 2010. v. 5, p. V5–484–V5–487. ISSN 2154-7491.

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