UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Análise de viabilidade em adicionar proteção à privacidade na camada de aplicação em redes smart home

Dissertação de Mestrado

Fernando Henrique Vieira Trindade

São Cristóvão – Sergipe

2022

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Fernando Henrique Vieira Trindade

Análise de viabilidade em adicionar proteção à privacidade na camada de aplicação em redes smart home

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de mestre em Ciência da Computação.

Orientador(a): Admilson Ribamar Ribeiro
Coorientador(a): Kalil Araujo Bispo

São Cristóvão – Sergipe

2022

Resumo

Smart Home ou casa inteligente é uma das tecnologias emergentes que mais cresce ao longo dos anos. Para prover comunicação a objetos inteligentes é preciso utilizar protocolos de comunicação "leves"para não os sobrecarregar. Devido à limitação de recursos de hardware em *Smart Home System (SHS)* requisitos como tempo de resposta, baixo processamento, e baixo consumo energético costumam ser priorizados em detrimento da segurança e privacidade. Todavia, neste tipo de ambiente, dados trafegados de forma insegura expõem a privacidade e intimidade dos usuários. Após uma revisão da literatura, identificou-se que grande parte das aplicações utilizando os protocolos *MQTT* e *CoAP*, dois dos principais protocolos da camada de aplicação, eram desenvolvidas de forma insegura. Posto isto, estes protocolos foram escolhidos para serem analisados quanto aos mecanismos de segurança e seus impactos no uso doméstico. Esta dissertação aplica métodos estatísticos em experimentos com os protocolos *MQTT* e *CoAP*, utilizando o microcontrolador *ESP32*, e mensura o impacto da adição de autenticidade e confidencialidade à camada de aplicação em *SHS* no tocante à performance, eficiência energética e usabilidade. Foram realizados testes de envio e recebimento de mensagens sem a utilização de qualquer proteção à privacidade e então comparados com novos testes utilizando *TLS* para o *MQTT* e *DTLS* para o *CoAP*. Além do incremento de segurança, também adicionou-se níveis de confiabilidade e ampliação do tamanho de *payload*. Para cada um destes cenários, avaliou-se o custo do consumo energético, o tempo de resposta entre o envio e o recebimento da mensagem, assim como a complexidade necessária para adicionar os procolos *TLS* e *DLS* como proteção do *MQTT* e *CoAP*, respectivamente. Os resultados obtidos mostram que o valor, na moeda Real do Brasil, do consumo de energia chegou a aumentar 2 centavos para o *CoAP* com *DTLS* e 7 centavos no cenário mais seguro utilizando *MQTT* com *QoS* 2 + *TLS* e o *payload* de 1 *kb*. O nível de complexidade aumentou de 4 pontos para 24, utilizando avaliação heurística desenvolvida neste trabalho. Quanto a performance no *MQTT*, identificou-se uma sobrecarga de até 1500% e no *CoAP* de 178% nos testes com maior segurança e confiabilidade. Ainda que os números pareçam expressivos, diante do impacto da falta de privacidade, os resultados demonstraram que o uso dos protocolos de segurança *TLS* e *DLTS* são viáveis no ambiente de *smart homes*.

Palavras-chave: *MQTT*. *TLS*. *CoAP*. *DTLS*. Privacidade. Segurança. *Smart Home*.

Abstract

Smart Home is one of the fastest-growing emerging technologies over the years. To provide communication to smart objects, it is necessary to use "light" communication protocols so as not to overload them. Due to limited hardware resources in SHS (smart home system), requirements such as response time, low processing, and low power consumption are often prioritized over security and privacy.However, in this type of environment, data traffic in an insecure way exposes the privacy and intimacy of users. After reviewing the literature, it was identified that applications using the MQTT and CoAP protocols (two of the main application layer protocols) were developed in an insecure way. That said, these protocols were chosen to be analyzed regarding security mechanisms and their impacts on domestic use. This dissertation applies statistical methods in experiments with MQTT and CoAP protocols, using the ESP32 microcontroller, and measures the impact of adding authenticity and confidentiality to the application layer in SHS in terms of performance, energy efficiency, and usability.Tests of sending and receiving messages without using any privacy protection were performed and then compared with new tests using TLS for MQTT and DTLS for CoAP. In addition to the increase in security, levels of reliability and expansion of the payload size were also added. For each of these scenarios, the cost of energy consumption, the response time between sending and receiving the message, as well as the complexity required to add the TLS and DLS protocols to protect the MQTT and CoAP respectively, were evaluated. The results obtained show that the value (in Brazilian Real currency) of energy consumption increased by 2 cents for CoAP with DTLS and 7 cents in the most secure scenario using MQTT with QoS2 + TLS and the 1kb payload. The level of complexity increased from 4 points to 24 using the heuristic evaluation developed in this work. As for the performance in MQTT an overhead of up to 1500% and in CoAP of 178% was identified in the tests with greater security and reliability. Although the numbers seem expressive, given the impact of the lack of privacy, the results showed that the use of TLS and DLTS security protocols are feasible in the smart home environment.

Keywords: *MQTT. TLS. CoAP. DTLS. Privacy. Security. Smart Home.*

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AMQP	*Advanced Message Queuing Protocol*
API	*Application Programming Interfacel*
BSD	*Berkeley Software Distribution*
CoAP	*Constrained Application Protocol*
CBOR	*Concise Binary Object Representationl*
DTLS	*Datagram Transport Layer Security*
ERP	*Enterprise Resource Planning*
E2E	*End-to-End*
EEPROM	*Electrically Erasable Programmable Read-only Memory*
HTTP	*Hypertext Transfer Protocol*
I2C	*Inter-Integrated Circuit)*
IANA	*Internet Assigned Numbers Authority*
IPsec	*IP Security Protocol*
ISO	*International Organization for Standardization*
IEEE	*Institute of Electrical and Electronic Engineers*
IoT	*Internet of Things*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
MQTT	*Message Queuing Telemetry Transport*
NFC	*Near Field Communication*
OSI	*Open System Interconnection*
PLR	*Packet-loss ratio*
PSK	*Pre-Shared Key*
QoS	*Quality of Service*

RAM	*Random Access Memory*
REST	*Representational State Transfer*
RFC	*Request for Comments*
RFID	*Radio-Frequency IDentification*
SBC	*Single Board computer*
SCTO	*Stream Control Transmission Protocol*
SEQ	*Single Ease Question*
SINTEF	*Stiftelsen for industriell og teknisk forskning*
SoC	*System on Chip*
SHS	*Smart Home Systems*
SMBUS	*System Management Bus*
SMS	*Short Message Service*
SPI	*Serial Peripheral Interface*
SRAM	*Static Random Access Memory*
TCP	*Transmission Control Protocol*
TTL	*Time to Live*
TLS	*Transport Layer Security*
TWI	*Two-Wire Interface*
UART	*Universal Asynchronous Receiver/Transmitter*
UDP	*User Datagram Protocol*
URI	*Uniform Resource Identifier*
XEP	*XMPP Extension Protocol*
XML	*EXtensible Markup Language*
XMPP	*Extensible Messaging and Presence Protocol*

Sumário

1 Introdução

1.1 Apresentação Geral

A Internet das coisas ou *Internet of Things (IoT)* é uma tecnologia em desenvolvimento que torna o mundo cada vez mais conectado. Essa nova categoria de redes de computadores expande a capacidade de comunicação e informação entre dispositivos e seres humanos. Assim, as redes que antes deveriam atender aos requisitos: *"any time"* (a qualquer momento) e *"any place"* (em qualquer lugar), agora também atendem ao requisito *"any thing"* (em qualquer coisa). Por outro lado, torna crítica a situação referente à segurança da informação e privacidade (SURYANEGARA; ASVIAL; RAHARYA, 2015).

A *IoT* influenciou uma massiva explosão no volume de dados trafegados na Internet e armazenados na nuvem. A organização de pesquisa norueguesa *Stiftelsen for industriell og teknisk forskning (SINTEF)* apontou que em meados do ano de 2015, 90% dos dados mundiais foram gerados a uma velocidade de mais de 205.000 gigabytes por segundo, o que era aproximadamente equivalente a 150 milhões de livros (DUBRAVAC; RATTI, 2015).

A Internet das Coisas é bastante ampla e pode ser subdividida em diversas áreas de aplicação, a exemplo de: cidades inteligentes, casas inteligentes ou *smart homes*, cuidados com a saúde ou *health care*, automação de indústria, automação da agropecuária, entre diversas outras. Segundo Gartner (2019), dentre as tantas subáreas da *IoT*, as que mais crescem são: casas inteligentes e cuidados com a saúde.

Smart Home é uma habitação que incorpora uma rede de comunicação que liga os aparelhos eléctricos e serviços essenciais, e permite que eles sejam controlados remotamente, monitorados ou acessados (KING, 2003). Com o advento dos microcontroladores integrados às placas de redes de baixo custo, houve uma crescente demanda de utensílios inteligentes e conectados. A produção em massa desses eletrônicos popularizou a utilização de objetos que trazem comodidade e conforto através da comunicação, monitoramento e interação com pessoas

comuns em suas residências. Os benefícios da utilização dessas tecnologias vão desde o controle de acesso físico, através de biometria, até o acionamento de aparelhos domésticos por comando de voz.

Diante deste cenário, onde a tecnologia está inserida no cotidiano das pessoas até mesmo nos momentos mais íntimos dentro de suas próprias casas e que estes dispositivos de baixo poder de processamento são limitados para processar uma camada robusta de segurança, é preciso ter bastante cuidado no que cerne à privacidade e segurança dos dados. Diversas pesquisas, sob diferentes perspectivas, já foram realizadas sobre segurança em redes *IoT* domésticas, mas algumas lacunas ainda permanecem e devem ser exploradas. Com o objetivo de entender melhor os problemas relacionados à segurança e privacidade, no contexto supracitado, foi realizado primeiramente um mapeamento sistemático onde foi detectado o desafio da privacidade em redes *IoT*. Posteriormente foi realizada uma revisão sistemática para identificar quais trabalhos estão abordando esse assunto e se ainda existe questões que precisam ser abordadas. O resultado do mapeamento e da revisão serviu de embasamento para esta pesquisa, que tem o objetivo de analisar a viabilidade de adicionar mecanismos de proteção à privacidade, na camada de aplicação, de redes IoT em *smart homes*.

1.2 Motivação

Um dos princípios fundamentais do conceito Internet das Coisas é construir uma solução adequada às necessidades das aplicações considerando as devidas restrições de requisitos não-funcionais. A *IoT* possui a característica de trabalhar com recursos escassos, ou seja, sistemas embarcados em dispositivos com pouco poder computacional e interligados em rede.

Dessa maneira, a escolha dos protocolos influencia drasticamente o consumo de energia e desempenho dos dispositivos de uma rede de Internet das Coisas. Pode-se notar a relevância deste tema ao perceber a expressividade da *IoT* no cenário atual. Segundo Gartner (2019), *IoT* está entre as 8 tecnologias em destaque para 2019. Além disso, estima-se que 26 bilhões de dispositivos *IoT* estiveram conectados em 2020. Isso resulta em US$ 1,9 trilhão em valor econômico global agregado, ademais, os provedores de produtos e serviços da *IoT* geraram receita incremental superior a US $300 bilhões, principalmente em serviços. Já Lueth (2020), acredita que até 2025 serão 22 bilhões de dispositivos. Desse modo, torna-se imprescindível escolher de maneira assertiva qual protocolo de comunicação e mecanismos de segurança devem ser utilizado, visto que a eficiência energética, taxa de transmissão e cobertura, são categóricos no sucesso da aplicação no contexto de *IoT*.

1.3 Justificativa

A partir da motivação levantada na seção 1.2, fica evidente a necessidade proteger, no que tange a privacidade, as redes *smart home* dada a sua relevância. Assim, há necessidade de que novas formas de proteção sejam desenvolvidas e que tenham a característica de levar em consideração requisitos de desempenho, consumo energético e usabilidade.

Para isso, torna-se necessário resolver questionamentos a cerca de: O ambiente de *smart home IoT* suporta, do ponto de vista de desempenho, níveis altos de proteção à privacidade?; O aumento no nível de proteção à privacidade é viável do ponto de vista de desempenho? (tempo de resposta); O aumento no nível de proteção à privacidade é viável do ponto de vista econômico? em quais dispositivos? (dispositivo para economizar energia pode ser seguro?) e O incremento de segurança é viável do ponto de vista do usuário? (usabilidade, configurações complexas). Dessa maneira, torna-se crucial responder estes questionamentos, para garantir que ambientes *smart home* seguros permitam que os usuários possam utilizar seus dispositivos com um nível de privacidade aceitável.

1.4 Objetivos

Este trabalho foi realizado com a finalidade de resolver alguns questionamentos a respeito dos mecanismos de proteção à privacidade nos protocolos da camada de aplicação em redes *IoT*. Para isso, foi primordial definir objetivos os quais nortearam as respostas aos questionamentos supracitados.

1.4.1 Objetivos Gerais

O objetivo geral deste trabalho é verificar a viabilidade de adicionar mecanismos de proteção à privacidade na camada de aplicação, em redes *IoT* no contexto de *smart home*. Este incremento de segurança tem como objetivo garantir a confidencialidade das informações trafegadas, sem comprometer os requisitos de qualidade relacionados a performance e eficiência energética a ponto de inviabilizar uma solução de *smart home*.

1.4.2 Objetivos Específicos

Para alcançar o objetivo principal, foram definidos os seguintes objetivos específicos:

- Levantar as principais vulnerabilidades relacionadas à privacidade na camada de aplicação de redes *IoT*.
- Levantar os principais protocolos da camada de aplicação (que rodam sobre *TCP* e *UDP*) e suas opções de confidencialidade para redes *smart home*.

- Analisar os possíveis impactos de um tráfego vulnerável no contexto de *smart home*.
- Verificar, por prototipação, a avaliação dos mecanismos de privacidade utilizando uma metodologia para aferimento de tempo de resposta e aumento do consumo de corrente elétrica.
- Apurar o cenário viável para que a camada de aplicação possa prover privacidade em dispositivos de *smart home*.

1.5 Estrutura do Documento

Para facilitar a navegação e melhor entendimento, este documento está estruturado em capítulos e seções, que são:

- Capítulo 1 - Introdução: apresenta as definições preliminares da literatura, problemática, argumentações e hipóteses sobre o tema, além dos objetivos;
- Capítulo 2 - Fundamentação teórica: expõe a contextualização teórica, com a revisão de literatura relacionada ao tema proposto;
- Capítulo 3 - Trabalhos relacionados: demonstra os trabalhos correlatos com a revisão de literatura adotada (Revisão Sistemática), bem como são apresentados os resultados dessa revisão e a síntese do processo de sumarização dos trabalhos estudados;
- Capítulo 4 - Metodologia: o hardware, bibliotecas e *firmwares*, assim como as configurações necessários para realizar a experimentação.
- Capítulo 5 - Protocolos, ameaças e mecanismos de segurança: consiste em uma abordagem mais detalhada dos protocolos *Message Queuing Telemetry Transport (MQTT)* e *Constrained Application Protocol (CoAP)*, assim como seus respectivos mecanismos de segurança *Transport Layer Security (TLS)*, *Datagram Transport Layer Security (DTLS)* e as principais vulnerabilidades. Nesta seção demonstra-se também as configurações relacionadas à segurança, privacidade e *Quality of Service (QoS)*;
- Capítulo 6 - Experimentação, análise de desempenho, usabilidade e custos: exposição dos resultados obtidos na experimentação referente a performance, usabilidade, custos e a avaliação dos resultados;
- Capítulo 7 - Conclusões e Considerações Finais.

2

Fundamentação Teórica

Neste capítulo são levantados conceitos, aplicações, características e arquitetura a respeito dos temas abordados. Dessa maneira, foi realizada uma consulta a literatura específica ao tema. Assim, foi possível modelar o conhecimento inicial necessário para este trabalho no que diz respeito a privacidade na camada de aplicações de redes *IoT* no contexto de *smart home* .

2.1 *IoT*

O termo *IoT* foi utilizado pela primeira vez em 1999 por Kevin Ashton, um britânico pioneiro nessa tecnologia. De acordo com Kevin, a Internet das Coisas é definida como um sistema de objetos físicos que se conectam à Internet através de um sensor, ou seja, consiste em máquinas inteligentes que interagem com outras máquinas, objetos, ambientes e infraestruturas. Essa tecnologia emergente vem impactando imensamente o cotidiano das pessoas e ajudando a tornar suas vidas mais prática e inteligente. A Internet das Coisas já tornou-se uma área fundamental para os negócios provendo transações em tempo real, automatizando processos em cadeias de produção e operações logísticas.

Consequentemente, essa nova tecnologia envolve coleta e gerenciamento de grandes volumes de dados, processamento e compartilhamento com outros sensores ou objetos. Toda essa interconexão cria diversas oportunidades de novos serviços. O processo de automatização reduz os custos com a mão de obra reduzindo assim o custo para produção de produtos industrializados e prestação de serviços. Por isso, tem se tornado um diferencial competitivo.

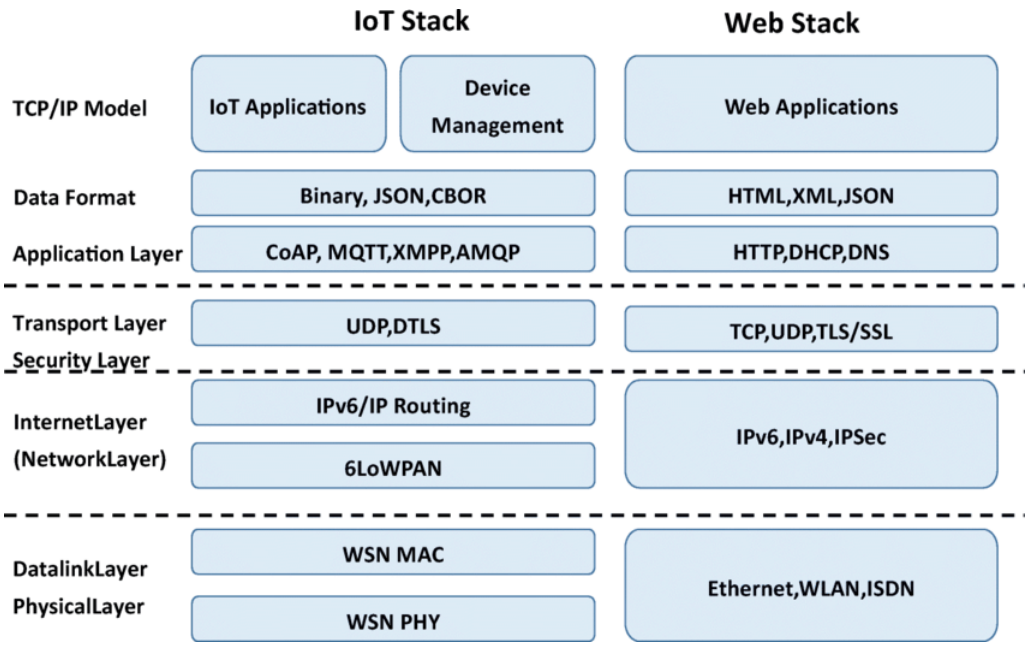
Deverá acontecer um grande crescimento no mercado global de *IoT*, aliado a uma vasta diversidade de dispositivos, chegando a 27.1 bilhões em final de 2025. Entretanto a grande quantidade de dados gerados por objetos *IoT* pode representar uma séria ameaça à privacidade e segurança das pessoas, dado que suas atividades podem ser monitoradas a qualquer hora e em qualquer lugar (LUETH, 2020). As potenciais ameaças de segurança que podem ser usadas para

prejudicar os consumidores são: acesso não autorizado e uso impróprio de informações pessoais, promoção de ataques a outros sistemas e o aumento dos riscos de segurança (RAO; SHAIK; REDDY, 2017).

2.1.1 Arquitetura e Protocolos

Para estabelecer uma interconexão em redes *IoT*, os dispositivos precisam seguir um conjunto de regras e especificações para comunicar-se entre si e com a Internet. Esse conjunto de regras são implementadas nos protocolos, que são comumente representados em camadas de uma arquitetura, onde cada camada é responsável por um conjunto específico de funções. Segundo Aitzaouiat et al. (2022), a arquitetura de Rede de Computadores é um conjunto de camadas e protocolos de rede, mais precisamente, é a interconexão lógica e física de todos os elementos, desde a geração de um sinal, até sua terminação. Na literatura é possível encontrar dois modelos para ilustrar as camadas de protocolos utilizadas para uma comunicação em redes de computadores, são eles: Modelo *OSI* e *TCP/IP*. Este trabalho propõe a estudar a camada de aplicação do modelo *TCP/IP* de redes *IoT*, que é bastante semelhante a arquitetura *TCP/IP* da Web. A Figura 1 exibe uma comparação entre o modelo *TCP/IP IoT* e o *TCP/IP* Web. É possível perceber que alguns protocolos, principalmente os da Camada de Aplicação Web, são substituídos por protocolos "mais leves"na camada equivalente da *IoT*.

Figura 1 – Modelo *TCP/IP*



Fonte: (AITZAOUIAT et al., 2022)

Dentre os protocolos mais utilizados na camada de aplicação de redes IoT destacam-se:

1. **MQTT**: O *Message Queue Telemetry Transport (MQTT)* é um protocolo de camada de aplicação projetado para dispositivos com recursos limitados. Ele utiliza uma arquitetura baseada em *publisher-subscriber* (publicadores-assinantes). Isso significa que quando um o cliente publica uma mensagem M para um determinado tópico T, todos os clientes inscritos no tópico T receberão a mensagem M. Assim como o *Hypertext Transfer Protocol (HTTP)*, o *MQTT* depende do protocolo de controle de transmissão *TCP/IP* assim como suas camadas subjacentes. Contudo, em comparação com o *HTTP*, o *MQTT* é projetado para ter uma menor sobrecarga de protocolo. Em outras palavras, ele é mais "enxuto", ou seja, consome menos recursos, com o objetivo de obter uma transmissão mais leve e que pode ser utilizada dispositivos com recursos restritos (THANGAVEL et al., 2014).

2. **CoAP**: *Constrained Application Protocol (CoAP)* é um protocolo da Camada de Aplicação desenvolvido para ser usado na comunicação entre dispositivos que possuem recursos restritos. Este protocolo é baseado em na arquitetura *REST* e suporta o modelo de solicitação-resposta como o *HTTP*. O *CoAP* também oferece suporte para publicação e assinatura arquitetura usando o método *GET*, similar ao *GET* do *HTTP* estendido. Porém, ao contrário do *MQTT*, o modelo publicador-assinante de *CoAP* utiliza *Universal Resource Identifier (URI)* em vez de tópicos. Isso significa que os assinantes irão se inscrever em um recurso específico indicado pelo *URI U*. Assim, quando um editor publica os dados D para o *URI U*, todos assinantes são notificados sobre o novo valor conforme indicado em D (THANGAVEL et al., 2014).

3. **AMQP**:O *Advanced Message Queuing Protocol (AMQP)* é um procotolo que foi desenhado para atender aos seguintes requisitos: segurança, confiabilidade, interoperabilidade, padronização e ser código aberto. Ele tem o propósito de prover comunicação entre dispositivos heterogêneos, utilizando o mesmo protocolo para que diversas linguagens de programação possam se comunicar facilmente. Além disso, possibilitam que servidores legados de mensagens trocar mensagens como se fossem um serviço de nuvem. Se propõe ainda a prover um serviço avançado de *publisher/subscriber* como também funcionalidade de mensagens transacionais (AMQP, 2020).

4. **XMPP**: O *Extensible Messaging and Presence Protocol (XMPP)* é um protocolo da Camada de Aplicação utilizado para troca de informações entre quaisquer entidades de rede em tempo real. O protocolo foi originalmente desenvolvido com o nome de *Jabber* dentro da comunidade de código aberto *Jabber* em 1999, e seu objetivo principal era resolver os problemas com mensagens instantâneas e aplicativos de presença daquela época. Desde então, o núcleo do protocolo *Jabber* foi revisado e formalizado pela *Internet Engineering Task Force (IETF)* e publicado sob seu nome atual *XMPP* na *RFC 6120* e *RFC 6121*. Além das especificações básicas do *XMPP* mencionadas, o *XMPP Standards Foundation (XSF)* desenvolveu e publicou mais de 300 protocolos de extensão *XMPP (XEPs)* para cobrir uma ampla variedade de cenários de aplicação. Dentre alguns exemplos,

destacam-se: o *XEP-0206*, que versa sobre *XMPP* para *BOSH*, protocolo similar ao *HTTP* para comunicações *XMPP*, *XEP-0060*, que versa sobre um protocolo *Publish-Subscribe* para mensagens *multicast* e o *XEP-0045*, que versa sobre um protocolo utilizado em salas de bate-papo com multiusuário autenticados (LAINE; SÄILÄ, 2012).

2.1.2 Aplicações da *IoT*

A diversidade de aplicações da *IoT* vem modificando a forma como nós interagimos com os dispositivos conectados. À medida que o custo de produção diminui, há uma maior aplicação dessas tecnologias, além de haver um aumento no retorno sobre investimento aplicado à ela. A Figura 2 demonstra algumas áreas em que existe aplicação de *IoT*.

Figura 2 – Aplicações da *IoT*

Fonte: Santos (2018, p. 3)

A sua adoção é cada vez mais aplicada em áreas como: indústria, agropecuária, saúde, entre outros. Dentre as principais, destacam-se:

- **Direção Assistida**: para melhorar a navegação e a segurança de carros, de trens e de ônibus, são utilizados sensores e atuadores que podem prover informações sobre acidentes, ruas interditadas, etc. Ademais, essas informações podem ser usadas para otimizar rotas (ATZORI; IERA; MORABITO, 2010).
- **Bilhetagem Móvel**: usuários de serviços de transporte podem utilizar o celular com tecnologia *Near Field Communication (NFC)* e simplesmente passar o celular em cima de um leitor de *NFC* e pagar o ticket do transporte (BING et al., 2011).
- ***E-health***: sensores coletam sinais vitais de pacientes e ajudam em diagnósticos provendo informações em tempo real. Como por exemplo contador de passos, batimentos cardíacos,

níveis de insulina, padrões de sono, entre outros. Dessa maneira, essa tecnologia objetiva o monitoramento contínuo da condição dos pacientes utilizando tecnologias ubíquas e sem fio (DATTA; SHARMA, 2017).

- ***Smart Home***: segurança, eficiência energética, conforto e assistência à idosos ou deficientes são algumas motivações para utilizar *IoT* em uma casa. Para a segurança, por exemplo, destacam-se aplicações de vigilância da casa, monitoramento de possíveis intrusões, alertas automatizados e chamadas automática para as autoridades. Já para a eficiência energética e conforto, destacam-se aplicações para ligar ou desligar luzes de acordo com a luminosidade do ambiente; abrir ou fechar cortinas de acordo com o as condições climáticas; ligar, desligar ou ajustar o ar condicionado de acordo alguma condição pré-estabelecida; desligar dispositivos não utilizados automaticamente para economizar energia e verificar a geladeira e adicionar a comida que falta à lista de compras. Para a assistência à idosos ou deficientes, destacam-se aplicações que utilizam recursos visuais ou sonoros para fornecer alertas para eventos e através de dispositivos vestíveis alertar a família ou autoridades no caso de um incidente (BASTOS; SHACKLETON; EL-MOUSSA, 2018).

- ***Smart City***: otimização de serviços públicos tradicionais, gestão de tráfego, monitoramento consumo de energia, monitorando a qualidade do ar e o ruído e redução de custos operacionais as algumas motivações para o uso de *IoT* em uma cidade são (BASTOS; SHACKLETON; EL-MOUSSA, 2018).

- **Agricultura**: o alcance da *IoT* pode definitivamente influenciar a vida do fazendeiro através da implantação de sensores no solo, que podem fornecer dados para ajudá-los a planejar a melhor época do ano para semear as sementes. Além disso, pode-se utilizar atuadores e sensores para irrigar e manter a muda com um crescimento saudável e livre de pragas, controlando a quantidade de inseticidas (CHAUDHARY et al., 2019). Ademais, existem plantações que utilizam tratores autônomos tanto para semear quanto para colher e *drones* que monitoram, através de imagens, o controle de pragas e o momento ideal da colheita.

- **Plantas industriais**: ambientes inteligentes também ajudam na melhoria da automação de plantas industriais, com uma implantação massiva de etiquetas com a tecnologia *Radio-Frequency Identification (RFID)*, associadas às peças de produção. Essas peças chegam ao ponto de processamento, a etiqueta é lida pelo leitor *RFID* e um evento é gerado pelo leitor com todos os dados necessários. Estas informações são trafegadas na rede e a máquina é notificada sobre este evento. Combinando os dados de produção da empresa e os dados da etiqueta *RFID*, a máquina saberá qual procedimento deverá realizar na peça. Em paralelo, um sensor sem fio montado na máquina monitora a vibração, e se ela exceder um limite específico, um evento é disparado imediatamente parar o processo. Ele é parte do controle de qualidade e uma vez que isto aconteça, o evento de emergência é propagado e dispositivos que assinam este serviço podem reagir de acordo com o

procedimento previamente planejado. A maquina recebe o desligamento de emergência, o evento e interrompe imediatamente o seu funcionamento. O gerente da planta industrial vê imediatamente o status do chamado através do sistema *Enterprise Resource Planning (ERP)*. Além de ver também os indicadores de produção, progresso, o status do dispositivo, bem como uma visão global de todos os elementos e os possíveis efeitos colaterais de uma produção atraso devido ao mau funcionamento de dispositivos no chão de fábrica (SPIESS et al., 2009).

2.2 Segurança e Privacidade

A Internet das Coisas vem impactando enormemente a vida das pessoas, pois facilita viver e trabalhar de maneira mais inteligente, bem como gerenciar e controlar de forma automatizada o ambiente. Além de oferecer dispositivos inteligentes para automatizar casas, a *IoT* é essencial para os negócios, pois fornece às empresas uma visão em tempo real de como seus sistemas realmente funcionam, fornecendo percepções sobre tudo, desde o desempenho das máquinas para a cadeia de suprimentos e operações logísticas (OGONJI; OKEYO; WAFULA, 2020). Consequentemente, esta nova realidade tecnológica envolve coleta e gerenciamento de grandes volumes de dados através de uma crescente rede de dispositivos e sensores. Esses dados acabam sendo compartilhados entre diversos serviços diferentes com a finalidade de criar um modelo inteligente que forneça cada vez mais valor agregado e customizado para o usuário.

A grande quantidade de dados gerados por objetos *IoT* pode representar uma séria ameaça à privacidade e segurança das pessoas, uma vez que suas atividades podem ser monitoradas a qualquer hora e em qualquer lugar (NEISSE et al., 2014). As potenciais ameaças de segurança que podem ser usadas para prejudicar os consumidores são: acesso não autorizado e uso impróprio de informações pessoais; promoção de ataques a outros sistemas e o aumento dos riscos de segurança. Para evitar essas ameaças é preciso garantir um fluxo contínuo de informações, ademais é necessário trafegar por um meio seguro e que se garanta a confidencialidade, a integridade e a disponibilidade.

2.2.1 Visão centrada no usuário

A maior parte dos estudos e discussões sobre *IoT* gira em torno de objetos inteligentes, enquanto o usuário não recebe a devida importância, já que as informações mais importantes são dos objetos para o usuário. Portanto, há necessidade de colocar usuários no centro das soluções de *IoT*. O usuário é neste contexto, um humano, definido por diferentes características: nome, idade, trabalho, nível escolar, interesses, domínios de especialização e preferências. Isso cria um perfil que pode ser representado em um sistema de computador e o usuário também é representado pelo contexto. Esse contexto inclui localização, atividade atual, objetos e outros usuários nas proximidades, e também o contexto social. Assim, o contexto social de um usuário é

representado por um conjunto de relações sociais com outros usuários, definindo assim uma rede de conexões entre usuários, comumente chamada de redes sociais. Dessa maneira, em sistemas inteligentes o usuário está, portanto, no centro desses processos, por isso que alguns paradigmas surgiram dando foco ao usuário. Dentre os trabalhos que focam na detecção de perfil de usuário destaca-se o trabalho de Ogonji, Okeyo e Wafula (2020). No ambiente *IoT*, a visão centrada no usuário está preocupada em capacitar os usuários a assumirem o controle do acesso aos seus dados e a sua privacidade. Isso ocorre porque os usuários compartilham suas informações mais pessoais com terceiros ou com dispositivos nem sempre confiáveis. Assim, é preciso prover meios que garantam que os dados estarão protegidos contra interceptações indevidas e que não serão compartilhados entre pessoas ou empresas não autorizadas.

2.2.2 Privacidade em *IoT*

A proteção da privacidade é muito ampla e multifacetada, além disso, seu conceito pode variar a depender da área de aplicação ou do contexto. Para Seliem, Elgazzar e Khalil (2018), privacidade significa que as informações sobre os indivíduos devem ser protegidos e não devem ser expostos sem consentimento em qualquer circunstância. Cada indivíduo tem o direito final de decidir com quem compartilhar seus dados. A confiança, em privacidade, é derivada de dois termos: transparência e consistência. Transparência significa que os dispositivos *IoT* que coletam informações, informam o usuário sobre quais dados foram coletados, a finalidade da coleta e como os dados coletados serão usados. Consistência significa que o comportamento dos dispositivos *IoT* atende consistentemente às expectativas do usuário. Por exemplo, se um usuário pede ao seu alto-falante inteligente para controlar a lâmpada da sala, não deve-se fazer nada não intencional, somente o específico da tarefa solicitada. Por fim, a segurança refere-se à proteção de dispositivos e conexão de acesso não autorizado. Com base nas definições acima mencionadas, fica evidente que privacidade é mais geral do que segurança e confiança (SELIEM; ELGAZZAR; KHALIL, 2018). Este trabalho propõe aprofundar análises no que se refere à segurança e consistência, uma vez que a transparência só pode ser garantida através de leis e regulamentações.

Objetos inteligentes normalmente têm recursos limitados com capacidades limitadas devido ao tamanho, peso, custo de aquisição e consumo energético. Por isso, costumam ter pouca memória, pouco processamento, e conectividade de rede limitada, como por exemplo do padrão *IEEE 802.15.4*. A especificação *IEEE 802.15.4* é restrita com em relação a baixas taxas de dados, que variam de 20 *Kbits/s(868 MHz)* a 250 *Kbits/s* (2,45 *GHz*); não é confiável por conta das perdas, quando comparado a links com fio; o tamanho do pacote é pequeno, 127 *bytes*, o que significa menos espaço para carga útil ao incluir outros cabeçalhos e possui um ciclo de alimentação restrito, já que objetos inteligentes visam economizar energia, permanecendo mais tempo em modo de baixa potência. Essas restrições afetam diretamente o tipo e complexidade de funcionalidades que os dispositivos *IoT* podem executar. Portanto, as limitações de recursos

tornam difícil proteger as camadas *IoT* individualmente. A Tabela 1 lista o número de dispositivos *IoT* vulneráveis por conta do protocolo *TLS* nos principais países do mundo em Setembro de 2017, totalizando 237.539 dispositivos (SELIEM; ELGAZZAR; KHALIL, 2018).

Tabela 1 – Vulnerabilidades do protocolo TLS.

Países	**Número de dispositivos vulneráveis**
Estados Unidos	57.598
China	17.455
Alemanha	17.273
França	10.708
Índia	9.427
Reino Unido	9.268
Rússia	7.897
Coreia do Sul	7.525
Brasil	7.095
Japão	5.302

Fonte: Seliem, Elgazzar e Khalil (2018)

2.2.3 Usabilidade

Usabilidade é frequentemente associada à interação entre usuários finais e interfaces de aplicações de tecnologia. Todavia o grau de dificuldade que um usuário pode apresentar ao utilizar um dispositivo de Internet das Coisas, pode variar desde a quantidade de etapas necessárias para a instalação básica, assim como pelo nível de proteção configurada. Alguns aparelhos inteligentes disponibilizam configurações que tornam o uso mais seguro e privativo. Em contra partida, exigem maior dedicação nos ajustes, e por vezes maior conhecimento técnico do usuário. Analisando o ciclo de vida de desenvolvimento de uma solução de *IoT*, é possível identificar que atores, os papéis que atuam nas fases do ciclo de desenvolvimento, podem ser considerados usuários da etapa que desempenham.

Em muitos casos os usuários de dispositivos de *smart home* são *makers*, ou seja, são queles que criam ou inventam coisas, utilizando artesanato ou tecnologia (CAMBRIDGE, 2021). Neste caso, a usabilidade deve ser considerada em nível de dificuldade técnica ao adicionar camadas de proteção durante o desenvolvimento. Este trabalho tem o foco de analisar somente os aspectos de dificuldades do usuário ao desenvolver, configurar ou utilizar um recurso de segurança à privacidade em ambientes *smart home*, os demais tópicos relacionados à usabilidade não foram abordados.

Usabilidade é um requisito não funcional de suma importância para qualquer produto onde ocorre uma interação homem-máquina. A Iso (1998) define usabilidade como a extensão na qual usuários específicos podem usar um produto para atingir objetivos específicos de forma eficaz, eficiente e satisfatória em um contexto específico de uso. De acordo com Nielsen e Budiu

(2013), um precursor nesta área, a usabilidade é um atributo de qualidade que avalia a facilidade de uso de uma interface, ou a medida da qualidade da experiência de um usuário ao interagir com um produto ou sistema. Além disso, para um sistema atingir um nível aceitável de usabilidade, é necessário atender aos seguintes objetivos:

- **Eficácia** - é a capacidade de resolver um problema para o qual o produto foi proposto. O sistema atende a especificação que resolve o problema do usuário. Na *IoT*, por exemplo, um dispositivo de lâmpada inteligente deve fornecer os meios para o usuário acender uma lâmpada por meio da comunicação com a Internet;
- **Eficiência** - Esta meta está vinculada aos recursos necessários para realizar uma determinada tarefa. As características que podem influenciar a eficiência de um sistema são o tempo de resposta que o sistema leva para dar feedback e o uso de recursos para executar a resposta. Por exemplo, em dispositivos inteligentes, é possível avaliar o tempo de leitura de um sensor ou o tempo de operação após o envio de um comando;
- **Segurança (confiabilidade)** - este item está diretamente vinculado à situações indesejáveis que podem ocorrer durante o uso de um aplicativo ou dispositivo. A segurança visa prevenir erros e reduzir riscos relacionados a ações drásticas, como desligar o sistema ou excluir algumas informações. A segurança e a confiabilidade no uso estão ligadas a requisitos como autenticidade e confidencialidade e evitam que o usuário crie situações de risco, como abrir acidentalmente a porta de sua casa;
- **Utilidade** - cada sistema possui uma utilidade. Além disso, o sistema deve possuir funcionalidade que justifique seu uso. O fato de que adicionar conectividade a um dispositivo deve agregar algum valor ao usuário, por exemplo, adicionar conectividade a uma lâmpada deve fornecer ao usuário a facilidade de programar a ativação ou desativação automática da lâmpada;
- **Aprendizagem** - é a capacidade que o sistema possui de ser intuitivo para o usuário. Uma lâmpada inteligente não deve ser tão complexa de usar a ponto de ser mais difícil que utilizar o disjuntor manual;
- **Memorização** - este objetivo está vinculado à facilidade do usuário em memorizar o processo necessário para utilizar um sistema. Em uma casa inteligente, um exemplo seria qual comando de voz é essencial memorizar para ativar uma lâmpada.

3

Trabalhos Relacionados

Nesta seção, elencamos e discutimos trabalhos de pesquisa que abordaram os assuntos que compõem o tema: **Mecanismos de proteção à privacidade** na **Camada de Aplicação** em redes *smart home*. Foi realizada uma Revisão Sistemática da Literatura e como critério de seleção, foram incluídos os trabalhos que utilizassem algum mecanismo que provesse maior privacidade ou detectasse vulnerabilidades, na camada de aplicação de redes *smart home*.

3.1 Revisão Sistemática

O levantamento bibliográfico se deu através de uma Revisão Sistemática da Literatura, a fim de manter um alto grau de rigor científico. Essa Revisão é um meio de identificar, avaliar e interpretar toda a pesquisa disponível e relevante para determinado tema ou questão de pesquisa específica (KITCHENHAM, 2004).

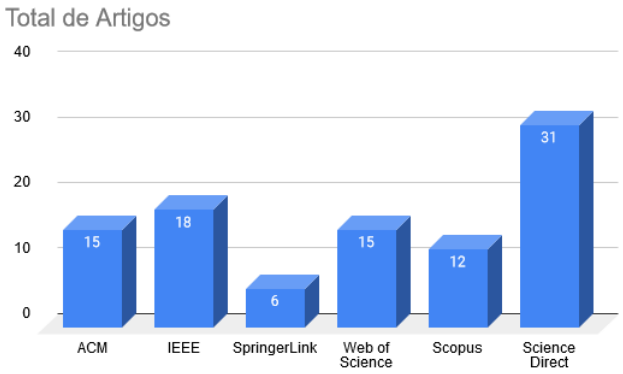
Para a realização desse levantamento bibliográfico utilizou-se as bases de dados *Springer Link* [1], *ACM Digital Library* [2], *IEEE Digital Library* [3], *ISI Web Of Science* [4], *Science Direct* [5] e *Scopus* [6]. Foram utilizados os seguintes termos de busca: *"privacy"*, *"application layer"*, *"smart home"*.

Esta Revisão Sistemática seguiu um protocolo rígido, com o objetivo de redução de viés de pesquisa, e foi dividida em três etapas: execução da busca, primeira etapa de seleção e segunda etapa de seleção. Para a etapa de execução da busca, foram realizados dois refinamentos na *string* de busca, com o objetivo de obter resultados mais significativos. Assim, a *string* de busca é então utilizada nas fontes selecionadas e os resultados obtidos armazenados na ferramenta online

[1] <https://link.springer.com/>
[2] <https://dl.acm.org/>
[3] <https://ieeexplore.ieee.org/Xplore/home.jsp>
[4] <https://webofknowledge.com>
[5] <https://www.sciencedirect.com/>
[6] <https://www.scopus.com/>

Parsifal [7]. A coleta de dados aconteceu no dia 06 de novembro de 2020, não foi definido nenhum limite temporal e foram selecionados somente artigos publicados em conferências ou periódicos. Dessa forma, a pesquisa retornou um total de 97 artigos, como pode ser visto na Figura 3, sendo 15 artigos na base *ACM Digital Library*, 18 artigos na base *IEEE Digital Library*, 31 artigos na base *Science Direct*, 12 artigos na base *Scopus*, 6 artigos na base *Springer Link* e 15 artigos na base *ISI Web Of Science*.

Figura 3 – Artigos levantados.

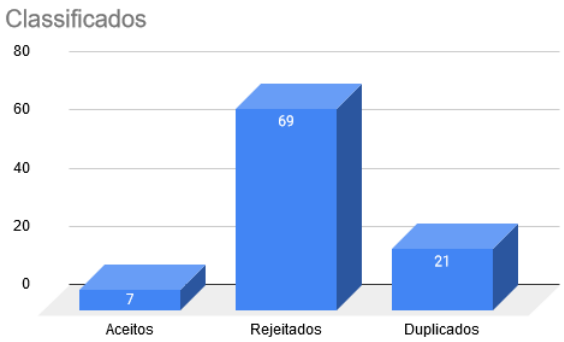


Fonte: Próprio autor

Após isso, foi realizada a primeira etapa de seleção onde foi aplicado aos artigos retornados na busca os critérios de inclusão e exclusão. Foram excluídos os artigos duplicados, artigos que não estavam escritos na língua inglesa, estudos não primários, artigos publicados apenas como resumos ou prefácio de periódicos e eventos, artigos com falta de disponibilidade para download, teses, dissertações, monografias de conclusão de curso e artigos fora da área de Ciências da Computação. Nessa etapa foram excluídos 21 artigos duplicados e 37 artigos que se encaixaram nos outros critérios de exclusão, resultando num total de 58 artigos excluídos, restando 39 artigos. Logo após, os artigos selecionados passaram para a primeira etapa de seleção, nela foi feita uma leitura dos títulos e dos resumos de cada estudo, a fim de detectar artigos fora do escopo da pesquisa. Nesta etapa foram excluídos 22 artigos, restando um total de 17 artigos. Após isso, os estudos selecionados passaram para a segunda etapa, a qual foi realizada a leitura completa dos artigos, nesta etapa foram excluídos 10 artigos, restando 7 artigos, os quais atendem os critérios formalizados no protocolo desta Revisão Sistemática. Do resultado inicial contendo 97 artigos, 7 foram aceitos e 90 foram excluídos, dentre os quais 21 estavam duplicados, conforme pode ser visto na Figura 4.

[7] <https://parsif.al/about/>

Figura 4 – Resultado da segunda etapa de seleção.



Fonte: Próprio autor

3.2 Trabalhos de Pesquisas Relacionados

O projeto de Pohl et al. (2018) compara o desempenho dos protocolos: *AMQP*, *MQTT* e *XMPP* no contexto de aplicações de monitoramento, no ambiente empresarial. Para para realização dos experimentos um banco de testes foi projetado com latência e taxa de perda de pacotes (*PLR*) variáveis. O *MQTT* tem melhor desempenho em relação às características de uso da largura de banda,confiabilidade, latência e taxa de transferência. Já o *AMQP* tem desempenho pior do que *MQTT* comparando o uso da largura de banda e a taxa de transferência. O protocolo *XMPP* apresenta os piores resultados em comparação com os demais. Além dos experimentos, os autores realizam um comparativo teórico entre os protocolos, considerando a seguintes características: segurança, escalabilidade, confiabilidade e flexibilidade.

O estudo de Prayogo, Mukhlis e Yakti (2019) propôs um esquema para transmitir um grande volume de dados de um publicador para um assinante, utilizando o dispositivo *IoT NodeMCU ESP8266*, que é um microcontrolador bastante conhecido. Para fim de comparações utilizou-se dois protocolos diferentes: o *MQTT* e o *CoAP*. O dispositivo *NodeMCU* pode atuar como publicador assim como assinante e também solicitar requisições para receber dados. Através da experimentação, observou-se as diferenças de tempo gasto, precisão e conveniência de uso entre *MQTT* e *CoAP*. O *MQTT* é mais preciso do que o *CoAP*, porém, o *CoAP* requer menos tempo para completar a transmissão. O uso de *MQTT* e *CoAP* também varia dependendo da finalidade e da forma de comunicação.

O artigo desenvolvido por Siddiqui et al. (2019) propõe usar a implementação de código aberto do *CoAP* e integrá-la com o *Datagram Transport Layer Security (DTLS)*, com a finalidade de criar um canal de transferência de dados seguro entre dispositivos *IoT*. Estudou-se o impacto da adição do *DTLS* ao *COAP*, através de experimentos utilizando dados reais, em dispositivos *IoT* com recursos limitados e software de código aberto. Os testes mostraram que utilizar uma implementação do *CoAP-DTLS* com um pacote de cifras de chave simétrica resultou em

custos de desempenho perceptíveis. Uma conexão segura com *DTLS* sobre *CoAP* consumia aproximadamente 10% mais energia do que uma conexão não segura. Além disso, os testes de latência revelaram um aumento de mais de 100% no tempo médio de latência para mensagens seguras em comparação com não seguras. Também foram encontrados alguns dos desafios de implementação ao desenvolver um ambiente de teste *IoT* real para experimentação segura.

Raza et al. (2017) implementaram o *CoAP* seguro, protegido por *DTLS*, para dispositivos *IoT* com restrição de recursos e um back-end na nuvem com a finalidade de avaliar os três modos de segurança do *CoAP*: chave pré-compartilhada, chave pública bruta e com base em certificado. Para este fim, utilizou-se uma configuração baseada em aplicações reais de *IoT* conectada à nuvem. Os autores estenderam o *SicsthSense*, uma plataforma em nuvem para a *IoT*, que utiliza o protocolo *CoAP* juntamente com uma implementação de *DTLS* para dispositivos *IoT* com recursos limitados. Os testes foram realizados utilizando criptografia simétrica e assimétrica. Para este trabalho os autores criaram a *SecureSense*, que é uma arquitetura de comunicação segura *End-to-End (E2E)* para a *IoT* e de código aberto, licenciada por *BSD*. Realizou-se um *benchmark* de avaliação que torna possível para fornecedores de serviços e produtos *IoT* contabilizar a sobrecarga de segurança ao usar os protocolos. As principais contribuições deste artigo são: implementação completa para modos de segurança *CoAP* para segurança *E2E IoT*; segurança e comunicação *IoT* protocolos para uma plataforma de nuvem para a *IoT*; avaliação experimental detalhada e *benchmarking* de segurança *E2E* entre uma rede de coisas inteligentes e uma plataforma em nuvem.

Protskaya e Veltri (2019) neste artigo, os autores apresentam um esquema para comunicação anonima dentro de redes *IoT* com base no protocolo *MQTT* usando o mecanismo de ponte dinâmica ou *dynamic bridging*. A solução proposta fornece anonimato para os clientes, criando um caminho através de um conjunto de *brokers* e solicitações de encapsulamento e criptografia de forma que nenhum dos participantes saiba qual cliente assina ou publica em determinado tópico. Uma variante do *MQTT*, o *MQTT-SN*, foi especificamente projetado para uso em ambiente de comunicação sem fio com suas principais limitações: altas taxas de falha, baixa largura de banda, curto comprimento do pacote. O *MQTT-SN* também é otimizado para implementação em dispositivos operados por bateria de baixo custo com recursos de processamento e armazenamento limitados. Suporte específico para hibernar os clientes através do envio de uma mensagem sem esperar qualquer confirmação.

A pesquisa de Jin, Kumar e Hong (2020) apresenta um *framework* de desenvolvimento de aplicativo chamado *Peekaboo*, que visa tornar mais fácil para os desenvolvedores obterem a granularidade dos dados que eles realmente precisam, em vez de sempre solicitar dados brutos, ao mesmo tempo que oferece suporte de arquitetura a recursos de privacidade aos aplicativos desenvolvidos. A filosofia do projeto arquitetônico do *Peekaboo* é o pré-processamento das tarefas de dados repetitivos, por exemplo, a detecção de rosto, extração de espectro de frequência. Esse pré-processamento ocorre do lado da nuvem em direção a um *hub* controlado pelo usuário e

suporte como um conjunto fixo de código aberto, reutilizável e operadores encadeados. Esses operadores pré-processam dados brutos para remover informações confidenciais desnecessárias do usuário antes do fluxo de dados para a nuvem (e fora do controle dos usuários), reduzindo assim os dados de saída e muitos riscos potenciais de privacidade para os usuários. Os aplicativos *IoT* desenvolvidos com o *Peekaboo* compartilham uma estrutura comum de operadores encadeáveis, tornando possível construir recursos de privacidade consistentes além de aplicativos individuais.

O projeto de Pannuto et al. (2018) propõe uma arquitetura feita sob medida para facilitar a criação de aplicativos *IoT* com maior segurança e privacidade. Essa arquitetura proposta fornece uma visão de acesso aos dispositivos através de portas específicas, encapsulando em tempo real o ponto de comunicação com cada dispositivo. A arquitetura também suporta a descoberta de dispositivos, interfaces compartilhadas entre dispositivos e uma especificação de aplicativo interface que promove a criação de aplicativos independentes de hardware capaz de operar mesmo quando os dispositivos mudam. Os autores fizeram alguns experimentos com a finalidade de provar a eficiência desta arquitetura.

3.3 Análise dos Trabalhos de Pesquisas Relacionados

No trabalho de Pohl et al. (2018) é feita uma comparação de desempenho entre alguns dos principais protocolos da camada de aplicação (*AMQP*,*MQTT* e *XMPP*) *IoT*. Os autores abordam um ambiente de automação em prédios empresariais, que muitas vezes se assemelham aos dispositivos utilizados em *smart homes*, porém, não levam em consideração requisitos não funcionais como segurança, privacidade e consumo de energia. Os autores sugerem como trabalho futuro fazer a mesma análise entretanto utilizando diferentes configurações de *QoS (Quality of Service)* e segurança.

Prayogo, Mukhlis e Yakti (2019) propôs uma comparação entre os protocolos: *MQTT* e *CoAP*, ambos da Camada de Aplicação *IoT*. Os autores utilizaram o *NodeMCU ESP8266*, que é um microcontrolador compatível com dispositivos *smart home*. Embora esta análise de desempenho seja interessante por comparar protocolos de arquiteturas diferentes, pois, um roda sobre *TCP* e o outro sobre *UDP*, eles não são considerados os mecanismos de proteção e sua sobrecarga sobre este tipo de comunicação. Devido a característica de cada protocolo, evidencia-se o uso mais adequado à situação em que serão utilizados.

O artigo desenvolvido por Siddiqui et al. (2019) propõe uma integração do *CoAP* com *DTLS* para prover um tunelamento criptografado por onde as mensagens devem ser trafegadas. Os experimentos demonstraram uma sobrecarga na comunicação ao utilizar o *DTLS*, o que resulta em uma degradação de performance e aumento do consumo energético. Embora o trabalho seja de grande relevância, os autores não abordam a questão da privacidade em ambientes *smart home*. Os experimentos realizados utilizaram a técnica de *Pre-Shared Key (PSK)* ou chaves pré-compartilhadas em um hardware bastante limitado chamado *SensorTag CC2650*, entretanto

em ambientes *smart home* é possível utilizar técnicas mais seguras, no que tange a privacidade, e hardwares com maior poder computacional, de baixo custo e consumo energético relativamente baixo.

Raza et al. (2017) implementaram o *CoAP* seguro, protegido por *DTLS*, e analisaram as três possíveis configurações de segurança do *CoAP*. O trabalho demonstra a sobrecarga da utilização de um protocolo de transporte, o *DTLS*, porém limita-se somente a esta solução, não analisando outro protocolo e desconsiderando a usabilidade.

Protskaya e Veltri (2019) apresentam uma forma de comunicação anônima em redes *IoT* utilizando o protocolo *MQT* com o mecanismo de ponte dinâmica. Esta solução é pertinente no tocante à privacidade, uma vez que consiste em um roteamento do tipo *onion* entre os *brokers*. Todavia não foi realizado um estudo elaborado sobre o impacto na performance, assim como na usabilidade do esquema proposto. Ainda que forneça uma contribuição bastante significativa no contexto de redes *IoT*, para que fosse aplicado em um ambiente *smart home*, seria necessário um estudo mais aprofundado sobre os custos da aquisição de diversos dispositivos e consumo energético destes.

A pesquisa de Jin, Kumar e Hong (2020) apresenta um *framework* de desenvolvimento de aplicações chamado *Peekaboo* e utilizado para rodar em *hub* de dispositivos *smart home*. O objetivo do algoritmo é pré-processar dados brutos e retirar informações que possam comprometer a privacidade dos usuários. Esta proposta almeja uma garantia de privacidade, contudo, não é apresentado uma análise detalhada do impacto gerado à comunicação. O *framework* incrementa uma complexidade considerável ao usuário de *smart home* e não explicita dados sobre o consumo energético.

O projeto de Pannuto et al. (2018) propõe uma arquitetura com uma camada de encapsulamento que abstrai a descoberta de novos dispositivos e o acesso a estes através de portas. Essa proposta pode ser promissora no tocante a orquestração de dispositivos, como também em padronizar a forma com que eles são acessados. Porém, os autores esclarecem que é preciso evoluir bastante no que se refere a autenticação, autorização, desempenho e consumo energético. Uma análise de desempenho aprofundada poderia evidenciar se este mecanismo seria viável ao ambiente de redes *IoT* no contexto de *smart home*.

Os trabalhos supracitados foram sumarizados e podem ser visualizados na Tabela 2. Como pode ser observado no texto acima, nenhum dos trabalhos prover uma análise completa sobre a privacidade na camada de aplicação *IoT*. Pohl et al. (2018) não consideram os principais requisitos de *QoS*. Em Prayogo, Mukhlis e Yakti (2019), os autores não analisam os custos com o consumo de energia. Já nos trabalhos de Siddiqui et al. (2019) e Raza et al. (2017), a eficiência energética é considerada, todavia, limitam-se a junção dos protocolo *CoAP + DTLS* sem compará-los com outros protocolos da Camada de Aplicação. Protskaya e Veltri (2019) e Jin, Kumar e Hong (2020) não realizaram um estudo sobre o impacto na performance e nem na usabilidade. Contudo, Pannuto et al. (2018) discorre sobre usabilidade, mas não aprofunda os

demais requisitos não funcionais e essenciais em ambientes *smart home*.

Tabela 2 – Comparação entre os trabalhos relacionados.

Autor (Ano)	**Desempenho**	**Con. energético**	**Segurança**	**Privacidade**	**Usabilidade**	***Smart Home***
(POHL et al., 2018)	X					
(PRAYOGO; MUKHLIS; YAKTI, 2019)	X					X
(SIDDIQUI et al., 2019)	X	X	X	X		X
(RAZA et al., 2017)	X	X	X	X		
(PROTSKAYA; VELTRI, 2019)			X	X		X
(JIN; KUMAR; HONG, 2020)			X	X		X
(PANNUTO et al., 2018)			X	X	X	X
Este Trabalho	X	X	X	X	X	X

Diferente dos trabalhos apresentados, este trabalho vislumbra uma avaliação de viabilidade em agregar mecanismos de proteção à privacidade sobre a Camada de Aplicação. Para isso, pretende-se analisar se esses mecanismos degradam a performance, como por exemplo o tempo de resposta e a perda de pacotes; se aumentam o consumo energético ou criam complexidade suficiente para inviabilizar soluções *smart home* que sejam seguras no que se refere à privacidade.

4

Protocolos, ameaças e mecanismos de segurança

Nesta seção, será abordado o conteúdo referente aos protocolos de rede utilizados para o estudo da viabilidade em adicionar proteção à privacidade na transferência de dados em redes *IoT*. Este trabalho se propôs à analisar dois dos protocolos mais utilizados nesse tipo de rede, o *CoAP* e o *MQTT*. O principal mecanismo de segurança para esses protocolos é adicionar uma camada de criptografia, tornando assim o tráfego dos dados privativo a quem tem as chaves para visualizá-los. Para que isto aconteça, no protocolo *CoAP* é preciso adicionar uma camada de criptografia utilizando o protocolo *DTLS*, dado que os pacotes *CoAP* trafegam sobre o protocolo de *UDP*. Já no caso do *MQTT*, o processo é o mesmo porém utilizando *TLS*, uma vez que trafega sobre o protocolo de transporte *TCP*.

4.1 *MQTT*

No final da década de 1990, Andy Stanford-Clark *(IBM)* e Arlen Nipper *(Cirrus Link)* inventaram o *MQTT* para monitorar oleodutos e gasodutos em redes de satélite. Eles projetaram este protocolo para ser aberto, simples e fácil de implementar. O resultado foi um protocolo extremamente leve, que minimiza a largura de banda da rede e os requisitos de recursos do dispositivo com alguma garantia de entrega confiável. O design permitiu que milhares de pequenos dispositivos fossem suportados a partir de um único servidor. Essas características tornam o *MQTT* ideal para uso em ambientes restritos e redes de baixa largura de banda, que possuem capacidade de processamento limitada, baixa capacidade de memória e alta latência, como a Internet das Coisas (HIVEMQ, 2022).

O protocolo *MQTT* tem se tornado um protocolo padrão para troca de mensagens de *IoT*. Padronizado pelo *OASIS* e *ISO*, o protocolo de publicação/assinatura fornece uma maneira escalável e confiável de conectar dispositivos pela Internet. Dentre as principais características do protocolo estão:

- Clientes *MQTT*: os clientes *MQTT* publicam uma mensagem para um *broker MQTT* e outros clientes assinam as mensagens que desejam receber. As implementações de clientes geralmente exigem um espaço mínimo, portanto, são adequadas para implantação em dispositivos restritos e são muito eficientes em seus requisitos de largura de banda.

- *Brokers*: os *brokers MQTT* recebem mensagens publicadas e despacham a mensagem para os clientes assinantes. Uma mensagem contém um tópico de mensagem que os clientes assinam e os *brokers* usam essas listas de assinaturas para determinar quais clientes irão receber as mensagens.

- *QoS*: o *MQTT* implementa 3 níveis de qualidade de serviço para acordo entre o emissor e o receptor: no máximo uma vez (nível 0), pelo menos uma vez (nível 1) e exatamente uma vez (nível 2). Esses níveis de *QoS* permitem aplicativos de *IoT* se tornem mais confiáveis. A infraestrutura de mensagens subjacente e permite que eles se adaptem a condições de redes não confiáveis.

- Persistência de sessão: o *MQTT* permite uma sessão persistente entre o cliente e o *broker*. Isso faz com que as sessões persistam mesmo se a rede estiver desconectada. Depois que a rede for reconectada, as informações para reconectar o cliente ao *broker* ainda existirão. Esse é um dos principais recursos que torna o protocolo *MQTT* mais eficiente que o *HTTP* para uso em redes não confiáveis.

- Mensagem retida: os clientes *MQTT* que assinam um novo tópico não têm informações sobre quando esperar a primeira mensagem que receberão. No entanto, um *broker* pode armazenar uma mensagem retida que pode ser enviada imediatamente após uma nova assinatura. Nesse caso, o cliente receberá pelo menos uma mensagem ao assinar o tópico.

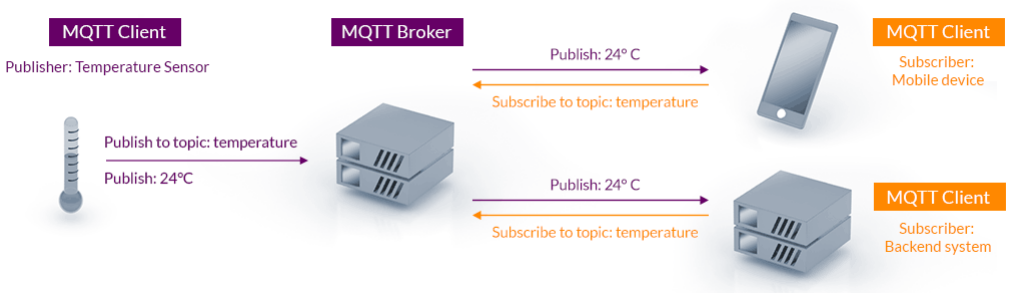
- Testamento: um cliente *MQTT* pode especificar para um *broker* uma mensagem, chamada de última vontade, que será enviada se o cliente *MQTT* se desconectar de forma inadequada. Isso permite a notificação de que um cliente foi desconectado seja mais elegante em todo o sistema.

4.1.1 Modelo de publicação e assinatura

O modelo de publicação e assinatura, também conhecido como pub/sub, é um padrão proposto para ser uma alternativa ao tradicional padrão da arquitetura cliente-servidor. No modelo cliente-servidor, um cliente se comunica diretamente com um *endpoint*. Já o padrão pub/sub, desacopla o cliente, ou seja, ele em momentos envia uma mensagem como publicador e em outros momentos recebe-as como assinantes. Os editores e assinantes nunca se contatam diretamente. Na verdade, eles nem sabem que o outro existe, pois, a conexão entre eles é feita por um terceiro componente (o *broker*). O trabalho do *broker* é filtrar todas as mensagens recebidas e distribuí-las corretamente aos assinantes. A figura Figura 5 demonstra a comunicação entre um sensor e seus

clientes através de um *broker* (MQTT, 2022). Um sensor de temperatura envia uma mensagem com o valor da temperatura lido para um *broker* através do comando *publish* em um tópico chamado *temperature*. O *broker* por sua vez repassa esta mensagem para todos os cliente clientes que assinam o tópico com o mesmo nome.

Figura 5 – Abstração de camadas do protocolo *CoAP*



Fonte: MQTT (2022)

O aspecto mais importante do pub/sub é a dissociação entre publicador da mensagem e o destinatário (assinante). Este desacoplamento tem várias dimensões:

- Desacoplamento de espaço: publicador e assinante não precisam se conhecer (por exemplo, não há troca de endereço *IP* e porta);
- Desacoplamento de tempo: o editor e o assinante não precisam ser executados ao mesmo tempo;
- Desacoplamento de sincronização: as operações em ambos os componentes não precisam ser interrompidas durante a publicação ou recebimento.

Em resumo, o modelo pub/sub remove a comunicação direta entre o publicador da mensagem e o destinatário/assinante. Além disso, a atividade de filtragem do *broker* permite controlar qual cliente/assinante recebe qual mensagem. O desacoplamento tem três dimensões: espaço, tempo e sincronização.

4.1.2 *MQTT QoS*

O nível de Qualidade de Serviço ou *QoS* é um acordo entre o remetente e o destinatário de uma mensagem que define a garantia de entrega desta mensagem entre eles. Quando se fala em *QoS* no *MQTT*, é necessário considerar os dois lados da entrega de mensagens:

- Entrega de mensagens do cliente de publicação para o *broker*;
- Entrega de mensagens do *broker* para o cliente assinante.

O cliente que publica a mensagem no *broker* define o nível de *QoS* da mensagem quando envia a mensagem ao *broker*. Este por sua vez transmite essa mensagem para clientes assinantes usando o nível de *QoS* que cada cliente assinante define durante o processo de assinatura. Se o cliente assinante definir uma *QoS* inferior à do cliente de publicação, o *broker* transmitirá a mensagem com a qualidade de serviço mais baixa.

QoS é um recurso chave do protocolo *MQTT*, pois, dá ao cliente o poder de escolher um nível de serviço que corresponda à confiabilidade da rede e à lógica do aplicativo. Assim, como o *MQTT* gerencia a retransmissão de mensagens e garante a entrega, mesmo quando o transporte subjacente não é confiável, o *QoS* ajuda a facilitar a comunicação em redes não confiáveis. Segundo HiveMQ (2022), existem 3 níveis de *QoS* no *MQTT*:

- Nível 0 - este nível é o menor deles, garante uma entrega com o melhor esforço e por isso, não há garantia de entrega. O destinatário não confirma o recebimento da mensagem e a mensagem não é armazenada e retransmitida pelo remetente. Este nível é frequentemente chamado de "disparar e esquecer", e fornece a mesma garantia que o protocolo *TCP* subjacente.

- Nível 1 - garante que uma mensagem seja entregue pelo menos uma vez ao receptor. O remetente armazena a mensagem até receber um pacote *PUBACK* do destinatário, que reconhece o recebimento da mensagem. É possível que uma mensagem seja enviada ou entregue várias vezes. O remetente utiliza o identificador de pacote em cada pacote para corresponder o pacote *PUBLISH* ao pacote *PUBACK* correspondente. Se o remetente não receber um pacote *PUBACK* em um período de tempo razoável, o remetente reenvia o pacote *PUBLISH*. Quando um receptor recebe uma mensagem com *QoS* 1, ele pode processá-la imediatamente. Por exemplo, se o receptor for um *broker*, o *broker* envia a mensagem para todos os clientes assinantes e então responde com um pacote *PUBACK*. Se o cliente de publicação enviar a mensagem novamente, ele definirá um sinalizador duplicado *(DUP)*. No *QoS* 1, esse sinalizador *DUP* é usado apenas para fins internos e não é processado pelo *broke*r ou cliente. O receptor da mensagem envia um *PUBACK*, independente da *flag DUP*.

- Nível 2 - o mais alto em *MQTT*. Este nível garante que cada mensagem seja recebida apenas uma vez pelos destinatários pretendidos. *QoS 2* é o nível de qualidade de serviço mais seguro e mais lento. Dessa forma, a garantia é fornecida por pelo menos dois fluxos de solicitação/resposta, que é um *handshake* de quatro partes, entre o remetente e o destinatário. O remetente e o destinatário utilizam o identificador de pacote da mensagem *PUBLISH* original para coordenar a entrega da mensagem. Quando um receptor recebe um pacote *QoS 2 PUBLISH* de um remetente, ele processa a mensagem de publicação de acordo e responde ao remetente com um pacote *PUBREC* que reconhece o pacote *PUBLISH*. Se o remetente não receber um pacote *PUBREC* do destinatário, ele enviará o pacote *PUBLISH*

novamente com um sinalizador duplicado *(DUP)* até receber uma confirmação. Uma vez que o remetente recebe um pacote *PUBREC* do destinatário, o remetente pode descartar com segurança o pacote *PUBLISH* inicial. O remetente armazena o pacote *PUBREC* do receptor e responde com um pacote *PUBREL*. Após o receptor receber o pacote *PUBREL*, ele pode descartar todos os estados armazenados e responder com um pacote *PUBCOMP* (o mesmo ocorre quando o remetente recebe o *PUBCOMP*). Até que o receptor conclua o processamento e envie o pacote *PUBLISH* de volta ao remetente, o receptor armazena uma referência ao identificador de pacote do pacote *PUBLISH* original. Esta etapa é importante para evitar o processamento da mensagem uma segunda vez. Depois que o remetente recebe o pacote *PUBCOMP*, o identificador do pacote da mensagem publicada fica disponível para reutilização. Quando o fluxo de *QoS 2* é concluído, ambas as partes têm certeza de que a mensagem foi entregue e o remetente tem a confirmação da entrega. Se um pacote se perder ao longo do caminho, o remetente é responsável por retransmitir a mensagem dentro de um período de tempo razoável. Isso é igualmente verdadeiro se o remetente for um cliente *MQTT* ou um *broker MQTT*. O destinatário tem a responsabilidade de responder a cada mensagem de comando de acordo.

4.1.3 Segurança *MQTT*

A segurança no *MQTT* é dividida em várias camadas e cada uma delas previne diferentes tipos de ataques. Para proteção à privacidade o próprio protocolo adota padrões de segurança de última geração: por exemplo, *TLS/SSL* o transporte seguro. Considerando que prover a transferência de dados de forma segura é uma tarefa bastante complexa, faz sentido utilizar padrões geralmente aceitos.

Quando a confidencialidade é o objetivo principal, o *TLS/SSL* é comumente utilizado para criptografia de transporte. Esse método é uma maneira segura e comprovada de garantir que os dados não possam ser lidos durante a transmissão e fornece autenticação de certificado de cliente para verificar a identidade de ambos os lados.

O *TLS* e o *SSL* fornecem um canal de comunicação seguro entre um cliente e um servidor. Em suas essências, eles são protocolos criptográficos que usam um mecanismo de *handshake* para negociar vários parâmetros e criar uma conexão segura entre o cliente e o servidor. Após a conclusão do *handshake*, uma comunicação criptografada entre cliente e servidor é estabelecida e nenhum invasor pode espionar qualquer parte da comunicação. Os servidores fornecem um certificado *X509*, que normalmente emitido por uma autoridade confiável e que os clientes utilizam para verificar a identidade do servidor.

O *MQTT* depende do protocolo de transporte *TCP*. Por padrão, as conexões *TCP* não usam uma comunicação criptografada. Para criptografar toda a comunicação *MQTT*, muitos *brokers*, como por exemplo o *Mosquitto*, permitem adicionar uma camada *TLS* ao *TCP* simples. A porta 8883 é padronizada para uma conexão *MQTT* segura. O nome padronizado na *Internet*

Assigned Numbers Authority (IANA) é *"secure-mqtt"*.

Há uma desvantagem em usar *MQTT* sobre *TLS*: a segurança tem um custo em termos de uso da *CPU* e sobrecarga de comunicação. Embora o uso adicional da *CPU* seja normalmente insignificante no *broker*, pode ser um problema para dispositivos muito restritos e que não são projetados para tarefas de computação intensiva. Técnicas como a retomada da sessão podem melhorar drasticamente o desempenho do *TLS*.

A sobrecarga de comunicação do *handshake TLS* pode ser significativa se as conexões do cliente *MQTT* forem de curta duração. Embora a quantidade de largura de banda necessária para este *handshake* dependa de muitos fatores, algumas medições mostram que estabelecer uma nova conexão *TLS* pode exigir até alguns *kilobytes* de largura de banda. Como cada pacote é criptografado, produz-se uma sobrecarga adicional em comparação com os pacotes não criptografados.

Ao usar conexões *TCP* de longa duração com *MQTT*, o que é recomendado, via de regra gera um custo insignificante. Em um cenário que lida com muitas reconexões e não pode usar a retomada da sessão, a sobrecarga pode ser significativa, especialmente se estiver publicando mensagens muito pequenas. Se cada byte conta para o caso de uso, o *TLS* pode não ser a melhor opção. Este trabalho mede essa sobrecarga para o cenário de uma comunicação em uma *smart home*.

4.2 *CoAP*

O *CoAP* é um protocolo de transferência de dados na Web, especializado para uso com nós e redes restritas na Internet das Coisas. Foi criado pelo *Constrained RESTful Environments (CoRE)* do *Internet Engineering Task Force (IETF)* e projetado para aplicações *M2M* (máquina para máquina). Ele fornece um modelo de interação de solicitação/resposta entre *endpoints* de aplicativos, oferece suporte à descoberta integrada de serviços e recursos que incluem os conceitos-chave da Web, como *URIs* e tipos de mídia da Internet. É projetado para interagir facilmente com *HTTP*, a fim de prover integração com a Web ao mesmo tempo em que atende a requisitos especializados para ambientes restritos como: suporte *multicast*, sobrecarga muito baixa e simplicidade (SHELBY; HARTKE; BORMANN, 2014).

Aprovado em 11/07/2013 como *draft-ietf-core-coap-18*, foi publicado como *RFC* 7252 em 26/06/2014. O atraso foi causado pela espera de algumas especificações de segurança relacionadas para concluir seu processo de publicação. O *CoAP* foi projetado para ser extensível, assim, vários registros são definidos para permitir a evolução do protocolo ao longo do tempo. Isso inclui códigos de resposta como o 404 (Não encontrado) e valores de formato de conteúdo para indicar tipos de mídia, como o código 50 para aplicativo/*json*. Eles são coletados em um registro da *IANA* para garantir que as especificações estejam disponíveis e que não sejam confundidas com a de outros protocolos (SHELBY; HARTKE; BORMANN, 2014).

Um objetivo de design do *CoAP* tem sido manter sobrecarga de mensagem pequena, limitando assim a necessidade de fragmentação. Ele foi desenvolvido para ser um protocolo genérico da Web para os requisitos especiais em ambiente restrito, especialmente considerando baixo consumo de energia em automação predial e outras aplicações *M2M*. O objetivo não é compactar indiscriminadamente o *HTTP*, mas sim para realizar um subconjunto *REST* comum com *HTTP*, porém otimizado para aplicações *M2M*. Embora o *CoAP* possa ser usado como um *HTTP* simplificado, é um protocolo mais compacto e também oferece recursos úteis para *IoT* como descoberta de dispositivos, suporte a *multicast* e trocas de mensagens assíncronas.

Segundo Shelby, Hartke e Bormann (2014), as principais características do protocolo são:

- Modelo *REST* para dispositivos restritos - assim como o *HTTP*, o *CoAP* é baseado no modelo *REST* largamente adotado. Assim, os servidores disponibilizam recursos em uma *URL* e os clientes acessam esses recursos usando métodos como *GET*, *PUT*, *POST* e *DELETE*.
- Feito para bilhões de clientes - a Internet das Coisas precisará de bilhões de nós e o *CoAP* foi projetado para funcionar em microcontroladores com apenas 10 *Kb* de *RAM* e 100 *Kb* de espaço de código.
- Protocolo padronizado - o *CoAP* foi desenvolvido como um documento de padrões da Internet, a *RFC 7252*. Questões difíceis, como controle de congestionamento foram tratadas usando as técnicas mais avançadas.
- Fácil de ser utilizado - do ponto de vista do desenvolvedor, o *CoAP* se parece muito com o *HTTP*. Obter um valor de um sensor não é muito diferente de obter um valor de uma *API* da web.
- Evita desperdícios - foi projetado para usar o mínimo de recursos, tanto no dispositivo quanto na rede. Em vez de uma pilha de transporte complexa, ele funciona com *UDP* no *IP*. Um cabeçalho fixo de 4 bytes e uma codificação compacta de opções permitem pequenas mensagens que causam pouca ou nenhuma fragmentação na camada de link. Muitos servidores podem operar de forma completamente sem estado.
- Seguro - a escolha padrão dos parâmetros *DTLS* do *CoAP* é equivalente a chaves *RSA* de 3072 *bits*, mas ainda funciona bem nos menores nós.
- Integrável com *HTTP* - como o *HTTP* e o *CoAP* compartilham o modelo *REST*, eles podem ser facilmente conectados usando *proxies* de protocolo cruzado agnósticos de aplicativos. Um cliente Web pode acessar dados de um sensor de forma transparente, por exemplo.
- Descoberta de dispositivos integrada - o diretório de recursos *CoAP* fornece uma maneira de descobrir as propriedades dos nós em sua rede.

- Diferentes tipos de *payload* - assim como o *HTTP*, o *CoAP* pode transportar diferentes tipos de carga útil e pode identificar qual tipo de carga útil está sendo usado. O *CoAP* integra-se com *XML*, *JSON*, *CBOR* ou qualquer outra formatação de dados.

4.2.1 Camada de Mensagem *CoAP*

O modelo de interação do protocolo *CoAP* é semelhante ao modelo cliente/servidor de *HTTP*. No entanto, as interações máquina a máquina normalmente resultam em uma implementação *CoAP*, atuando em funções de cliente e servidor. A solicitação *CoAP* é equivalente à de *HTTP* e é enviada por um cliente para solicitar uma ação, usando um código de método, em um recurso, que é identificado por um URI, em um servidor. O servidor então envia uma resposta com um código de resposta e pode também incluir uma representação de recursos.

Ao contrário do *HTTP*, o *CoAP* lida com essas trocas de forma assíncrona em um transporte orientado a *datagramas*, como *UDP*. Isso é feito logicamente usando uma Camada de Mensagens que suporta adição de confiabilidade de forma opcional. Assim, o *CoAP* define quatro tipos de mensagens:

- *Confirmable* - as mensagens que requerem confirmação são chamadas de "confirmáveis". Dessa maneira, quando nenhum pacote é perdido durante o tráfego, cada mensagem confirmável solicita exatamente uma mensagem de retorno do tipo (*Acnowledgement (Ack*) ou do tipo *Reset*.

- *Non-confirmable* - diferente das confirmáveis algumas mensagens não exigem um confirmação do tipo *Acnkowledgement*, isso acontece geralmente em aplicações onde as mensagens são frequentemente repetidas, como por exemplo, o envio de leitura de um sensor.

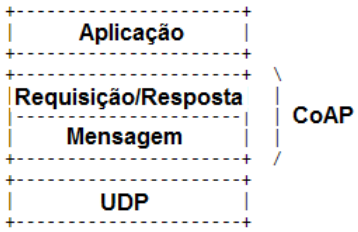
- *Acknowledgment* - este tipo de mensagem é a confirmação do recebimento de uma mensagem do tipo *Confirmable*. Ela não indica se uma operação foi realizada com sucesso ou não, apenas informa que a mensagem foi recebida. Por questão de otimização, este tipo de mensagem também pode carregar a resposta de alguma solicitação, quando isso acontece é denominado de resposta *Piggybacked*.

- *Reset* - indica que uma mensagem (*Confirmable* ou *Non-confirmable*) foi recebida, mas ainda falta algum contexto para ser processado. Isso geralmente acontece quando o nó que está recebendo a mensagem foi reiniciado e perdeu algum "estado"necessário para interpretar a mensagem. É possível provocar uma mensagem de *reset* enviando uma mensagem do tipo confirmável vazia. É útil também para verificar a disponibilidade de um *endpoint CoAP* funcionando como se fosse uma mensagem de *ping*

Pode-se pensar no *CoAP* utilizando uma abordagem de duas camadas. Uma camada de mensagens usada para lidar com *UDP* e suas interações assíncronas, e outra de solicitação/resposta/

utilizando códigos de método e de resposta. No entanto, ele é um protocolo único. A Figura 6 demonstra uma divisão lógica dessas camadas.

Figura 6 – Abstração de camadas do protocolo CoAP

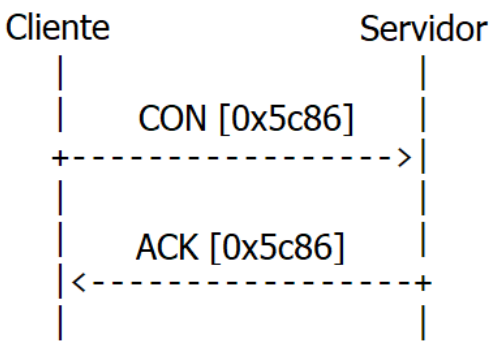


Fonte: Própria

O *CoAP* usa um cabeçalho binário de comprimento fixo curto (4 bytes), que pode ser seguido por opções binárias compactas e uma carga útil. Este formato de mensagem é compartilhado tanto para solicitações como para respostas. Cada mensagem contém um identificador (*ID*), usado para detectar duplicatas e outro para definir o nível de confiabilidade, que é opcional. O *ID* da mensagem é compacto, seu tamanho de 16 *bits* permite até cerca de 250 mensagens por segundo de um *endpoint* para outro com a configuração padrão do parâmetros do protocolo.

A confiabilidade é fornecida marcando uma mensagem como confirmável ou *Confirmable (CON)*. Essa mensagem é retransmitida utilizando um tempo limite padrão e recuo exponencial entre as retransmissões, até que o destinatário envie uma mensagem de confirmação *ACK* com o mesmo *ID* de mensagem. Quando um destinatário não consegue processar uma mensagem confirmável, ou seja, nem mesmo é capaz de fornecer uma resposta de erro adequada, responde com uma mensagem de redefinição *RST* em vez de uma confirmação *ACK*. Na Figura 7, é possível visualizar uma transmissão de uma mensagem do tipo confirmável.

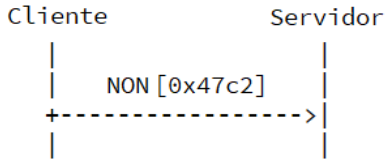
Figura 7 – Mensagem confirmável CoAP



Fonte: Própria

Uma mensagem que não requer transmissão confiável pode ser enviada como uma mensagem não-confirmável ou *NON*, por exemplo, a mensagem de cada medição única de um fluxo de dados do sensor. Elas não são confirmadas, mas ainda têm um *ID* de mensagem para detecção de mensagens duplicadas, o qual pode ser visto na Figura 8. Quando um destinatário não consegue processar uma mensagem não-confirmável, ele pode responder com uma mensagem de *reset RST*. Assim, como o *CoAP* é executado sobre o *UDP*, ele também suporta o uso de *multicast*.

Figura 8 – Mensagem não confirmável CoAP

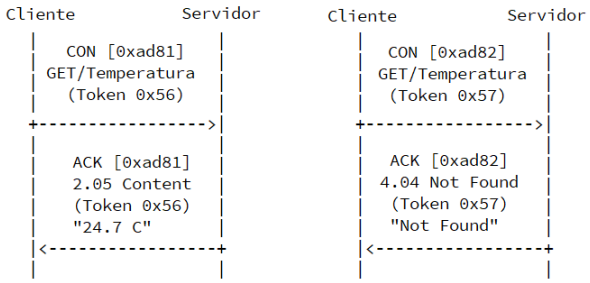


Fonte: Própria

4.2.2 Camada de Requisições e Respostas

A semântica de requisição e resposta *CoAP* é transportada em mensagens que incluem os códigos de métodos, por exemplo os métodos *GET* e *POST*, ou códigos de respostas, como os 200 e 404. Informações opcionais como a *URI* e o tipo de mídia de carga útil são transportados como opções. Um *token* é incluído para realizar a correspondência entre solicitações e respostas, independente de mensagens subjacentes, porém, o conceito de *token* é diferente do identificador da mensagem. Dois exemplos para uma solicitação básica utilizando o método *GET* com resposta *piggybacked* são mostrados na Figura 9, um bem sucedido, e um resultando em uma resposta 4.04 (não encontrado).

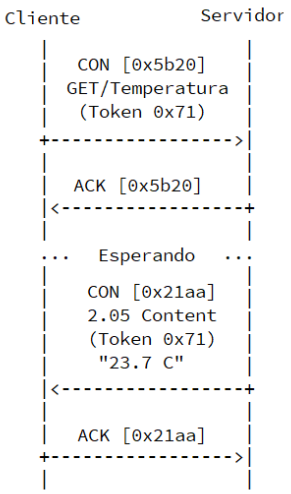
Figura 9 – Solicitação com resposta Piggybacked



Fonte: Própria

Se o servidor não puder responder imediatamente a uma solicitação realizada com uma mensagem *CON*, ele simplesmente responde com uma mensagem vazia (*Empty Message*) de confirmação para que o cliente possa parar de retransmitir o pedido. Quando a resposta estiver pronta, o servidor a envia em um nova mensagem confirmável, que por sua vez precisa ser reconhecida pelo cliente. Isso é chamado de "resposta separada"e a Figura 10 ilustra esta situação.

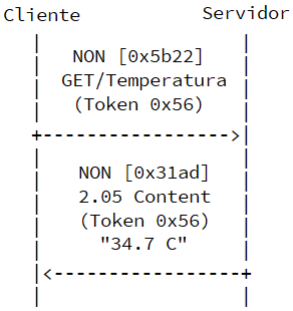
Figura 10 – Solicitação com resposta Separada



Fonte: Própria

Se uma solicitação for enviada em uma mensagem não-confirmável, a resposta é enviada usando uma nova mensagem não-confirmável, entretanto o servidor pode, ao invés disso, enviar uma mensagem confirmável. Este tipo de troca é ilustrada na Figura 11

Figura 11 – Solicitação NON com resposta CON



Fonte: Própria

Embora o *CoAP* utilize os métodos *GET*, *PUT*, *POST* e *DELETE* de forma semelhante ao *HTTP*, porém não são exatamente iguais, algumas diferenças podem ser consultadas em detalhes

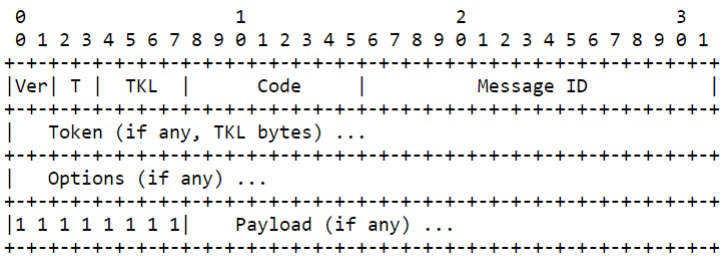
na seção 5.8 da *RFC 7252* (SHELBY; HARTKE; BORMANN, 2014).

4.2.3 Formato da Mensagem *CoAP*

O *CoAP* é baseado na troca de mensagens compactas que por padrão são transportados sobre *UDP*, ou seja, cada mensagem ocupa a seção de dados de um datagrama. Mas também, pode ser utilizado sobre um datagrama criptografado, no caso do *DTLS*. Assim como poderia ser utilizado sobre outros protocolos de transporte, tais como: *SMS*, *TCP* ou *SCTP*, todavia não fazem parte do escopo deste trabalho.

As mensagens são codificadas em um formato binário simples. O modelo da mensagem começa com um cabeçalho de 4 bytes de tamanho fixo. Em seguida, por um valor de *token* de comprimento variável, que pode ter entre 0 e 8 bytes de comprimento. Uma sequência de zeros ou opções sucede o *token*, podendo ainda conter um *payload* no resto do datagrama. Dessa maneira, o modelo do cabeçalho *CoAP* pode ser visto na Figura 12.

Figura 12 – Cabeçalho *CoAP*



Fonte: (SHELBY; HARTKE; BORMANN, 2014)

Os principiais campos que compõem o formato do cabeçalho de uma requisição *CoAP* saõ:

- *Ver* - este campo é representado por um inteiro sem sinal de 2 bits. Indica a versão do *CoAP*. As implementações desta especificação devem definir este campo para 1 (01 binário). Outros valores são reservados para versões futuras. Mensagens com números de versão desconhecidos devem ser ignoradas silenciosamente.
- *T* - indica o tipo da mensagem, se ela é *CON* (0), *NON* (1), *ACK* (2) ou *RST* (3)
- *TKL* - este campo é um inteiro sem sinal de 4 bits e indica o tamanho do *token*.
- *Code* - os 3 bits mais significativos indicam se é uma solicitação (0), uma resposta bem sucedida (2), erro de resposta do cliente (4), ou um erro do servidor (5) e os demais são reservados.

- *Message ID* - é o identificador da mensagem, ou seja, campo utilizado para detectar duplicatas ou correspondências entre mensagens *ACK/RST* e *CON/NON*.

4.2.4 Segurança no protocolo *CoAP*

A especificação *RFC 7252* define quatro modos de segurança que podem ser configurados em um tráfego de mensagens utilizando o protocolo *CoAP*. Durante a fase de *handshake* o dispositivo solicitante fornece as informações de segurança de que necessita, incluindo informações de chave e listas de controle de acesso. Ao final desta fase os dispositivos cliente e servidor terão acordado um dos quatro modos de segurança possíveis para então continuar a comunicação seguindo a regra estabelecida. Os quatro modos de segurança especificados são:

- *NoSec* ou sem segurança - a troca de mensagens é realizada em texto pleno e sem nenhum tipo de segurança estabelecido, ou seja, com o *DTLS* desabilitado.
- *PreSharedKey* ou chave compartilhada - neste modo o *DTLS* é habilitado e exite uma lista de chaves previamente compartilhadas. Quando uma chave é escolhida ela serve para comunicação entre apenas dois nós, em uma relação de 1 para 1.
- *RawPublicKey* ou chave pública pura - neste modo o *DTLS* está habilitado e o dispositivo tem um par de chaves assimétricas sem um certificado (uma chave pública bruta). O dispositivo possui uma identidade calculada a partir da chave pública e uma lista de identidades dos nós que ela pode se comunicar.
- *Certificate* ou com certificado - neste modo o *DTLS* está habilitado e o dispositivo possui um par de chaves assimétricas com certificado *X.509* assinado por uma entidade autenticadora de confiança. O dispositivo também possui uma lista de cadeia de confiança que pode ser usada para validar o certificado.

Os três modos de segurança são obtidos utilizando o *DTLS* e são utilizados através de uma *URL* que inicia com *"coaps"* e pela porta padrão *CoAP* protegida por *DTLS*. O resultado é uma associação de segurança que pode ser usada para autenticar, dentro dos limites do modelo de segurança, e autorizar a comunicação segura entre os nós. O *CoAP* não fornece, intrinsecamente, uma comunicação segura, dependendo portanto da utilização de camadas como *DTLS*, *IPSec* ou até mesmo mecanismos de criptografia nos *payloads* das mensagens. Esta pesquisa comparou o *overhead* gerado ao utilizar o modo seguro com certificado (*Certificate Mode*), quando comparado ao modo sem segurança (*NoSec Mode*).

5
Metodologia

Para demonstrar o experimento, se faz necessário apresentar o ambiente no qual os testes foram realizados. Nesta seção serão abordados: hardware, bibliotecas e *firmwares*, assim como as configurações necessárias para realizar a experimentação.

5.1 Hardware

Foram utilizados dois *NodeMCUs ESP32* para executar os códigos clientes, um para o protocolo *MQTT* e outro para o *CoAP*, como também foram acrescidas as proteções de *TLS* e *DTLS* respectivamente. Assim, o hardware utilizado como cliente tinha o objetivo de realizar uma coleta de medições de tempo de resposta. Visando a não interferência no processamento do lado do cliente e consequentemente no tempo de resposta das requisições, a medição de energia foi realizada utilizando um *Arduino Nano V3*. As aplicações do lado do servidor foram executadas em um *Raspberry Pi 3 model B* ligado por cabo a um roteador *mesh Intelbras Twibi Giga*. Os microcontroladores *ESP32* conectaram-se ao mesmo roteador através de conexão wi-fi.

5.1.1 *Arduino Nano V3*

De acordo com Nano (2021), o *Arduino Nano* é uma placa de desenvolvimento e prototipação com o microcontrolador *ATmega328* embarcado. O *Arduino Nano* trabalha com a velocidade de *clock* em 16 *MHz*, possui um consumo de energia de 19 mAh e pode ser adquirido por menos dois dólares. Além disso, pode ser alimentado por meio da conexão *USB Mini-B*, fonte de alimentação externa não regulada de 6-20 V (pino 30) ou fonte de alimentação externa regulada de 5 V (pino 27). A fonte de alimentação é selecionada automaticamente para a fonte de tensão mais alta. O microcontrolador *ATmega328* possui 32 *KB* de memória interna, os quais 2 *KB* são utilizados para o *bootloader*. Ele também possui 2 *KB* de *SRAM* e 1 *KB* de *EEPROM*.

Cada um dos 14 pinos digitais no *Nano* pode ser usado como entrada ou saída, usando as

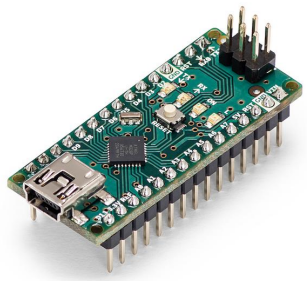
funções *pinMode*, *digitalWrite* e *digitalRead*. Cada pino opera a 5V, pode fornecer ou receber no máximo 40 mA e tem um resistor *pull-up* interno, que é desconectado por padrão, de 20-50 kOhms. Além disso, o *Nano* possui 8 entradas analógicas e cada uma fornece 10 *bits* de resolução, ou seja, 1024 valores diferentes. Por padrão, eles medem do aterramento a 5V, embora seja possível alterar a extremidade superior de sua faixa usando a função *analogReference*. Os pinos analógicos 6 e 7 não podem ser usados como pinos digitais.

O *Arduino Nano* possui vários recursos para se comunicar com um computador, com outro *Arduino* ou com outros microcontroladores. O *ATmega328* fornece comunicação serial *UART TTL* de 5V e está disponível nos pinos digitais 0 (*RX*) e 1 (*TX*). Ademais, também suporta comunicação *I2C*, *TWI* e *SPI*. Um *FTDI FT232RL* na placa canaliza essa comunicação serial por *USB* e os *drivers FTDI*, incluídos com o software *Arduino*, fornecem uma porta de comunicação virtual para o software no computador. Ele é o *ArduinoIDE* e inclui um monitor serial que permite que dados textuais simples sejam enviados de e para a placa *Arduino*. Esse software também inclui uma biblioteca *Wire* para simplificar o uso do barramento *I2C*. Os *LEDs RX* e *TX* na placa piscarão quando os dados estiverem sendo transmitidos através do chip *FTDI* e a conexão *USB* para o computador, mas não para comunicação serial nos pinos 0 e 1. Além disso, uma biblioteca chamada *SoftwareSerial* permite a comunicação serial em qualquer um dos pinos digitais do *Nano*.

O *Arduino Nano* pode ser programado com o software *Arduino IDE*. Além disto, o *ATmega328* no *Arduino Nano* vem pré-programado com um *bootloader*, que é um instalador de programas e que permite que se carregue um novo código de programa sem o uso de um programador de hardware externo.

O *Nano*, que pode ser visualizado na Figura 13, foi utilizado neste trabalho para processar isoladamente os dados coletados do módulo de consumo de energia. A escolha desta placa se deu devido ao baixo custo, poder de processamento suficiente para o experimento, facilidade na integração com o módulo de medição e facilidade no desenvolvimento do código para este fim.

Figura 13 – Arduino Nano V3



Fonte: (NANO, 2021)

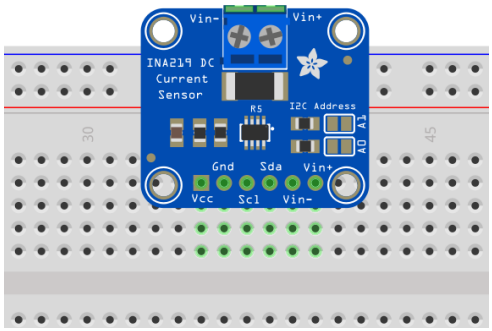
5.1.2 *INA219*

A empresa Texas (2022) descreve este módulo como um medidor de corrente e monitor de energia com uma interface compatível com I2C ou *SMBUS*. O dispositivo monitora a queda de tensão e a tensão de alimentação do barramento, com tempos de conversão e filtragem programáveis. Assim, um valor de calibração programável, combinado com um multiplicador interno, permite leituras diretas de corrente em amperes e um registro de multiplicação adicional calcula a potência em *watts*. Além disso, a interface é compatível com *I2C* ou *SMBUS* e possui 16 endereços programáveis.

O *INA219* está disponível em duas classes: A e B. A versão da classe B tem maior precisão e especificações de precisão mais altas. Ademais, ele detecta a resistência em barramentos que podem variar de 0 a 26 V. O dispositivo utiliza uma única alimentação de 3 a 5,5 V, consumindo no máximo 1 mA de corrente de alimentação e opera de –40 °C a 125 °C.

A placa Sensor de Corrente *DC INA219 I2C*, que pode ser vista na Figura 14, facilita os projetos envolvendo este componente, pois já engloba o circuito integrado e os componentes passivos necessários para seu funcionamento. Ela possui um resistor *shunt* de 0.100R (100 miliOhm), que permite medição de correntes máximas de +- 3,2 A (tensão máxima de +-320 mV no resistor *shunt*) e também terminais para conexão com o microcontrolador e com a fonte que irá alimentar a carga a ser monitorada. A placa possui também dois conjuntos de *pads/jumpers* que permitem modificar o endereço *I2C* do dispositivo, ou seja, podemos utilizar até 4 dispositivos ao mesmo tempo em uma mesma linha *I2C* (FLOP, 2017).

Figura 14 – Módulo INA219 Texas Intruments



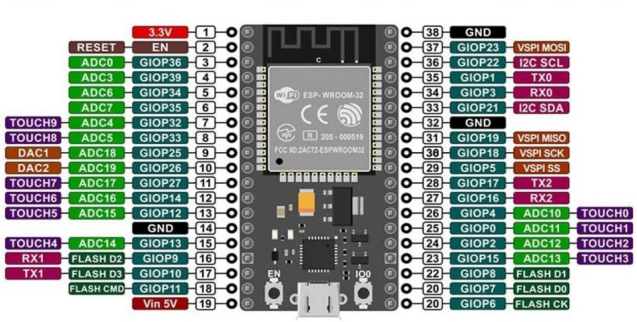
Fonte: (TEXAS, 2022)

Este módulo foi escolhido por ser de fácil utilização e integração com o *Arduino*, além de ter uma boa resolução na medição de tensão e corrente *DC*. Com isso, tornou-se possível isolar o circuito de medição de consumo, do microcontrolador *NodeMCU*, que foi utilizado nos experimentos de *overhead* aplicando protocolos de segurança. Desta forma, a medição torna-se confiável e não interfere no processamento do experimento.

5.1.3 *NodeMCU ESP32*

O *ESP32* é um *System On Chip (SoC)* ou Sistema Microcontrolador em um Chip com vários recursos embutidos: wi-fi (banda de 2,4 GHz), *Bluetooth*, dois núcleos *Xtensa LX6* de 32 *bits* e vários periféricos. A Figura 15 mostra os pinos de entrada/saída do *NodeMCU ESP32*. A versão utilizada nos experimentos foi um kit de desenvolvimento que integra a versão *ESP32-D0WD-V3* do microcontrolador.

Figura 15 – *NodeMCU ESP32 I/O Pin Datasheet*



Fonte: (PRAYOGO; MUKHLIS; YAKTI, 2019)

Alguns atributos desta plataforma foram avaliados para defini-la como hardware dos experimentos do lado do cliente *IoT*, entre elas estão:

- Design robusto - é capaz de funcionar de forma confiável em ambientes industriais, com uma temperatura de operação variando de –40 ° C a + 125 ° C. Alimentado por circuitos de calibração avançados, o *ESP32* pode remover dinamicamente imperfeições do circuito externo e se adaptar às mudanças nas condições externas. Atributo mais do que suficiente para utilização em *smart home*.

- Baixo consumo energético - projetado para dispositivos móveis, eletrônicos vestíveis e aplicativos *IoT*, o *ESP32* atinge um consumo de energia ultra baixo. O *ESP32* também inclui recursos como contador de *clock* de baixa granularidade, vários modos de energia e escalonamento de energia dinâmico.

- Alto nível de integração - é altamente integrável com circuitos projetados em placas de circuito impresso, possui antena embutida, amplificador de potência, amplificador de recepção de baixo ruído, filtros e módulos de gerenciamento de energia. Possui ambiente de desenvolvimento próprio, chamado de *ESP-IDF*, mas pode ser facilmente integrado com projetos utilizando *Arduino IDE*.

- Modo de funcionamento híbrido - o *ESP32* pode funcionar como um sistema autônomo completo ou como um dispositivo escravo para um *MCU host*, reduzindo a sobrecarga da pilha de comunicação no processador do aplicativo principal. O *ESP32* pode fazer

interface com outros sistemas para fornecer funcionalidade wi-fi e *Bluetooth* por meio de suas interfaces *SPI/I2C/UART*.

- Baixo custo de aquisição - o microcontrolador já é embarcado em um kit de desenvolvimento pode ser comprado por menos de 2 dólares.
- Poder de processamento - o modelo escolhido possui dois núcleos *Xtensa LX6* de 32 *bits*, hardware exclusivo para aceleração de cálculos cartográficos e pode trabalhar a uma frequência de até 240 *MHz*.

Para desenvolver os aplicativos para o *ESP32* a *IDE* da *Espressif ESP-IDF* foi utilizada neste trabalho (ESPRESSIF, 2021).

5.1.4 *Raspberry pi 3 model B*

Fabricada pela empresa *Raspberry Pi Foundation*, o Raspberry pi 3 model B é um *Single Board computer (SBC)*, ou seja, um computador de placa única, com um processador *Broadcom BCM2837 quad-core* 1,2*GHz* de 64 *bits*, 1*GB RAM*, *BCM43438 wireless LAN* e *Bluetooth Low Energy (BLE)*. O sistema operacional utilizado foi o *Raspbian*. Nessa pesquisa, o *Raspbery* foi utilizado como servidor e nele foram instalados o *broker mosquitto*, para utilização do *MQTT*, e a biblioteca *LibCoAP* pra rodar um código de servidor *CoAP*. A Figura 16 mostra o *Raspberry pi 3 model B* (FOUNDATION, 2021).

Figura 16 – Raspberry pi 3 model B

Fonte: (FOUNDATION, 2021)

5.1.5 Roteador Twibi Giga

O roteador wi-fi 5 *Mesh – Twibi Giga* da Intelbras utiliza a tecnologia de redes *mesh* para melhorar a conexão entre os dispositivos. O wi-fi *mesh* é uma tecnologia de redes sem fio que utiliza uma topologia de malha, onde todos os nós são conectados entre si, agindo como uma única rede sem fio com vários pontos espalhados. Neste modelo, a taxa de transferência é de até 867*Mbps* em 5*GHz(802.11ac)* e 300*Mbps* em 2,4 *GHz (802.11n)*, além de garantir

até 60 usuários conectados, possuindo uma área de cobertura do sinal wi-fi de até 180m² em ambientes abertos (INTELBRAS, 2021). Dentre as principais especificações disponibilizadas pelo fabricante estão, destacam-se:

- Cobertura - Módulo unitário Até 180m², kit 2 unidades Até 360m²;
- Número de dispositivos conectados - Até 60;
- Antenas - Duas antenas internas de 3*dBi*;
- Portas - Duas portas 10/100/1000 *Mbps*;
- *Chipset* - *Chipset Realtek® RTL8197FS+RTL8812BR+RTL8363*;
- Memória - Memória *flash* 16 *MB*, memória *RAM* 128 *MB*;
- Padrões - *IEEE 802.11ac/a/n 5 GHz, IEEE 802.11b/g/n 2.4 GHz, IEEE 802.11v/r (roaming)*;
- Modo do Rádio - *MU-MiMo, Beamforming*;
- Frequência de Operação - 2.4 *GHz*, 5 *GHz*;
- Segurança - *WPA2-PSK*;

Uma imagem ilustrativa do roteador pode ser vista em Figura 17

Figura 17 – Twibi Giga

Fonte: (INTELBRAS, 2021)

5.2 Bibliotecas e códigos

Para execução dos experimentos foi oportuno utilizar algumas bibliotecas e códigos de exemplo fornecidos pelos fabricantes de hardwares ou por desenvolvedores dos protocolos. Nesta seção será explanado quais foram esses códigos e como foram configurados ou modificados para atender aos experimentos.

5.2.1 *Adafruit INA219*

A *Adafruit* foi fundada em 2005 pelo engenheiro do *MIT, Limor Ladyada Fried* e seu objetivo era criar o melhor lugar online para aprender eletrônica, além de fazer os melhores produtos projetados para fabricantes de todas as idades e níveis de habilidade. Nos últimos 10 anos, a empresa cresceu para mais de 100 funcionários no coração de Nova York, com uma fábrica de mais de 15.000 metros quadrados. Ao longo dos anos, expandiu as ofertas para incluir ferramentas, equipamentos e eletrônicos que *Limor* seleciona, testa e aprova pessoalmente antes de entrar na loja. O sensor *INA219* utilizado no experimento de medição de consumo energético utiliza a biblioteca (*Adafruit_INA219.h*) e código de exemplo fornecidos pela *Adafruit* (ADAFRUIT, 2021).

Esta biblioteca utiliza o protocolo de comunicação de hardware *I2C* para fazer leituras do módulo de medição e enviar para o *Arduino Nano*. Para utilizar os códigos de exemplo é preciso instalar a biblioteca na *IDE* do *Arduino*, para isso basta ir no menu "Ferramentas", após isso, clicar em "Gerenciar Bibliotecas", pesquisar por *INA219* e clicar em instalar a versão escolhida e fornecida pela *Adafruit*. Após instalar, basta ir no menu "Abrir", depois clicar em "Exemplos", "Adafruit INA219"e "getcurrent". As modificações realizadas nesse exemplo foram comentários nas linhas que exibem os valores medidos, deixando somente o valor medido da corrente do circuito, para facilitar a exportação dos dados para uma planilha. A parte comentada pode ser vista na Figura 18.

Figura 18 – Código de medição de corrente.

```
void loop(void)
{
  float shuntvoltage = 0;
  float busvoltage = 0;
  float current_mA = 0;
  float loadvoltage = 0;
  float power_mW = 0;

  shuntvoltage = ina219.getShuntVoltage_mV();
  busvoltage = ina219.getBusVoltage_V();
  current_mA = ina219.getCurrent_mA();
  power_mW = ina219.getPower_mW();
  loadvoltage = busvoltage + (shuntvoltage / 1000);

 // Serial.print("Bus Voltage:   "); Serial.print(busvoltage); Serial.println(" V");
 // Serial.print("Shunt Voltage: "); Serial.print(shuntvoltage); Serial.println(" mV");
 // Serial.print("Load Voltage:  "); Serial.print(loadvoltage); Serial.println(" V");
  Serial.print("Current:       "); Serial.print(current_mA); Serial.println(" mA");
//  Serial.print("Power:         "); Serial.print(power_mW); Serial.println(" mW");
  Serial.println("");

  delay(2000);
}
```

Fonte: Própria

5.2.2 *LibCoAP*

Os códigos tanto da aplicação cliente, no *NodeMCU*, quanto no lado servidor, exacutado no *Raspberry Pi*, foram derivados dos exemplos dessa biblioteca. A *Libcoap* implementa na

linguagem *C* a especificação da *RFC 7252* que define o protocolo *CoAP* e suas características. Entre os principais atributos dessa implementação estão:

- Multiplataforma - foi projetada para ser executada em dispositivos embarcados, bem como em sistemas de computador de ponta com *POSIX OS*. Assim, você pode desenvolver e testar seus aplicativos *CoAP* em seu laptop e, em seguida, movê-los para sua plataforma de destino facilmente.
- Interoperável - a *libcoap* participou com sucesso em várias soluções utilizando *CoAP* e está em uso diário por vários projetos *IoT* na indústria e na academia.
- Diversas extensões - a biblioteca fornece a funcionalidade principal para o desenvolvimento de servidores e clientes *CoAP* com uso eficiente de recursos, incluindo monitoramento de recursos, transferência em bloco, *FETCH/PATCH*, *No-Response*, *TCP* e *Hop-Limit*. Implementações de exemplo mostram como esses recursos podem ser usados em aplicativos.
- Códigos de exemplos - o *coap-client* é uma ferramenta semelhante a um *wget*, utilizada para gerar solicitações simples para recuperação e modificação de recursos em um servidor remoto. Já o *coap-server* é um aplicativo de servidor básico que ilustra vários recursos do lado do servidor do *libcoap*, enquanto *coap-rd* implementa um diretório de recursos *CoAP* simples.
- Segurança - a biblioteca foi projetada para oferecer suporte à segurança da Camada de Transporte utilizando estruturas como *GnuTLS*, *OpenSSL*, *Mbed TLS* ou *tinydtls*.

5.2.3 Aplicação cliente *Libcoap*

Com a finalidade de experimentação dos argumentos propostos para essa dissertação, utilizou-se o exemplo *coap_client* fornecido pelo *framework ESP-IDF* como base, dado que foi o único exemplo de *CoAP + DTLS* funcional encontrado. O código utilizado foi modificado para que ao invés de enviar uma única requisição para o servidor, enviasse a cada segundo uma requisição repetindo este processo 100 vezes. Ao enviar uma requisição, é iniciado um contador de tempo que só finaliza a contagem assim que a resposta retorna do servidor. Este tempo de requisição e resposta é impresso no console e coletado para análise de *Round-trip time (RTT)*. A Figura 19 mostra as principais alterações grifadas em vermelho.

Figura 19 – Exemplo código cliente *CoAP*

```
int cont = 0;


while(cont <= 99){
    request = coap_new_pdu(session);
    if (!request) {
        ESP_LOGE(TAG, "coap_new_pdu() failed");
        goto clean_up;
    }
    request->type = COAP_MESSAGE_CON;
    request->tid = coap_new_message_id(session);
    request->code = COAP_REQUEST_GET;
    coap_add_optlist_pdu(request, &optlist);

    resp_wait = 1;
    tempo inicial = esp_timer_get_time();
    coap_send(session, request);

    wait_ms = COAP_DEFAULT_TIME_SEC * 1000;

    while (resp_wait) {
        int result = coap_run_once(ctx, wait_ms > 1000 ? 1000 : wait_ms);

        if (result >= 0) {
            if (result >= wait_ms) {
                ESP_LOGE(TAG, "select timeout");
                break;
            } else {
                wait_ms -= result;
            }
        }
    }
    sleep(1);
    cont++;
```

Fonte: Própria

Uma segunda versão desse código foi utilizada adicionando as configurações de *PSK* e *PKI* com certificado autoassinado para fornecer a comunicação segura entre cliente e servidor. Além disso, foi utilizado um par de chaves *RSA* de 2048 bits, que foram geradas com o auxilio do *framework OpenSSL* através da linha de comando da Figura 20.

Figura 20 – Comando para gerar certificado.

```
C:\ESP_IDF\examples\protocols\coap_client> openssl req -x509 -sha256 -nodes -newkey rsa:2048 -keyout keyfile.pem -out certfile.pem
```

Fonte: Própria

É importante ressaltar que o certificado gerado é autoassinado, considerado suficiente para ser utilizado nos experimentos. O ideal seria envolver uma assinatura de autoridade certificadora válida, mas, por motivo de custos ele não foi utilizado. Por esse motivo, os certificados *coap_ca.pem* e *coap_client.crt* são idênticos. Para fins de organização, estes arquivos foram armazenados no diretório *certs*, na pasta raiz do projeto.

5.2.4 *ESP-MQTT*

ESP-MQTT é uma biblioteca que implementa o lado cliente do protocolo *MQTT*. Fornecida pela plataforma *EDP-IDF* da Espressif (2021), esta biblioteca disponibiliza códigos de exemplos

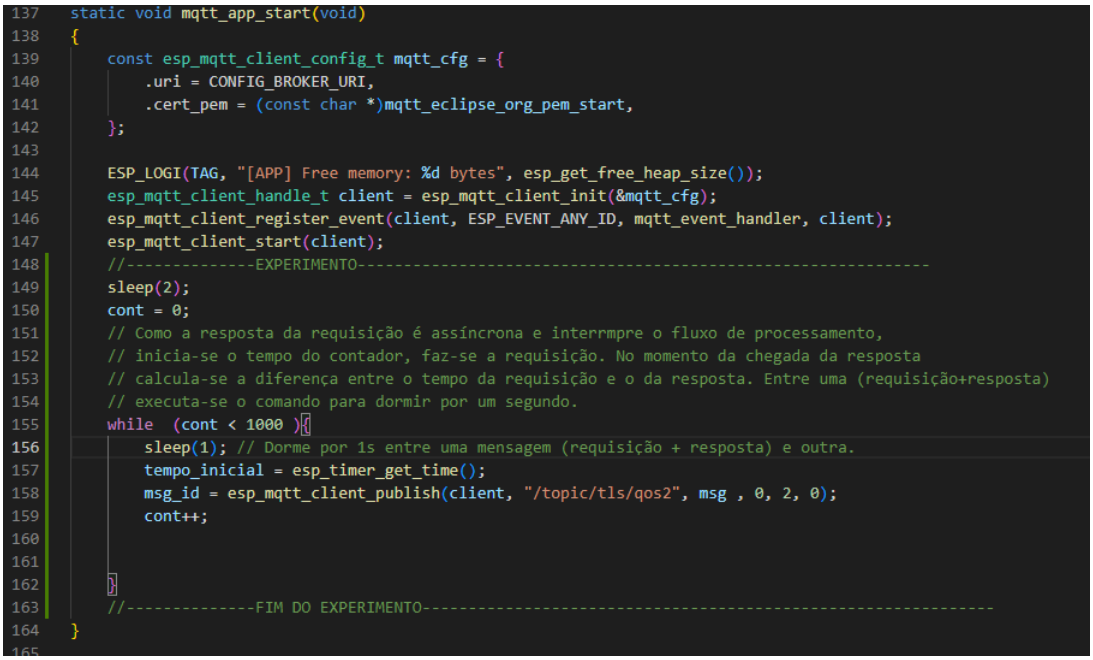
para utilizar o *NodeMCU ESP32* como cliente *MQTT*. Dentre as principais características desse pacote estão:

- Suporta *MQTT* sobre *TCP*, *SSL* com *mbedtls*, *MQTT* sobre *Websocket*, *MQTT* sobre *Websocket Secure*;
- Fácil de configurar com *URI*;
- Várias instâncias (vários clientes em uma aplicação);
- Suporta assinatura, publicação, autenticação, e todos os 3 níveis de *QoS*;
- Permite mensagens de *Last Will Testammnt (LWT)* para notificar os outros clientes quando um cliente se desconecta de maneira inesperada.

5.2.5 Aplicação cliente *MQTT*

Objetivando investigar os argumentos propostos para essa dissertação, utilizou-se os exemplos *mqtt_ssl* e *mqtt_tcp* fornecidos pela plataforma *ESP-IDF*, como base para os experimentos. Os códigos utilizados foram modificados para que ao invés de enviar uma única requisição para o servidor enviasse a cada segundo uma requisição repetindo este processo 100 vezes. Ao enviar uma requisição, é iniciado um contador de tempo que só finaliza a contagem assim que a resposta retorna do servidor. Este tempo de requisição e resposta é impresso no console e coletado para análise de *RTT*. A Figura 21 mostra as principais alterações no trecho de código entre os comentários: *//–EXPERIMENTO—* e *//—FIM DO EXPERIMENTO—*.

Figura 21 – Exemplo código cliente *MQTT*



```
static void mqtt_app_start(void)
{
    const esp_mqtt_client_config_t mqtt_cfg = {
        .uri = CONFIG_BROKER_URI,
        .cert_pem = (const char *)mqtt_eclipse_org_pem_start,
    };

    ESP_LOGI(TAG, "[APP] Free memory: %d bytes", esp_get_free_heap_size());
    esp_mqtt_client_handle_t client = esp_mqtt_client_init(&mqtt_cfg);
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, client);
    esp_mqtt_client_start(client);
    //--------------EXPERIMENTO-------------------------------------------------------------
    sleep(2);
    cont = 0;
    // Como a resposta da requisição é assíncrona e interrmpre o fluxo de processamento,
    // inicia-se o tempo do contador, faz-se a requisição. No momento da chegada da resposta
    // calcula-se a diferença entre o tempo da requisição e o da resposta. Entre uma (requisição+resposta)
    // executa-se o comando para dormir por um segundo.
    while  (cont < 1000 ){
        sleep(1); // Dorme por 1s entre uma mensagem (requisição + resposta) e outra.
        tempo_inicial = esp_timer_get_time();
        msg_id = esp_mqtt_client_publish(client, "/topic/tls/qos2", msg , 0, 2, 0);
        cont++;


    }
    //--------------FIM DO EXPERIMENTO-------------------------------------------------------
}
```

Fonte: Própria

5.2.6 *Mosquitto*

O *broker Mosquitto* é utilizado para implementar o protocolo *MQTT* e por ser leve, torna-o compatível para ser empregado na Internet das Coisas. Ele responsável por gerir as publicações e as assinaturas do *MQTT*, ou seja, sua função é intermediar e automatizar as mensagens recebidas para depois enviar ao seus destinatários (ECLIPSE, 2021). No estudo do desempenho do *MQTT* e do *MQTT + TLS*, o *EsP-IDF* é utilizado para executar o código cliente no *NodeMCU ESP32*. Assim, o *NodeMCU* atua como dispositivo que envia uma mensagem, registra o momento do envio e aguarda o recebimento da mesma mensagem, contabilizando o tempo de duração (envio + recebimento) ao receber a mensagem.

5.2.7 Arquitetura Experimental *MQTT*

No *MQTT*, o *NodeMCU* atua como publicador e assinante de um tópico específico chamado de *"/topic/qos0"*. A mensagem que consiste de dado útil possui uma *String* com o texto "Teste *RTT MQTT QoS* 0"e servidor utilizado no experimento foi o *Mosquitto*. Por fim, o tempo de resposta ou atraso será a diferença entre o momento em que o dado é enviado do cliente (*NodeMCU*), vai para o servidor (*Mosquitto*) e é recebido pelo cliente que também é assinante do tópico.

5.2.8 Arquitetura Experimental *MQTT + TLS*

Para configurar a segurança *TLS* no *Mosquitto* é fundamental criar um autoridade certificadora, chaves de servidor e certificados. O software *OpenSSL* é usado para gerar as chaves e os certificados. O primeiro passo é criar uma chave para simular autoridade certificadora que será responsável pela validação do certificado. Depois disso, é criado um certificado para a autoridade certificadora. E em seguida, criamos a chave destinada ao servidor. Ao concluir o processo, faremos a configuração do servidor para exigir o certificado.

5.2.9 Arquitetura Experimental *Mosquitto*

Por padrão o *Mosquitto* vem configurado com a porta 1883 para ser utilizado no *MQTT* puro. Após isso, para utilizar a criptografia *TLS*, o arquivo de configuração que esta presente no diretório do *Mosquitto* deve ser alterado para executar na porta 8883. Ao abrir o arquivo *Config*, é necessário alterar a porta para 8883, como mostra a Figura 22.

Figura 22 – Configuração de porta *Mosquitto.config*

```
200  # ================================================================
201  # Default listener
202  # ================================================================
203
204  # IP address/hostname to bind the default listener to. If not
205  # given, the default listener will not be bound to a specific
206  # address and so will be accessible to all network interfaces.
207  # bind_address ip-address/host name
208  #bind_address
209
210  # Port to use for the default listener.
211  port 8883
```

Fonte: Própria

Depois, precisamos adicionar o caminho onde está o certificado emitido pela autoridade certificadora chamado *"ca.crt"*, a chave do servidor *"server.key"* e o arquivo para validar o certificado do servidor *"server.crt"*, como mostra na Figura 23.

Figura 23 – Certificados - *Mosquitto.config*

```
# ----------------------------------------------------------------
# Certificate based SSL/TLS support
# ----------------------------------------------------------------
# The following options can be used to enable SSL/TLS support for
# this listener. Note that the recommended port for MQTT over TLS
# is 8883, but this must be set manually.
#
# See also the mosquitto-tls man page.

# At least one of cafile or capath must be defined. They both
# define methods of accessing the PEM encoded Certificate
# Authority certificates that have signed your server certificate
# and that you wish to trust.
# cafile defines the path to a file containing the CA certificates.
# capath defines a directory that will be searched for files
# containing the CA certificates. For capath to work correctly, the
# certificate files must have ".crt" as the file ending and you must run
# "openssl rehash <path to capath>" each time you add/remove a certificate.
#cafile
#capath
cafile C:\Program Files\mosquitto\certs\ca.crt

# Path to the PEM encoded server certificate.
#certfile
certfile C:\Program Files\mosquitto\certs\server.crt

# Path to the PEM encoded keyfile.
#keyfile
keyfile C:\Program Files\mosquitto\certs\server.key

```

Fonte: Própria

Feito isso, precisamos informar qual versão do *TLS*, conforme visto na Figura 24. Ao final, o certificado deve ser copiado para o cliente.

Figura 24 – TLS - Mosquitto.config

```
# This option defines the version of the TLS protocol to use for this listener.
# The default value allows all of v1.3, v1.2 and v1.1. The valid values are
# tlsv1.3 tlsv1.2 and tlsv1.1.
tls_version tlsv1.2
```

Fonte: Própria

6

Experimentação, análise de desempenho, usabilidade e custos

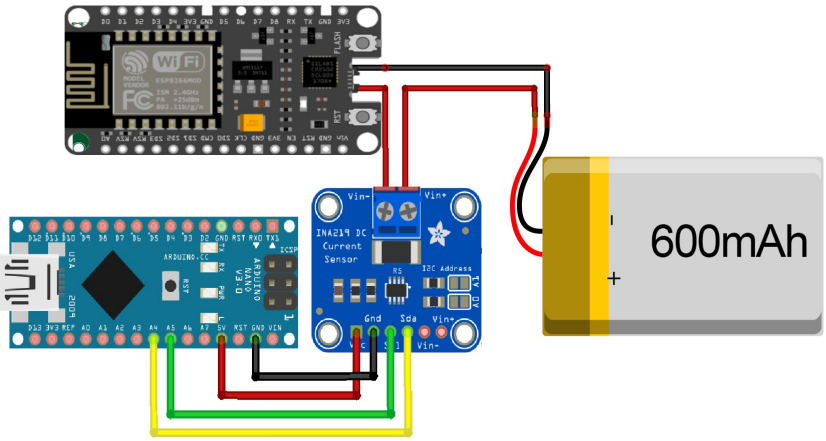
Neste capítulo, será abordado detalhes da execução dos experimentos, assim como, será feita uma análise estatísticas dos resultados aferidos. Atendendo aos questionamentos propostos pelo trabalho, efetuou-se estudos que visam avaliar a viabilidade do ponto de vista econômico, de performance e de facilidade de uso. O objetivo é responder se a sobrecarga de segurança em relação à privacidade, na comunicação utilizando os protocolos escolhidos, pode inviabilizar o uso em microcontroladores de baixo custo em uma rede *smart home*.

6.1 Análise de medição de consumo energético

Para medir o consumo energético da execução dos experimentos no lado cliente (*NodeMCU*), um segundo microcontrolador do modelo *Arduino Nano* foi utilizado juntamente com o módulo *Ina219*, medidor de corrente *DC*, utilizando o protocolo de comunicação *I2C*. A escolha do uso de um segundo microcontrolador apenas para a medição de corrente, se deu devido a necessidade de isolar o circuito de medição em relação ao circuito do experimento. Desta maneira, torna-se possível garantir que todo consumo medido será apenas da execução dos protocolos e suas sobrecargas de segurança no *ESP32*. Além disso, com o propósito de checar qualquer tipo de problema de calibração no sensor *Ina219*, um multímetro foi utilizado pare conferir a medição entre a fonte de alimentação e o *ESP32*, o que apresentou resultados idênticos nas duas medições.

A Figura 25 mostra o *ESP32* ligado à uma fonte de alimentação externa, representado pela bateria de 600 *mAh*, e o sensor *Ina219* ligado em série, realizando assim, a medição de toda carga que o microcontrolador demanda da bateria. O *Arduino* por sua vez é alimentado pela porta *USB* de um computador, que além de fornecer energia para ele também recebe os dados coletados pelo sensor.

Figura 25 – Circuito de medição de corrente



Fonte: Própria

6.1.1 Análise comparativa de consumo energético no protocolo *CoAP*

As medições de corrente elétrica aferidas através do módulo *Ina219* utilizam a unidade de mediada *mA* (*miliampere*). Foram coletadas 100 observações em *mA* para cada amostra de cada experimento. Para o total de 5 amostras por experimento, calculou-se a média entre as observações e por fim a média entre as amostras. Foi optado pelo índice de média por este ser sensível aos valores extremos, o que reflete a realidade quando se refere a custo financeiro por consumo de energia. A Tabela 3 denota as médias em cada experimento com o protocolo *CoAP*.

Tabela 3 – Consumo energia *CoAP*.

CoAP sem Segurança	***CoAP + DTLS***	***CoAP + DTLS + Payload 1kb***
34,4*mA*	38,3*mA*	40,52*mA*

Visando calcular a carga total de energia consumida nos experimentos, fez-se necessário aplicar algumas conversões de escalas e unidades de medidas. Para exemplificar os cálculos efetuados em cada experimento, lista-se a seguir o passo a passo utilizando a primeira média (protocolo *CoAP* sem segurança):

- 34,4 *mA* / 1000 = 0,0344 *A* (conversão de *miliampere* para *ampere*);
- 3,3 *v* * 0,35 *A* = 0,11352 *W* (cálculo da potência em *watts* utilizando lei de *ohm*);
- 0,1155 *W* / 1000 = 0,00011352 *kW* (conversão de *watt* para *Kilowatt*);
- 0,0001155 *kW* *24h = 0,00272448 *kWh* (consumo diário em *kilowatt* hora);
- 0,002772 *kWh* * 30 dias = 0,0817344 *kWh* (consumo mensal);

- 0,08316 *kwh* * R$ 1,2 (taxa aplicada pela companhia de energia) = 0,09808 (aproximadamente 9 centavos de reais por mês) .

Seguindo as mesmas etapas para as demais médias, obteve-se os seguintes resultados:

- Para *CoAP + DTLS*, o custo em reais foi de: 0,091 *KWh* que custa 0,109 reais ou 10 centavos;
- Para *CoAP + DTLS + Payload 1 kb*, o custo foi de: 0,0962 *kWh* que custa 0,115 reais ou 11 centavos.

6.1.2 Análise comparativa de consumo energético no protocolo *MQTT*

Assim como na avaliação de consumo do protocolo *CoAP*, foram coletadas 100 observações em *mA* para cada amostra de cada experimento. Com um total de 5 amostras por experimento, calculou-se a média entre as observações e por fim a média entre as amostras. Foi optado pelo índice de média por este ser sensível aos valores extremos, o que reflete a realidade quando se refere a custo financeiro por consumo de energia. A Tabela 4 denota as médias em cada experimento com o protocolo *MQTT*.

Tabela 4 – Consumo energia *MQTT*

MQTT Qos 0	***MQTT Qos 0 + TLS***	***MQTT Qos 1 + TLS***	***MQTT Qos 2 + TLS***	***MQTT Qos2 + TLS + Payload 1kb***
42 *mA*	44,9 *mA*	48,7 *mA*	53,9 *mA*	68,5 *mA*

Seguindo as mesmas etapas de conversões e cálculos aplicadas ao *CoAP* obteve-se os seguintes resultados para os experimentos com *MQTT*:

- Para *MQTT QoS 0* o custo foi de: 0,09979 *kWh* que custa 0,1197 reais, quase 12 centavos de reais;
- Para *MQTT QoS 0 + TLS* o custo foi de: 0,1066 *kWh* que custa 0,128 reais, quase 13 centavos;
- Para *MQTT QoS 1 + TLS* o custo foi de: 0,1157 *kWh* que custa 0,138 reais, quase 14 centavos;
- Para *MQTT QoS 2 + TLS* o custo foi de: 0,1280 *kWh* que custa 0,153 reais, 15 centavos;
- Para *MQTT QoS 2 + TLS + Payload 1kb* o custo foi de: 0,1627 *kWh* que custa 0,195 reais, 19 centavos.

6.2 Avaliação de Usabilidade

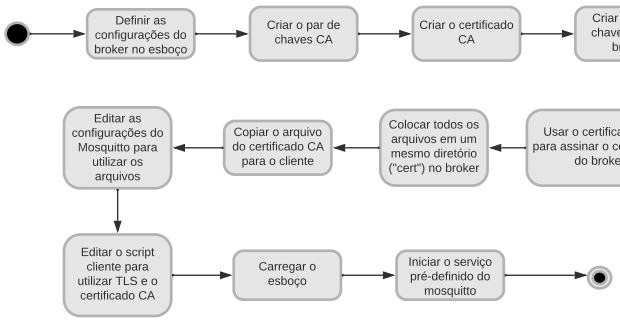
Nesta seção será feita uma avaliação de usabilidade do fluxo necessário para prover comunicação através dos protocolos *CoAP* e *MQTT* sem segurança, assim como, com a comunicação com segurança, utilizando os protocolos *TLS* e *DTLS*. Dessa maneira, do lado cliente o *ESP32* e lado servidor o *Raspberry Pi* utilizando o *broker Mosquitto*, para trabalhar com *MQTT* e *LibCoAP*, para trabalhar com *CoAP*.

Utilizando os exemplos dos experimentos realizados neste trabalho, realizou-se uma análise heurística das etapas necessárias para que os usuários possam configurar e usar um ambiente mínimo proposto. O objetivo desta avaliação é mensurar o aumento no nível de dificuldade em criar: certificados auto assinados, chaves públicas, privadas e configurá-los tanto no lado cliente quanto no lado servidor, afim de fornecer uma troca de mensagens com garantia de privacidade.

Neste experimento de usabilidade, efetuou-se uma análise das etapas necessárias para que um usuário *maker* possa, por conta própria, gerar as chaves e certificados essenciais ao uso do *TLS/DTLS*. O movimento *maker*, ou movimento do tipo faça você mesmo, é uma tendência onde os usuários criam seus próprios dispositivos e consequentemente geram alternativas às soluções de baixa privacidade das grandes corporações. Portanto, a análise da diferença de complexidade entre a utilização dos protocolos sem proteção e com proteção através de *TLS/DTLS* torna-se relevante, mesmo considerando que é necessário algum conhecimento básico prévio sobre essas tecnologias. A Figura 26 e a Figura 27 mostram os passos necessários para implementar uma comunicação utilizando *MQTT* e *MQTT* + *TLS*, respectivamente.

Figura 26 – Fluxo MQTT

Figura 27 – Fluxo MQTT+TLS



A primeira diferença notável está no número de etapas, o primeiro fluxo de trabalho contou com apenas três etapas, enquanto o segundo possui onze etapas. Para medir a complexidade dos fluxos, foi realizada uma avaliação heurística baseada no trabalho de Nielsen e Budiu (2013), no trabalho de Tedesco e Tullis (2006) e na métrica *Single Ease Question (SEQ)*. Dessa maneira, foi utilizada uma pontuação de complexidade para cada tarefa, neste método de 1 a 5, onde 1 seria pouco complexa e 5 muito complexa. A Tabela 5 mostra a distribuição das pontuações em cada uma das tarefas dos fluxos. Os passos para configurar uma conexão com certificado digital no protocolo *CoAP* foram os mesmos, porém substituindo o *broker* pela aplicação servidora.

Tabela 5 – Pontuação de usabilidade *MQTT* e *CoAP*.

Etapas sem segurança	**Pontos**	**Etapas com segurança**	**Pontos**
Adicionar *URL* do *broker*/servidor/servidor no cliente	2	Adicionar *URL* do *broker*/servidor no cliente	2
Carregar o *firmware* no cliente	1	Criar o par de chaves auto assinados	3
Iniciar o *broker*/servidor	1	Criar par de chaves do *broker*/servidor	3
		Usar o certificado auto assinado para assinar o certificado do *broker*/servidor	4
		Adicionar os arquivos de chaves e certificados ao diretório *certs*	1
		Copiar a chave pública para o *firmware* cliente	1
		Editar as configurações do *broker*/servidor para usar *TLS*	4
		Editar o *firmware* cliente para usar *TLS*	3
		Carregar o *firmware* no cliente	1
		Iniciar o *broker*/servidor com as configurações personalizadas	2
Total	4		24

6.3 Avaliação de Tempo de Resposta

Para a avaliação do tempo de resposta, foram realizados um total de 8 experimentos, refletindo 8 cenários, cada um com 5 amostras e cada amostra teve 100 observações do tempo de resposta. Esses experimentos resultaram em um total de 500 observações em cada cenário, permitindo determinar adequadamente a distribuição da média amostral. Este procedimento foi realizado para avaliar o tempo de resposta de uma requisição do cliente para o servidor, no caso do *CoAP*, ou uma requisição realizada de um cliente publicando em um *broker MQTT* e recebendo a resposta através da assinatura do mesmo tópico publicado.

Dessa forma, foram analisados os cenários onde a requisição fosse feita sem a camada de segurança em ambos os protocolos e com a camada segurança, com o objetivo de verificar a confiabilidade da informação quando aplicada à segurança. Testou-se também cenários seguros com adição de *QoS* no *MQTT* e mensagens confirmáveis no *CoAP*. Por último, adicionou-se mais informações ao *payload*, objetivando analisar se o tamanho deste poderia inviabilizar a comunicação segura. Os dados estatísticos demonstrados a seguir serão das 5 amostras de cada cenário, em seguida será demonstrada uma análise comparativa entre a média das observações de cada amostra, tornando assim uma amostra média, definida como:

$$Média = \frac{Amostra1 + Amostra2 + Amostra3 + Amostra4 + Amostra5}{5} \tag{6.1}$$

6.3.1 Resultados *RTT* do protocolo *MQTT*

Os experimentos desta subseção foram realizados com a finalidade de comparar a performance entre o *MQTT* inseguro, e este com os mecanismos de segurança e confiabilidade.

6.3.1.1 *MQTT QoS* 0

O primeiro experimento realizado com o protocolo *MQTT* foi a execução do algoritmo que faz a requisição sem segurança e sem nenhuma garantia de qualidade em em nível de serviço, denominou-se o experimento de *MQTT* com *QoS* 0. Assim, a Tabela 6 exibe os principais índices estatísticos calculados a partir das observações das cinco amostras neste cenário. Analisando os índices de tendência centrais (média, mediana e moda), percebe-se um equilíbrio entre eles. A média de tempo de resposta de uma requisição nesse cenário é de aproximadamente 11 *ms*, o valor que mais se repete é também de 11 *ms* e o tempo de resposta que central definido pela moda é de aproximadamente 9 *ms*. Essas informações apontam para uma conexão com tempo de resposta rápido. Como pode ser visto, o tempo mínimo em cada observação foi de 7 *ms* e o máximo na casa dos 20 *ms*.

Com a finalidade de entender melhor a distribuição dos dados coletados sobre o *RTT* desta conexão, calculou-se índices de variabilidade, desvios e dispersão. O desvio padrão, que

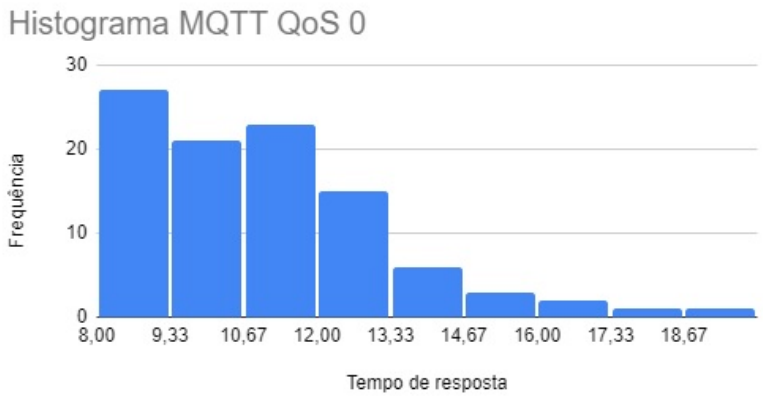
Tabela 6 – Tabela de resultados *RTT* usando *MQTT* com *QoS* 0.

***MQTT QoS* 0**	**Amostra 1**	**Amostra 2**	**Amostra 3**	**Amostra 4**	**Amostra 5**
Média	10,98	10,92	10,85	11.32	10,88
Mediana	11	11	11	11	10,5
Moda	9	10	9	9	9
Variação	5,8	3	3,7	7,5	4,9
Desvio Padrão	2,4	1,7	1,9	2,7	2,2
Mínimo	7	8	7	7	7
Máximo	21	17	18	21	21
Quartil 1	9	10	9	9	9
Quartil 2	11	11	11	11	10,5
Quartil 3	12	12	12	12	12
Distância Inter-Quartil	3	2	3	3	3
Distância Semi-Quartil	1,5	1	1,5	1,5	1,5

indica a dispersão em torno da média amostral a partir de de uma coleta aleatória, indicou que dado uma observação qualquer no conjunto da amostra, o seu valor pode diferir da média em mais ou menos 3 *ms*, que denota um tempo de resposta rápido.

Além disso, foram calculados os quantis, para que facilite o entendimento da distribuição probabilística. O Quartil 1 aponta que 25% das observações tiveram tempo de resposta abaixo dos 10 *ms*. O Quartil 2 indica que 50% tiveram *RTT* abaixo dos 11 *ms*. Já o Quartil 3 corrobora com 75% foram abaixo de 12 *ms*, o que confirma uma conexão rápida e estável. A Figura 28 exibe uma distribuição da frequência de tempo de resposta de forma visual, é notável que os maiores tempos de resposta acontecem com menos frequência.

Figura 28 – Histograma *MQTT Qos0*



6.3.1.2 *MQTT* com *SSL* e *QoS* 0

O segundo experimento realizado com o protocolo *MQTT*, além de utilizar segurança com *TLS*, porém sem nenhuma garantia de qualidade em nível de serviço, Assim, denominou-se o experimento de *MQTT* com *TLS* e *QoS* 0. A Tabela 7 exibe os principais índices estatísticos calculados a partir das observações das cinco amostras neste cenário. Analisando os índices de tendência centrais percebe-se que existe harmonia entre eles. A média de tempo de resposta de

uma requisição nesse cenário é de aproximadamente 13 *ms*, o valor que mais se repete é também de 13 *ms* e o tempo de resposta que central definido pela moda é de aproximadamente 13 *ms*, o que sugere uma distribuição próxima à distribuição normal. Essas informações apontam para uma conexão com tempo de resposta rápido. Como pode ser visto, o tempo mínimo em cada observação foi de 8 *ms* e o máximo na casa dos 24 *ms*.

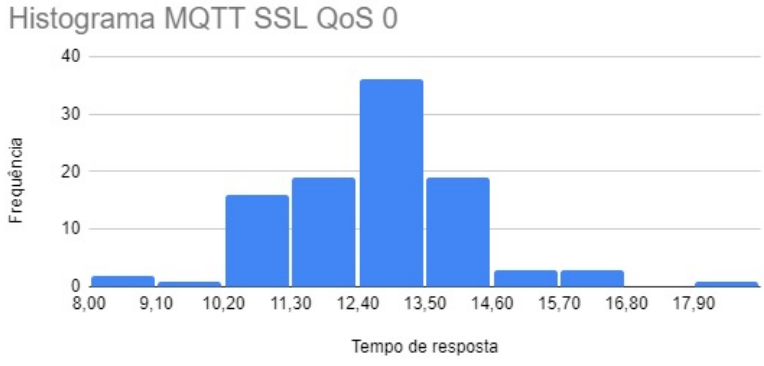
Tabela 7 – Tabela de resultados *RTT* usando *MQTT* com *TSL* e *QoS* 0.

***MQTT* com *SSL QoS* 0**	**Amostra 1**	**Amostra 2**	**Amostra 3**	**Amostra 4**	**Amostra 5**
Média	13,38	12,80	12,76	12,03	12,11
Mediana	13	13	13	12	12
Moda	13	13	13	1	11
Variação	5,42	2,28	1,92	1,52	2,05
Desvio Padrão	2,33	1,51	1,38	1,23	1,43
Mínimo	8	8	8	8	9
Máximo	24	18	16	17	19
Quartil 1	12	12	12	11	11
Quartil 2	13	13	13	12	12
Quartil 3	14	14	14	13	13
Distância Inter-Quartil	2	2	2	2	2
Distância Semi-Quartil	1	1	1	1	1

Objetivando entender melhor a distribuição dos dados coletados sobre o *RTT* desta conexão, calculou-se índices de variabilidade, desvios e dispersão. O desvio padrão indicou que dado uma observação qualquer no conjunto da amostra, o seu valor pode diferir da média em mais ou menos 2,3 *ms*, o que ainda é um tempo de resposta rápido.

Ademais, foram calculados os quantis, com o objetivo de facilitar o entendimento da distribuição probabilística. O Quartil 1 aponta que 25% das observações tiveram tempo de resposta abaixo dos 12 *ms*. O Quartil 2 indica que 50% tiveram *RTT* abaixo dos 13ms. Já o Quartil 3 corrobora com 75% foram abaixo de 14 *ms*, o que confirma uma conexão rápida e estável. A Figura 29 exibe uma distribuição da frequência de tempo de resposta de forma visual. A partir dela, é notável que os maiores tempos de resposta acontecem com menos frequência e os menores também, sugerindo que a maior parte das observações estão em torno da média.

Figura 29 – Histograma *MQTT* + *SSL QoS* 0



6.3.1.3 *MQTT* com *TSL* e *QoS* 1

O terceiro experimento realizado com o protocolo *MQTT*, utilizando segurança através do protocolo *TLS*, porém adicionando a qualidade de serviço nível 1 *QoS* 1. Dessa maneira, denominou-se o experimento de *MQTT* com *TLS* e *QoS* 1.

A tabela Tabela 8 exibe os principais índices estatísticos calculados a partir das observações das cinco amostras neste cenário. Analisando os índices de tendência centrais, percebe-se que a média destoa bastante dos demais índices. Isto ocorre devido à valores extremos causados pela retransmissão de pacotes no *QoS* 1, como a média é sensível aos extremos, o valor desta difere bastante da moda e da mediana. A média de *RTT* nesse cenário é de aproximadamente 50 *ms*, a mediana por volta dos 17 *ms* e o tempo de resposta que central definido pela moda é de aproximadamente 16 *ms*.

Tabela 8 – Tabela de resultados *RTT* usando *MQTT* com *TSL* e *QoS* 1.

***MQTT* com *SSL* *QoS* 1**	**Amostra 1**	**Amostra 2**	**Amostra 3**	**Amostra 4**	**Amostra 5**
Média	39,70	51,98	44,31	64,92	46,86
Mediana	17	17	17	17	16
Moda	16	16	16	16	16
Variação	5260,77	7698,98	6235,12	9897,04	6785,03
Desvio Padrão	72,53	87,74	78,96	99,48	82,37
Mínimo	14	14	14	14	14
Máximo	273	274	271	273	271
Quartil 1	16	16	16	16	16
Quartil 2	17	17	17	17	16
Quartil 3	18	18	18	19	18
Distância Inter-Quartil	2	2	2	3	2
Distância Semi-Quartil	1	1	1	1,5	1

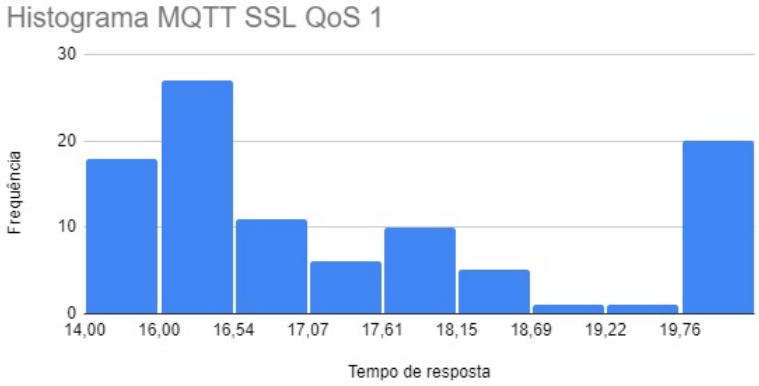
Analisando somente a média, percebe-se um aumento considerável no tempo de resposta, porém a maior parte das observações ainda são com *RTTs* consideravelmente rápidos. O tempo mínimo de resposta foi de 14 *ms* e o máximo por volta de 273 *ms*. Com a finalidade de entender melhor a distribuição dos dados coletados sobre o *RTT* desta conexão, calculou-se índices de variabilidade, desvios e dispersão. O desvio padrão indicou que dado uma observação qualquer no conjunto da amostra, o seu valor pode diferir da média em mais ou menos 90 *ms*, o que indica uma variabilidade considerável.

Além disso, foram calculados os quantis para que facilite o entendimento da distribuição probabilística. O Quartil 1 aponta que 25% das observações tiveram tempo de resposta abaixo dos 16 *ms*. O Quartil 2 indica que 50% tiveram *RTT* abaixo dos 17 *ms*. Já o Quartil 3 constata que 75% foram abaixo de 18 *ms*, o que demonstra uma conexão geralmente rápida com alguns momentos mais lentos.

A Figura 30 exibe uma distribuição da frequência de tempo de resposta de forma visual. Assim, é possível notar que uma quantidade razoável de observações destoam bastante das demais, como explicado anteriormente este tempo extra é causado pelo trabalho em que o protocolo tem

para garantir pelo menos uma vez a entrega da mensagem.

Figura 30 – Histograma *MQTT + SSL QoS* 1



6.3.1.4 *MQTT* com *SSL* e *QoS* 2

O terceiro experimento realizado com o protocolo *MQTT* e executado utilizando a segurança do protocolo *TLS*, além disso, foi utilizado o nível 2 de qualidade de serviço. Assim, denominou-se o experimento de *MQTT com TLS e QoS 2*.

A Tabela 9 exibe os principais índices estatísticos calculados a partir das observações das cinco amostras neste cenário. Analisando os índices de tendência centrais, percebe-se que os valores são demasiadamente altos se comparados com tempo mínimo de resposta. Isto ocorre devido aos envios das mensagens de confirmação e envio e recebimento no *QoS* 2. Neste nível de *QoS*, o *broker* retém a mensagem até que haja confirmação de recebimento, evitando assim duplicatas. A média de *RTT* nesse cenário é de aproximadamente 260 *ms*, a mediana por volta dos 268 *ms* e o tempo de resposta que central definido pela moda é de aproximadamente 268 *ms*.

Tabela 9 – Tabela de resultados *RTT* usando *MQTT* com *SSL* e *QoS* 2.

MQTT com SSL QoS 2	**Amostra 1**	**Amostra 2**	**Amostra 3**	**Amostra 4**	**Amostra 5**
Média	261,33	260,92	272,21	261,11	260,88
Mediana	268	268	272	268	268
Moda	267	267	268	268	268
Variação	1719,27	1769,39	2019,40	1740,79	1767,86
Desvio Padrão	41,46	42,06	44,94	41,72	42,05
Mínimo	25	22	30	25	22
Máximo	283	287	365	278	275
Quartil 1	267	267	269	267	267
Quartil 2	268	268	271,5	268	268
Quartil 3	270	269	293	269	269
Distância Inter-Quartil	3	2	24	2	2
Distância Semi-Quartil	1,5	1	12	1	1

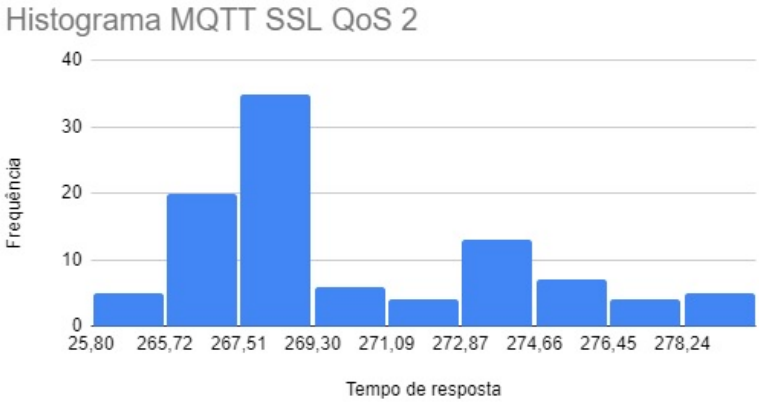
Analisando o tempo mínimo de resposta, que foi de 22 *ms* e o máximo de 365 *ms*, constata-se que o uso da segurança através do *TLS* associado ao *QoS* 2 produz uma sobrecarga no tempo de resposta, mesmo nas transmissões em que o pacote é enviado e recebido sem nenhuma

interrupção. Assim, objetivando entender melhor a distribuição dos dados coletados sobre o *RTT* desta conexão, calculou-se índices de variabilidade, desvios e dispersão. O desvio padrão indicou que dado uma observação qualquer no conjunto da amostra, o seu valor pode diferir da média em mais ou menos 45 *ms*, o que indica uma variabilidade elevada.

Ademais, foram calculados os quantis para que facilite o entendimento da distribuição probabilística. O Quartil 1 aponta que 25% das observações tiveram tempo de resposta abaixo dos 267 *ms*. O Quartil 2 indica que 50% tiveram *RTT* abaixo dos 271ms. Já o Quartil 3 corrobora com 75% foram abaixo de 293 *ms*, o que demonstra uma conexão mais lenta comparada aos cenários supracitados.

A Figura 31 exibe uma distribuição da frequência de tempo de resposta de forma visual. Dessa maneira, é notável que os maiores tempos de resposta acontecem com maior frequência e os menores já são bem raros, sugerindo que a maior parte das observações estão em torno da média, que neste cenário foi pouco afetada pelos extremos.

Figura 31 – Histograma *MQTT* + *TLS* + *QoS* 2



6.3.1.5 MQTT com TSL e QoS 2 + Payload 1024 bytes

O terceiro experimento realizado com o protocolo *MQTT* foi a execução do algoritmo que faz a requisição com segurança *TLS* utilizando o nível dois de qualidade de serviço, denominou-se o experimento de *MQTT* com *TLS*, *QoS* 2 e *payload* de 1024 bytes.

A tabela Tabela 10 exibe os principais índices estatísticos calculados a partir das observações das cinco amostras neste cenário. Analisando os índices de tendência centrais, percebe-se que os valores destes são demasiadamente altos se comparados com os experimentos anteriores. Isto ocorre devido ao tamanho do *payload*, causando fragmentação nos pacotes que são enviados em pedaços que precisam ser remontados. A média de *RTT* nesse cenário é maior que 300 *ms*, a mediana passa dos 300 *ms* e o tempo de resposta médio mais frequente também é próximo dos 300 *ms* na maioria das amostras.

Analisando o tempo mínimo de resposta que foi de 234ms e o máximo de 984ms, constata-se que o uso da segurança através do *TLS* associado ao *QoS* 2 e um *payload* de 1024

Tabela 10 – Tabela de resultados *RTT* usando *MQTT* com *TSL*, *QoS* 2 e *Payload* de 1024 bytes.

MQTT* + *TSL* + *QoS* 2 + 1024 bytes de *payload	**Amostra 1**	**Amostra 2**	**Amostra 3**	**Amostra 4**	**Amostra 5**
Média	330,83	302,25	373,45	302,38	327,61
Mediana	317	277,5	344,5	299,5	307
Moda	294	269	269	312	307
Variação	7971,31	6194,59	10871,02	4334,76	10029,61
Desvio Padrão	89,28	78,71	104,26	65,84	100,15
Mínimo	245	251	240	251	234
Máximo	868	790	743	787	984
Quartil 1	287	267	308	266	276
Quartil 2	317	277	344	299	307
Quartil 3	348	306	394	316	438
Distância Inter-Quartil	61	38,5	86,25	49,75	71,25
Distância Semi-Quartil	30,5	19,25	43,125	24,875	35,625

bytes produzem uma sobrecarga no tempo de resposta mesmo nas transmissões em que o pacote é enviado e recebido sem nenhuma interrupção.

Para entender melhor a distribuição dos dados coletados sobre o *RTT* desta conexão, calculou-se índices de variabilidade, desvios e dispersão. O desvio padrão indicou, dado uma observação qualquer no conjunto da amostra, que o seu valor pode diferir da média em até 104 *ms*, o que indica uma variabilidade elevada.

Além disso, foram calculados os quantis para que facilite o entendimento da distribuição probabilística. O Quartil 1 aponta que 25% das observações tiveram tempo de resposta abaixo dos 308 *ms*. O Quartil 2 indica que 50% tiveram *RTT* abaixo dos 344 *ms*. Já o Quartil 3 explicita que 75% foram abaixo de 438ms, o que demonstra uma conexão mais lenta comparada aos cenários supracitados. A Figura 32 exibe uma distribuição da frequência de tempo de resposta de forma visual, é notável que na maior parte das observações o tempo de resta é elevado em comparação com os experimentos anteriores, não houve observações com *RTT* abaixo dos 200 *ms*.

Figura 32 – Histograma *MQTT* + *TSL* +*QoS* 2 + *Payload* 1024 bytes

6.3.1.6 Análise comparativa dos experimentos com *MQTT*

Após finalizar todos os experimentos com o protocolo *MQTT*, coletar os resultados e gerar dados estatísticos, analisou-se comparativamente os resultados entre os tempos de resposta de cada cenário. A tabela Tabela 11 exibe a frequência, em porcentagem, de observações em cada categoria de tempo de resposta.

Tabela 11 – Tabela da distribuição da frequência de *RTT*.

Categorias *RTT*	***Qos* 0**	***Qos* 0 + *TLS***	***Qos* 1 + *TLS***	***QoS* 2 + *TLS***	***QoS* 2 + *TLS* + *Payload* 1 *kb***
8-12ms	80%	37%	-	-	-
12-25ms	20%	63%	81%	1%	-
25-50ms	-	-	-	2%	-
50-70ms	-	-	4%	-	-
70-100ms	-	-	1%	-	-
100-200ms	-	-	3%	-	-
200-350ms	-	-	11%	97%	79%
350-500ms	-	-	-	-	17%
500-700ms	-	-	-	-	2%
700-100ms	-	-	-	-	2%

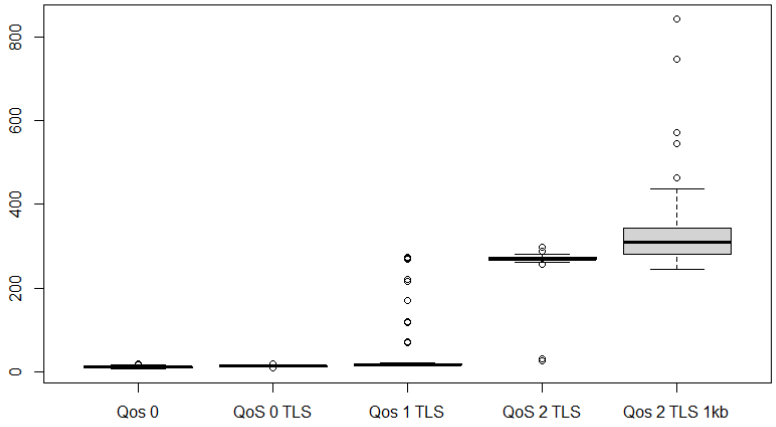
Observando a tabela, é possível constatar nitidamente uma sobrecarga nos tempos de resposta. A medida que se adiciona a proteção à privacidade com o uso do *TLS*, e aumenta-se o nível de confiabilidade na entrega dos pacotes através dos níveis *QoS*, a sobrecarga fica ainda mais evidente. No experimento em que foi acrescido o tamanho do *payload*, percebeu-se esforço ainda maior para enviar e receber uma mensagem.

Os dados demonstram que 80% das mensagens sem criptografia e qualidade de serviço *Qos* 0 apresentaram *RTT* entre 8 e 12 *ms* e os 20% restantes entre 12 e 25 *ms*. Ao adicionar a camada de proteção do *TLS*, no caso *QoS 0 + TLS*, apenas 37% apresentaram resposta entre 8 e 12 *ms* os outros 63% entre 12 e 25 *ms*. Em seguida, ao configurar o nível de confiabilidade para *QoS* 1, mantendo o *TLS*, não houve mensagens com tempo de resposta menor que 12 *ms* e 81% das mensagens retornaram entre 12 e 25 *ms*. Ainda mantendo o *TLS* e configurando nível de *QoS* para 2, as mensagens foram devolvidas, quase que na totalidade das observações (97%) entre 200 e 350 *ms*. Por fim adicionando uma quantidade significativa de dados ao pacote, quase 80% das medições ficaram entre 200 e 350 *ms*, porém 21% foram acima de meio segundo.

Considerando apenas as médias de *RTT* em cada experimento, o *MQTT* sem segurança com qualidade de serviço zero sofre uma sobrecarga de de 14,8% em relacao ao *MQTT* com *Qos* 0 e *TLS*, que por sua vez é 392,8% mais rápido que o *MQTT* com *QoS* 1 e *TLS*, e este é 531,3% mais rápido que o*MQTT* com *Qos* 2 e *TLS*, que por fim é 124,7% mais rápido do que esta mesma configuração porém com um *payload* de 1024 bytes. Contudo, a média é um índice bastante sensível aos extremos e ao modificar nível de *QoS*, percebe-se que algumas observações destoam bastante das demais, em estatística isso é chamado de *outliers*. Essas discrepâncias influenciam o valor da média, tonando-a um índice que não reflete a maior parte das observações. O gráfico de *boxplot* da Figura 33, que considera a mediana e os quartis, esclarece a real distribuição das

amostras.

Figura 33 – *Boxplot MQTT*



Baseando-se pelo gráfico, onde os traços mais escuros representam a mediana e os pontos são os *outliers*, torna-se evidente que a mediana é o índice mais adequado para analisar os dados, e a sobrecarga maior acontece a a partir do *QoS 2*.

6.3.2 Resultados *RTT* do protocolo *CoAP*

O primeiro experimento realizado com o protocolo *CoAP* foi a execução do algoritmo que faz a requisição sem segurança *DTLS*, porém, utilizando mensagens do tipo confirmável. Assim, denominou-se o experimento de *CoAP sem DTLS*.

Dessa maneira, a Tabela 12 exibe os principais índices estatísticos calculados a partir das observações das cinco amostras neste cenário. Analisando os índices de tendência centrais percebe-se um equilíbrio entre eles. A média de tempo de resposta de uma requisição nesse cenário é de aproximadamente 7 *ms*, o valor que mais se repete é também de 7 *ms* e o tempo de resposta que central definido pela moda é de aproximadamente 7 *ms*. Essas informações apontam para uma conexão com tempo de resposta rápido. Como pode ser visto, o tempo mínimo em cada observação foi de 4 *ms* e o máximo na casa dos 21 *ms*.

Tabela 12 – Tabela de resultados *RTT* usando *CoAP*.

CoAP	**Amostra 1**	**Amostra 2**	**Amostra 3**	**Amostra 4**	**Amostra 5**
Média	6,65	7,65	7,45	7,79	7,48
Mediana	6	7	7	7	7
Moda	6	7	6	7	6
Variação	1,91	4,09	3,77	4,21	5,16
Desvio Padrão	1,38	2,02	1,94	2,05	2,27
Mínimo	4	4	4	5	6
Máximo	14	18	15	17	21
Quartil 1	6	6	6	6,75	6
Quartil 2	7	7	7	7	7
Quartil 3	8	9	8,25	9	8
Distância Inter-Quartil	1	3	2,25	2,25	2
Distância Semi-Quartil	0,5	1,5	1,125	1,125	1

Com a finalidade de entender melhor a distribuição dos dados coletados sobre o *RTT* desta conexão, calculou-se índices de variabilidade, desvios e dispersão. O desvio padrão, calculado como a raiz quadrada da variância para manter a mesma unidade de medida e que indica a dispersão em torno da média amostral a partir de de uma coleta aleatória, indicou que dado uma observação qualquer no conjunto da amostra, o seu valor pode diferir da média em mais ou menos 2 *ms*, o que ainda é um tempo de resposta rápido.

Além disso, foram calculados os quantis, para que facilite o entendimento da distribuição probabilística. O Quartil 1 aponta que 25% das observações tiveram tempo de resposta abaixo dos 7 *ms*. O Quartil 2 indica que 50% tiveram *RTT* abaixo dos 8ms. Já o Quartil 3 demonstra que 75% foram abaixo de 9 *ms*, o que confirma uma conexão rápida e estável. A Figura 34 exibe uma distribuição da frequência de tempo de resposta de forma visual.Dessa maneira, é notável que os maiores tempos de resposta acontecem com menos frequência e os menores também, sugerindo que a maior parte das observações estão em torno da média.

Figura 34 – Histograma *CoAP*

6.3.2.1 Resultados *RTT* do protocolo *CoAP* com *DTLS*

O segundo experimento realizado com o protocolo *CoAP* foi a execução do algoritmo que faz a requisição com segurança *DTLS*, utilizando mensagens do tipo confirmável. Dessa maneira, denominou-se o experimento de *CoAP* com *DTLS*.

Assim, a Tabela 13 exibe os principais índices estatísticos calculados a partir das observações das cinco amostras neste cenário. Analisando os índices de tendência centrais, percebe-se um equilíbrio entre eles. A média de tempo de resposta de uma requisição nesse cenário é de aproximadamente 10 *ms*, o valor que mais se repete é também de 9 *ms* e o tempo de resposta que central definido pela moda é de aproximadamente 9 *ms*. Essas informações apontam para uma conexão com tempo de resposta rápido. Além disso, como pode ser visto na Tabela 13, o tempo mínimo em cada observação foi de 6 *ms* e o máximo na casa dos 27 *ms*.

Tabela 13 – Tabela de resultados *RTT* usando *CoAP* com *DTLS*.

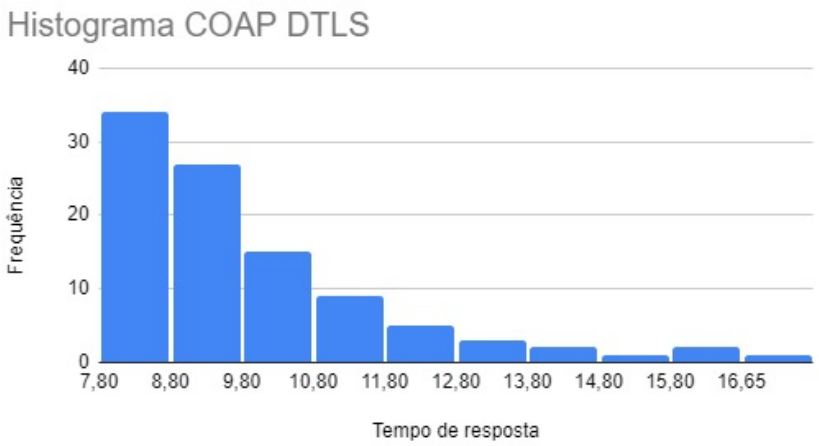
CoAP* com *DTLS	**Amostra 1**	**Amostra 2**	**Amostra 3**	**Amostra 4**	**Amostra 5**
Média	10,13	9,52	10,10	9,58	9,48
Mediana	9	9	9	9	9
Moda	9	8	9	8	8
Variação	7,77	3,48	6,96	3,22	2,72
Desvio Padrão	2,79	1,87	2,64	1,79	1,65
Mínimo	7	6	8	7	8
Máximo	21	17	27	15	16
Quartil 1	8	8	9	8	8
Quartil 2	9	9	10	9	9
Quartil 3	11	10	11	10	10
Distância Inter-Quartil	3	2	2	2	2
Distância Semi-Quartil	1,5	1	1	1	1

Com a finalidade de entender melhor a distribuição dos dados coletados sobre o *RTT* desta conexão, calculou-se índices de variabilidade, desvios e dispersão. O desvio padrão que indica a dispersão em torno da média amostral a partir de de uma coleta aleatória, indicou que dado uma observação qualquer no conjunto da amostra, o seu valor pode diferir da média em mais ou menos 3 *ms*, o que ainda é um tempo de resposta rápido.

Foram calculados os quantis para que facilite o entendimento da distribuição probabilística. O Quartil 1 aponta que 25% das observações tiveram tempo de resposta abaixo dos 9 *ms*. O Quartil 2 indica que 50% tiveram *RTT* abaixo dos 10 *ms*. Já o Quartil 3 corrobora com 75% abaixo de 11ms, o que confirma uma conexão rápida e estável.

A Figura 35 exibe uma distribuição da frequência de tempo de resposta de forma visual, é notável que os maiores tempos de resposta acontecem com menos frequência e os menores estão em torno da média.

Figura 35 – Histograma *CoAP + DTLS*



6.3.2.2 Resultados *RTT* do protocolo *CoAP* com *DTLS* e *payload* de 1024 bytes

O terceiro experimento realizado com o protocolo *CoAP* foi a execução do algoritmo que faz a requisição com segurança *DTLS*, *payload* de 1024 bytes e utiliza mensagens do tipo confirmável. Assim, denominou-se o experimento de *CoAP* com *DTLS* e *payload* de 1024 bytes.

Dessa maneira, a Tabela 14 exibe os principais índices estatísticos calculados a partir das observações das cinco amostras neste cenário. Analisando os índices de tendência centrais, percebe-se uma discrepância de valores entre eles. A média de tempo de resposta de uma requisição nesse cenário é de aproximadamente 20 *ms*, o valor que mais se repete é também de 14 *ms* e o tempo de resposta que central definido pela moda é de aproximadamente 17 *ms*. Essas informações apontam para uma conexão com tempo de resposta rápido na maioria das observações. Porém, como pode ser observado, o tempo mínimo em cada observação foi de 12 *ms* e o máximo na casa dos 333 *ms*, o que indica uma maior variabilidade.

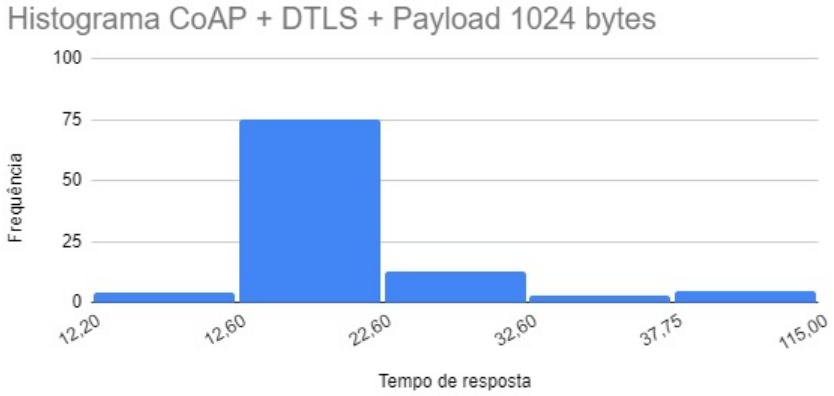
Tabela 14 – Tabela de resultados *RTT* usando *CoAP* com *DTLS*.

***CoAP +DTLS + payload* 1024 bytes**	**Amostra 1**	**Amostra 2**	**Amostra 3**	**Amostra 4**	**Amostra 5**
Média	27,57	18,01	19,87	19,19	18,29
Mediana	16	16	17	17	17
Moda	14	14	14	14	14
Variação	2127,88	37,95	117,77	49,19	31,22
Desvio Padrão	46,13	6,16	10,85	7,01	5,59
Mínimo	12	14	12	12	12
Máximo	333	47	103	46	46
Quartil 1	14	14	14	15	15
Quartil 2	15,5	16	17	17	17
Quartil 3	20,25	20	22	21	21
Distância Inter-Quartil	6,25	6	8	6,25	7
Distância Semi-Quartil	3,125	3	4	3,125	3,5

Com a finalidade de entender melhor a distribuição dos dados coletados sobre o *RTT* desta conexão, calculou-se índices de variabilidade e desvios e dispersão. O desvio padrão indicou que dado uma observação qualquer no conjunto da amostra, o seu valor pode diferir da média no pior caso em 46 *ms*, o que ainda é um tempo de resposta rápido.

Ademais, foram calculados os quantis para que facilite o entendimento da distribuição probabilística. O Quartil 1 aponta que 25% das observações tiveram tempo de resposta abaixo dos 15 *ms*. O Quartil 2 indica que 50% tiveram *RTT* abaixo dos 17 *ms*. Já o Quartil 3 corrobora com 75% abaixo de 22 *ms*, o que confirma uma conexão rápida e estável. Assim, a Figura 36 exibe uma distribuição da frequência de tempo de resposta de forma visual e é notável que os maiores tempos de resposta acontecem com menos frequência e os menores estão em torno da média.

Figura 36 – Histograma *CoAP* + *DTLS* + *Payload* 1024 bytes



6.3.2.3 Análise comparativa dos experimentos com *CoAP*

Ao completar todos os experimentos com o protocolo *CoAP* e coletar os resultados e gerar dados estatísticos, analisou-se comparativamente os resultados entre os tempos de resposta de cada cenário. A Tabela 15 exibe a frequência em porcentagem de observações em cada categoria de tempo de resposta. A partir dela é possível identificar com clareza um aumento no tempo de resposta a medida que é adicionada uma camada de proteção, por conta da utilização do *DTLS*, e aumenta-se o nível confidencialidade dos dados trafegados nos pacotes. Entretanto, nota-se uma sobrecarga e além disso, no experimento em que foi adicionando dados ao tamanho do *payload*, percebeu-se esforço ainda maior para enviar e receber uma mensagem.

Dessa maneira, os dados demonstram que 73% das mensagens sem criptografia apresentaram *RTT* entre 4 e 8 *ms* e somente 1% das observações atingiram entre 16 e 32 *ms*. Ao adicionar

Tabela 15 – Tabela da distribuição da frequência de *RTT CoAP*.

Categorias *RTT*	***CoAP***	***CoAP + DTLS***	***CoAP + DTLS + Payload* 1kb**
4-8ms	73%	21%	-
8-16ms	26%	77%	39%
16-32ms	1%	2%	54%
32-64ms	-	-	5%
64-128ms	-	-	2%

a camada de proteção, apenas 21% apresentaram resposta entre 4 e 8 *ms*, as observações entre 8 e 16 *ms* foram 77% das ocorrências. Por fim, adicionando uma quantidade significativa de dados ao pacote, não houve *RTT* menor que 8 *ms*, 93% das medições ficaram entre 8 e 32 *ms*, 5% entre 32 e 64 *ms* e somente 2% entre 64 e 128 *ms*.

Assim, tendo em vista que nos três experimentos utilizando *CoAP* os valores de índices de tendencias centrais apresentaram valores semelhantes, considerou-se a média como índice representativo das amostras. O gráfico de *boxplot* da Figura 37 expõe a distribuição das amostras.

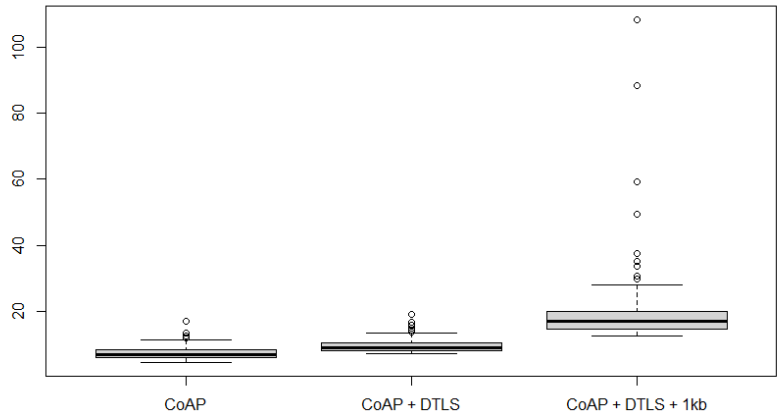


Figura 37 – *Boxplot CoAP*

7

Conclusões e considerações Finais.

Este trabalho teve como objetivo verificar a viabilidade do uso de mecanismos de proteção à privacidade nos protocolos da Camada de Aplicação utilizados em redes *IoT* no contexto de *smart home*. Para atender ao propósito deste estudo de forma propícia, foi necessário analisar a comunicação utilizando esses protocolos através dos critérios de usabilidade, performance e eficiência energética. Assim, essa proteção conta com a finalidade de garantir a privacidade no ambiente doméstico, garantido a facilidade de utilização, bom desempenho e custo baixo. Para isto, foram definidos como objetivos específicos:

- Levantar as principais vulnerabilidades relacionadas à privacidade na Camada de Aplicação de redes *IoT*;
- Levantar os principais protocolos da Camada de Aplicação que rodam sobre *TCP* e *UDP* e suas opções de confidencialidade para redes *smart home*;
- Analisar os possíveis impactos de um tráfego vulnerável no contexto de *smart home*;
- Verificar, por prototipação, a avaliação dos mecanismos de privacidade, utilizando uma metodologia para aferimento de tempo de resposta, usabilidade e aumento do consumo de corrente elétrica;
- Apurar o cenário viável para que a camada de aplicação possa prover privacidade em dispositivos de *smart home*.

Primeiramente foi realizado um Mapeamento Sistemático da Literatura. Dessa maneira, foram incluídos os trabalhos que utilizassem algum mecanismo que provesse maior privacidade ou detectasse vulnerabilidades, na Camada de Aplicação de redes *smart home*. Além disso, buscou-se trabalhos que dissertassem sobre os principais protocolos *IoT* utilizados pra automações residenciais. Este mapeamento foi necessário para embasar os experimentos que seriam

necessários para analisar o problema suscitado, que derivou os objetivos dessa pesquisa. Em seguida, foi realizada uma Revisão Sistemática da Literatura e como critério de seleção foram incluídos os trabalhos que utilizassem métodos de avaliação dos principais protocolos e suas camadas de segurança. Ao término da revisão, decidiu-se analisar um protocolo da Camada de Aplicação que utilizasse *TCP*. Assim, foi elencado o *MQTT* e outro que utilizasse *UDP* para transportar os pacotes, e neste caso definiu-se o *CoAP*.

Os experimentos não foram realizados com a finalidade de comparar o *MQTT* com o *CoAP*, mas sim qual seria a sobrecarga em cada um deles ao utilizar *TLS* e *DTLS*, respectivamente como suas camadas de proteção. Os resultados evidenciaram que utilizando *CoAP*, o aumento de consumo energético em cada conexão, no período de um mês, foi de menos de 1 centavo de reais ao utilizar o *DTLS* com *payload* pequeno. Além disso, a diferença entre a adoção de segurança na comunicação no cenário com maior sobrecarga (*CoAP* + *DTLS* + *payload* de 1 *kb*) foi de 2 centavos em um mês, quando comparado ao *CoAP* sem segurança, o que caracteriza um aumento de 20%, mas financeiramente irrisório. Já nos experimentos com *MQTT*, ao comparar a utilização do *MQTT* com *QoS* 0 e sem segurança com o cenário mais custoso (*MQTT* com *QoS* 2 + *TLS* + *payload* 1 *kb*), o aumento de consumo foi de 7 centavos de reais em um mês. Partindo de 12 centavos para 19 centavos, um aumento de 58,33% em um mês de utilização de cada dispositivo. Embora em termos de porcentagem o aumento tenha sido expressivo, o custo de 7 centavos para ter segurança e qualidade de serviço ainda é viável.

Para garantir que o incremento de proteção à privacidade não inviabilize o uso dos protocolos de segurança, é preciso analisar se ao adicionar proteção não se acrescenta nenhuma dificuldade de uso para o usuário final, intrínseca ao processo de adição de camadas. Para tal, criou-se uma metodologia de pontuação de dificuldade para cada etapa de configuração sem segurança e comparou-se com as etapas pra geração das chaves e certificados da criptografia assimétrica, assim como as configurações inerentes ao processo. Os resultados demonstraram que tanto para o *MQTT* como para o *CoAP*, o nível de dificuldade em pontos para uma utilização sem segurança foi de 4 pontos, e ao utilizar segurança aumentou-se para 24 pontos de dificuldade. Este aumento torna-se significativo para quem proverá a solução de segurança, entretanto, uma vez configurados, os protocolos sob proteção são completamente transparentes ao usuário final. Dessa forma, isto torna viável o uso de segurança quando analisado através do prisma da usabilidade.

Ademais, ainda que factível através das observações dos critérios já analisados, o tempo de resposta das requisições também foi avaliado estatisticamente. Dessa maneira, finalidade dessa avaliação foi de elucidar os questionamentos sobre a utilização do *TLS* e *DTLS* como segurança dos protocolos *MQTT* e *CoAP* num ambiente de *smart home*. Os experimentos de tempo de resposta no *MQTT* demonstraram que ao utilizar as medianas das amostras como índice, observou-se:

1. Apenas adicionando o *TLS* no nível de *Qos* 0, a sobrecarga é de 17%;

2. Do *QoS* 0 com *TLS* para o *Qos 1* com *TLS* a sobrecarga foi de 33%;
3. Do *QoS* 1 com *TLS* para *Qos* 2 com *TLS* a sobrecarga foi de 1500%;
4. Acrescendo o *payload* para 1024 bytes e utilizando *Qos* 2 com *TLS* a sobrecarga foi de 15%.

Dado o exposto, ainda que a diferença do tempo de *RTT* do cenário mais rápido para o mais lento tenha sido consideravelmente alta, analisando pelo prisma do contexto da uma *smart home*, o tempo de resposta em todos os casos foi menor que 1 *s*. Assim, é viável a utilização de segurança através do *TLS* com garantia de entrega de pacotes e um *payload* de 1024 bytes.

Os experimentos realizados para o protocolo *CoAP* demonstraram que:

1. Apenas adicionando o *DTLS* , a sobrecarga é de 32,43%;
2. Acrescendo o *payload* para 1024 bytes e utilizando *DTLS* a sobrecarga foi de 178%, em relação ao anterior;
3. Comparando o uso do *DTLS*, e *DTLS* + *payload* de 1 *kb* a sobrecarga foi de 110%

Posto isso, ainda que a diferença do tempo de *RTT* do cenário mais rápido para o mais lento tenha sido consideravelmente elevada, analisando pela óptica de uma *smart home*, o tempo de resposta em todos os casos também foi menor que um segundo. Dessa maneira, é viável a utilização de segurança através do*DTLS* com confirmação de entrega de pacotes e um *payload* de 1024 bytes.

7.1 Contribuições

As principais contribuições deste trabalho foram analisar o consumo energético através de um método isolado de aferimento e co-relacionar usabilidade ao incremento de segurança no processo de desenvolvimento. Além disso, calcular o custo (financeiro) de consumo de energia de um dispositivo seguro no ambiente de *smart home* e prover análise de viabilidade de uso da camada de segurança no ambiente doméstico através de análise estatística, utilizando um *SoC* de baixo custo e amplamente difundido.

7.2 Dificuldades e Limitações

Algumas dificuldades e limitações foram encontradas ao decorrer do trabalho. A primeira foi a inutilização da *IDE* do *Arduino* para escrever os códigos dos experimentos, pois, algumas bibliotecas disponíveis nesta *IDE* eram muito básicas e não disponibilizavam camadas de segurança com *TLS* e *DTLS*. Por esse motivo, os códigos foram escritos utilizando a *IDE Esp-IDF*, software utilizado para escrever as bibliotecas nativas do *ESP 32*.

7.3 Trabalhos Futuros

Nesta seção são levantados algumas possibilidades de trabalhos futuros para dar sequência e aprofundar o estudo:

- Analisar a viabilidade (desempenho, custo e usabilidade) utilizando diferentes *cipher suites* do *TLS* e *DTLS*;
- Analisar se a utilização do *Cryptographic Hardware Acceleration (CHA)*, um núcleo de aceleração de criptografia, na arquitetura dos ESPs32 acarretaria em resultados mais performáticos do que quando comparados com outro microcontroladores sem esta tecnologia;
- Implementar novos experimentos utilizando outros protocolos da Camada de Aplicação como por exemplo: *XMPP*, *AMQP*, *DDS* entre outros;
- Realizar uma análise mais detalhada no lado *server* ou *broker*, já que este trabalho focou no gargalo pelo lado cliente das aplicações.

Referências

ADAFRUIT. *INA219 High Side DC Current Sensor Breakout - 26V ±3.2A Max - STEMMA QT*. [S.l.], 2021. Disponível em: <https://www.adafruit.com/product/904>. Acesso em: 06 dez. 2021. Citado na página 51.

AITZAOUIAT, C. E. et al. Machine learning based prediction and modeling in healthcare secured internet of things. *Mobile Networks and Applications*, Springer, v. 27, n. 1, p. 84–95, 2022. Citado na página 17.

AMQP. *Advanced Message Queuing Protocol*. [S.l.], 2020. Disponível em: <https://www.amqp.org>. Acesso em: 06 dez. 2021. Citado na página 18.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer networks*, Elsevier, v. 54, n. 15, p. 2787–2805, 2010. Citado na página 19.

BASTOS, D.; SHACKLETON, M.; EL-MOUSSA, F. Internet of things: A survey of technologies and security risks in smart home and city environments. IET, 2018. Citado na página 20.

BING, K. et al. Design of an internet of things-based smart home system. In: IEEE. *2011 2nd International Conference on Intelligent Control and Information Processing*. [S.l.], 2011. v. 2, p. 921–924. Citado na página 19.

CAMBRIDGE, D. *Maker*. [S.l.], 2021. Disponível em: <https://dictionary.cambridge.org/pt/dicionario/ingles/maker>. Acesso em: 01 nov. 2021. Citado na página 23.

CHAUDHARY, S. et al. Craiot: Concept, review and application (s) of iot. In: IEEE. *2019 4th international conference on internet of things: Smart innovation and usages (IoT-SIU)*. [S.l.], 2019. p. 1–4. Citado na página 20.

DATTA, P.; SHARMA, B. A survey on iot architectures, protocols, security and smart city based applications. In: IEEE. *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. [S.l.], 2017. p. 1–5. Citado na página 20.

DUBRAVAC, S.; RATTI, C. The internet of things: evolution or revolution? *AIG White Paper*, 2015. Citado na página 12.

ECLIPSE, F. *Eclipse Mosquitto*. [S.l.], 2021. Disponível em: <https://mosquitto.org/>. Acesso em: 06 dez. 2021. Citado na página 55.

ESPRESSIF. *ESP32*. 2021. Disponível em: <<https://www.espressif.com/en/products/socs/esp32>.> Citado 2 vezes nas páginas 49 e 53.

FLOP, F. *Medindo corrente e tensão com o módulo INA219*. [S.l.], 2017. Disponível em: <https://www.filipeflop.com/blog/medindo-corrente-e-tensao-modulo-ina219/>. Acesso em: 06 dez. 2021. Citado na página 47.

FOUNDATION, R. P. *Raspberry Pi 3 Model B*. [S.l.], 2021. Disponível em: <https://www.raspberrypi.com/products/raspberry-pi-3-model-b/>. Acesso em: 06 dez. 2021. Citado na página 49.

GARTNER. *Gartner Says that IoT is top 8 Supply Chain Technology Trends in 2019*. [S.l.], 2019. Disponível em: <https://www.gartner.com/en/newsroom/press-releases/2019-04-24-gartner-identifies-the-top-8-supply-chain-technology-0>. Acesso em: 05 aug. 2021. Citado 2 vezes nas páginas 12 e 13.

HIVEMQ. *MQTT Essentials*. [S.l.], 2022. Disponível em: <https://www.hivemq.com/mqtt/mqtt-protocol/>. Acesso em: 01 jan. 2022. Citado 2 vezes nas páginas 32 e 35.

INTELBRAS. *Datasheet Twibi Giga*. [S.l.], 2021. Disponível em: <https://www.intelbras.com/pt-br/roteador-wireless-mesh-twibi-giga#beneficios>. Acesso em: 06 dez. 2021. Citado na página 50.

ISO, I. 9241-11: 1998 ergonomic requirements for office work with visual display terminals (vdts)–part 11: Guidance on usability. *Geneve, CH: ISO*, 1998. Citado na página 23.

JIN, H.; KUMAR, S.; HONG, J. Providing architectural support for building privacy-sensitive smart home applications. In: *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. [S.l.: s.n.], 2020. p. 212–217. Citado 3 vezes nas páginas 28, 30 e 31.

KING, N. Smart home–a definition. *Intertek Research and Testing Center*, p. 1–6, 2003. Citado na página 12.

KITCHENHAM, B. © kitchenham, 2004 procedures for performing systematic reviews. Citeseer, 2004. Citado na página 25.

LAINE, M.; SÄILÄ, K. Performance evaluation of xmpp on the web. *Aalto Univ. Tech. Rep.*, Citeseer, 2012. Citado na página 19.

LUETH, K. L. *State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time ?* [S.l.], 2020. Disponível em: <https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/>. Acesso em: 15 jan. 2021. Citado 2 vezes nas páginas 13 e 16.

MQTT. *MQTT: The Standard for IoT Messaging*. [S.l.], 2022. Disponível em: <https://mqtt.org/>. Acesso em: 01 feb. 2022. Citado na página 34.

NANO, A. *Arduino Nano*. [S.l.], 2021. Disponível em: <https://store.arduino.cc/products/arduino-nano>. Acesso em: 06 dez. 2021. Citado 2 vezes nas páginas 45 e 46.

NEISSE, R. et al. Dynamic context-aware scalable and trust-based iot security, privacy framework. *Chapter in Internet of Things Applications-From Research and Innovation to Market Deployment, IERC Cluster Book*, 2014. Citado na página 21.

NIELSEN, J.; BUDIU, R. *Mobile usability*. [S.l.]: MITP-Verlags GmbH & Co. KG, 2013. Citado 2 vezes nas páginas 24 e 62.

OGONJI, M. M.; OKEYO, G.; WAFULA, J. M. A survey on privacy and security of internet of things. *Computer Science Review*, Elsevier, v. 38, p. 100312, 2020. Citado 2 vezes nas páginas 21 e 22.

PANNUTO, P. et al. A modular and adaptive architecture for building applications with connected devices. In: IEEE. *2018 IEEE International Conference on Industrial Internet (ICII)*. [S.l.], 2018. p. 1–12. Citado 3 vezes nas páginas 29, 30 e 31.

POHL, M. et al. Performance evaluation of application layer protocols for the internet-of-things. In: IEEE. *2018 Sixth International Conference on Enterprise Systems (ES)*. [S.l.], 2018. p. 180–187. Citado 4 vezes nas páginas 27, 29, 30 e 31.

PRAYOGO, S. S.; MUKHLIS, Y.; YAKTI, B. K. The use and performance of mqtt and coap as internet of things application protocol using nodemcu esp8266. In: IEEE. *2019 Fourth International Conference on Informatics and Computing (ICIC)*. [S.l.], 2019. p. 1–5. Citado 5 vezes nas páginas 27, 29, 30, 31 e 48.

PROTSKAYA, Y.; VELTRI, L. Broker bridging mechanism for providing anonymity in mqtt. In: IEEE. *2019 10th International Conference on Networks of the Future (NoF)*. [S.l.], 2019. p. 110–113. Citado 3 vezes nas páginas 28, 30 e 31.

RAO, T. V. N.; SHAIK, K. S.; REDDY, A. J. R. Design of architecture for efficient integration of internet of things and cloud computing. *International Journal of Advanced Research in Computer Science*, International Journal of Advanced Research in Computer Science, v. 8, n. 3, 2017. Citado na página 17.

RAZA, S. et al. Securesense: End-to-end secure communication architecture for the cloud-connected internet of things. *Future Generation Computer Systems*, Elsevier, v. 77, p. 40–51, 2017. Citado 3 vezes nas páginas 28, 30 e 31.

SANTOS, M. C. Estudos sobre educação a distância e internet das coisas: perspectivas, possibilidades e desafios. In: *24º Congresso Internacional ABED de Educação a Distância. Florianópolis*. [S.l.: s.n.], 2018. Citado na página 19.

SELIEM, M.; ELGAZZAR, K.; KHALIL, K. Towards privacy preserving iot environments: a survey. *Wireless Communications and Mobile Computing*, Hindawi, v. 2018, 2018. Citado 2 vezes nas páginas 22 e 23.

SHELBY, Z.; HARTKE, K.; BORMANN, C. *The constrained application protocol (CoAP)*. [S.l.], 2014. Citado 3 vezes nas páginas 37, 38 e 43.

SIDDIQUI, F. et al. Secure and lightweight communication in heterogeneous iot environments. *Internet of Things*, Elsevier, p. 100093, 2019. Citado 4 vezes nas páginas 27, 29, 30 e 31.

SPIESS, P. et al. Soa-based integration of the internet of things in enterprise services. In: IEEE. *2009 IEEE international conference on web services*. [S.l.], 2009. p. 968–975. Citado na página 21.

SURYANEGARA, M.; ASVIAL, M.; RAHARYA, N. System engineering approach to the communications technology at unmanned aircraft system (uas). In: IEEE. *2015 IEEE International Symposium on Systems Engineering (ISSE)*. [S.l.], 2015. p. 475–480. Citado na página 12.

TEDESCO, D.; TULLIS, T. A comparison of methods for eliciting post-task subjective ratings in usability testing. *Usability Professionals Association (UPA)*, v. 2006, p. 1–9, 2006. Citado na página 62.

TEXAS, I. *INA219 Data sheet*. [S.l.], 2022. Disponível em: <https://www.ti.com/product/INA219/>. Acesso em: 01 jan. 2022. Citado na página 47.

THANGAVEL, D. et al. Performance evaluation of mqtt and coap via a common middleware. In: IEEE. *2014 IEEE ninth international conference on intelligent sensors, sensor networks and information processing (ISSNIP)*. [S.l.], 2014. p. 1–6. Citado na página 18.