Pós-Graduação em Ciência da Computação

Thays Melo de Moraes

Avaliação de Desempenho de Protocolos de Comunicação para Aplicações IoT

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2019

Thays Melo de Moraes

Avaliação de Desempenho de Protocolos de Comunicação para Aplicações IoT

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação, do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Avaliação de Desempenho e Dependabilidade
Orientador: Eduardo Antônio Guimarães Tavares

Recife
2019

Catalogação na fonte
Bibliotecária Mariana de Souza Alves CRB4-2106

M827a Moraes, Thays Melo de
Avaliação de Desempenho de Protocolos de Comunicação para Aplicações IoT – 2019.
109f.: il., fig., tab.

Orientador: Eduardo Antônio Guimarães Tavares
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da computação. Recife, 2019.
Inclui referências e apêndices.

1. Avaliação de Desempenho e Dependabilidade 2. Internet das Coisas. 3. Protocolo AMQP. 4. Protocolo MQTT. I. Tavares, Eduardo Antônio Guimarães (orientador). II. Título.

004.029 CDD (22. ed.) UFPE-MEI 2019-148

Thays Melo de Moraes

“Avaliação de Desempenho de Protocolos de Comunicação para Aplicações IoT”

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

Aprovado em: 28 de junho de 2019.

_____

Orientador: Prof. Dr. Eduardo Antonio Guimarães Tavares

BANCA EXAMINADORA

_____

Prof. Dr. José Augusto Suruagy Monteiro
Centro de Informática / UFPE

_____

Prof. Dr. Ricardo Massa Ferreira Lima
Centro de Informática / UFPE

_____

Profa. Dra. Erica Teixeira Gomes de Sousa
Departamento de Computação / UFRPE

Dedico este trabalho à minha família, meu namorado, meu orientador e meus amigos que foram fundamentais diante de todas as dificuldades no decorrer desta caminhada.

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me dado a vida e pelas oportunidades que me ofereceu. Agradeço aos meus pais, meu irmão e todos da minha família, por todo o apoio que tive principalmente por compreenderem e me apoiarem nesta etapa da minha vida.

Agradeço ao meu amigo e namorado, Sergio, por toda a paciência que ele teve, por todas as conversas nos momentos de dúvida, por sempre me dar forças nas horas em que eu precisei, sempre acreditando em mim.

Agradeço ao Professor Eduardo, por ter acreditado e depositado sua confiança em mim ao me orientar, por ter paciência ao compartilhar seu conhecimento nas aulas e no decorrer deste período.

Agradeço a todos os professores com quem tive aulas durante o mestrado. Todos foram essenciais para minha formação. Agradeço à FACEPE pelos recursos fornecidos através da bolsa de pesquisa.

Agradeço a todos os meus amigos do laboratório de pesquisa da pós-graduação, que me ajudaram e me apoiaram. E a todos os meus amigos que convivem comigo, que me incentivaram a fazer o mestrado, enfim todos os que fazem parte da minha vida.

Obrigada por fazerem parte da minha vida.

"Transmita o que aprendeu. Força, mestria. Mas fraqueza, insensatez, fracasso também. Sim, fracasso acima de tudo. O maior professor, o fracasso é."
(Yoda, Star Wars: Os Últimos Jedi) (STAR..., 2017)

RESUMO

Nos últimos anos, um crescente número de sistemas para Internet das Coisas (IoT) foram desenvolvidos em um ritmo sem precedentes, e esse crescimento tende a continuar. Em consequência, espera-se também um aumento na quantidade de dispositivos conectados à Internet. Esses objetos podem ser implementados em diversas áreas de atuação como transporte, habitação, assistência médica, agricultura, entre outros. Dessa forma, múltiplos estudos são direcionados para essas áreas. Devido a características particulares presentes nestes equipamentos, como por exemplo, a sua heterogeneidade e recursos computacionais limitados, a comunicação de dispositivos IoT é uma função significativa dos sistemas, para os quais, protocolos de comunicação distintos têm sido propostos, geralmente sendo implementados diretamente entre os dispositivos ou através de servidores na nuvem. Assim, é importante adotar uma estrutura de comunicação que não sobrecarregue a rede e os equipamentos, pois muitos dos ambientes IoT além de possuírem características específicas, frequentemente comunicam-se através de redes sem fio e intermitentes. Ponderando essa questão, este trabalho apresenta uma avaliação de desempenho em protocolos de comunicação para aplicações IoT. Os protocolos selecionados para o estudo foram AMQP, CoAP e MQTT e a metodologia utilizada baseou-se em um projeto de experimentos. Dois experimentos foram realizados utilizando um sistema de sensores que transmitem informações através de uma rede sem fio para um servidor. O primeiro experimento avalia a rede em condições consideradas corriqueiras e o segundo experimento analisa o desempenho considerando falhas na conexão da rede. As métricas usadas para avaliação consistem no consumo de banda, tamanho da mensagem e perda de pacotes, como também, foi realizada uma análise da correlação entre essas variáveis. Os resultados apontam que o protocolo CoAP é o mais indicado nos experimentos propostos, apresentando os melhores resultados, apesar do valor para perda de pacotes ser um pouco maior comparado ao protocolo MQTT. Além disso, os resultados indicam uma correlação positiva entre o tamanho da mensagem e o consumo de banda.

Palavras-chaves: Avaliação de Desempenho. Internet das Coisas. AMQP. MQTT. CoAP.

ABSTRACT

In recent years, an increasing number of IoT (IoT) systems have been developed at an unprecedented pace, and this growth tends to continue. As a consequence, an increase in the number of devices connected to the Internet is also expected. These objects can be implemented in various areas such as transportation, housing, healthcare, agriculture, among others. Therefore, multiple studies are directed to these areas. Due to the particular characteristics present in these types of equipment, such as their heterogeneity and limited computational resources, IoT device communication is a significant function of the systems, for which distinct communication protocols are proposed, usually, implemented through servers in the cloud or directly between the devices themselves. In this way, it is important to adopt a communication structure that does not overload the network and equipment, as many IoT environments, besides having specific characteristics, often communicate over wireless and intermittent networks. Considering this issue, this thesis presents a performance evaluation of communication protocols for IoT applications. The protocols selected for the study were AMQP, CoAP and MQTT and the methodology used were based on a design of experiments. Two experiments were performed using a sensor system that transmits information over a wireless network to a server. The first experiment evaluates the network under usual conditions and the second experiment analyzes the performance considering failures in the network connection. The metrics used for evaluation consist of bandwidth consumption, message size, and packet loss, as well as a correlation analysis between these variables. The results indicate that the CoAP protocol is the most indicated in the proposed experiments, presenting the best results, although the value for packet loss is slightly higher compared to the MQTT protocol. Besides, the results indicate a positive correlation between message size and bandwidth consumption.

Key-words: Performance Evaluation. Internet of Things. AMQP. MQTT. CoAP.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACK	*Acknowledgment*
AMQP	*Advanced Message Queuing Protocol*
ANOVA	Análise de Variância
AP	*Access Point*
CoAP	*Constraint Application Protocol*
CON	*Conformable*
CoRE	*Constrained RESTful Environments*
DDS	*Data Distribution System*
DoE	*Design of Experiments*
HTTP	*Hypertext Transfer Protocol*
IBM	*International Business Machines*
IC	Intervalo de Confiança
IDE	Ambiente de Desenvolvimento Integrado
IETF	*Internet Engineering Task Force*
IoT	*Internet of Things*
LDR	Resistor Dependente de Luz
M2M	Máquina a Máquina
M2S	Máquina a Servidor
MQTT	*Message Queue Telemetry Transport*
MTTF	*Mean Time To Failure*
MTTR	*Mean Time To Repair*
NON	*Non-Conformable*
OASIS	*Organization for the Advancement of Structured Information Standards*
OSI	*Open System Interconnection*
PIR	Infravermelho Passivo
QoS	*Quality of Service*
REST	*Representational State Transfer*
RFC	*Request for Comments*
RFID	*Radio-Frequency IDentification*

RSSF	Rede de Sensores Sem Fio
RST	*Reset*
RTT	*Round Trip Time*
S2S	Servidor a Servidor
SBC	Computador de Placa Única
SO	Sistema Operacional
TCP	*Transmission Control Protocol*
TCP/IP	Protocolo de Controle de Transmissão/Protocolo de Controle de Transmissão
TTF	*Time to Failure*
TTR	*Time to Repair*
UDP	*User Datagram Protoco*l
URI	*Uniform Resource Identifier*
XMPP	*Extensible Messaging and Presence Protocol*

LISTA DE SÍMBOLOS

α	Nível de significância
l	Fator
k	Níveis do fator
s	Número de replicações
t_{TTF}	Tempo para falhar
t_{TTR}	Tempo para reparo
rnd	Número aleatório gerado a partir de uma distribuição uniforme
r^2	Coeficiente de determinação

SUMÁRIO

1 INTRODUÇÃO

O presente capítulo apresenta inicialmente a contextualização e a motivação no qual idealizou-se este trabalho. Em seguida, discorre a respeito dos objetivos geral e específicos a serem alcançados. Por fim, denota a descrição da estrutura desta dissertação.

1.1 CONTEXTUALIZAÇÃO E MOTIVAÇÃO

Consideráveis avanços tecnológicos transcorreram nos últimos anos em relação às tecnologias de informação e comunicação permitindo, por exemplo, o surgimento de dispositivos portáteis, veículos autônomos e tecnologia *wearable*. Essas tecnologias frequentemente comunicam-se através de redes sem fio, e nesse âmbito, encontram-se também *Radio-Frequency IDentification* (RFID) e Rede de Sensores Sem Fio (RSSF). Dentre os exemplos relatados e inúmeros outros existentes, uma característica concomitante entre eles são de oferecer algum tipo de serviço ao usuário com intuito de facilitar a execução de atividades, como também auxiliar na tomada de decisões a serem desempenhadas pelo ser humano.

Juntamente com esses avanços, ocorreu um considerável aumento no número de equipamentos conectados à Internet. Segundo Egham (2017), aproximadamente 20,4 bilhões de dispositivos estarão conectados e em uso no ano 2020. A Rede de Sensores Sem Fio (RSSF) cresce progressivamente para atender a novos desafios, onde uso deles nos sistemas, ocorre de forma invisível nos ambientes que nos cercam (GUBBI et al., 2013). É possível citar como exemplo, sistemas para *healthcare*, ou seja, aplicações para assistência médica, onde sensores de temperatura e frequência cardíaca, realizam o monitoramento dos pacientes, possibilitando um atendimento mais preciso (XU; HE; LI, 2014).

A conectividade e a comunicação desses dispositivos com outros dispositivos ou servidores compõem um conceito importante, denominado *Internet of Things* (IoT) (Internet das Coisas). O termo surgiu inicialmente no final da década de 1990 com Kevin Ashton, que propôs a definição para IoT como um conjunto de objetos conectados e comunicantes através da tecnologia RFID (LI; XU; ZHAO, 2015). Contudo, à medida que pesquisas foram sendo desenvolvidas ao longo dos anos, o significado desse termo passou por modificações e, atualmente, não existe uma definição comum e sobre o que IoT realmente engloba (WORTMANN; FLÜCHTER, 2015).

De acordo com Lee, Bae e Kim (2017), o conceito é simples e pode consistir em qualquer objeto conectado que se comunique usando uma rede, permitindo que as comunicações entre eles forneçam qualquer serviço para os usuários. Ademais, embora a definição do que significa “Coisas” tenha se alterado à medida que a tecnologia foi evoluindo, o objetivo determinante de obter e compartilhar uma informação sem a ajuda da intervenção humana permanece igual (GUBBI et al., 2013). O que é possível afirmar com relação à sua definição

e sua principal característica é a composição de objetos conectados à rede que interagem entre si ou diretamente com usuários e servidores.

Atualmente, a Internet das Coisas desempenha uma importante função em ambientes diversificados, como representado na Figura 1. Nesses setores que incluem transporte, educação, automação industrial, além de respostas a emergência como desastres naturais e provocados pelo homem, possuem como uma das principais finalidades a tomada de decisão o qual seja difícil ser realizada de forma humana (AL-FUQAHA et al., 2015). A Internet das Coisas também pode operar em conjunto com *big data*, onde sensores e atuadores geram grandes quantidades de dados e esses dados são usados para descobrir novas oportunidades de negócio, como também, na resolução de problemas em diversas áreas, como por exemplo, mudanças no comportamento de clientes dentro do mercado (LEE; LEE, 2015).

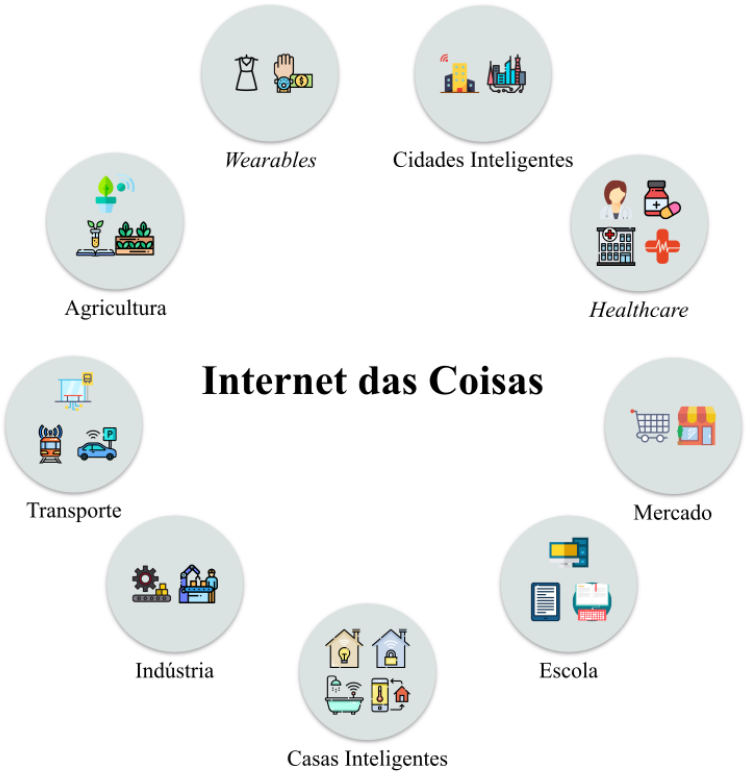


Figura 1 – Áreas de atuação da Internet das Coisas

Fonte: Adaptado de Al-Fuqaha et al. (2015)

Na agricultura, por meio de monitoramento do solo, é possível contribuir para um menor custo no processo de cultivo e melhor qualidade dos produtos. No âmbito das cidades inteligentes, através de sensores de rede sem fio, pode-se monitorar estacionamentos de veículos possibilitando que usuários passem menos tempo em busca de uma vaga ou encontrem seu carro rapidamente (ARASTEH et al., 2016). Dessa forma, em cada uma dessas áreas, a tecnologia voltada para Internet das Coisas pode contribuir de alguma

forma, estando sempre em desenvolvimento e possibilitando a abertura de muitas outras oportunidades (SINGH; TRIPATHI; JARA, 2014).

Como consequência desta diversidade de áreas que a Internet das Coisas pode auxiliar, existe também uma variedade em suas pesquisas para resolução de novos desafios que surgem em conjunto com as inovações. Um dessas pesquisas envolve a comunicação entre os dispositivos IoT, pois, devido à sua natureza remota e o continuado uso de redes sem fio para comunicabilidade, os sistemas IoT devem ser capazes de lidar com conexões potencialmente intermitentes e de baixa largura de banda (CHEN; KUNZ, 2016). Ademais, a Internet das Coisas transforma objetos considerados simples em dispositivos interconectados em sistemas inteligentes (YASSEIN; SHATNAWI et al., 2016). Contudo, esses dispositivos geralmente são heterogêneos e, de acordo com os requisitos da aplicação, é necessário um mecanismo de interação comum (YASSEIN; SHATNAWI et al., 2016).

Dentro dessa aposta, além de *softwares*, *frameworks* e arquiteturas, podemos citar um fator importante relacionado, o qual são os protocolos associados a esses ambientes. Frequentemente, um ambiente desenvolvido para *Internet of Things* necessita de uma lista de protocolos diferentes da pilha de protocolos utilizados em outros modelos como *Open System Interconnection* (OSI) e Protocolo de Controle de Transmissão/Protocolo de Controle de Transmissão (TCP/IP), como ilustrado na Figura 2 (ASIM, 2017). Nessa imagem podemos identificar as Camadas de Aplicação, Transporte, Internet e Física/Link e em cada uma delas existe um conjunto de protocolos associados. O presente estudo, irá focar nos protocolos da camada de aplicação para comunicação, onde de uma forma geral estes protocolos apenas substituem os protocolos de camada de aplicação TCP/IP na estrutura de IoT (ASIM, 2017).

Camadas	Protocolos
Camada de Aplicação	CoAP, MQTT, XMPP, AMQP, RESTful, Websockets
Camada de Transporte	UDP, DTLS
Camada de Internet	RPL, 6LoWPAN
Camada Física/Link	IEEE 802.15 Series, IEEE 802.11 Series

Figura 2 – Pilha de Protocolos IoT

Fonte: Adaptado de Asim (2017)

Protocolos de comunicação da camada de aplicação para aplicações IoT são extremamente relevantes, pois a maioria dos objetos são de natureza distinta, e para interagir precisam de uma estrutura comum em sua comunicação (JAIKAR; IYER, 2018). Além disso, para a transferência de informações mediante a coleta de dados através de sensores desti-

nado outros dispositivos e servidores, é requerido um protocolo que seja eficiente e capaz de trabalhar com recursos limitados (THANGAVEL et al., 2014). Em virtude disso, é importante a realização de pesquisas acerca do desempenho destes protocolos, para apontar quais deles são os mais indicados para cada ambiente.

Atualmente, alguns protocolos de comunicação foram projetados para atenderem a demanda proposta de sistemas IoT e comunicação Máquina a Máquina (M2M), como *Message Queue Telemetry Transport* (MQTT) (BANKS; GUPTA, 2015), *Advanced Message Queuing Protocol* (AMQP) (STANDARD, 2012) e *Constraint Application Protocol* (CoAP) (SHELBY; HARTKE; BORMANN, 2014). Esses protocolos citados têm um importante papel na contribuição do desempenho da comunicação, visto que muitos dispositivos, a exemplo dos sensores, e a própria rede, podem ter recursos limitados (THANGAVEL et al., 2014; NAIK, 2017).

Um profundo entendimento do sistema de Internet das Coisas e seus requisitos de compartilhamento de dados é essencial para selecionar um protocolo apropriado (NAIK, 2017). Por exemplo, uma sobrecarga pode impactar negativamente o desempenho do sistema se o protocolo adotado não for eficiente para a rede específica. Nos últimos anos, algumas pesquisas foram realizadas para avaliar os protocolos IoT no contexto do desempenho. No entanto, alguns problemas não foram resolvidos, como o comportamento do protocolo em caso de falhas de rede.

1.2 OBJETIVOS

O principal objetivo deste trabalho é avaliar o desempenho dos protocolos da camada de aplicação comumente utilizados em sistemas de Internet das Coisas, avaliando as métricas de consumo de banda, tamanho da mensagem e perda de pacotes.

Como objetivos específicos, este trabalho propõe:

- Propor uma metodologia para avaliação de desempenho de protocolos da camada de aplicação utilizados em aplicações IoT;
- Avaliar qual dos protocolos da camada de aplicação é o mais indicado para utilização do ambiente IoT projetado para o estudo;
- Avaliar a existência de correlação entre as métricas adotadas.

1.3 ESTRUTURA DA DISSERTAÇÃO

A seguir descreve-se o contexto dos capítulos que seguem posteriormente neste trabalho.

O Capítulo 2 fornece o referencial teórico necessário para compreensão deste trabalho. Primeiramente temos uma breve descrição a respeito do conceito de Internet das Coisas. Em seguida, são descritas informações sobre os protocolos estudados (AMQP, CoAP e

MQTT) englobando o formato da mensagem, arquitetura, comunicação e a forma de entrega das mensagens. Posteriormente, é exposta uma sucinta explicação a respeito da Avaliação de Desempenho em sistemas computacionais. Por fim, é explicado brevemente o conceito de *Design of Experiments* e apresentado o modelo de projeto de experimentos utilizado no estudo.

No Capítulo 3 é apresentado um resumo dos principais trabalhos encontrados na literatura, associados a esta dissertação e as principais diferenças das pesquisas com o presente estudo.

No Capítulo 4 é descrita a metodologia adotada para pesquisa. Inicialmente explicamos a configuração dos experimentos, e posteriormente é relatada as configurações de *hardware* e *software* utilizadas nos experimentos, como também as bibliotecas utilizadas nos algoritmos. Em seguida as ferramentas utilizadas para os experimentos e coleta dos dados são especificadas. Por fim explana-se a utilização da metodologia que é baseada no projeto de experimentos DoE.

O Capítulo 5 apresenta os resultados obtidos no presente estudo. O resultado de cada experimento é apresentado, sendo no primeiro experimento descrevendo os resultados para o consumo de banda e tamanho da mensagem e, no segundo experimento, além destas duas métricas também é descrita a perda de pacotes. Por fim é realizada uma discussão dos resultados.

O Capítulo 6 apresenta as conclusões do trabalho, limitações e trabalhos futuros.

2 REFERENCIAL TEÓRICO

O presente capítulo aponta os principais conceitos básicos dos assuntos explanados nesta dissertação. Primeiramente temos uma breve descrição a respeito do conceito de Internet das Coisas, complementando o que foi relatado no Capítulo 1. Em seguida, são descritas informações sobre os protocolos estudados (AMQP, CoAP e MQTT) englobando o formato da mensagem, arquitetura, comunicação e a forma de entrega das mensagens. Posteriormente, é exposta uma sucinta explicação a respeito da Avaliação de Desempenho em sistemas computacionais. Por fim, é explicado brevemente o conceito de Projeto de Experimentos e apresentado o modelo de projeto de experimentos utilizado no estudo.

2.1 INTERNET DAS COISAS

Atualmente estamos inseridos numa sociedade dependente das Tecnologias da Informação e Comunicação, e estas cada vez mais vem crescendo e tem se ampliado. A exemplo dos telefones, que antigamente tinham como principal função a de telefonar, atualmente são denominados smartphones e oferecem diversas funcionalidades diferenciadas aos usuários para o trabalho, a diversão ou até mesmo auxiliar na resolução de problemas, tornando muitas vezes a função de telefonar a menos importante.

Os smartphones assim como outros dispositivos estão inseridos no conceito de *Internet of Things* (Internet das Coisas). Historicamente, este conceito surgiu inicialmente na década de 1990 na apresentação de uma torradeira que estava conectada à rede e recebia requisições para o preparo de torradas (LI; XU; ZHAO, 2015; MATTERN; FLOERKEMEIER, 2010). Atualmente, a ideia é incorporar diversos dispositivos a rede, comunicando-se entre si com o objetivo de proporcionar ao usuário uma nova forma de ver o mundo e revolucionar suas vidas.

Além de incorporar esses dispositivos, o IoT transforma aqueles objetos muitas vezes considerados tradicionais em inteligentes, capacitando os recursos do objeto de ouvir, ver, e comunicar com outros dispositivos, como também incluindo uma computação inteligente (RATHORE et al., 2016).

Segundo (ROMAN; NAJERA; LOPEZ, 2011) o desenvolvimento do IoT faz com que tudo o que é real se torne virtual e essas entidades virtuais podem produzir e consumir serviços e colaborar para um objetivo comum, tornando-a localizável endereçável e legível dentro da Internet.

Desse modo, o desenvolvimento da Internet das Coisas, acaba englobando diversas tecnologias, aplicações e protocolos novos, projetados para atenderem às novas necessidades, e com isso cada vez mais o escopo aumenta, abrindo caminho para oportunidades de pesquisa, oferecendo desafios (MATTERN; FLOERKEMEIER, 2010; ROMAN; NAJERA; LOPEZ,

2011).

2.2 PROTOCOLOS

Atualmente existem diversos protocolos presentes na camada de aplicação, e em sua maioria, eles concentram-se em questões acerca da comunicação Máquina a Máquina (M2M). Al-Fuqaha et al. (2015) relata os cinco protocolos de comunicação (*Data Distribution System* (DDS), *Extensible Messaging and Presence Protocol* (XMPP), *Advanced Message Queuing Protocol* (AMQP), *Constraint Application Protocol* (CoAP) e *Message Queue Telemetry Transport* (MQTT)) mais indicados para a transmissão de mensagens entre dispositivos IoT. Outros autores como Yassein, Shatnawi et al. (2016), Mun, Dinh e Kwon (2016), Yokotani e Sasaki (2016), incluem em suas pesquisas *WebSocket* e HTTP, contudo esses protocolos não são específicos para comunicação de aplicações IoT.

Dos protocolos citados, esta dissertação seleciona três deles voltados essencialmente para ambientes de Internet das Coisas, que são: AMQP, CoAP e MQTT. Esses protocolos foram selecionados de acordo com alguns fatores, entre eles, o escopo do protocolo, a disponibilidade de código aberto e bibliotecas e o suporte da comunidade e documentação. Em virtude disso, o protocolo XMPP não foi utilizado devido ao fato de que este protocolo é voltado principalmente para *chats*, e o protocolo DDS dispõe de suporte à comunidade e disponibilidade de código limitada.

2.2.1 *Advanced Message Queuing Protocol*

Advanced Message Queuing Protocol (AMQP) é um protocolo de aplicação para troca de mensagens. De acordo com Frigieri, Mazzer e Parreira (2015), o AMQP é um protocolo de padrão aberto de mensagens de *middleware* baseado no paradigma de filas de mensagens orientadas a tópicos, em que produtos escritos para diferentes plataformas e em diferentes linguagens podem trocar mensagens. Conforme sua documentação, Standard (2012), o AMQP é um protocolo de internet aberto para mensagens de negócios. No padrão AMQP, os produtos de *middleware* escritos para diferentes plataformas e em diferentes idiomas podem enviar mensagens de um para outro (FERNANDES et al., 2013).

Esse protocolo foi criado em 2003 por John O'Hara no JPMorgan Chase (NAIK, 2017), e atualmente sua documentação padronizada encontra-se em *Organization for the Advancement of Structured Information Standards* (OASIS). O principal objetivo do AMQP é entregar as mensagens sem perda e fornecer segurança e interoperabilidade (PONNUSAMY; RAJAGOPALAN, 2018). Ele define um protocolo de nível de fio binário que permite a troca confiável de mensagens de negócios entre duas partes, possui uma arquitetura em camadas e a especificação é organizada como um conjunto de partes que reflete essa arquitetura (STANDARD, 2012).

O AMQP suporta a comunicação *Publisher/Subscriber* (Publicação/Assinatura), possui alguns níveis de serviço de entrega como até uma vez, ao menos uma vez e exatamente uma vez, semelhante ao protocolo MQTT (descrito posteriormente neste capítulo, na Seção 2.2.3). As mensagens trocadas entre os editores e assinantes são por meio do *Transmission Control Protocol* (TCP), e é fornecida uma conexão ponto-a-ponto confiável (PONNUSAMY; RAJAGOPALAN, 2018).

2.2.1.1 Formato da Mensagem

Uma mensagem AMQP possui diversas informações acerca da infraestrutura, regras de roteamento, política de armazenamento, entre outras informações (STANDARD, 2012; MAZZER; FRIGIERI; PARREIRA, 2018). A Figura 3 ilustra o formato da mensagem do protocolo AMQP.

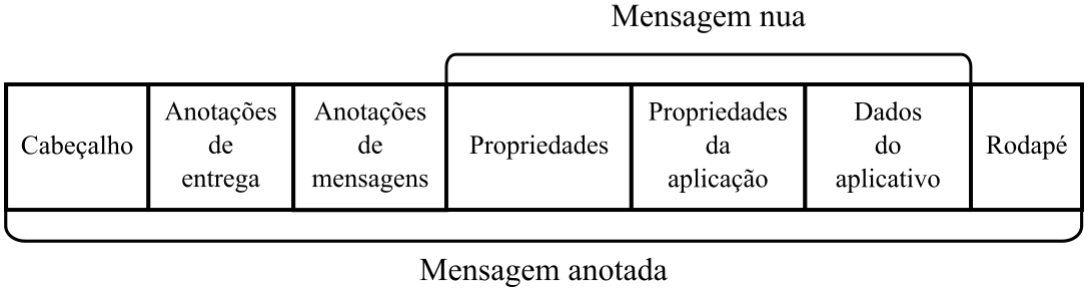


Figura 3 – Formato da mensagem AMQP

Fonte: Adaptado de Standard (2012)

De acordo com Standard (2012), uma mensagem comentada consiste em um conjunto de uma mensagem nua mais as seções para anotação, que podem ser classificadas em anotações que trafegam com a mensagem indefinidamente e as anotações que são consumidas pelo próximo nó. Ademais, temos o cabeçalho. No cabeçalho os parâmetros de entrega são transferidos incluindo informações sobre durabilidade, prioridade, tempo de vida, primeiro adquirente e contagem de entrega (AL-FUQAHA et al., 2015). Já a mensagem nua, consiste em três seções: propriedades, propriedades do aplicativo e dados do aplicativo, ou seja, o *Payload* (STANDARD, 2012).

2.2.1.2 Arquitetura e Funcionamento

A arquitetura do AMQP, é constituída por meio de uma comunicação *Publisher/Subscriber* (Publicação/Assinatura) composta por editores, *broker* e assinantes, como ilustrado na Figura 4.

Os publicadores, também chamados de produtores, que podem ser um sensor de temperatura, por exemplo, são os componentes que irão produzir os dados e enviá-los para o *broker*. No *broker* temos dois componentes importantes: as *queues* e a *exchange*.

Uma *queue* (fila), tem como papel realizar o armazenamento das mensagens tanto em memória quanto em disco, até que estas sejam enviadas em sequência para os respectivos consumidores (CARMO, 2012). A permanência desse armazenamento depende da implementação da fila de mensagens, seja para armazenar no disco ou exclusivamente na memória (VINOSKI, 2006).

Uma *exchange* possui como finalidade receber as mensagens dos produtores, e realizam o roteamento para as filas, a partir de um conjunto de regras ou critérios preestabelecidos (AL-FUQAHA et al., 2015; VINOSKI, 2006; CARMO, 2012).

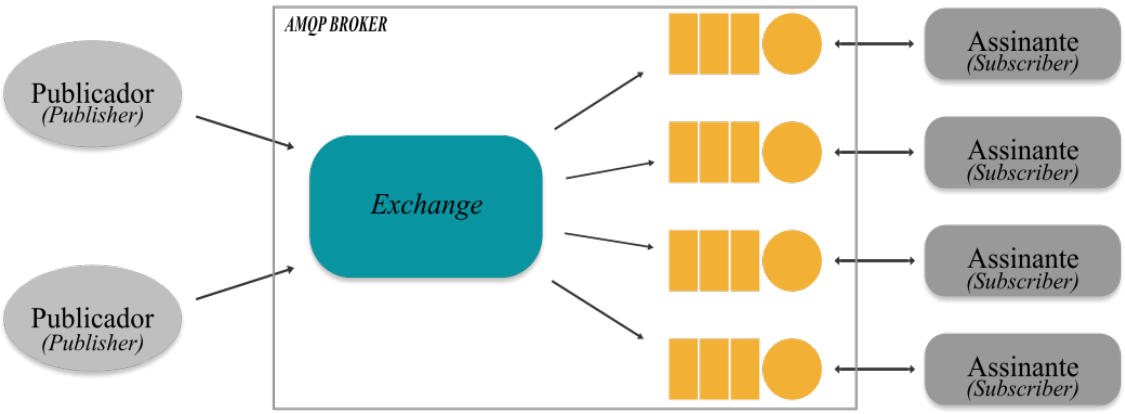


Figura 4 – Arquitetura AMQP

Fonte: Adaptado de Standard (2012)

2.2.1.3 Qualidade de Serviço

A Qualidade de Serviço no AMQP, ocorre durante a transmissão das mensagens por meio de uma *tag* de entrega, a qual também é utilizada para realizar o rastreamento do estado de entrega quando a mensagem está sendo transmitida.

Segundo Standard (2012), existem três tipos de garantia de entrega:

***At-least-once* (No máximo uma vez)** – Ocorre quando o aplicativo de envio atribui a *tag* de entrega como entregue antes da transferência da mensagem se iniciar. Ou seja, o remetente indica antes de transmitir a mensagem que deletou as informações sobre ela e, portanto, não será possível realizar a retransmissão destes dados. Se o destinatário receber, ele não precisará responder com a atualização do estado de entrega.

***At-most-once* (Pelo menos uma vez)** – Ocorre quando o aplicativo de envio não atribui a *tag* estado de entrega como *delivered* e o aplicativo de recebimento opte por liquidar imediatamente após o processamento da mensagem em vez de aguardar o remetente liquidar primeiro. Destarte, se a mensagem for perdida, o remetente irá

perceber que a *tag* de entrega não foi modificada e concluirá que a entrega foi perdida, executando assim, a retransmissão da mensagem. O destinatário por sua vez, poderá receber essa mensagem de forma duplicada.

***Exactly-once* (Exatamente uma vez)** – Ocorre quando o aplicativo de envio é liquidado (*settled=True*) no mesmo momento em que o destinatário atinge um estado de terminal, e o aplicativo de recebimento é liquidado (*settled=True*) quando o remetente se estabelece.

As Figuras a seguir ilustram o comportamento da garantia de entrega No máximo uma vez (Figura 5) e Pelo menos uma vez (Figura 6).

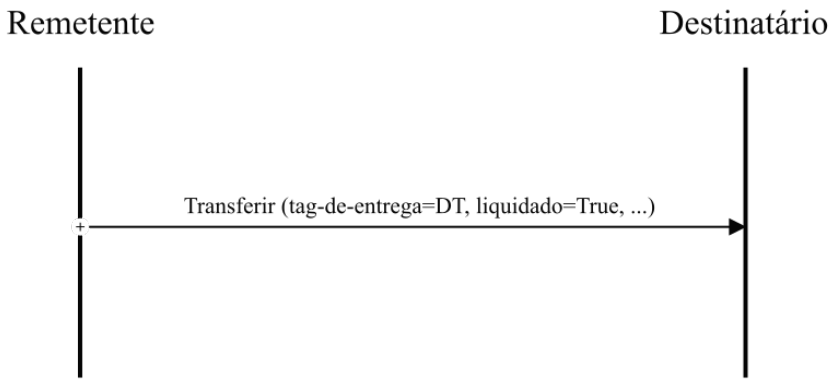


Figura 5 – Garantia de entrega no máximo uma vez AMQP

Fonte: Adaptado de Standard (2012)

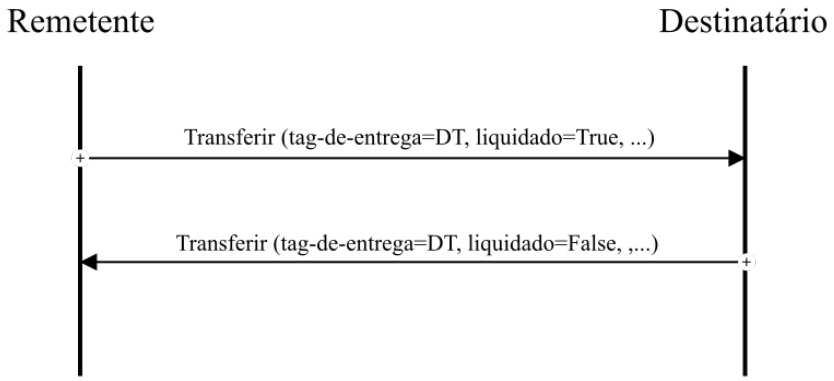


Figura 6 – Garantia de entrega pelo menos uma vez AMQP

Fonte: Adaptado de Standard (2012)

2.2.2 *Constrained Application Protocol*

Constraint Application Protocol (CoAP) é um protocolo que foi projetado para ser usado na comunicação de dispositivos com recursos limitados (THANGAVEL et al., 2014). Esse protocolo foi criado por um grupo da *Internet Engineering Task Force* (IETF) composto por Carstem Bormann, Andrew McGregor e Barry Leiba chamado *Constrained RESTful Environments* (CoRE) em 2010, e em 2014 sua documentação foi lançada na *Request for Comments* (RFC) 7252 (MARTINS; ZEM, 2016).

Um dos objetivos do projeto CoAP era o de minimizar a sobrecarga de mensagens e limitar a fragmentação de pacotes (CARO et al., 2013). O CoAP foi desenvolvido principalmente para interoperar com o *Hypertext Transfer Protocol* (HTTP) e *Web RESTful* por meio de *proxies* simples (NAIK, 2017).

De acordo com Shelby, Hartke e Bormann (2014), um dos principais objetivos do CoAP é projetar um protocolo web genérico para os requisitos especiais de ambientes restritos, especialmente, considerando energia, automação predial e outras aplicações M2M. Também, segundo Shelby, Hartke e Bormann (2014), o objetivo do CoAP não é simplesmente compactar o HTTP, mas sim, para realizar um subconjunto REST comum com o HTTP mais otimizado para aplicativos M2M.

2.2.2.1 Formato da Mensagem

Como mencionado anteriormente, o CoAP foi desenvolvido para ambientes restritos, assim a comunicação entre o cliente e servidor é realizada por meio de mensagens compactas. As mensagens CoAP são codificadas em um formato binário simples (SHELBY; HARTKE; BORMANN, 2014). O formato da mensagem começa com um cabeçalho de 4 bytes de tamanho fixo e é seguido por um valor Token de tamanho variável, que pode ter entre 0 e 8 bytes de comprimento (SHELBY; HARTKE; BORMANN, 2014). A Figura 7 apresenta o formato da mensagem CoAP.

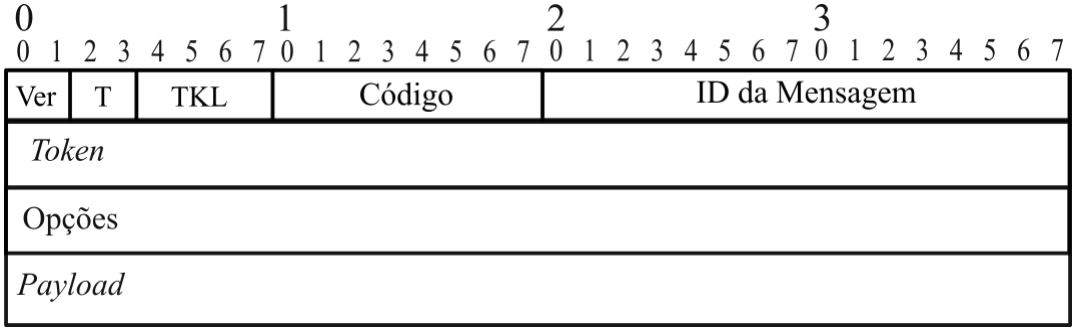


Figura 7 – Formato da mensagem CoAP

Fonte: Adaptado de Shelby, Hartke e Bormann (2014)

Segundo Shelby, Hartke e Bormann (2014), os campos do cabeçalho CoAP são definidos por:

Ver (Versão) – Indica a versão do CoAP que está sendo utilizada.

T (Tipo) – Indica qual o tipo de mensagem que está sendo transmitida: *Conformable*, *Non-Conformable*, *Acknowledgement*, *Reset*.

TKL (Tamanho do Token) – Indica o tamanho to Token.

Código – Indica o tipo de código. Seja um código de requisição (GET, POST, PUT, DELETE), ou um código de resposta de uma requisição.

ID da Mensagem – Este ID tem como objetivo detectar mensagens duplicadas e responder as mensagens, como por exemplo, responder mensagens com *Acknowledgement*.

2.2.2.2 Métodos

No CoAP, é proposto uma semântica semelhante ao HTTP, utilizando para si os métodos GET, POST, PUT e DELETE. Assim, esses métodos possuem propriedades semelhantes às propriedades dos métodos análogos ao HTTP, como por exemplo seguro, ou seja, somente para recuperação; e idempotente, isto é, possui o mesmo efeito ao ser chamado diversas vezes (JOSHI; KAUR, 2015; SHELBY; HARTKE; BORMANN, 2014). Contudo, ao contrário do HTTP que funciona por meio do TCP, o CoAP funciona por meio do *User Datagram Protocol* (UDP). Esses métodos citados são modificados para atender aos requisitos de Internet das Coisas, como baixo consumo de energia e operação na presença de links com e sem ruído (AL-FUQAHA et al., 2015). A seguir, temos a definição dos métodos que esse protocolo utiliza para a comunicação entre cliente e servidor de acordo com Shelby, Hartke e Bormann (2014):

GET – Recupera uma representação para as informações que correspondem ao recurso identificado pelo *Uniform Resource Identifier* (URI) da solicitação realizada pelo cliente. Esse método é seguro e idempotente.

POST – Solicita que a representação incluída no pedido seja processada resultando geralmente em uma criação ou atualização de um recurso. Esse método não é seguro e não é idempotente.

PUT – Solicita que um recurso identificado pela solicitação URI seja criado ou atualizado com uma representação em anexo. Esse método não é seguro, mas é idempotente.

DELETE – Solicita que o recurso identificado pelo URI seja excluído. Esse método não é seguro, mas é idempotente.

O Token é gerado pelo cliente e é utilizado para relacionar requisições e respostas. Após o Token vem o campo opções, se houver, e em seguida, vem o *Payload*, caso exista também.

A cada requisição realizada, é retornado um código de resposta. Esses códigos podem ser visualizados na Tabela 1.

Tabela 1 – Código de Resposta do CoAP

Código de Resposta	**Definição**
2.xx	**Sucesso**
2.01	*Created*
2.02	*Deleted*
2.03	*Valid*
2.04	*Changed*
2.05	*Content*
4.xx	**Erro no Cliente**
4.00	*Bad Request*
4.01	*Unauthorized*
4.02	*Bad Option*
4.03	*Forbidden*
4.04	*Not found*
4.05	*Method not allowed*
4.06	*Not Acceptable*
4.12	*Precondition Failed*
4.13	*Request Entity Too Large*
4.15	*Unsupported Content-Format*
5.xx	**Erro no Servidor**
5.00	*Internal Server Error*
5.01	*Not Implemented*
5.02	*Bad Gateway*
5.03	*Service Unavailable*
5.04	*Gateway Timeout*
5.05	*Proxying Not Supported*

Fonte: Adaptado de Shelby, Hartke e Bormann (2014)

2.2.2.3 Comunicação

De acordo com Yassein, Shatnawi et al. (2016), os recursos mais importantes no CoAP são simplicidade e confiabilidade, onde é oferecida a troca de mensagens de forma assíncrona. As mensagens são trocadas entre os pontos finais, e são utilizadas para transportar solicitações e respostas (SHELBY; HARTKE; BORMANN, 2014).

Como o CoAP trabalha com UDP e este não é inerentemente confiável, o CoAP fornece seu próprio mecanismo de confiabilidade, feito com o uso de mensagens *Conformable* (CON) e *Non-Conformable* (NON) (THANGAVEL et al., 2014). Esse mecanismo de confiabilidade é leve sem tentar recriar o conjunto completo de recursos de um transporte como o TCP (SHELBY; HARTKE; BORMANN, 2014).

Ao configurar a confiabilidade NON, o servidor não envia uma mensagem de ACK para o cliente confirmando que recebeu a requisição, como visualizado na Figura 8. Neste caso, dado a circunstância de que o servidor não possa processar uma mensagem, ele envia uma mensagem de *Reset* (RST) para o cliente com o ID da mensagem recebida (SHELBY; HARTKE; BORMANN, 2014).

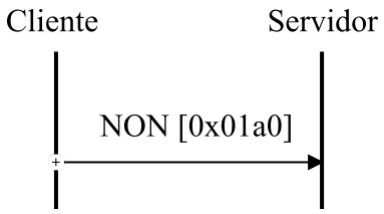


Figura 8 – Troca de Mensagens *Non-Conformable*

Fonte: Adaptado de Shelby, Hartke e Bormann (2014)

Quando se faz necessário que o remetente tenha uma confirmação de que a requisição foi recebida pelo destinatário, a confiabilidade é configurada como CON. Destarte, o servidor envia uma mensagem de ACK para o cliente, informando que a requisição foi recebida, como pode ser visto nas Figura 9 e 10.

Na configuração *Conformable* (CON), o envio da mensagem de ACK pode ter dois tipos diferentes: *Piggybacked* e *Separate Response*. No *Piggybacked*, a resposta da solicitação enviada pelo cliente é retornada no mesmo pacote do ACK, como pode ser visto na Figura 9, ou seja, simultaneamente com a mensagem de confirmação de recebimento do pacote, é enviada também a resposta da solicitação realizada. Diante disso, o cliente retransmitirá a solicitação se a mensagem de *Acknowledgment* (ACK) transportando a resposta for perdida (SHELBY; HARTKE; BORMANN, 2014).

Contudo, existem casos em que o processamento da requisição precise de um pouco mais de tempo para sua finalização. Neste contexto, caso a resposta continue *Piggybacked* é possível ocorrer uma quantidade considerável de retransmissão de pacotes desnecessariamente. A solução para este caso é por meio do *Separate Response* onde, nesse caso, uma resposta é enviada confirmando o recebimento da mensagem de requisição, como é apresentada na Figura 10. A mensagem que o servidor responde é um pacote ACK vazio para que o cliente possa parar de retransmitir a solicitação (SHELBY; HARTKE; BORMANN, 2014). Assim que o servidor finaliza o processamento dessa solicitação, a resposta é enviada para o cliente em uma nova mensagem. O servidor aguarda a confirmação de que a resposta chegou ao seu remetente e o cliente, por sua vez, ao receber a resposta da

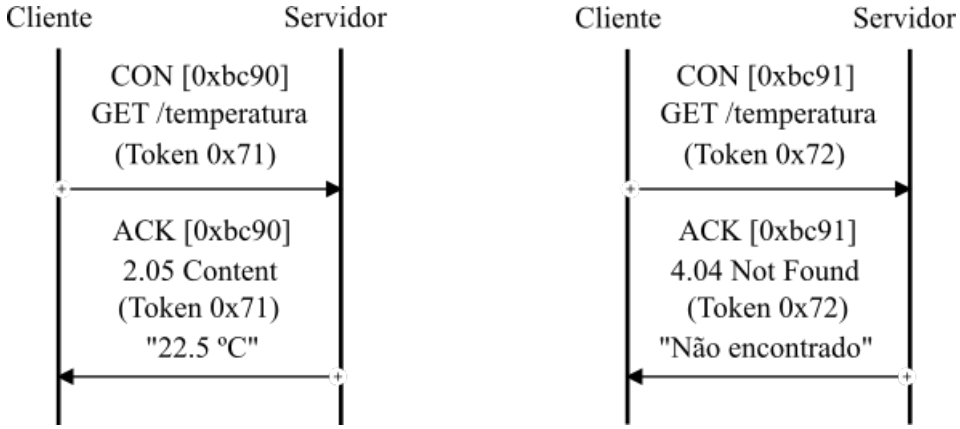


Figura 9 – Troca de mensagem *Conformable Piggybacked*

Fonte: Adaptado de Shelby, Hartke e Bormann (2014)

requisição realizada por ele, deverá enviar uma mensagem de ACK vazia, confirmando assim, que a resposta foi entregue.

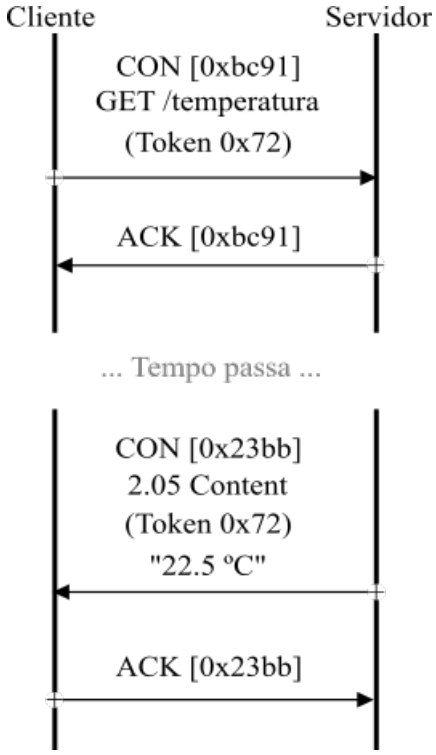


Figura 10 – Troca de mensagem *Conformable Separate Response*

Fonte: Adaptado de Shelby, Hartke e Bormann (2014)

2.2.3 *Message Queuing Telemetry Transport*

Message Queue Telemetry Transport (MQTT) é um protocolo de aplicação que foi projetado para atuar em dispositivos que possuem poucos recursos computacionais. Ele foi criado, no final da década de 1990, pela *International Business Machines* (IBM), e em

2013, foi documentado como protocolo na *Organization for the Advancement of Structured Information Standards* (OASIS) (YASSEIN et al., 2017).

Inicialmente, sua aplicação original era vincular sensores em *pipelines* de petróleo a satélites, e como seu nome sugere, ele é um protocolo de mensagem com suporte para a comunicação assíncrona entre as partes (YUAN, 2017). A operação de conexão usa um mecanismo de roteamento (um-para-um, um-para-muitos, muitos-para-muitos) e utiliza padrões de roteamento Máquina a Máquina (M2M), Máquina a Servidor (M2S) e Servidor a Servidor (S2S) (AL-FUQAHA et al., 2015; YASSEIN et al., 2017).

O MQTT possui características intrínsecas, que o tornam uma opção valiosa em ambientes de Internet das Coisas que comumente dispõem de baixa largura de banda, são intermitentes, entre outros fatores (MANSO et al., 2018; BELLAVISTA; ZANNI, 2016). Este protocolo, assim como o protocolo AMQP, trabalha sobre o *Transmission Control Protocol* (TCP).

2.2.3.1 Formato da Mensagem

O protocolo MQTT funciona realizando diversas trocas de pacotes de controle de forma definida (BANKS; GUPTA, 2015), e como ilustrado na Figura 11, o cabeçalho deste protocolo é composto por um cabeçalho fixo, um cabeçalho variável e o *Payload*.

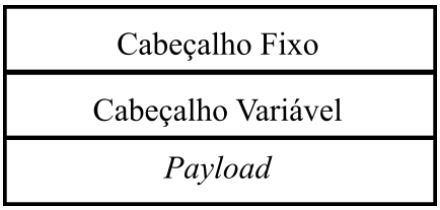


Figura 11 – Formato geral da mensagem MQTT

Fonte: Adaptado de Banks e Gupta (2015)

O cabeçalho fixo possui um comprimento de 2 bytes, o qual pode ser visualizado na Figura 12, onde o byte 1 contém o tipo da mensagem representado por um valor não assinado de 4 bits, sendo esses os bits iniciais (GRGIĆ; ŠPEH; HEđI, 2016). Conforme Banks e Gupta (2015), esses tipos de mensagem estão relacionados a algumas solicitações, respostas, ACK, tipo de publicação, entre outros e encontram-se listados na Tabela 2.

Bit	7	6	5	4	3	2	1	0
Byte 1	Tipo da Mensagem				DUP	QoS		Retain
Byte 2	Largura Restante							

Figura 12 – Cabeçalho fixo MQTT

Fonte: Adaptado de Banks e Gupta (2015)

Em seguida, existe o campo DUP que sinaliza quando o cliente ou servidor tenta entregar mais de uma vez uma mensagem do tipo *PUBLISH*, *PUBREL*, *SUBSCRIBE* ou *UNSUBSCRIBE*, ou seja, uma possível retransmissão da mensagem (GRGIĆ; ŠPEH; HEđI, 2016). Posteriormente, 2 bits são reservados para definir o nível de *Quality of Service* (QoS) (Qualidade de Serviço), que serão explicados posteriormente mais detalhadamente. O último bit para o byte 1 é reservado à opção *Retain*. Quando esse marcador é ativado, essa mensagem deve ser retida no servidor mesmo após ser entregue aos assinantes, para o caso de um novo assinante se inscrever poder receber essa mensagem (MARTINS; ZEM, 2016). A Largura Restante é o número de bytes restantes da mensagem atual (BANKS; GUPTA, 2015).

O conteúdo do cabeçalho variável, apresentado na Figura 11, varia dependendo do tipo de pacote, sendo possível conter algumas informações importantes, como por exemplo, a versão do protocolo que está sendo utilizada ou um identificador de pacote, e em seguida tem-se o campo de *Payload*.

Tabela 2 – Tipos de pacotes de controle MQTT

Nome	**Valor**	**Descrição**
Reserved	0	Reservado
CONNECT	1	Solicitação do cliente para se conectar ao servidor
CONNACK	2	ACK da conexão
PUBLISH	3	Mensagem Publicação
PUBACK	4	ACK da Publicação
PUBREC	5	Publicação recebida (garantia de entrega parte I)
PUBREL	6	Publicação liberada (garantia de entrega parte II)
PUBCOMP	7	Publicação completa (garantia de entrega parte III)
SUBSCRIBE	8	Solicitação de assinatura do cliente
SUBACK	9	ACK de assinatura
UNSUBSCRIBE	10	Solicitação de cancelamento de assinatura do cliente
UNSUBACK	11	ACK de cancelamento de assinatura
PINGREQ	12	Solicitação PING
PINGRESP	13	Resposta PING
DISCONNECT	14	Cliente disconectado
Reserved	15	Reservado

Fonte: Adaptado de Banks e Gupta (2015)

2.2.3.2 Arquitetura e Comunicação

A comunicação do protocolo MQTT, ocorre por meio da arquitetura Publicação/Assinatura, ilustrada na Figura 13. Nela é possível observar três componentes principais:

Publisher (Publicador), o *Broker* (Intermediário) e o *Subscriber* (Assinante). Segundo Xu, Mahendran e Radhakrishnan (2016), cada componente é definido por:

Publisher – Os Publicadores, como o próprio nome indica, são os dispositivos que geram dados. Os dados são publicados seguindo o formato de tópicos.

Broker – O Intermediário age como um nó central realizando a comunicação entre Publicadores e Assinantes.

Subscriber – Os Assinantes recebem as mensagens publicadas, de acordo com o tópico em que elas se inscreveram.

Para publicar os dados por meio do MQTT, um *Pulisher*, que pode ser um sensor, por exemplo, inicialmente realiza uma solicitação para conexão com o *Broker*. Após realizada a conexão alguns elementos são estabelecidos: QoS, tópico e mensagem.

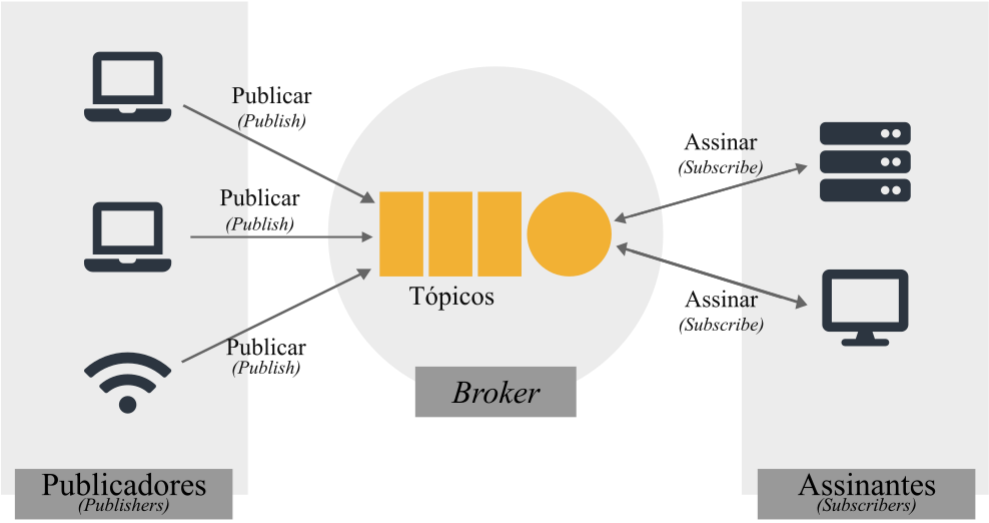


Figura 13 – Arquitetura MQTT

Fonte: Adaptado de Al-Fuqaha et al. (2015)

O nível da Qualidade de Serviço irá definir se as mensagens irão ser entregues ao menos uma vez, pelo menos uma vez ou exatamente uma vez, como relatado posteriormente nesta seção. O tópico é definido como uma cadeia que é usada pelo intermediário para decidir qual assinante receberá a mensagem, sendo utilizado pelo *Broker* para decidir qual assinante encaminhará a mensagem (TANTITHARANUKUL et al., 2016). Um exemplo que pode ser citado é a utilização de um sensor que publica dados por meio do MQTT fazendo uso do seguinte tópico:

Tópico: *sensor*

Ademais, um tópico pode possuir um ou mais níveis hierárquicos, onde cada nível é separado por uma barra (/) (TANTITHARANUKUL et al., 2016). Utilizando o mesmo exemplo citado anteriormente, este sensor pode publicar mensagens empregando o seguinte tópico:

Tópico: *sensor/temperatura*

Neste caso, somente os *Subscribers* que estão inscritos nesse tópico é que receberão as mensagens publicadas pelo sensor. A mensagem consiste basicamente na informação que será publicada pelo *Publisher*.

O *Subscriber* inicialmente também solicita uma conexão com o *Broker*. Após a conexão ser realizada, ele vai assinar um determinado tópico, para posteriormente receber informações sobre o tópico assinado. Por exemplo, caso deseje receber informações sobre o sensor de temperatura, sua inscrição será por meio da expressão:

Tópico: *sensor/temperatura*

Seguindo este exemplo de sensores, também é possível receber informações das publicações de todos os sensores, por meio do seguinte tópico:

Tópico: *sensor/#*

Além disso, o *Subscriber* pode solicitar o cancelamento da inscrição para remover uma solicitação de mensagens e caso não deseje mais receber informações poderá se desconectar do *Broker* (BANKS; GUPTA, 2015). Como pode ser observado na Figura 13, ao ser publicada uma mensagem, o *Broker* irá transmitir essa mensagem a todos os interessados que possuem assinatura no tópico, sendo possível lidar com milhares de dispositivos conectados ao mesmo tempo (YASSEIN et al., 2017).

2.2.3.3 Qualidade de Serviço

Como relatado anteriormente, o protocolo MQTT possui três níveis de Qualidade de Serviço. Estes são: Até uma vez, Ao menos uma vez e Exatamente uma vez. Eles serão detalhados a seguir.

QoS 0 - Até uma vez – De acordo com Banks e Gupta (2015), nesse nível de QoS nenhuma mensagem de resposta é enviada pelo receptor e nenhuma tentativa é executada pelo remetente, ou seja, o remetente envia e não recebe nenhuma confirmação de que a mensagem enviada chegou ao seu destinatário. Isto significa que uma mensagem enviada pode ou não chegar aos inscrito do tópico. Segundo Manso et al. (2018), nesse nível, as mensagens são entregues de acordo com o melhor esforço da rede e a mensagem não é armazenada. A Figura 14 ilustra a troca de mensagens para QoS-0.

Qos 1 - Ao menos uma vez – Neste nível, uma mensagem de resposta é enviada confirmando o recebimento da mensagem, garantindo assim, que a mensagem chegue ao destinatário pelo menos uma vez. O remetente deve tratar o pacote como *unacknowledged* até que tenha recebido o pacote *PUBAKC* correspondente ao receptor (BANKS; GUPTA, 2015). Desse modo, pode ocorrer o recebimento de pacotes duplicados. De acordo com Bellavista e Zanni (2016), em QoS-1, as mensagens devem ser armazenadas localmente no remetente, até que tenham sido entregues ao seu receptor, de modo que sejam viáveis possíveis retransmissões. Após o recebimento da mensagem *PUBACK*, a mensagem é descartada, como visto na Figura 15.

QoS 2 - Exatamente uma vez – Esta é considerada a mais alta Qualidade de Serviço, sendo recomendado para os casos em que a perda de pacotes e a duplicação de pacotes não sejam aceitáveis, ocorrendo assim, a entrega de exatamente uma única mensagem ao destinatário. O remetente deve tratar o pacote como *unacknowledged* até que tenha recebido o pacote *PUBREC* correspondente, e após o recebimento desse pacote, deve enviar um pacote *PUBREL* tratando esse pacote como *unacknowledged* até que receba um pacote *PUBCOMP* correspondente (BANKS; GUPTA, 2015). Após o recebimento desse último pacote a mensagem é deletada, como pode ser observado na Figura 16. Devido a esta série de troca de pacotes confirmando o recebimento, há uma sobrecarga associada a ela. Ademais, é possível ocorrer um atraso relativo na entrega das mensagens, contudo não há perda de pacotes (GRGIĆ; ŠPEH; HEđI, 2016).

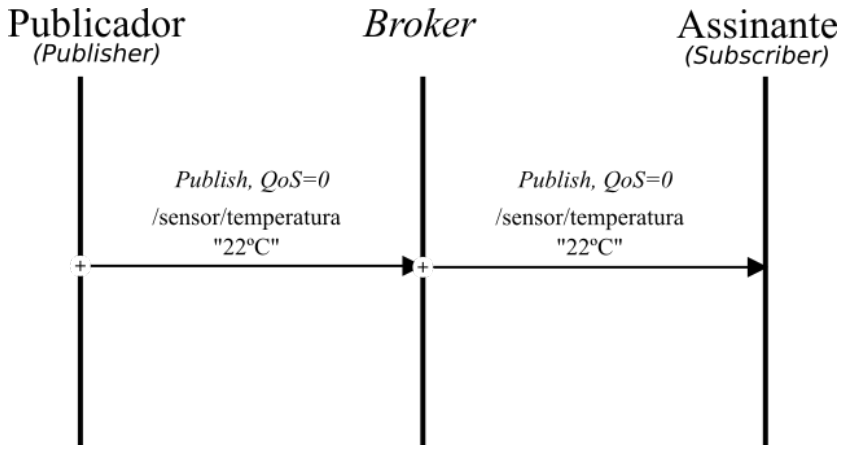


Figura 14 – Qos 0 - Até uma vez MQTT

Fonte: Adaptado de Banks e Gupta (2015)

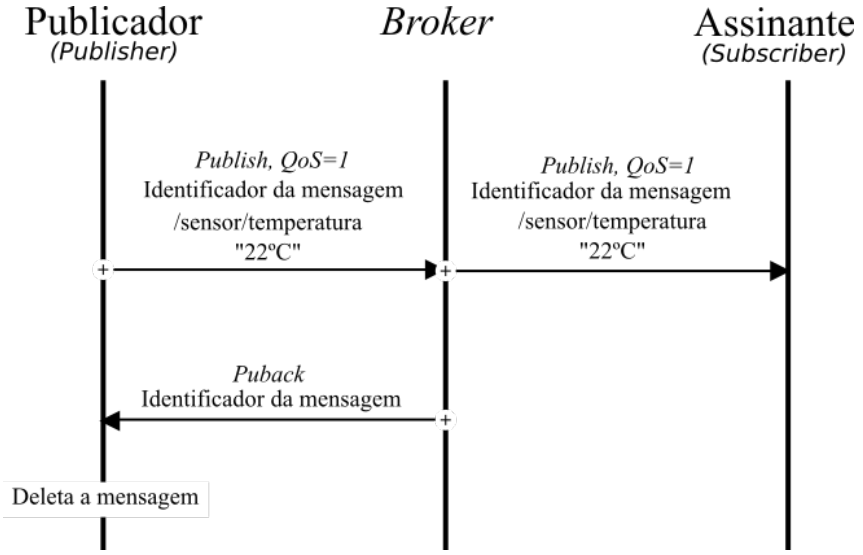


Figura 15 – Qos 1 - Ao mesmo uma vez MQTT

Fonte: Adaptado de Banks e Gupta (2015)

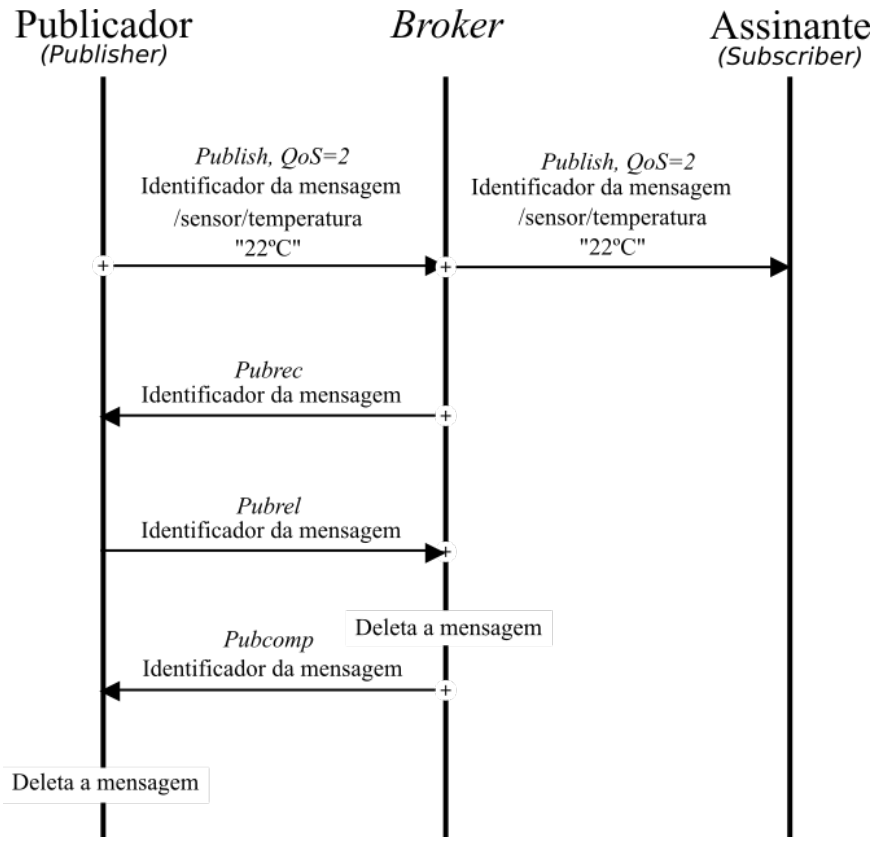


Figura 16 – Qos 2 - Exatamente uma vez MQTT

Fonte: Adaptado de Banks e Gupta (2015)

2.3 AVALIAÇÃO DE DESEMPENHO

A análise ou avaliação de desempenho é um fator fundamental em qualquer estágio do ciclo de vida de um sistema, como também pode ser considerado um critério-chave no design, aquisição e uso de softwares (CARVALHO, 2005; JAIN, 1990). Desse modo, ela possui um importante papel na projeção e análise de softwares para evitar futuramente altos custos, como também buscar um melhor desempenho no funcionamento. De acordo com Jain (1990), a avaliação de desempenho pode ser considerada uma arte, e assim, ela requer um conhecimento íntimo do sistema que está sendo modelado e uma seleção cuidadosa da metodologia, carga de trabalho e ferramentas.

O principal objetivo de uma análise de desempenho é auxiliar engenheiros, analistas e projetistas a obter conclusões objetivas, a partir de um conjunto de parâmetros fornecidos, acerca do sistema ou configuração que está sendo analisada, comparando diversas alternativas para encontrar a que melhor atenda aos seus requisitos (CARVALHO, 2005; JAIN, 1990).

A avaliação de desempenho de sistemas consiste em um conjunto de critérios e técnicas que podem ser classificadas em modelagem ou a medição (JOHN; EECKHOUT, 2018). No contexto da modelagem, encontram-se a modelagem analítica e a simulação, em que é requisitado uma abstração do sistema analisado, o qual inclui diversos níveis de detalhamento, dependendo do objetivo a ser investigado, os principais elementos e eventos condizentes ao comportamento do sistema e relevantes na análise, e encontram-se também a utilização de modelos formais e funções matemáticas (CARVALHO, 2005; JOHN; EECKHOUT, 2018).

No contexto da medição, geralmente uma pequena parte do sistema é selecionada para a realização da análise de desempenho, devido ao fato que muitas vezes, as condições de se avaliar o sistema completo são complexas e custosas. Desse modo, é imprescindível um estudo criterioso para a seleção das técnicas, carga de trabalho e ferramentas a serem utilizadas para realização da medição (LILJA, 2005).

Em um contexto geral, para todos os processo de avaliação de desempenho a serem realizados, é fundamental a realização de um estudo aprofundado dos requisitos do sistema, dos parâmetros e variáveis existentes, carga de trabalho e ferramentas a serem utilizadas, como também como será executada essa avaliação para obtenção resultados corretos.

2.3.1 Projets de Experimentos (*Design of Experiments*)

O Projeto de Experimentos é uma parte essencial para a projeção, planejamento e execução de projetos. Atualmente, também pode ser visto como parte do processo científico, e como uma das maneiras pelas quais aprendemos sobre como os sistemas ou processos funcionam (MONTGOMERY, 2017). É possível dizer que o DoE é uma técnica utilizada para a elaboração de um planejamento de como será realizado um determinado experimento,

contendo informações importantes como as variáveis, os parâmetros, fatores envolvidos, como também a quantidade de amostras a serem utilizadas. Ao se conduzir um experimento com todos os passos e processos bem definidos os resultados obtidos serão confiáveis e consistentes.

Para utilizar a abordagem estatística ao projetar e analisar um experimento, é necessário que todos os envolvidos no experimento tenham uma ideia clara antes do que exatamente será estudado, como como também quais os dados devem ser coletados e analisados (MONTGOMERY, 2017). Dessa forma Montgomery (2017) propõem um conjunto de passos para a realização do projeto de experimentos:

1. Caracterização do problema;
2. Identificação da variável de resposta;
3. Seleção dos fatores, níveis e intervalos a serem utilizados no experimento;
4. Escolha do design experimental;
5. Condução do experimento;
6. Análise estatística dos dados;
7. Conclusões e recomendações.

Antes de chegar nas etapas de condução e análise dos dados, é importante conhecer bem os conceitos e definir bem as variáveis de resposta, os fatores e níveis. A Figura 17 apresenta o processo de experimentação, em que podemos visualizar a existência de fatores que podemos controlar e fatores incontroláveis. De acordo com Jain (1990), Juristo e Moreno (2013) alguns termos são essenciais para o DoE. A seguir estão alguns desses termos:

Variável de Resposta – é o resultado obtido de um experimento. Uma variável de resposta é característica do experimento, fase, produto ou recurso que é medida para testar os efeitos das variações provocadas de um experimento para outro.

Fator – cada variável de desenvolvimento do projeto a ser estudada que afeta a variável de resposta e tem várias alternativas.

Parâmetro – qualquer característica do projeto de software que deve ser invariável ao longo da experimentação.

Níveis – os valores que um fator pode assumir.

Unidade Experimental – qualquer entidade usada para o experimento, ou seja, os objetos nos quais o experimento é executado.

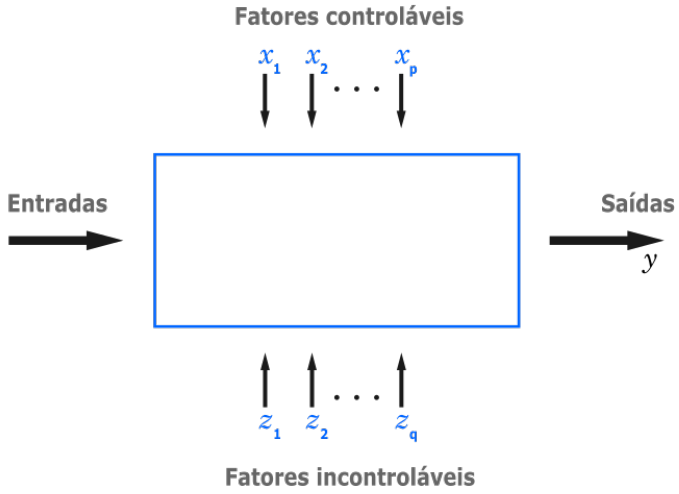


Figura 17 – Modelo geral de um processo

Fonte: Adaptado de Montgomery (2017)

De acordo com a quantidade de fatores, um *Design of Experiments* l^k é indicado. Os projetos de experimentos que usam apenas um fator são utilizados para comparar várias alternativas, ou seja, níveis de uma única variável categórica, e para isto utiliza-se do modelo descrito pela seguinte Equação (MONTGOMERY, 2017):

$$y_{ij} = \mu_i + \epsilon_{ij} \begin{cases} i = 1, 2, \cdots, a \\ j = 1, 2, \cdots, n \end{cases} \tag{2.1}$$

onde y_{ij} é a ij-ésima observação, μ é a média i-ésima com o fator no nível e ϵ_{ij} é o erro aleatório que incorpora todas as variabilidades.

Esta equação também é adotada para a Análise de Variância (ANOVA) de um único fator, devido a apenas um fator ser investigado e seus respectivos níveis comparados, sendo extremamente importante que os experimentos sejam realizados de forma aleatória para que a unidade experimental tão uniforme quanto possível (MONTGOMERY, 2017). Dispondo dessas informações, este estudo tem como base o projeto de experimentos contendo um fator.

2.4 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados alguns dos conceitos fundamentais acerca deste trabalho. Primeiramente, uma introdução sobre o conceito de Internet das Coisas foi explanada. Em seguida, foram descritas informações sobre os protocolos AMQP, CoAP e MQTT que são utilizados para o estudo. Posteriormente, foi apresentada uma breve explicação a respeito da Avaliação de Desempenho em sistemas computacionais. Por fim, abordou-se um resumo do conceito de *Design of Experiments*, sendo apresentado o modelo de projeto de experimentos utilizado no estudo.

3 TRABALHOS CORRELATOS

Neste capítulo, é exposto um resumo dos trabalhos encontrados na literatura referente ao assunto desta dissertação. Existem pesquisas efetuam uma análise comparativa dos protocolos da camada de aplicação para ambientes de Internet das Coisas, o qual são selecionados os considerados mais expressivos. Alguns desses trabalhos realizam a comparação dos protocolos mediante estudo do estado da arte e revisão da literatura, enquanto outros utilizam *middlewares* como interface comum a um conjunto de protocolos. Ademais, outras pesquisas executam experimentos em simuladores de redes ou emuladores e, temos também, as que avaliam os protocolos a partir de experimentos em ambientes semelhantes aos reais. No final deste capítulo é apresentada uma tabela comparativa destes principais trabalhos relatados.

3.1 AVALIAÇÃO DE PROTOCOLOS IOT

No tocante aos trabalhos referentes à comparação através do estado da arte, podemos citar dois principais, Naik (2017) e Yassein, Shatnawi et al. (2016). Naik (2017) aborda a questão do desafio na padronização de um protocolo de mensagens no uso de aplicações IoT, e possui como objetivo investigar os prós e os contras dos protocolos de mensagens amplamente aceitos e emergentes para os sistemas de Internet das Coisas, além de apresentar uma avaliação comparativa desses protocolos que são o MQTT, CoAP, AMQP e HTTP. Primeiramente é exposta as principais características de cada protocolo observado e, em seguida, a partir de evidências obtidas na literatura, uma comparação é realizada apresentando os pontos fortes e as limitações de cada protocolo baseado em latência, segurança, consumo de banda, consumo de energia, entre outros critérios.

Yassein, Shatnawi et al. (2016) citam o desafio existente na comunicação efetiva entre objetos heterogêneos para ambientes IoT, e apresenta como objetivo a promoção de uma comparação entre os protocolos MQTT, AMQP, CoAP, XMPP, *RESTful Services*, DDS e *WebSocket*. Essa comparação é efetuada com base na arquitetura, modelo de comunicação, camada de transporte utilizada e também é descrito um resumo relatando as especificidades de cada protocolo e para quais ambientes eles são mais favoráveis.

Na Tabela 3, dispomos de uma comparação entre esses dois trabalhos citados que são referentes ao estado da arte, com base nos protocolos listados e possíveis critérios analisados. Devido as estas duas pesquisas possuírem como base uma investigação teórica, não foram relatados quais protocolos são melhores, mas apresentados os pontos fortes e pontos fracos de cada um deles.

Tabela 3 – Comparação dos Trabalhos de Naik (2017), Yassein, Shatnawi et al. (2016)

Autor	**Protocolos**	**Métricas**
Naik (2017)	MQTT CoAP AMQP HTTP	Tamanho da mensagem Consumo de energia Largura de Banda Confiabilidade/QoS Segurança Uso M2M/IoT
Yassein, Shatnawi et al. (2016)	CoAP MQTT XMPP RESTFUL AMQP DDS Websocket	Baixa potência Grande volume de atualizações de dados

Fonte: Elaborado pela autora

Em relação aos trabalhos que utilizam *middleware*, podemos citar Thangavel et al. (2014), o qual relata a importância de protocolos de aplicação eficientes em termos de largura de banda e energia para transmissão de dados, visto que a redes de sensores sem fio operam com dispositivos que possuem recursos limitados. Para isto, a pesquisa apresenta a criação e desenvolvimento de um *middleware* comum aos dois protocolos selecionados para estudo: MQTT e CoAP. Com base nesse *middleware* efetuou-se uma avaliação de desempenho de ambos os protocolos, utilizando como métricas o *delay* e o total de bytes transferidos por mensagem. Os resultados obtidos mostraram que para baixa perda de pacote, as mensagens transferidas por meio do MQTT obtiveram menores atrasos do que no CoAP. Contudo, em altas perdas de pacotes, o MQTT apresentou os maiores atrasos. Neste trabalho, também foi inferido que o CoAP gera menos carga de tráfego extra na transmissão de dados confiáveis (THANGAVEL et al., 2014), todavia, a influência de desconexão da rede não foi observada na pesquisa.

A pesquisa de Luzuriaga et al. (2015) apresenta o problema existente dos protocolos MQTT e AMQP terem sido pouco explorados na área de redes móveis e redes veiculares. Deste modo, é proposta uma avaliação de desempenho dos dois protocolos em um ambiente experimental controlado, com alguns componentes importantes, como: cliente, servidor e *Access Point* (AP). No experimento, as mensagens contiveram um tamanho prefixado transmitidas do remetente ao destinatário e as métricas avaliadas foram o *jitter* e *workload boundary*. Como resultados obtidos os autores recomendam o uso do protocolo AMQP em ambientes onde os requisitos primordiais sejam a confiabilidade e escalabilidade, e o uso do protocolo MQTT para ambientes restritos. No entanto, não foi abordado como essas

métricas reagem a falhas recorrentes na rede.

Mijovic, Shehu e Buratti (2016) abordam a padronização de uma pilha de protocolos de comunicação, onde esta possa ser eficiente para os dispositivos que estão inseridos dentro de cenários de Internet das Coisas. A partir desta temática, o trabalho realiza uma avaliação de desempenho de três protocolos de comunicação MQTT, CoAP e *WebSocket*, e nos experimentos foram utilizados alguns componentes que realizam a troca de mensagens sendo avaliadas a eficiência do protocolo e média do *Round Trip Time* (RTT). Como resultado, os autores apontam que o CoAP possui uma maior eficiência do protocolo e um menor RTT seguido da tecnologia *WebSocket*. Contudo no presente artigo, não foi abordada a questão de falhas na conexão da rede.

Chen e Kunz (2016) discutem a eficiência necessária para comunicação Máquina a Máquina (M2M), devido aos sensores remotos e dispositivos de *gateway* estarem comumente conectados por meio de links de comunicação sem fio, onde essas conexões conservam as características de baixa largura de banda, não confiáveis ou intermitentes. Nesse sentido, a pesquisa tem como intuito realizar uma análise de desempenho dos protocolos MQTT, CoAP, DDS e *Custom UDP* em um ambiente voltado para área médica. No ambiente desenvolvido para o estudo, alguns componentes representam os sensores dos pacientes que enviam informações para o servidor, que por sua vez, disponibiliza os dados para outros componentes que representam os cuidadores. As métricas consideradas para estudo foram o consumo de banda, a latência e a perda de pacote e, a partir dos resultados obtidos, os autores afirmam que o protocolo DDS apresenta os melhores resultados em relação à latência e à confiabilidade, embora apresente uma alta taxa de consumo de banda. No entanto, utilizou-se um tamanho fixo do pacote, assim não há uma alteração do tamanho do pacote em casos de retransmissão dos dados, e a alteração da taxa de perda de pacote não foi considerada na pesquisa.

O trabalho de Collina et al. (2014) aborda a relevância do desempenho de protocolos de comunicação em redes com links de alto atraso, dando como exemplo, as redes sem fio intermitentes e redes por satélite. Desse modo, o propósito da pesquisa é prover uma análise de desempenho dos protocolos MQTT e CoAP sobre determinadas condições de tráfego da rede, perda de pacote e atraso no link, considerando uma estrutura que consiste na comunicação entre sensores e servidor por meio de uma conexão sem fio. A partir dos resultados, os autores concluem que o protocolo CoAP apresentou melhores resultados para a arquitetura proposta. No entanto, não foi considerada a influência do tamanho do pacote nos casos de retransmissão dos dados.

Yokotani e Sasaki (2016) descrevem a questão da sobrecarga da rede na transferência de dados ao se utilizar o protocolo HTTP em ambientes IoT. Com isso, é proposta a possibilidade de aplicar o protocolo MQTT para a comunicação entre estas redes e, para isto, a pesquisa realiza uma comparação entre os protocolos HTTP e MQTT tomando como métricas o tamanho do *payload*, bytes transmitidos, entre outras. Como resultados,

é apontado que o protocolo MQTT tem um desempenho melhor que o HTTP. Todavia, o artigo não considerou outros protocolos da camada de aplicação para Internet das Coisas e o estudo do desempenho na ocorrência de falha na rede.

Kayal e Perros (2017) relatam em sua pesquisa que diversos protocolos voltados para Internet das Coisas foram incorporados para melhorar a comunicação entre dispositivos, porém o desempenho deles ainda precisam ser mais estudados. Para isso, a pesquisa propõe uma comparação de desempenho de quatro protocolos (CoAP, MQTT, XMPP e *WebSocket*) em um aplicativo de estacionamento inteligente, avaliando o tempo médio de resposta de acordo com a utilização do servidor. Os resultados revelaram que para uma baixa utilização do servidor, o protocolo CoAP apresenta um melhor resultado em relação aos outros, seguido do protocolo XMPP; e para os ambientes em que há um aumento na utilização do servidor, a tecnologia mais indicada passa a ser o *WebSocket*. Contudo, o trabalho não avalia a perspectiva da instabilidade da rede no momento do envio das informações sobre disponibilidade de vagas no estacionamento.

Marques e Kniess (2018) relatam que os protocolos desenvolvidos para redes de Internet das Coisas necessitam ser eficientes, e a partir disso, o trabalho apresenta uma análise de desempenho dos protocolos HTTP, MQTT e CoAP, com o objetivo de indicar quais protocolos são os mais indicados para aplicações IoT. Dois experimentos foram realizados utilizando microcontroladores e dois servidores, sendo um local e outro remoto, para a realização dos testes. As métricas avaliadas foram o *Round Trip Time* (RTT), o tamanho da mensagem e a sobrecarga. Como resultados, o protocolo CoAP foi o mais indicado para implantação de aplicações IoT. No entanto, não foi avaliado em casos em que há falhas na rede.

Diferentemente, esta dissertação propõe uma avaliação de desempenho considerando o tamanho da mensagem, a taxa de transmissão e a perda de pacotes em dois experimentos. Sendo um deles em um ambiente em que não há falhas recorrentes na rede e o outro com injeção de falhas, para avaliar e comparar o comportamento, e a capacidade do protocolo através de medições em uma rede sem fio.

Alguns protocolos de comunicação comumente relatados na literatura, a exemplo do *Data Distribution System* (DDS) e *Extensible Messaging and Presence Protocol* (XMPP), não foram utilizados devido a algumas limitações. No XMPP, podemos citar o fato de que este protocolo é voltado, principalmente, a aplicações de troca de mensagens instantâneas e o DDS, dispõe de poucas implementações *open source*, e estas não possuem documentação abrangente.

3.2 COMPARAÇÃO DOS TRABALHOS

A Tabela 4 reúne os parâmetros adotados em cada um dos trabalhos relacionados mencionados, bem como, as métricas abordadas nesta dissertação. Nesta tabela são descritas as principais pesquisas citadas na Seção anterior.

Também são apresentadas se nestas pesquisas ocorreu algum estudo acerca da falha, onde em Thangavel et al. (2014), Chen e Kunz (2016), Collina et al. (2014), apresentaram a falha como um parâmetro, em que variam em seus experimentos o percentual de perda de pacotes para verificar sua influência nas métricas estudadas.

Ao se observar os protocolos selecionados para estudo, é possível identificar que alguns autores como Mijovic, Shehu e Buratti (2016), Kayal e Perros (2017) utilizaram a tecnologia *websocket* para comparação (inclusa na coluna Outros). Contudo, esses autores apresentaram resultados diferenciados em que Kayal e Perros (2017) relatam que o *websocket* é o mais indicado para ambientes em que ocorre uma alta utilização do servidor. Os autores Mijovic, Shehu e Buratti (2016) apontam essa tecnologia como a segunda mais indicada após o protocolo CoAP.

Tabela 4 – Comparação entre esta Dissertação e Trabalhos Relacionados

Autor	**Avaliação de Desempenho**	**Avaliação na ocorrência de falha**	**Protocolos**						**Métricas**
			MQTT	**AMQP**	**CoAP**	**XMPP**	**DDS**	**Outros**	
Thangavel et al. (2014)	✓	✓	✓	-	✓	-	-	-	*Delay*, Total de bytes transferidos por mensagem
Luzuriaga et al. (2015)	✓	-	✓	✓		-	-	-	*Jitter*, *Workload boundary*
Mijovic, Shehu e Buratti (2016)	✓	-	✓	-	✓	-	-	*Websocket*	Eficiência do protocolo, RTT médio
Chen e Kunz (2016)	✓	✓	✓	-	✓	-	✓	*Custom UDP*	Consumo de energia, Latência experiente, Perda de pacote experiente
Collina et al. (2014)	✓	✓	✓	-	✓	-	-	-	Latência no nível da aplicação, *Throughput* normalizado
Yokotani e Sasaki (2016)	✓	-	✓	-	-	-	-	HTTP	Tamanho do *payload*, Bytes transmitidos, RTT
Kayal e Perros (2017)	✓	-	✓	-	✓	✓	-	*Websocket*	Tempo médio de resposta, Utilização do servidor
Marques e Kniess (2018)	-	-	✓	-	✓	-	-	HTTP	Tamanho da mensagem, Sobrecarga, RTT
Este trabalho	✓	✓	✓	✓	✓	-	-	-	*Throughput*, Tamanho da mensagem, Perda de pacote

Fonte: Elaborado pela autora

3.3 CONSIDERAÇÕES FINAIS

Este capítulo apresentou um resumo dos trabalhos encontrados na literatura no tocante ao estudo de protocolos da camada de aplicação para ambientes IoT. Inicialmente, descreveu-se sobre os trabalhos que possuem como base o estudo do estado da arte e revisão da literatura, indicando os pontos fortes e fracos de cada protocolo. Em seguida, apresentou-se uma pesquisa referente ao estudo de protocolos utilizando *middlewares* como a principal interface usada, sendo esta comum a todas os protocolos. Posteriormente, abordou-se as pesquisas que utilizam a execução de seus experimentos em simuladores de redes ou emuladores e também, as que avaliam os protocolos a partir de experimentos com base em ambientes reais. Por fim, foi apresentada uma tabela comparativa destes principais trabalhos relatados.

4 AMBIENTE DE EXPERIMENTAÇÃO E METODOLOGIA

O presente capítulo apresenta a metodologia proposta para a análise da avaliação de desempenho dos protocolos da camada de aplicação voltado para ambientes de Internet das Coisas. Primeiramente é explicada uma visão geral dos experimentos propostos no ambiente IoT selecionados para estudo e os principais elementos que os compõem. Em seguida, são discutidas as bibliotecas e as ferramentas necessárias para o desenvolvimento e execução dos experimentos. Posteriormente são apresentadas as ferramentas aplicadas para coleta das amostras e suas respectivas análises. Por fim, a metodologia baseada no projeto de experimentos *Design of Experiments* é apontada.

4.1 AMBIENTE E CONFIGURAÇÕES DOS EXPERIMENTOS

O ambiente de experimentação do presente estudo possui como base um ambiente IoT genérico, que pode ser visto na Figura 18. Nessa Figura, o ambiente contém sensores que enviam dados constantemente para os interessados em receberem essas informações, e esses dados podem ser servidores, clientes ou até mesmo outros sensores. A partir desse cenário mais geral, um projeto menor foi criado com o intuito de tornar a execução do conjunto de experimentos para a avaliação dos protocolos viável e possibilitar um resultado mais preciso.

Como relatado anteriormente, dois experimentos foram projetados para análise dos protocolos. Ambos experimentos consistem em um sistema composto por três sensores: movimento, luminosidade, temperatura e umidade. Estes sensores coletam dados do ambiente em que estão inseridos e enviam para um servidor central, onde nos casos em que o protocolo utilize a arquitetura *Publisher/Subscriber*, esse servidor também pode ser denominado de *Broker*.

Cada protocolo possui um mapeamento diferente. Em vista disso, o protocolo MQTT adota a arquitetura *Publisher/Subscriber* em que os sensores são os *Publishers*, o servidor é o *Broker* e localmente existem também os assinantes que recebem as informações enviadas pelos sensores. *Publisher/Subscriber* também é adotado para o AMQP e o mesmo mapeamento é utilizado. O CoAP utiliza o paradigma Cliente/Servidor e o servidor é responsável por receber dados dos sensores. Apesar da arquitetura dos protocolos MQTT e AMQP serem diferentes do protocolo CoAP, isto não influencia no momento da execução, pois o servidor assume o papel do *broker*. Ademais, todos os protocolos funcionam de forma assíncrona.

A partir do sistema representativo criado, foi escolhido que as informações enviadas pelos remetentes necessariamente cheguem aos destinatários. Para isso, as configurações do protocolo também foram ajustadas de forma que uma mensagem do sensor chegue

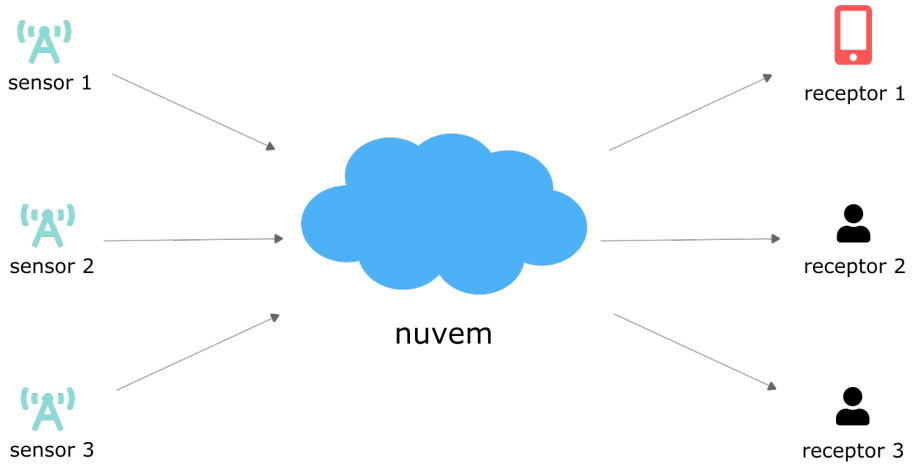


Figura 18 – Ambiente IoT genérico

Fonte: Elaborado pela autora

ao destino pelo menos uma vez. No MQTT, a configuração é QoS-1 (citada na Seção 2.2.3) e, para AMQP, a configuração é QoS-pelo menos uma vez (apresentada na Seção 2.2.1). Para o CoAP, a configuração realizada é *Non-Conformable* com a resposta em *Piggybacked* (mencionada na Seção 2.2.2). A opção *keep alive* também foi alterada em todos os protocolos. As tentativas de conexão foram ajustadas para manter o aplicativo tentando se conectar sempre que uma rota falha, ou seja, deixe de ter conexão para transmissão dos dados.

4.1.1 Experimento I

No primeiro experimento, a finalidade é analisar o desempenho dos protocolos em uma rede que não possua falhas recorrentes na rede, assim, os sensores efetuam a coleta e transmitem os dados para o servidor por meio de uma única rota, como pode ser visto na Figura 19.

Cada sensor realiza a coleta do dado a cada 5 segundos e logo após transmite ao destinatário. Além disso, cada amostra desse experimento é obtida com uma duração de 5 minutos entre cada uma delas. A abordagem de cada amostra ter duração de 5 minutos adotada no estudo, deve-se ao fato de que 5 minutos não é um tempo longo, mas dado que as coletas são realizadas a cada 5 segundos, nesse tempo é possível obter uma quantidade de dados suficientes para avaliação.

A partir de cada amostra conseguida, selecionou-se os dados referentes apenas à comunicação e transmissão entre os remetentes e o destinatário. Em seguida é calculado um valor médio de cada métrica, como por exemplo, o resultado do número de Bytes pelo número de pacotes transmitidos, obtendo-se assim, o tamanho da mensagem. Cada

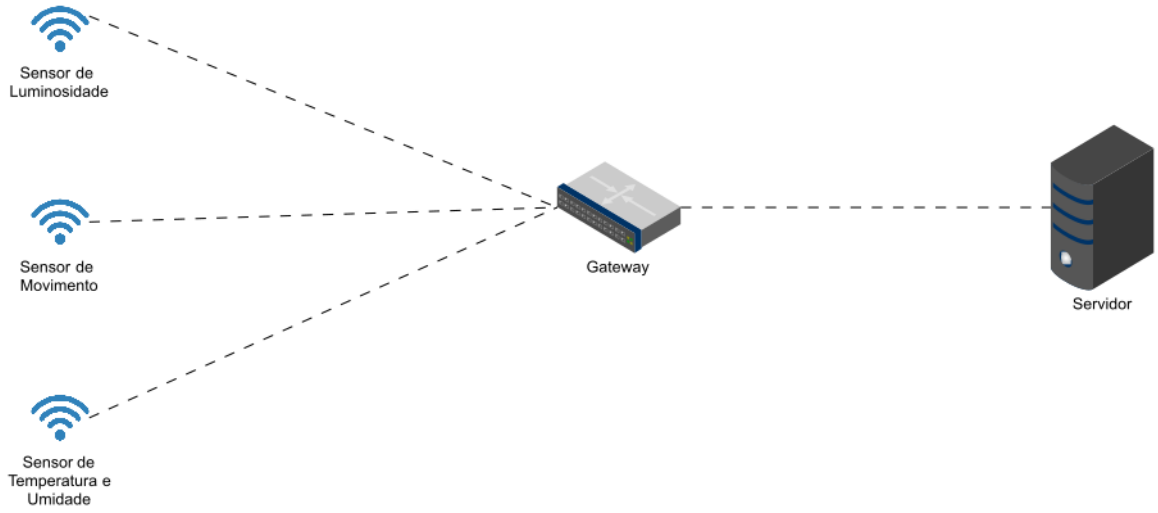


Figura 19 – Configuração do Experimento I

Fonte: Elaborado pela autora [1]

mensagem é gerada automaticamente pelos remetentes, e consiste no dado útil coletado pelos sensores e as informações de cada protocolo utilizado de todas as camadas.

4.1.2 Experimento II

O segundo experimento traz como principal característica a injeção de falha na rede. Assim, tem como objetivo avaliar o desempenho de cada protocolo considerando falhas na rede durante a transmissão dos dados. Essas falhas na rede constituem na interrupção da conexão, ou seja, a desconexão da rede sem fio. Diante disso, os sensores realizam a cada 5 segundos uma nova coleta, tal qual ocorre no Experimento I, e transmitem para o servidor por meio de dois caminhos/rotas disponíveis para a comunicação, como visualizado na Figura 20.

Sempre que ocorre uma falha de transmissão, a aplicação IoT redireciona os dados para uma rota alternativa, ou seja, ao ocorrer uma interrupção na comunicação, a aplicação redireciona a comunicação dos dados por meio da outra rota disponível. Este caminho alternativo, torna-se então a rota principal para transferência dos dados, e caso ocorra a falha de transmissão nesta rota, a aplicação IoT redireciona novamente a comunicação dos dados por meio da rota alternativa.

Os três sensores executam uma coleta de dados acerca do ambiente a cada 5 segundos e, logo em seguida, transmite ao destinatário. Além disso, cada amostra desse experimento é obtida com base de uma duração de 10 minutos entre elas. Esse tempo é pelo fato da transmissão sofrer com recorrentes falhas na rede.

[1] Figura criada no site: https://www.gliffy.com

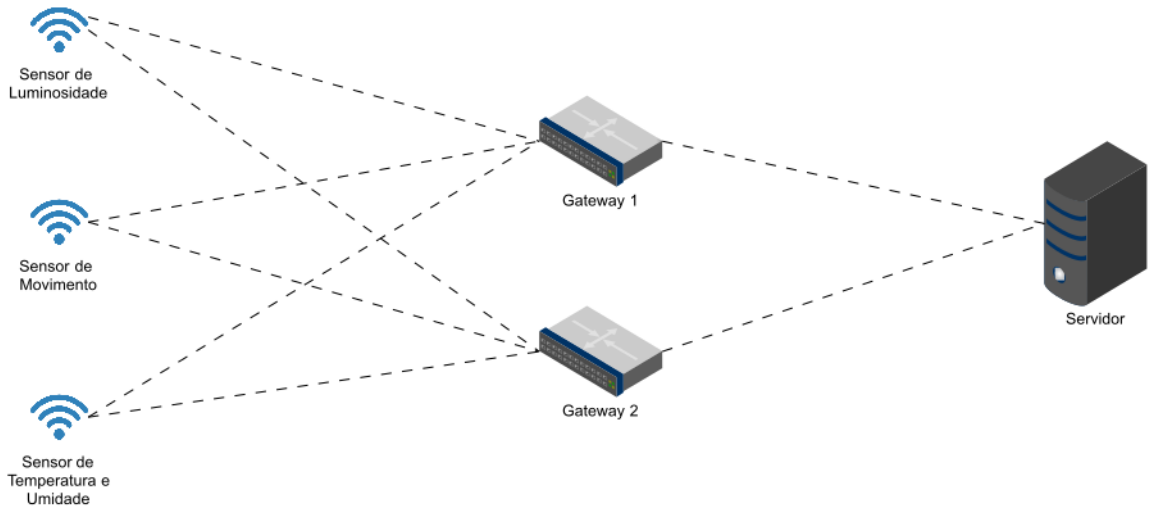


Figura 20 – Configuração do Experimento II

Fonte: Elaborado pela autora [1]

Dado cada amostra obtida, foram selecionados os dados referentes apenas à comunicação e transmissão entre os remetentes e o destinatário, e em seguida é calculado um valor médio de cada métrica, como por exemplo, o resultado do número de Bytes pelo total de tempo utilizado para transmissão, obtendo-se assim, o consumo de largura de banda da rede. Assim como no Experimento I, mensagem é gerada automaticamente pelos remetentes, e consiste no dado útil coletado pelos sensores e as informações de cada protocolo utilizado de todas as camadas. Caso ocorra uma retransmissão de uma determinada mensagem, essa informação acerca de que o pacote é uma retransmissão de dados, também é incluída no tamanho da mensagem.

Para a injeção de falha em uma rota de rede, é adotado o Algoritmo 1, na qual o tempo até a falha é calculado usando a Equação 4.1, e assume-se uma distribuição exponencial comumente utilizado no estudo de intervalos de chegada de pacotes em um servidor e tempo de falha e reparo (TOBIAS; TRINDADE, 2011; TRIVEDI, 1982):

$$t_{TTF} = -MTTF \times ln[1 - rnd] \tag{4.1}$$

onde rnd é o número aleatório gerado a partir da descrição da primeira linha do Algoritmo 1 e MTTF é *Mean Time To Failure* adotado para a falha, ou seja, é o tempo médio para um componente falhar (JAIN, 1990; MACIEL et al., 2012). Para o experimento, o MTTF adotado é de 60 segundos, com o objetivo de tornar viável o tempo total de execução dos experimentos.

Após a ocorrência da falha, a rota é reparada seguindo o Algoritmo 2, o qual o tempo

Algoritmo 1: Falha

```
1 while True do
2     Gerar um número aleatório entre 0 e 1 seguindo uma distribuição uniforme;
3     Calcular o Time to Failure (TTF);
4     Aguardar o TTF (em segundos);
5     Interromper a conexão;
```

de reparo é calculando por meio da Equação 4.2, onde assim como o cálculo do TTF, também se assume uma distribuição exponencial (TOBIAS; TRINDADE, 2011; TRIVEDI, 1982):

$$t_{TTR} = -MTTR \times ln[1 - rnd] \tag{4.2}$$

onde rnd é o número aleatório produzido exposto no Algoritmo 2, descrito na linha 2, e MTTR é *Mean Time To Repair*, ou seja, o tempo médio para reparo (MACIEL et al., 2012; TRIVEDI, 1982). Para este experimento, assume-se que o valor do MTTR é igual a 2 segundos que, assim como o tempo adotado para o MTTF, teve como objetivo de tornar viável o tempo total de execução dos experimentos.

Algoritmo 2: Reparo

```
1 while True do
2     Gerar um número aleatório entre 0 e 1 seguindo uma distribuição uniforme;
3     Calcular o Time to Repair (TTR);
4     Aguardar o TTR (em segundos);
5     Restaurar a conexão;
```

4.2 ARQUITETURA BÁSICA

Os ambientes IoT possuem algumas características próprias, sendo uma delas a questão que grande parte dos seus dispositivos possuem recursos limitados. Neste contexto, para proporcionar que o ambiente para a execução dos experimentos seja adequado a esses atributos, foram levantados alguns requisitos de hardware, software, como também selecionadas bibliotecas específicas que englobem Internet das Coisas e os protocolos estudados neste trabalho.

4.2.1 Hardware

Três sensores de baixo custo e baixo consumo de energia foram selecionados para o estudo, os quais obtêm informações acerca da luminosidade, detecção de movimento, temperatura e umidade.

O sensor de luminosidade selecionado foi o módulo sensor LDR (Resistor Dependente de Luz) (BANZI; SHILOH, 2015). Este módulo é constituído de um sensor LDR acoplado a um módulo com outros componentes, como por exemplo, um ajuste de sensibilidade da luminosidade, como visto na Figura 21. A obtenção dos dados ocorre por meio de respostas binárias 0 e 1, para luminosidade e não luminosidade, respectivamente.

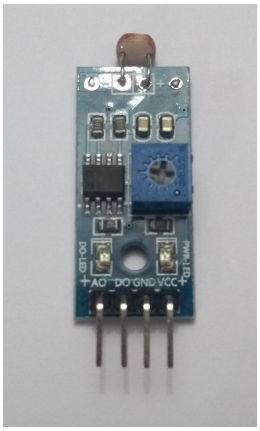
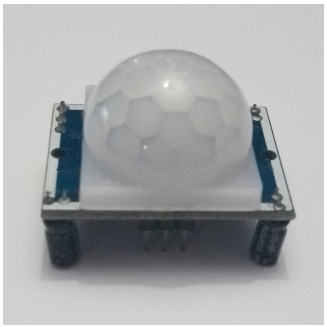


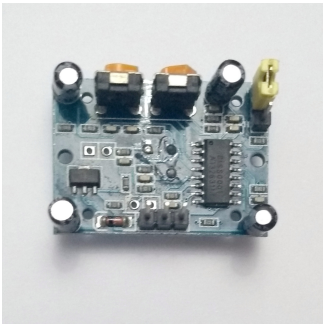
Figura 21 – Sensor LDR

Fonte: Elaborado pela autora

O sensor de movimento utilizado foi o sensor detector de movimento PIR HC-SR501 (Infravermelho Passivo), como visualizado na Figura 22 (OLIVEIRA et al., 2018). A resposta ocorre por meio de um valor *booleano*, onde *True* é retornado quando existe movimento, e *False* quando não há detecção de movimento no ambiente.



(a) Sensor PIR visto na parte superior



(b) Sensor PIR visto na parte inferior

Figura 22 – Sensor PIR HC-SR501

Fonte: Elaborado pela autora

Para as informações acerca da temperatura e umidade, utilizou-se um sensor DHT11, ilustrado na Figura 23, o qual consegue obter as duas informações, e estes dados são transmitidos por meio de um valor inteiro não negativo (GAY, 2018).

Estes sensores encontram-se conectados a um Raspberry Pi 3 Model B. O Raspberry Pi é desenvolvido pela Fundação Raspberry Pi, que tem como objetivo fornecer computadores de baixo custo e alto desempenho que as pessoas possam usar para aprendizado

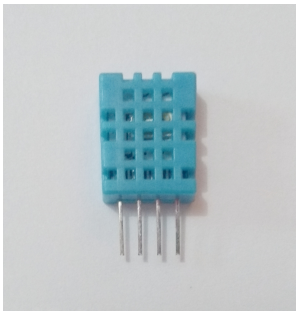


Figura 23 – Sensor DHT11

Fonte: Elaborado pela autora

de computação, resolução de problemas e diversão de forma simples e criativa (Raspberry Pi, 2019).

Brown (2016) divulgou um resultado obtido a partir de uma pesquisa no portal *Linux.com*, acerca dos principais projetos de Computador de Placa Única (SBC) selecionados com base em um Sistema Operacional (SO) *Android* ou *Linux*, que também sejam *open source* e tenham um bom suporte à comunidade. O resultado desta pesquisa afirmou que o Raspberry Pi encontra-se entre os primeiros colocados na preferência de dispositivos Computador de Placa Única (SBC).

A escolha por esse SBC deve-se ao fato dele ser conhecido mundialmente, encontra-se de acordo com os requisitos características para dispositivos IoT, é *open source*, possui documentação e suporte à comunidade expressiva e numerosa. Sua configuração é constituída por:

- CPU Quad Core 1.2GHz Broadcom BCM2837 64bit;
- 1GB RAM;
- Rede sem fio BCM43438;
- 100 Base Ethernet.

Dessa forma, é possível visualizar na Figura 24 os sensores conectados por meio de fios ao Raspberry Pi, com uma *protoboard* auxiliando na organização das conexões entre esses dispositivos.

4.2.2 Software

Para o Raspberry Pi 3 Model B, foi utilizado o Sistema Operacional (SO) Raspbian Stretch Kernel 4.14, e no servidor foi adotada uma máquina física Intel Core i3-3217U, 1,80 GHz, com 8GB de RAM executando um SO Ubuntu 18.04 LTS. A comunicação sem fio empregada tem como configuração 802.11n com uma frequência de 2.4GHz.

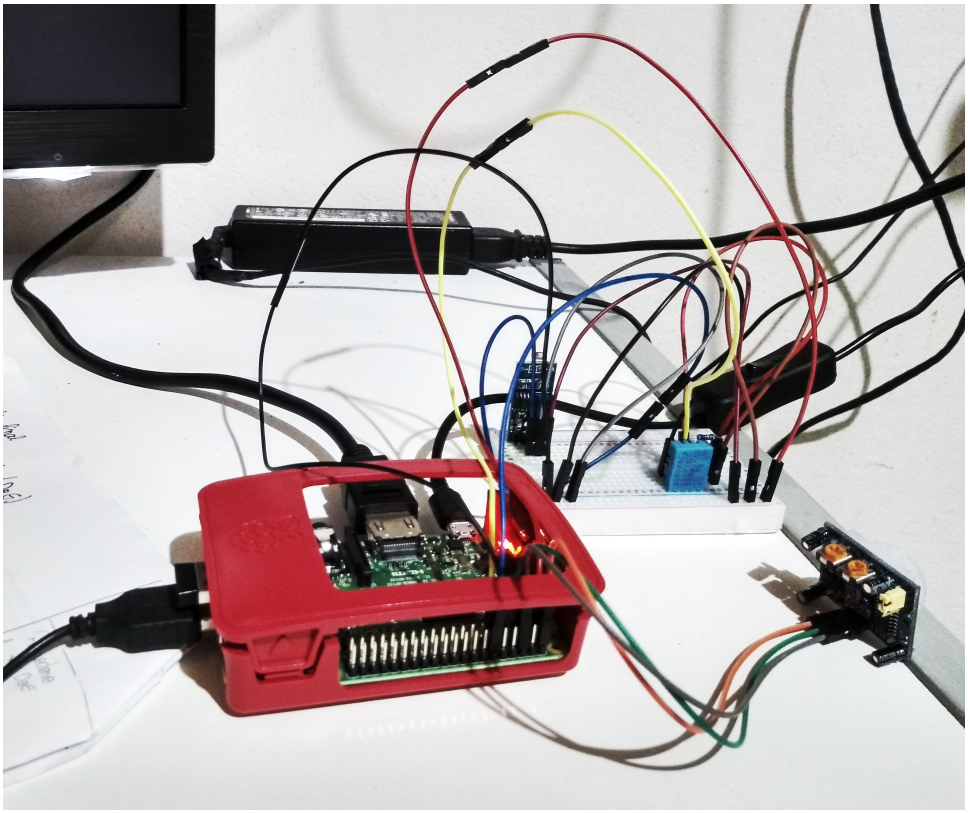


Figura 24 – Ambiente real dos Experimentos

Fonte: Elaborado pela autora

No tocante à coleta dos pacotes transmitidos, esta foi realizada por meio da ferramenta Wireshark (COMBS et al., 2008), e a respeito do cálculo das métricas, foi utilizada a linguagem de programação R e o Ambiente de Desenvolvimento Integrado (IDE) RStudio.

Para a comunicação entre sensores e servidor, a linguagem de programação selecionada foi Python 3.6. Como relatado anteriormente, os dados são enviados para os destinatários interessados. Neste caso, estes interessados encontram-se localmente no servidor, sendo efetuado um armazenamento destes dados recebidos utilizando o MongoDB 4.0 (MONGODB, 2018) por meio do *docker mvertes/alpine-mongo* (MVERTES, 2015).

Com relação às implementações de cada protocolo, que encontram-se no Apêndice A, empregou-se implementações *open source*, mais especificamente RabbitMQ (Inc. Pivotal Software, 2014), CoAPthon3 (TANGANELLI; VALLATI; MINGOZZI, 2015) e Mosquitto (LIGHT, 2017) para AMQP, CoAP e MQTT, respectivamente, sendo todos eles na linguagem Python 3.6, como citado na Seção 4.2. No desenvolvimento do código, no protocolo AMQP foi utilizado a biblioteca *python3-pika 0.9.14* (Python Community, 2014). Para o protocolo MQTT foi utilizada a biblioteca *paho-mqtt 1.4.0* (Python Community, 2008). Existem outras implementações e bibliotecas referentes ao desenvolvimento de aplicações IoT para estes protocolos. Todavia, as adotadas para este trabalho possuem um bom suporte da comunidade e uma boa documentação.

4.3 COLETA E FILTRO DOS DADOS

Como relatado anteriormente, a captura dos dados sobre os pacotes transmitidos na rede foi efetuada por meio da ferramenta Wireshark (COMBS et al., 2008). Durante a execução de cada experimento, o Wireshark realizou a coleta do tráfego da rede obtendo 30 amostras para cada protocolo, e em vista disso, foi identificado que seria necessário efetuar uma seleção de pacotes com a finalidade de excluir os dados transmitidos, e possivelmente, capturados pela ferramenta que não sejam importantes para o estudo ou possam influenciar erroneamente nos resultados.

Dessa forma, cada amostra resultante apresenta apenas informações acerca da transmissão de mensagens do protocolo estudado entre os sensores e o servidor, como pode ser observada na Figura 36, localizada no Apêndice B. Em seguida, foram retirados alguns dados importantes, como quantidade de pacotes, quantidade de Bytes transmitidos, entre outros, para o cálculo das métricas.

4.4 METODOLOGIA

Os experimentos propostos para estudo são voltados para as áreas de casas e cidades inteligentes. Isso deve-se ao fato de que o IoT possui potencial do desenvolvimento de novas aplicações principalmente nessas áreas de atuação devido à capacidade de realizar sensoriamento remoto e oferecer serviços personalizados (BORGIA, 2014).

Um projeto de experimentos (*Design of Experiments*) foi adotado para a condução dos experimentos utilizando fatorial l^k com r replicações (MONTGOMERY, 2017). Para o presente estudo, empregou-se o projeto de um fator, em que usualmente é utilizado para comparar várias alternativas de uma única variável categórica (JAIN, 1990).

O fator definido é o protocolo, este englobando três níveis ($k = 3$) que correspondem aos protocolos da camada de aplicação voltados para ambientes IoT, ou seja, este fator representa os protocolos selecionados para estudo, o quais são: AMQP, CoAP e MQTT. Ademais, foram realizados dois conjuntos de experimentos.

O primeiro experimento tem como objetivo avaliar o desempenho de cada protocolo em uma rede que não esteja sujeita a falhas na rede. Assim como relatado na Seção 4.1.1, os sensores que coletam informações do local em que estão inseridos, transmitem por meio do Raspberry via uma rede sem fio para um servidor, que recebe estas informações. Para cada nível do fator foi realizado essa transmissão de dados para obtenção de informações acerca da rede.

O segundo experimento possui como finalidade avaliar o desempenho de cada protocolo considerando falhas na rede durante a transmissão dos dados. Essas falhas na rede consistem em interrupções da conexão da rede sem fio, como relatado na Seção 4.1.2. Assim, caso da rota que esteja sendo transmitida os dados seja desconectada, ocorrerá uma retransmissão de possíveis dados perdidos através de uma outra rota, e as novas

coletas dos sensores também serão enviadas por meio desta rota disponível. Assim como no Experimento I, para cada nível foi executado uma série de transmissões de dados dos sensores.

Em ambos os experimentos, foram utilizadas 30 replicações/amostras ($r = 30$), onde essa quantidade de amostras foram usadas com a finalidade de se obter valores médios (com uma distribuição aproximadamente normal) e reduzir o impacto dos ruídos na medição. As métricas adotadas para avaliação são o consumo de banda (expresso em Bytes/segundo) e tamanho da mensagem (expresso em Bytes) para os dois experimentos. Além disso, no segundo experimento também é adotada para estudo o percentual de perda de pacotes. Para ambos é realizada uma pesquisa acerca de uma possível ocorrência de correlação entre as métricas selecionadas para estudo.

4.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou a metodologia proposta para a análise da avaliação de desempenho de protocolos da camada de aplicação desenvolvidos para ambientes de Internet das Coisas. Inicialmente, foi apresentado um ambiente geral, que foi considerada como base para o planejamento e criação dos ambientes experimentais. Em seguida, foram descritas as linguagens, bibliotecas e ferramentas utilizadas nos experimentos e coletas de dados. Por fim a metodologia utilizada foi apresentada.

5 RESULTADOS EXPERIMENTAIS

Este capítulo apresenta os resultados obtidos a partir dos experimentos realizados, conforme descrito no Capítulo 4. Inicialmente são apresentadas as configurações utilizadas em cada um dos experimentos, assim como os objetivos de cada um deles. Em seguida, são descritos os resultados para o consumo de banda, tamanho da mensagem e correlação obtidos no experimento I. Posteriormente, são expostos os resultados para o consumo de banda, tamanho da mensagem, perda de pacote e correlação entre as variáveis resultantes do experimento II. Por fim, é realizada uma discussão acerca dos resultados.

5.1 CONFIGURAÇÕES GERAIS

Um fator foi considerado para estudo, compreendendo três níveis: AMQP, CoAP e MQTT. Como explicado anteriormente, dois experimentos foram desenvolvidos para avaliar o desempenho dos protocolos de comunicação em ambientes de Internet das Coisas. O primeiro experimento, teve como objetivo avaliar o desempenho de cada protocolo em uma rede que não disponha de falhas. O segundo experimento possuiu como finalidade avaliar o desempenho de cada protocolo considerando falhas na rede durante a transmissão dos dados.

No ambiente experimental, relatado na Seção 4.1, algumas configurações foram adotadas, com intuito de evitar interferência nos resultados. Uma dessas é o parâmetro *keep alive*, em que foi alterado com o propósito de manter a conexão em todos os nós do ambiente, evitando uma possível reinicialização dos algoritmos.

Para ambos os experimentos realizados, a configuração de entrega de mensagens, foi semelhante, com a finalidade de obter entregas semelhantes. No protocolo AMQP utilizou-se a garantia de entrega *At-most-once* (Pelo menos uma vez), no protocolo CoAP optou-se pela troca de mensagens *Conformable* com o tipo da resposta *Piggybacked*, o cuja resposta da solicitação efetuada pelo cliente é realizada no mesmo pacote, e para o protocolo MQTT empregou-se a Qualidade de Serviço 1 - Ao menos uma vez. A partir dessas configurações, assegura-se que as mensagens cheguem ao destinatário ao menos uma vez, sendo possível a retransmissão de dados.

A duração de cada amostra é de 5 minutos e 10 minutos para os Experimentos I e II, respectivamente, e ambos os resultados foram apresentados por meio da ANOVA (Análise de Variância), com um nível de significância $\alpha = 0{,}05$ (MONTGOMERY; RUNGER, 2010). Também foi adotado um Intervalo de Confiança (IC), o qual está relacionado a um intervalo de valores inferior e superior em torno da estimativa pontual, onde almeja-se ter capturado o parâmetro populacional (FERNANDES, 2017).

5.2 EXPERIMENTO I

O primeiro experimento, como relatado anteriormente, consistiu em analisar o desempenho de cada protocolo em uma rede que não apresente falhas, onde foram avaliados o consumo de banda e o tamanho da mensagem.

A Tabela 5 exibe os resultados da ANOVA, no qual *df* é o grau de liberdade e obteve o valor 2, *F-stat.* é a estatística F com o respectivo *p-value*. Estes foram obtidos com base no procedimento de Tukey, um teste *post-hoc* (MONTGOMERY, 2017). Ao observar os valores dessa Tabela, é possível afirmar, sobretudo a partir dos valores para *p-value*, que existem diferenças significativas entre pelo menos duas as amostras consideradas, para ambas as métricas apresentadas (consumo de banda e tamanho da mensagem).

As Tabelas 6 e 7 apresentam os resultados gerais para as amostras consideradas em cada protocolo, seguido de um Intervalo de Confiança de 95%, em que o consumo de banda é definido por Bytes por segundo (B/s) e o tamanho da mensagem por Bytes por pacote (B/pkg).

Na Tabela 6, ao observar atentamente os valores de IC, podemos identificar um intervalo de confiança estreito principalmente entre os protocolos CoAP e MQTT, em que desse modo, é possível identificar que há uma probabilidade de 95% do valor de consumo de banca encontrar-se dentro deste intervalo.

Ao observar a Tabela 7, o IC para todos os protocolos também possui um intervalo estreito, e ao analisar o intervalo de confiança para o protocolo CoAP, é possível identificar uma diferença de apenas $0,01$ entre o IC mínimo e o IC máximo, o que nos permite afirmar que este protocolo é regular em sua seu tamanho de mensagem para a transmissão de dados, corroborado pelo fato dele utilizar o UDP para suas transmissões.

Tabela 5 – ANOVA Experimento I

Consumo de Banda			**Tamanho da mensagem**		
df	**F-stat**	**p-value**	**df**	**F-stat**	**p-value**
2	2320	$< 0{,}001$	2	30936	$< 0{,}001$

Fonte: Elaborado pela autora

Tabela 6 – Consumo de Banda para o Experimento I

Protocolo	**Média**	**IC (95%)**
AMQP	93,34	[91,50;95,25]
CoAP	37,75	[37,14;38,37]
MQTT	58,81	[57,70;59,96]

Fonte: Elaborado pela autora

Tabela 7 – Tamanho da Mensagem para o Experimento I

Protocolo	**Média**	**IC (95%)**
AMQP	139,78	[138,98;140,58]
CoAP	61	[61;61,01]
MQTT	90	[89,98;90,02]

Fonte: Elaborado pela autora

5.2.1 Consumo de Banda

Considerando a análise do consumo de largura de banda, o protocolo AMQP apresentou o maior resultado, como pode ser visualizado na Figura 25, com um valor de $93,34$ B/s (Bytes por segundo). Este protocolo funciona sobre o TCP na camada de transporte assim como o MQTT, porém, apesar dessa similaridade, o MQTT alcançou uma menor média com um valor de $58,81$ B/s.

Dentre os três protocolos avaliados, o CoAP denotou o menor valor com uma média de $37,75$ B/s. Considerando estas circunstâncias, é possível declarar que o protocolo CoAP possui um menor impacto no consumo da largura de banda da rede, pois uma menor quantidade de Bytes são transferidos durante um período de tempo. Assim, este protocolo pode ser o mais indicado, caso este recurso seja relevante e fundamental para aplicações IoT que sejam restringidos por limitações de rede. O mesmo não é possível afirmar em relação ao protocolo AMQP, tornando-o assim, o menos indicado para ambientes semelhantes ao desenvolvido no corrente experimento.

Uma das possíveis causas do protocolo CoAP ter obtido um resultado melhor, é pelo fato deste protocolo adotar o protocolo UDP em sua camada de transporte. E uma das possíveis causas para o resultado obtido pelo protocolo AMQP é devido à configuração da transmissão de suas mensagens, além da quantidade de informações adicionais que encontram-se juntamente com o cabeçalho do protocolo.

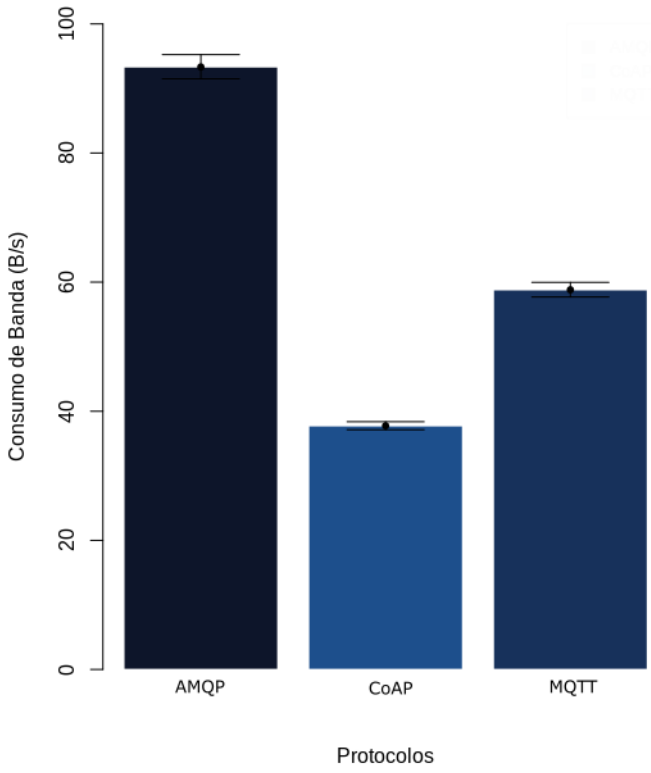


Figura 25 – Experimento I: Consumo de Banda

Fonte: Elaborado pela autora

5.2.2 Tamanho da Mensagem

Ao analisar o tamanho da mensagem é possível identificar que o protocolo CoAP alcançou o menor valor como resultado, com uma média de 61 Bytes por pacote, como ilustrado na Figura 26. O protocolo MQTT obteve uma média de 90 Bytes por pacote, sendo possível justificar este resultado pelo fato deste protocolo funcionar sobre o TCP, o que não ocorre com o CoAP que funciona sobre UDP.

Em contrapartida, o protocolo AMQP, que também funciona sobre o TCP, atingiu o valor mais alto como resultado com uma média de $139,78$ Bytes por pacote. Tal situação pode estar relacionada às configurações de cada protocolo, em que o protocolo AMQP requer uma configuração mais detalhada para a comunicação, o que torna o tamanho do seu pacote maior.

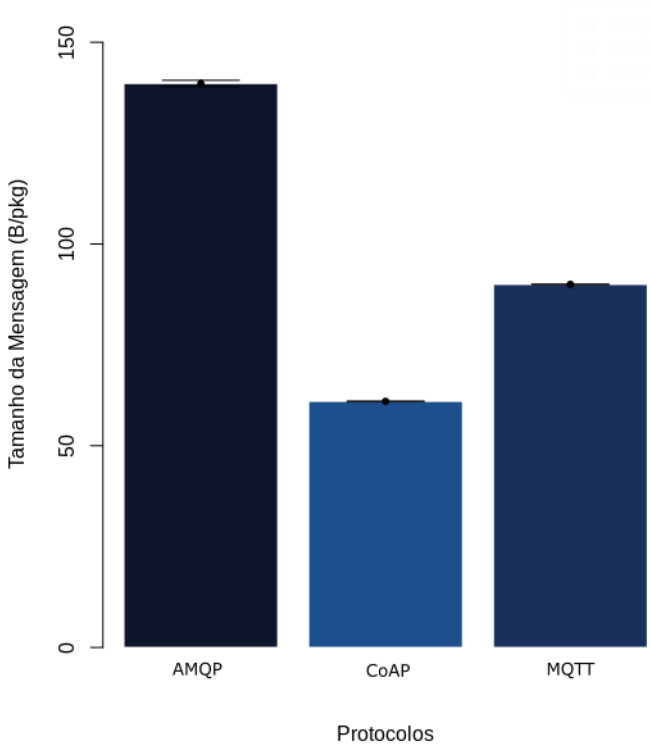


Figura 26 – Experimento I: Tamanho da Mensagem

Fonte: Elaborado pela autora

Considerando que as amostras coletadas possuem outros protocolos de outras camadas associadas à mensagem transmitida, realizou-se uma fragmentação, agregando a carga útil e todas as informações pertencentes ao protocolo estudado, ou seja, todas as informações pertencentes ao cabeçalho fixo e variável. Os resultados obtidos, encontram-se na Tabela 8. Ao analisar essa Tabela, é possível identificar que o Intervalo de Confiança possuem uma diferença pequena entre o IC máximo e o IC mínimo, principalmente em relação ao protocolo CoAP, cujo valor da diferença é $0,01$. Isto nos permite corroborar a questão deste protocolo possuir regularidade em seu tamanho de mensagem, ou seja, o tamanho da mensagem considerando a carga útil e os dados do protocolo possuem pouca variação.

Tabela 8 – Experimento I: Dados do Protocolo + Carga Útil

Protocolo	**Média**	**IC (95%)**
AMQP	73,55	[72,79;74,32]
CoAP	19	[19;19,01]
MQTT	23,98	[23,97;24]

Fonte: Elaborado pela autora

Neste contexto, o protocolo AMQP obteve uma média de 73, 55 Bytes por pacote, sucedendo novamente o maior valor comparado aos outros protocolos, como visualizado na Figura 27. O MQTT teve uma média de 23, 98 Bytes por pacote, e o CoAP resultou no menor valor dentre os três protocolos, com uma média de 19 Bytes por pacote.

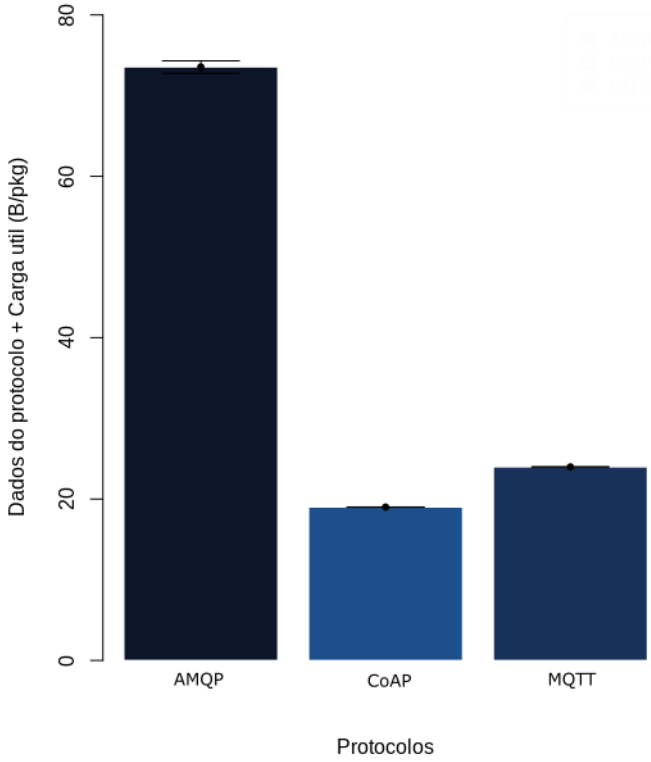


Figura 27 – Experimento I: Tamanho da Mensagem considerando Dados do Protocolo e Carga Útil

Fonte: Elaborado pela autora

Nesta circunstância, é possível afirmar que o protocolo CoAP é considerado o protocolo mais leve comparado aos outros, sendo assim, o mais indicado para ambientes IoT que possuam restrições de transferência de mensagens. Uma das possíveis justificativas para estes resultados, é que, como citado anteriormente, o protocolo CoAP funciona sobre o UDP na camada de transporte. Contudo existe uma outra questão que pode ter influenciado nos resultados, que são as configurações existentes e exigidas em cada um dos protocolos. Além disso, existe o tamanho do cabeçalho fixo e cabeçalho variável em cada um deles, como mencionado na Seção 2.2, onde cada cabeçalho exige um conjunto de itens a serem configurados.

5.2.3 Correlação

A seguir, é explorada a correlação entre o tamanho da mensagem e o consumo de banda, englobando todos os níveis do fator estudado, cujos resultados foram observados anteriormente. A correlação tem como finalidade verificar a existência de alguma relação entre as variáveis, e a observação para esse experimento foi realizada a partir da correlação de Pearson (BARBETTA; REIS; BORNIA, 2004; MUKAKA, 2012).

A Figura 28 apresenta a correlação entre o tamanho da mensagem e o consumo de banda para os protocolos AMQP, CoAP e MQTT, considerando os resultados obtidos no Experimento I, onde é possível verificar a existência de uma forte correlação positiva entre as variáveis, confirmada pelo coeficiente de determinação, cujo valor resultante é $r^2 = 0,979$.

Ao se observar o eixo y, que corresponde ao consumo de banda, a reta de regressão linear atinge valores um pouco acima de 90 B/s que são aproximados aos exibidos na Figura 25, em que mostra os resultados médios para consumo de banda. A partir disso, pode-se afirmar que o tamanho das mensagens enviadas pelos sensores impactam diretamente no aumento do consumo de banda da rede, fazendo com que mais dados sejam transmitidos e consuma uma maior largura de banda.

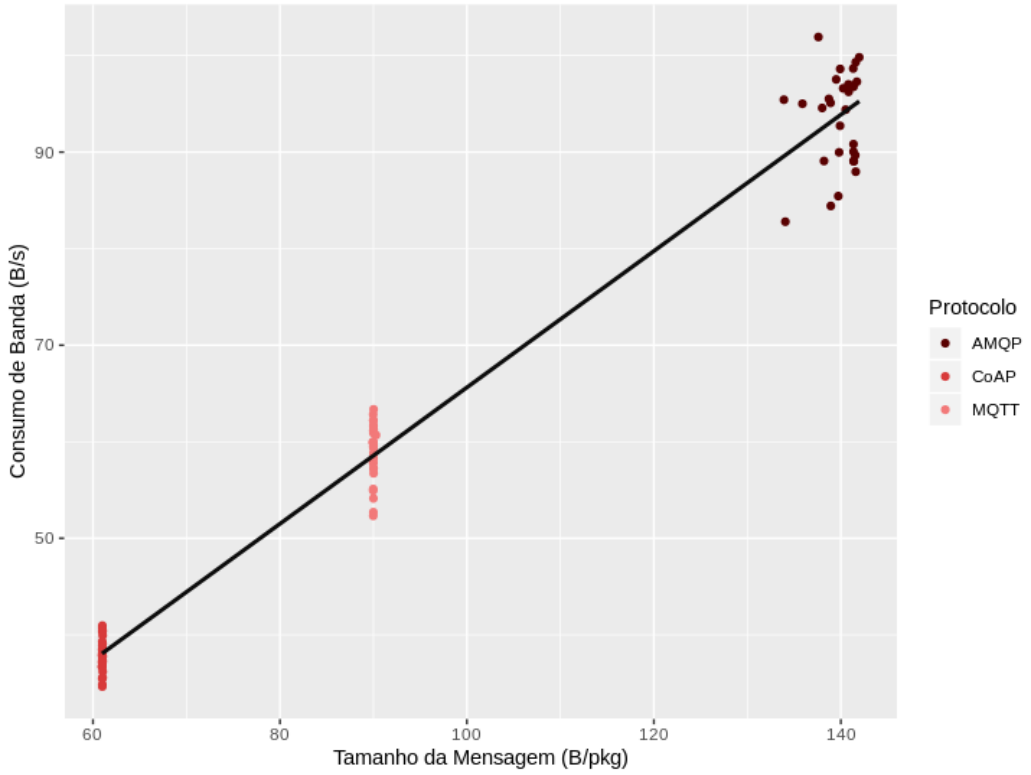


Figura 28 – Experimento I: Correlação Tamanho da Mensagem x Consumo de Banca

Fonte: Elaborado pela autora

5.3 EXPERIMENTO II

No segundo experimento, cuja finalidade foi avaliar o desempenho dos protocolos considerando falhas na rede durante a comunicação entre os sensores e o servidor, ou seja, foi aplicada a injeção de falhas em rotas da rede como relatado na Seção 4.1. Além de terem sido avaliados o consumo de banda e o tamanho da mensagem, também analisou-se a perda de pacote.

Os resultados para ANOVA são expostos na Tabela 9, em que *df* é o grau de liberdade obtido, cujo o valor resultante é 2; *F-stat.* é a estatística F, e temos também os valores para *p-value* que, para todas as métricas avaliadas resultou em um valor $< 0,001$. Todos os resultados também foram obtidos com base no procedimento de Tukey, um teste *post-hoc* (MONTGOMERY, 2017).

Ao examinar os valores desta Tabela, é possível identificar através dos valores para *p-value*, que existem diferenças significativas entre pelo menos duas amostras para todas as métricas estudadas neste experimento (Consumo de Banda, Tamanho da Mensagem e Perda de Pacotes).

Os resultados gerais obtidos para cada uma das métricas podem ser observadas nas Tabelas 10, 11 e 12, em que utiliza-se um Intervalo de Confiança de 95%, onde o consumo de banda é definido por Bytes por segundo (B/s) e o tamanho da mensagem por Bytes por pacote (B/pkg) e a taxa de perda de pacotes é apresentada em percentual (%).

Na Tabela 10, ao observar atentamente os valores de IC, podemos identificar um intervalo de confiança pequeno para os protocolos CoAP e MQTT, em que podemos dizer que existe uma probabilidade de 95% do valor se encontrar dentro deste intervalo. No protocolo AMQP o intervalo de confiança mínimo e máximo apresenta uma diferença maior, onde, a partir disso podemos identificar uma maior variabilidade no consumo de largura de banda para este protocolo.

O Intervalo de Confiança para o tamanho da mensagem apresentado na Tabela 11 apresenta um valor estreito para todos os protocolos. Assim, é possível identificar pouca variabilidade entre o conjunto de resultados obtidos a partir das amostras coletadas.

Na Tabela 12 também podemos identificar um IC pequeno para todos os protocolos e, além disso, podemos afirmar inicialmente que existe uma diferença significativa entre o protocolo AMQP e os protocolos CoAP e MQTT, onde o resultado para AMQP foi consideravelmente maior.

Tabela 9 – ANOVA Experimento II

Consumo de Banca			**Tamanho da mensagem**			**Perda de pacote**		
df	**F-stat**	**p-value**	**df**	**F-stat**	**p-value**	**df**	**F-stat**	**p-value**
2	2739	<0,001	2	25868	<0,001	2	11886	<0,001

Fonte: Elaborado pela autora

Tabela 10 – Consumo de Banda para o Experimento II

Protocolo	**Média**	**IC (95%)**
AMQP	100,37	[97,61;103,3]
CoAP	35,18	[34,52;35,86]
MQTT	59,48	[58,88;60,08]

Fonte: Elaborado pela autora

Tabela 11 – Tamanho da Mensagem para o Experimento II

Protocolo	**Média**	**IC (95%)**
AMQP	94,57	[94,21;94,94]
CoAP	63,7	[63,64;63,75]
MQTT	92,18	[92,12;92,25]

Fonte: Elaborado pela autora

Tabela 12 – Perda de Pacote para o Experimento II

Protocolo	**Média (%)**	**IC (95%)**
AMQP	45,28	[44,52;46,04]
CoAP	1	[0,68;1,33]
MQTT	00,48	[0,35;0,61]

Fonte: Elaborado pela autora

5.3.1 Consumo de Banda

Os resultados obtidos acerca do consumo de banda são exibidos na Figura 29. É possível verificar que o protocolo AMQP apresentou a maior média com um valor de $100,37$ B/s (Bytes por segundo), assim como obteve o maior valor no Experimento I, visto na Figura 25. Tal circunstância, neste experimento, pode estar relacionada às retransmissões adicionais, que podem incluir dados duplicados recebidos pelo servidor. Por conseguinte, o AMQP possivelmente não seja adequado para aplicações em que a rede é suscetível a falhas.

O protocolo MQTT apresentou uma média de $59,48$ B/s, todavia, ao se comparar este resultado com o Experimento I, o valor para o consumo de banda obteve um pequeno aumento, porém, este valor encontra-se dentro do Intervalo de Confiança apresentado na Tabela 6.

Diferentemente dos outros, o protocolo CoAP obteve o menor consumo de largura de banda, com uma média de $35,18$ B/s, e comparando com o Experimento I, esse protocolo também apresentou um valor menor. Nesse contexto, é possível afirmar que o protocolo CoAP é capaz de lidar com problemas na rede, sem aumentar o tráfego desta, sendo o

mais indicado para redes passíveis de falhas.

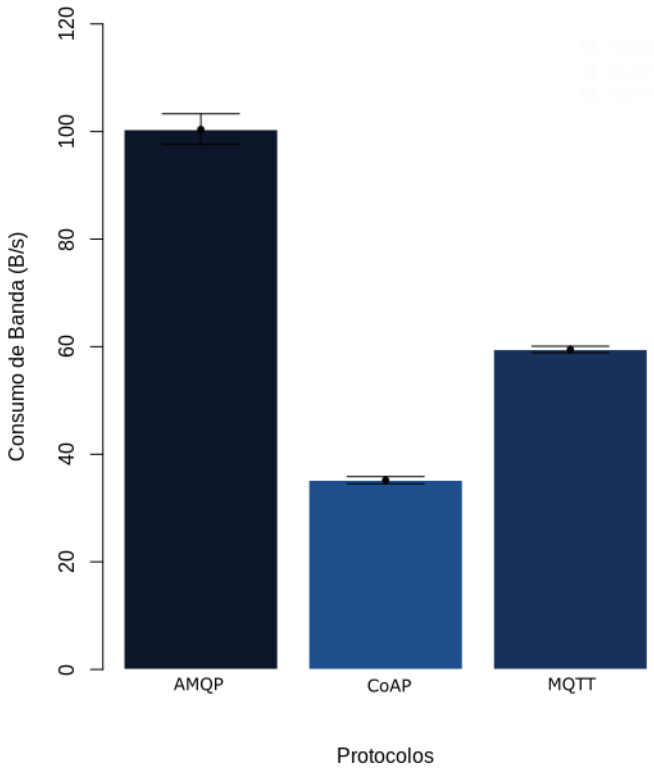


Figura 29 – Experimento II: Consumo de Banda

Fonte: Elaborado pela autora

5.3.2 Tamanho da mensagem

A Figura 30 exibe os resultados dos três protocolos estudados para o tamanho da mensagem. O protocolo AMQP resultou novamente em uma maior média para o tamanho das mensagens, com um valor de $94,57$ Bytes por pacote. Contudo, ao se comparar este resultado com o obtido no Experimento I, visualizado na Figura 26, é perceptível que esta média é menor. Tal situação deve-se ao fato de que, quando o AMQP constata algum problema na conexão, que resulta em várias tentativas de retransmissão dos dados, essa retransmissão é realizada sem que todos os dados sejam enviados. No momento em que constata-se que a conexão foi restaurada, os dados voltam a serem transmitidos em sua completude.

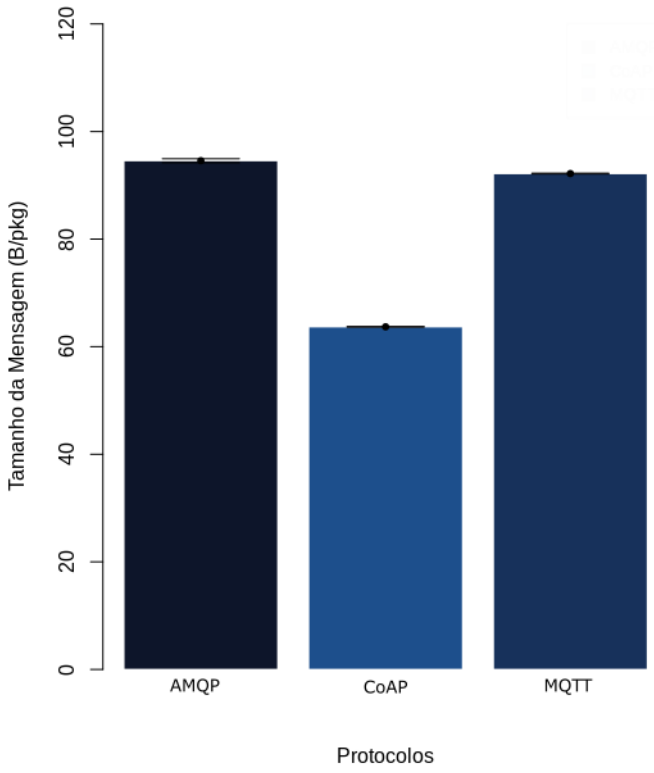


Figura 30 – Experimento II: Tamanho da Mensagem

Fonte: Elaborado pela autora

O protocolo MQTT obteve uma média para o tamanho da mensagem de 92, 18 Bytes por pacote, que comparado ao Experimento I, sofreu um pequeno aumento. Novamente o protocolo CoAP apresentou um menor valor comparado aos outros dois, com uma média de 63, 70 Bytes por pacote, sendo possível afirmar neste caso, que a falha na rede não influencia no tamanho da mensagem para o CoAP, ao contrário dos outros protocolos. Uma possível justificativa é a adoção do UDP que o CoAP realiza e que os outros dois diferentemente, adotam TCP na camada de transporte, e este por sua vez possui seu próprio mecanismo de confirmação de recebimento de mensagem.

Considerando novamente que as amostras coletadas possuem outros protocolos de outras camadas associadas à mensagem transmitida, efetuou-se uma fragmentação, agregando a carga útil e todas as informações pertencentes ao protocolo estudado, o qual os resultados gerais encontrados, podem ser vistos na Tabela 13, com um Intervalo de Confiança de 95%. Ao analisar esta Tabela, é possível identificar que o Intervalo de Confiança possuem uma diferença pequena entre o IC máximo e o IC mínimo. Com isso, é possível afirmar que os protocolos juntamente com a carga útil possuem pouca variação.

Tabela 13 – Experimento II: Dados do Protocolo + Carga Útil

Protocolo	**Média**	**IC (95%)**
AMQP	26,56	[26,20;26,93]
CoAP	19	[19;19,02]
MQTT	24,18	[24,12;24,25]

Fonte: Elaborado pela autora

Nesta circunstância, o protocolo AMQP obteve uma média de $26,56$ Bytes por pacote, decorrendo outra vez no maior valor comparado aos outros protocolos, como visualizado na Figura 31. É possível identificar que este valor é consideravelmente menor quando comparado ao valor resultante no Experimento I, apresentado na Tabela 8, o que corrobora a afirmação citada anteriormente, de que ao perceber problemas na comunicação o protocolo reduz a quantidade de dados enviados até que a conexão seja restaurada.

O MQTT, por sua vez, resultou em uma média de $24,18$ Bytes por pacote, já o CoAP, apresentou novamente o menor valor, com uma média de 19 Bytes por pacote. Comparado estes resultados aos valores obtidos no Experimento I, vistos na Figura 27, os valores médios para os protocolos CoAP e MQTT são semelhantes. Ao executar o teste de Tukey, foi constatado que as amostras para todos os grupos são estatisticamente diferentes com um $p\text{-}value < 0,001$.

Desta forma, é possível afirmar que os protocolos CoAP e MQTT não sofrem interferências no tamanho da mensagem em ambientes suscetíveis a falha. No entanto, o mesmo não ocorre com o protocolo AMQP que apresentou uma diminuição no valor.

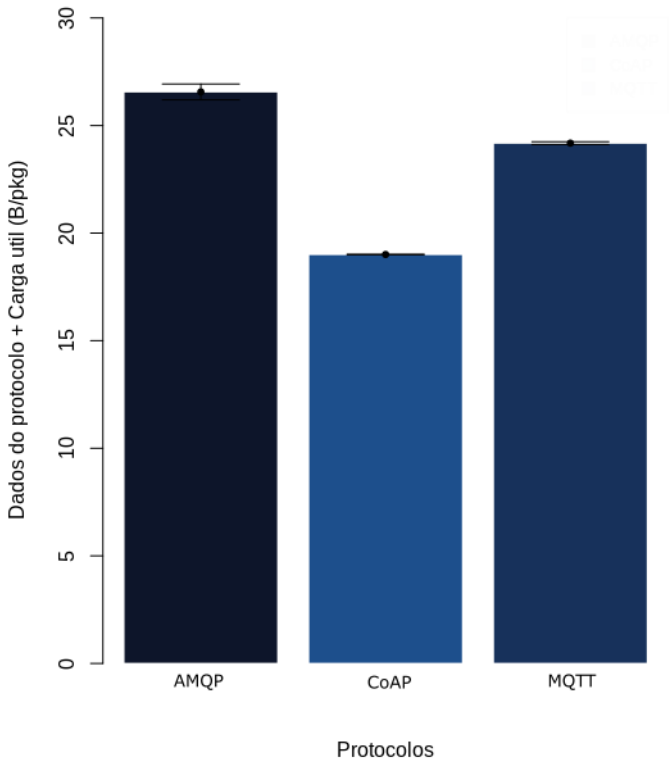


Figura 31 – Experimento II: Tamanho da Mensagem considerando Dados do Protocolo e Carga Útil

Fonte: Elaborado pela autora

5.3.3 Perda de Pacotes

Com relação à perda de pacotes, os resultados podem ser visualizados na Figura 32, sendo possível afirmar que os protocolos MQTT e CoAP apresentaram as menores taxas de perda de pacote. O MQTT resultou em uma taxa de $0,48\%$, e o CoAP, apresentou uma taxa de 1%.

Ao se comparar as médias das amostras obtidas para estes dois protocolos por meio do procedimento de Tukey, foi constatado que as médias não são estatisticamente diferentes, com um *p-value* $= 0,267$. Por outro lado, o AMQP incorreu em taxa de $45,28\%$ de perda de pacote, mesmo com a redução dos dados transmitidos nas retransmissões quando o protocolo detecta problemas na conexão.

Assim, é possível comprovar que esses dois protocolos obtiveram os melhores resultados, sendo estes os mais indicados para aplicações e ambientes IoT, onde a perda de pacotes ínfima seja um fator fundamental para a transmissão dos dados. Em contrapartida, o protocolo AMQP resultou em uma alta taxa de perda de pacote, sendo assim, o menos indicado para ambientes semelhantes ao experimento proposto neste estudo.

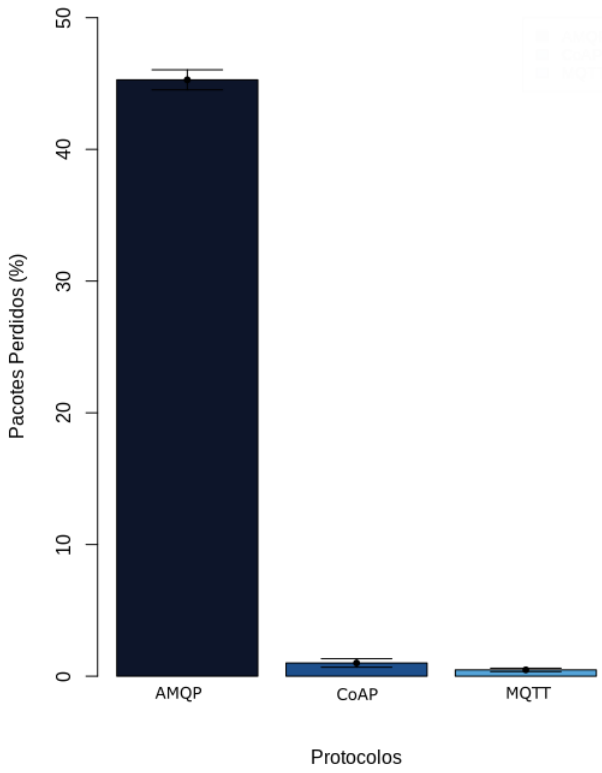


Figura 32 – Experimento II: Perda de Pacote

Fonte: Elaborado pela autora

5.3.4 Correlação

A seguir, é examinada a correlação entre o tamanho da mensagem e o consumo de banda, entre o tamanho da mensagem e a perda de pacotes, e entre o consumo de banda e a perda de pacotes, envolvendo os três níveis do fator protocolos, cujos resultados obtidos foram exibidos anteriormente. A análise da correlação para o presente experimento, foi feita por meio da correlação de Pearson, e tem como propósito analisar se existe alguma ligação entre o aumento ou a diminuição das variáveis (BARBETTA; REIS; BORNIA, 2004; MUKAKA, 2012).

A Figura 33 apresenta a correlação entre o tamanho da mensagem e o consumo de banda para os protocolos AMQP, CoAP e MQTT, apontando assim, uma correlação moderada positiva confirmada pelo coeficiente de determinação, cujo valor obtido foi $r^2 = 0,662$. Dessa forma, é possível afirmar que o aumento da média do tamanho da mensagem, não afeta de modo considerável no aumento do consumo de largura de banda da rede.

Na análise da relação entre o tamanho da mensagem e a perda de pacote, visualizado na Figura 34, observa-se uma correlação relativamente fraca positiva, ou seja, o resultado para o coeficiente de determinação (r^2) apontou um valor menor que $0,50$, mais especificamente

$r^2 = 0,301$. Mesmo com um tamanho de mensagem maior para o protocolo MQTT, isto não influencia na taxa de perda de pacotes, que para esse protocolo apresentou uma média de $0,48\%$, como exibido na Figura 32. Neste sentido, é possível informar que o aumento do tamanho da mensagem exerce pouquíssima influência na taxa de perda de pacotes.

A Figura 35 ilustra a relação entre o consumo de banda e a perda de pacote para os protocolos AMQP, CoAP e MQTT considerando os resultados obtidos no experimento. A partir da observação dessa Figura, é possível observar uma forte correlação positiva entre as variáveis, confirmada pelo coeficiente de determinação cujo valor obtido é $r^2 = 0,834$, ou seja, o coeficiente apontou um valor consideravelmente maior que $0,50$. Ao se observar o eixo y que corresponde à perda de pacotes, é possível identificar que a reta de regressão linear atinge valores pouco acima de 40%, o qual corresponde aproximadamente aos valores apresentados para as perdas de pacotes exibidas na Figura 25. Neste contexto, é possível afirmar que o consumo de banda exerce uma forte influência no aumento da perda de pacotes da rede.

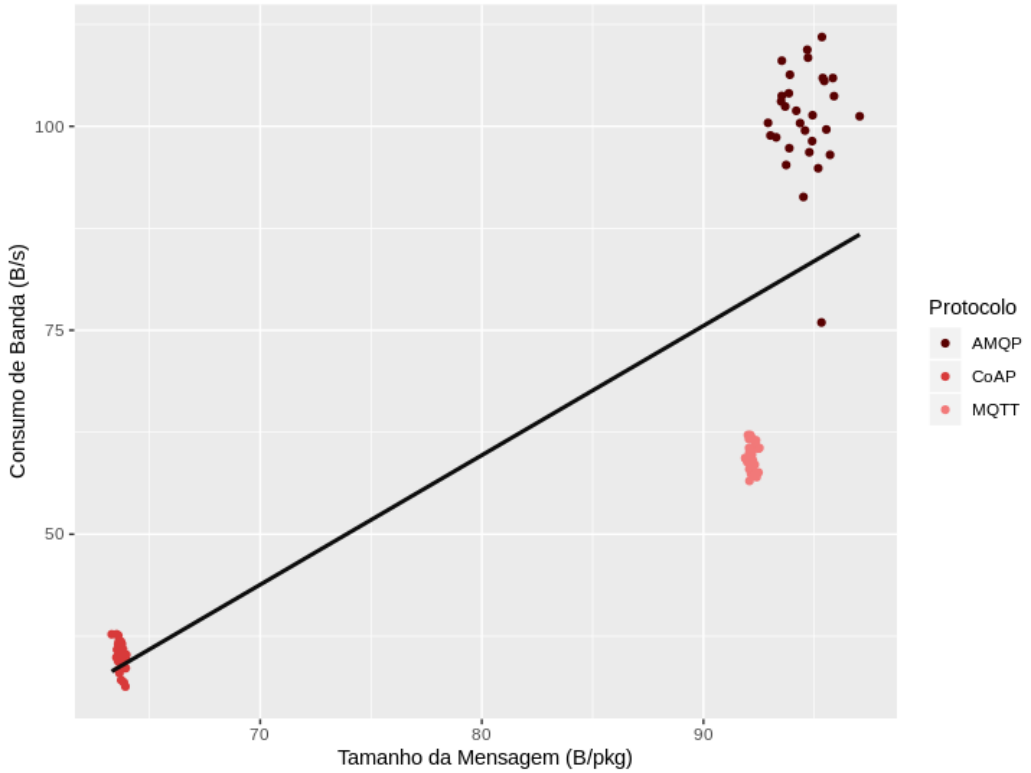


Figura 33 – Experimento II: Correlação Tamanho da Mensagem x Consumo de Banda

Fonte: Elaborado pela autora

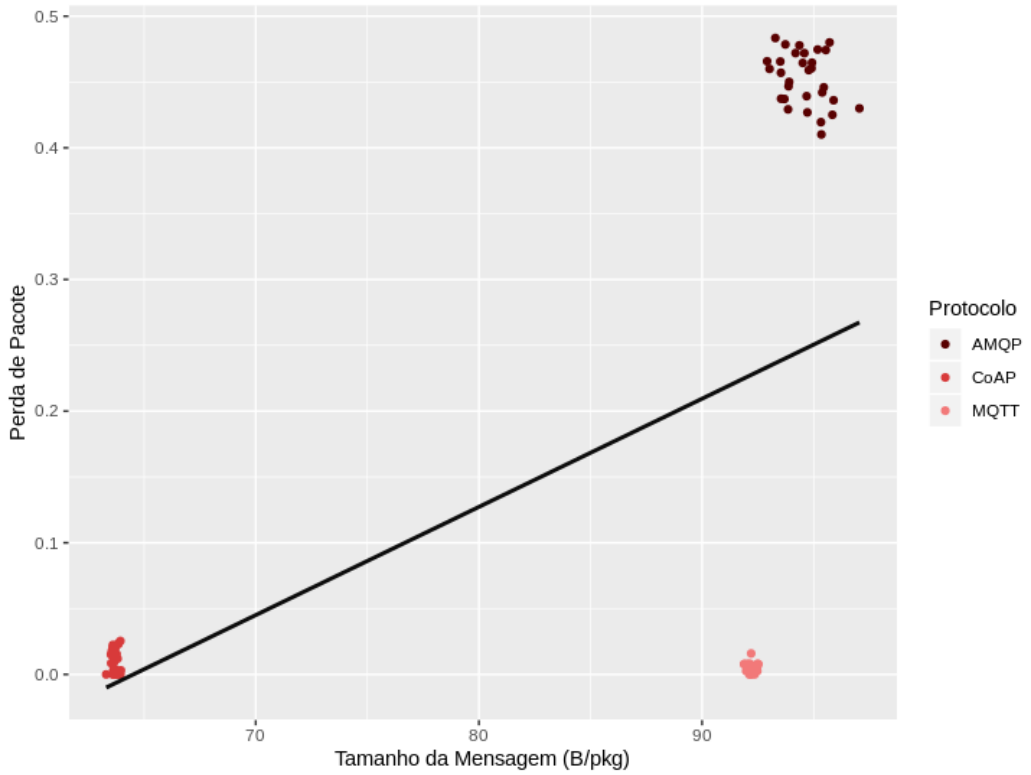


Figura 34 – Experimento II: Correlação Tamanho da Mensagem x Perda de Pacote

Fonte: Elaborado pela autora

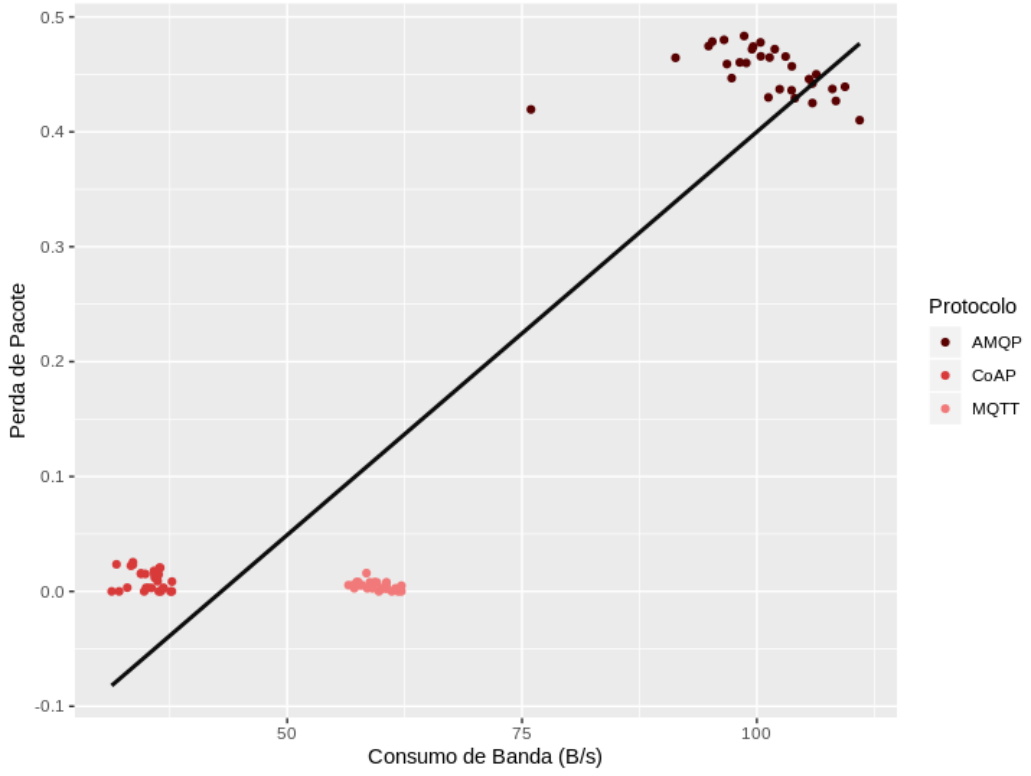


Figura 35 – Experimento II: Correlação Consumo de Banda x Perda de Pacote

Fonte: Elaborado pela autora

5.4 DISCUSSÃO

No presente estudo desta dissertação, dois experimentos foram realizados, sendo um considerando um ambiente comum e outro considerando interrupções recorrentes na rede. Em um contexto geral, os resultados de ambos os experimentos apresentaram em comum os melhores resultados para um mesmo protocolo. Os resultados apontaram também o mesmo protocolo, que pode-se considerar o menos ideal para aplicação no ambiente proposto.

No contexto do Experimento I, em que o ambiente não possui de interrupções de conexão na rede, o protocolo CoAP alcançou os melhores resultados, para ambas as métricas analisadas, com uma média de $37,75$ B/s para o consumo de banda e 61 B/pkg para o tamanho do pacote, seguido pelo protocolo MQTT, com os valores de $58,81$ B/s e 90 para o consumo de banda e tamanho da mensagem, respectivamente. Ao analisar apenas os dados referentes ao protocolo e à carga útil transmitida, os dois protocolos persistem nos seus resultados, com os valores médios de 19 B/pkg e $23,98$ B/pkg para CoAP e MQTT.

Outras pesquisas relatadas no Capítulo 3, como as de Mijovic, Shehu e Buratti (2016), Collina et al. (2014), Marques e Kniess (2018) relatam que os seus resultados também apresentaram o protocolo CoAP como um dos mais indicados comparados a outros protocolos, utilizando outras métricas para estudo.

Nesta perspectiva, é possível sugerir o protocolo CoAP para ser aplicado em ambientes IoT similares ao desenvolvido no experimento, uma vez que este parece ser um protocolo viável, no sentido que utiliza menos recursos de rede do que os outros protocolos. Em contrapartida, o protocolo AMQP produz mensagens maiores, o que afeta a quantidade de dados a serem transmitidos pela rede, consumindo assim, uma maior largura de banda.

No tocante à falha na rede, desenvolvido no Experimento II, o protocolo CoAP decorreu novamente em um melhor desempenho, obtendo os melhores resultados para as métricas analisadas, com médias de $35,18$ B/s e $63,7$ B/s, seguido também do protocolo MQTT. Com relação à perda de pacotes, o protocolo MQTT apresentou um percentual menor que o CoAP, com percentuais de $0,48\%$ e $0,01\%$, respectivamente, porém os resultados estatísticos obtidos a partir do teste de Tukey, apontaram que ambos os protocolos possuem perda de pacote similares.

Algumas pesquisas relatadas no Capítulo 3, como Collina et al. (2014), Chen e Kunz (2016) também relatam a questão de falhas na rede, porém em seus experimentos a falha é tratada como um parâmetro, avaliado outras métricas a partir de um percentual crescente de perda de pacotes. O experimento realizado nesta dissertação avalia o percentual de perda de pacotes a partir das interrupções recorrentes na rede. Apesar desta diferença, os autores Collina et al. (2014) também concluiu que o protocolo CoAP apresentou os melhores resultados.

Neste contexto, o protocolo CoAP é o mais adequado na aplicabilidade de ambientes IoT com falhas recorrentes na rede, pois mostrou que utiliza melhor a largura de banda

da rede. Embora o protocolo AMQP utilize uma manipulação para a transmissão de dados quando identifica falhas na comunicação, reduzindo assim, a quantidade de dados a serem trafegados, este não obteve bons resultados principalmente na questão de perda de pacotes. Destarte, este protocolo não é ideal para aplicações IoT com configurações semelhantes ao desenvolvido no corrente experimento.

Com base em ambos os experimentos produzidos e executados, o CoAP aparenta ser um protocolo adequado para aplicativos com recursos de rede restritos e ambientes IoT que dispunha principalmente de sensores. No entanto, o presente trabalho não ajustou configurações específicas de cada protocolo, exceto as configurações de comunicação para entrega de mensagens e o tempo de vida para continuação de uma mesma comunicação, mesmo com falhas na rede. É possível que outros resultados possam ser obtidos ajustando-se as configurações particulares de cada protocolo, contudo está fora do escopo deste trabalho.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este capítulo apresenta as conclusões obtidas na presente dissertação. Inicialmente, são apresentadas as conclusões dos trabalhos realizados. Em seguida, são descritas as limitações encontradas no decorrer da pesquisa. Por fim, são elencados os trabalhos futuros.

6.1 TRABALHO REALIZADO

O termo Internet das Coisas (IoT) surgiu há algumas décadas, e cada vez mais tem se expandindo e tem abrangido diversas áreas. Dentro deste contexto, estão inclusos vários sensores, tecnologias *wearable*, carros autônomos, *smart homes* e diversos outros dispositivos que encontram-se conectados à rede e conversam entre si, compondo assim a comunicação Máquina a Máquina (M2M). Desse modo, pesquisas relatam que bilhões de dispositivos estarão conectados e comunicando-se em alguns anos.

Devido a essa perspectiva da quantidade de dispositivos conectados e pelo fato da heterogeneidade existente entre eles, existem muitos desafios e pesquisas associadas, como por exemplo, a comunicação desses objetos nas aplicações desenvolvidas para os ambientes IoT. Muitas pesquisas foram dedicadas a projetar e avaliar protocolos de comunicação para sistemas baseados em Internet das Coisas, com intuito de se obter protocolos mais eficientes, pois uma sobrecarga na rede por causa de um protocolo inapropriado para um ambiente, poderá impactar negativamente na transmissão de dados, aumentar a largura de banda e gerar uma alta taxa de perda de pacotes.

Neste contexto, este trabalho teve como objetivo realizar um estudo acerca da avaliação de desempenho de protocolos de comunicação da camada de aplicação voltados para Internet das Coisas, onde três protocolos foram selecionados para pesquisa sendo eles o AMQP, CoAP e MQTT, que são comumente citados na literatura. A metodologia desenvolvida, descrita no Capítulo 4, foi baseada no projeto de experimentos *Design of Experiments* considerando o protocolo como fator com três níveis com 30 amostras obtidas para análise das métricas. Ademais, dois experimentos foram projetados e desenvolvidos para a avaliação, fornecendo informações importantes de cada protocolo relacionado ao consumo de banda, tamanho da mensagem e perda de pacotes devido a falhas na rede.

No primeiro experimento, cujo ambiente não dispõe de falhas na rede, os resultados alcançados mostraram que o protocolo CoAP obteve os melhores valores, tanto para o consumo de banda quanto para o tamanho da mensagem, com diferenças significativas em comparação aos outros protocolos. Assim, é possível afirmar, que este protocolo é o mais indicado para ambientes IoT similares ao desenvolvido no experimento, uma vez que este utiliza menos recursos de rede do que os outros protocolos. Em contrapartida, o protocolo AMQP produz mensagens maiores, e possui um alto valor no consumo de

banda, afetando na quantidade de dados a serem transmitidos pela rede, consumindo assim uma maior largura de banda. Ademais, identificou-se uma forte correlação entre as variáveis, constatando que o tamanho da mensagem impacta diretamente no aumento do consumo de banda, fazendo com que o aumento na quantidade de dados transmitidos consuma mais largura de banda.

O segundo experimento considera a injeção de falhas na rede, e apontou que o protocolo CoAP obteve os melhores resultados para o tamanho da mensagem e o consumo de banda novamente. Ao se analisar a taxa de perda de pacotes, o protocolo MQTT apresentou o menor valor seguido do protocolo CoAP. Contudo os valores obtidos não apresentam diferenças estatisticamente significativas. Nesta circunstância, sugere-se a utilização do protocolo CoAP, pois ele é capaz de lidar com problemas de conexão da rede sem aumentar o tráfego desta, como também não possui alta taxa de perda de pacote, o que é considerado essencial para a transmissão de dados em diversas aplicações IoT, sendo assim o mais indicado para redes passíveis de falhas.

Este trabalho mostrou que, apesar dos protocolos estudados serem voltados para Internet das Coisas, alguns deles são mais indicados que outros, para ambientes com sensores simples e poucos recursos, mostrando assim a importância de estudos voltados para essa área.

6.2 LIMITAÇÕES

No decorrer do desenvolvimento deste trabalho, algumas limitações foram encontradas, principalmente devido ao tempo limitado para elaboração de mais experimentos, em razão de alguns imprevistos como a necessidade de criar uma rede sem fio exclusiva com intuito de evitar influências nos resultados obtidos em consequência a fatores externos. Um outro fator contribuinte, foi o processo de adição de equipamentos que se enquadram dentro das características de IoT, e em virtude disso, ocorreu a ausência de mais equipamentos em outras potências energéticas e capacidades computacionais, e a análise de outras métricas. Outros protocolos de comunicação voltados para aplicações IoT, como por exemplo, os protocolos *Extensible Messaging and Presence Protocol* (XMPP) e *Data Distribution System* (DDS), foram considerados inicialmente, porém, estes possuem um suporte à comunidade regular, principalmente na questão ambientes constituídos por sensores. Também não foi possível apresentar configurações do ambiente adicionais.

6.3 TRABALHOS FUTUROS

Os resultados apresentados neste trabalho, induzem a realização de estudos que proporcionem a continuidade desta pesquisa para avaliação de protocolos em aplicações de Internet das Coisas. A seguir, são listados alguns itens a serem considerados para trabalhos futuros:

- Estudar outros protocolos e tecnologias de troca de mensagens que não sejam exclusivos para aplicações IoT, a exemplo do *Websocket* relatado nas pesquisas de Kayal e Perros (2017), Mijovic, Shehu e Buratti (2016), Yassein, Shatnawi et al. (2016), com o objetivo de verificar se essas tecnologias possuem melhores resultados quando comparadas aos protocolos estudados no presente estudo;

- Replicar esses experimentos em um ambiente de redes móveis, a exemplo das redes 4G e 5G, de modo a comparar os resultados com os resultados obtidos;

- Desenvolver novos experimentos adicionando novos equipamentos em outras capacidades energéticas e computacionais, como Arduinos, Sensores sem Fio, tendo como objetivo de verificar o comportamento destes protocolos em uma rede mista com vários dispositivos tornando mais semelhante a uma rede IoT encontrada no cotidiano;

- Replicar esses experimentos adicionando outras métricas, como por exemplo o consumo de energia, a quantidade de processamento utilizado, entre outras, com a finalidade de avaliar quais desses protocolos influenciam mais nas questões de hardware dos equipamentos;

- Replicar os experimentos, realizando modificações nos algoritmos de envio e recebimento de dados dos protocolos em relação às suas configurações, como por exemplo do tipo de garantia de entrega e configurações de segurança, com intuito de verificar o comportamento dos protocolos ao se alterar essas informações.

- A partir de pesquisas acerca das Redes de Petri e Cadeias de Markov, buscar criar modelos de simulação para avaliação do desempenho com base no ambiente proposto neste trabalho.

REFERÊNCIAS

AL-FUQAHA, A.; GUIZANI, M.; MOHAMMADI, M.; ALEDHARI, M.; AYYASH, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, IEEE, v. 17, n. 4, p. 2347–2376, 2015.

ARASTEH, H.; HOSSEINNEZHAD, V.; LOIA, V.; TOMMASETTI, A.; TROISI, O.; SHAFIE-KHAH, M.; SIANO, P. Iot-based smart cities: a survey. In: IEEE. *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*. [S.l.], 2016. p. 1–6.

ASIM, M. A survey on application layer protocols for internet of things (iot). *International Journal of Advanced Research in Computer Science*, International Journal of Advanced Research in Computer Science, v. 8, n. 3, 2017.

BANKS, A.; GUPTA, R. Mqtt version 3.1. 1 plus errata 01. *OASIS Standard*, 2015.

BANZI, M.; SHILOH, M. *Primeiros Passos com o Arduino–2ª Edição: A plataforma de prototipagem eletrônica open source*. [S.l.]: Novatec Editora, 2015.

BARBETTA, P. A.; REIS, M. M.; BORNIA, A. C. *Estatística: para cursos de engenharia e informática*. [S.l.]: Atlas São Paulo, 2004. v. 3.

BELLAVISTA, P.; ZANNI, A. Towards better scalability for iot-cloud interactions via combined exploitation of mqtt and coap. In: IEEE. *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*. [S.l.], 2016. p. 1–6.

BORGIA, E. The internet of things vision: Key features, applications and open issues. *Computer Communications*, Elsevier, v. 54, p. 1–31, 2014.

BROWN, E. *Top 10 Best Open-Spec Hacker SBCs | Linux.com | The source for Linux information*. 2016. Disponível em: <https://www.linux.com/news/raspberry-pi-stays-top-survey-81-open-spec-sbcs>. Acesso em: 25 abr. 2019.

CARMO, T. d. R. *Uso do padrão AMQP para transporte de mensagens entre atores remotos*. Tese (Doutorado) — Universidade de São Paulo, 2012.

CARO, N. D.; COLITTI, W.; STEENHAUT, K.; MANGINO, G.; REALI, G. Comparison of two lightweight protocols for smartphone-based sensing. In: IEEE. *2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*. [S.l.], 2013. p. 1–6.

CARVALHO, G. *Modelagem e análise de desempenho de esquemas de alocação de recursos em redes móveis celulares*. Tese (Doutorado) — Tese de Doutorado, Instituto de Tecnologia-Programa de Pós-Graduação em Engenharia Elétrica-Universidade Federal do Pará, 2005.

CHEN, Y.; KUNZ, T. Performance evaluation of iot protocols under a constrained wireless access network. In: IEEE. *Selected Topics in Mobile & Wireless Networking (MoWNeT), 2016 International Conference on*. [S.l.], 2016. p. 1–7.

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

COMBS, G. et al. *Wireshark*. 2008. Disponível em: <https://www.wireshark.org/>. Acesso em: 25 abr. 2019.

EGHAM. *Gartner Says 8.4 Billion Connected "Things"Will Be in Use in 2017, Up 31 Percent From 2016*. 2017. Disponível em: <https://www.gartner.com/newsroom/id/3598917>. Acesso em: 15 jun. 2018.

FERNANDES, J. L.; LOPES, I. C.; RODRIGUES, J. J.; ULLAH, S. Performance evaluation of restful web services and amqp protocol. In: IEEE. *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*. [S.l.], 2013. p. 810–815.

FERNANDES, S. *Performance Evaluation for Network Services, Systems and Protocols*. [S.l.]: Springer, 2017.

FRIGIERI, E. P.; MAZZER, D.; PARREIRA, L. M2m protocols for constrained environments in the context of iot: A comparison of approaches. In: *International Telecommunications Symposium*. [S.l.: s.n.], 2015.

GAY, W. Dht11 sensor. In: *Advanced Raspberry Pi*. [S.l.]: Springer, 2018. p. 399–418.

GRGIĆ, K.; ŠPEH, I.; HEđI, I. A web-based iot solution for monitoring data using mqtt protocol. In: IEEE. *2016 International Conference on Smart Systems and Technologies (SST)*. [S.l.], 2016. p. 249–253.

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

Inc. Pivotal Software. *Messaging that just works — RabbitMQ*. 2014. Disponível em: <https://www.rabbitmq.com/>. Acesso em: 31 jan. 2019.

JAIKAR, S. P.; IYER, K. R. A survey of messaging protocols for iot systems. p. 510–514, 02 2018.

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

JOHN, L. K.; EECKHOUT, L. *Performance evaluation and benchmarking*. [S.l.]: CRC Press, 2018.

JOSHI, M.; KAUR, B. P. Coap protocol for constrained networks. *IJWMT-International Journal of Wireless and Microwave Technologies (IJWMT)*, v. 5, n. 6, p. 1–10, 2015.

JURISTO, N.; MORENO, A. M. *Basics of software engineering experimentation*. [S.l.]: Springer Science & Business Media, 2013.

KAYAL, P.; PERROS, H. A comparison of iot application layer protocols through a smart parking implementation. In: IEEE. *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*. [S.l.], 2017. p. 331–336.

LEE, I.; LEE, K. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, Elsevier, v. 58, n. 4, p. 431–440, 2015.

LEE, S.; BAE, M.; KIM, H. Future of iot networks: A survey. *Applied Sciences*, Multidisciplinary Digital Publishing Institute, v. 7, n. 10, p. 1072, 2017.

LI, S.; XU, L. D.; ZHAO, S. The internet of things: a survey. *Information Systems Frontiers*, Springer, v. 17, n. 2, p. 243–259, 2015.

LIGHT, R. A. Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, The Open Journal, v. 2, n. 13, 2017.

LILJA, D. J. *Measuring computer performance: a practitioner's guide*. [S.l.]: Cambridge university press, 2005.

LUZURIAGA, J. E.; PEREZ, M.; BORONAT, P.; CANO, J. C.; CALAFATE, C.; MANZONI, P. A comparative evaluation of amqp and mqtt protocols over unstable and mobile networks. In: IEEE. *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*. [S.l.], 2015. p. 931–936.

MACIEL, P. R.; TRIVEDI, K. S.; MATIAS, R.; KIM, D. S. Dependability modeling. In: *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*. [S.l.]: IGI Global, 2012. p. 53–97.

MANSO, M.; JOHNSEN, F. T.; LUND, K.; CHAN, K. Using mqtt to support mobile tactical force situational awareness. In: IEEE. *2018 International Conference on Military Communications and Information Systems (ICMCIS)*. [S.l.], 2018. p. 1–6.

MARQUES, V. de F.; KNIESS, J. Uma avaliação de desempenho de protocolos de camada de aplicação para internet das coisas. *Anais do Computer on the Beach*, p. 120–129, 2018.

MARTINS, I. R.; ZEM, J. L. Estudo dos protocolos de comunicação mqtt e coap para aplicações machine-to-machine e internet das coisas. *Revista Tecnológica da Fatec Americana*, v. 3, n. 1, p. 24, 2016.

MATTERN, F.; FLOERKEMEIER, C. From the internet of computers to the internet of things. In: *From active data management to event-based systems and more*. [S.l.]: Springer, 2010. p. 242–259.

MAZZER, D.; FRIGIERI, E. P.; PARREIRA, L. F. C. G. Protocolos m2m para ambientes limitados no contexto do iot: Uma comparação de abordagens. 2018.

MIJOVIC, S.; SHEHU, E.; BURATTI, C. Comparing application layer protocols for the internet of things via experimentation. In: IEEE. *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2016 IEEE 2nd International Forum on*. [S.l.], 2016. p. 1–5.

MONGODB. *The most popular database for modern apps | MongoDB*. 2018. Disponível em: <https://www.mongodb.com/>. Acesso em: 30 abr. 2019.

MONTGOMERY, D. C. *Design and analysis of experiments*. [S.l.]: John wiley & sons, 2017.

MONTGOMERY, D. C.; RUNGER, G. C. *Applied statistics and probability for engineers*. [S.l.]: John Wiley & Sons, 2010.

MUKAKA, M. M. A guide to appropriate use of correlation coefficient in medical research. *Malawi Medical Journal*, Medical Association of Malawi, v. 24, n. 3, p. 69–71, 2012.

MUN, D.-H.; DINH, M. L.; KWON, Y.-W. An assessment of internet of things protocols for resource-constrained applications. In: IEEE. *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*. [S.l.], 2016. v. 1, p. 555–560.

MVERTES. *Docker Hub MongoDB 4.0 Alpine*. 2015. Disponível em: <https://hub.docker.com/r/mvertes/alpine-mongo>. Acesso em: 20 dez. 2018.

NAIK, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In: IEEE. *Systems Engineering Symposium (ISSE), 2017 IEEE International*. [S.l.], 2017. p. 1–7.

OLIVEIRA, R. N.; ROTH, V.; HENZEN, A. F.; SIMAO, J. M.; NOHAMA, P.; WILLE, E. C. G. Notification oriented paradigm applied to ambient assisted living tool. *IEEE Latin America Transactions*, IEEE, v. 16, n. 2, p. 647–653, 2018.

PONNUSAMY, K.; RAJAGOPALAN, N. Internet of things: A survey on iot protocol standards. In: *Progress in Advanced Computing and Intelligent Engineering*. [S.l.]: Springer, 2018. p. 651–663.

Python Community. *MQTT Python client library 1.4.0*. 2008. Disponível em: <https://pypi.org/project/paho-mqtt/>. Acesso em: 16 out. 2018.

Python Community. *Pika 0.9.14: Python Package Index*. 2014. Disponível em: <https://pypi.org/project/python3-pika/>. Acesso em: 16 out. 2018.

Raspberry Pi. *Raspberry Pi — Teach, Learn, and Make with Raspberry Pi*. 2019. Disponível em: <https://www.raspberrypi.org/>. Acesso em: 25 abr. 2019.

RATHORE, M. M.; AHMAD, A.; PAUL, A.; RHO, S. Urban planning and building smart cities based on the internet of things using big data analytics. *Computer Networks*, Elsevier, v. 101, p. 63–80, 2016.

ROMAN, R.; NAJERA, P.; LOPEZ, J. Securing the internet of things. *Computer*, IEEE, n. 9, p. 51–58, 2011.

SHELBY, Z.; HARTKE, K.; BORMANN, C. *The constrained application protocol (CoAP)*. [S.l.], 2014.

SINGH, D.; TRIPATHI, G.; JARA, A. J. A survey of internet-of-things: Future vision, architecture, challenges and services. In: IEEE. *2014 IEEE World Forum on Internet of Things (WF-IoT)*. [S.l.], 2014. p. 287–292.

STANDARD, O. Oasis advanced message queuing protocol (amqp) version 1.0. *International Journal of Aerospace Engineering Hindawi www. hindawi. com*, v. 2018, 2012.

STAR Wars: Os Últimos Jedi. [S.l.]: Estados Unidos: Walt Disney Studios Motion Pictures., 2017.

TANGANELLI, G.; VALLATI, C.; MINGOZZI, E. Coapthon: Easy development of coap-based iot applications with python. In: IEEE. *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. [S.l.], 2015. p. 63–68.

TANTITHARANUKUL, N.; OSATHANUNKUL, K.; HANTRAKUL, K.; PRAMOKCHON, P.; KHOENKAW, P. Mqtt-topic naming criteria of open data for smart cities. In: IEEE. *2016 International Computer Science and Engineering Conference (ICSEC)*. [S.l.], 2016. p. 1–6.

THANGAVEL, D.; MA, X.; VALERA, A.; TAN, H.-X.; TAN, C. K.-Y. Performance evaluation of mqtt and coap via a common middleware. In: IEEE. *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*. [S.l.], 2014. p. 1–6.

TOBIAS, P. A.; TRINDADE, D. *Applied reliability*. [S.l.]: Chapman and Hall/CRC, 2011.

TRIVEDI, K. S. *Probability and statistics with reliability, queuing, and computer science applications*. [S.l.]: Wiley Online Library, 1982. v. 13.

VINOSKI, S. Advanced message queuing protocol. *IEEE Internet Computing*, IEEE, n. 6, p. 87–89, 2006.

WORTMANN, F.; FLÜCHTER, K. Internet of things. *Business & Information Systems Engineering*, Springer, v. 57, n. 3, p. 221–224, 2015.

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

XU, Y.; MAHENDRAN, V.; RADHAKRISHNAN, S. Towards sdn-based fog computing: Mqtt broker virtualization for effective and reliable delivery. In: IEEE. *2016 8th International Conference on Communication Systems and Networks (COMSNETS)*. [S.l.], 2016. p. 1–6.

YASSEIN, M. B.; SHATNAWI, M. Q.; ALJWARNEH, S.; AL-HATMI, R. Internet of things: Survey and open issues of mqtt protocol. In: IEEE. *2017 International Conference on Engineering & MIS (ICEMIS)*. [S.l.], 2017. p. 1–6.

YASSEIN, M. B.; SHATNAWI, M. Q. et al. Application layer protocols for the internet of things: A survey. In: IEEE. *Engineering & MIS (ICEMIS), International Conference on*. [S.l.], 2016. p. 1–4.

YOKOTANI, T.; SASAKI, Y. Comparison with http and mqtt on required network resources for iot. In: IEEE. *Control, Electronics, Renewable Energy and Communications (ICCEREC), 2016 International Conference on*. [S.l.], 2016. p. 1–6.

YUAN, M. *Conhecendo o MQTT*. 2017. Disponível em: <https://www.ibm.com/developerworks/br/library/iot-mqtt-why-good-for-iot/index.html>. Acesso em: 31 jan. 2019.

APÊNDICE A – CÓDIGOS PYTHON DO RASPBERRY PI

Listing A.1 – Código em Python para o protocolo AMQP no Experimento I (Publicador).

```
1
# Codigo publisher.
3 # Codigo baseado no tutoriais do rabbitmq.
import pika
5 import time
import Adafruit_DHT
7 from gpiozero import MotionSensor
import RPi.GPIO as GPIO
9
# IP do servidor.
11 amqp_server = "10.0.0.106"

13 # Conexao com o servidor rabbit.
credentials = pika.PlainCredentials("sensor", "sensor")
15
# Tempo entre as mensagens.
17 timer_publisher = 5

19 # Topicos.
topic_pub = "sensor.#"
21 topic_temp = "sensor.temperature"
topic_hum = "sensor.humidity"
23 topic_pir = "sensor.motion"
topic_ldr = "sensor.light"
25
# Definindo o tipo de sensor que esta sendo utilizado.
27 sensor = Adafruit_DHT.DHT11

29 # Dados da temperatura e umidade.
data_dht11 = []
31
# GPIO o qual sensor esta conectado.
33 gpio_dht11 = 17
gpio_ldr = 18
35 gpio_pir = MotionSensor(4)

37 # Configuracao para capturar os dados do sensor.
GPIO.setmode(GPIO.BCM)
39 GPIO.setup(gpio_ldr, GPIO.IN)

41 ## Coletando dados dos sensores ##

43 # Metodo para coletar se o local tem luminosidade.
def collectLight(gpio_ldr):
45     result_ldr = GPIO.input(gpio_ldr)
    if result_ldr == 0:
47         print("Luminosidade")
    else:
49         print("Sem Luminosidade")
    return result_ldr
```

```
51
# Metodo para coletar se existe movimento.
53 def collectMotion(gpio_pir):
    result_motion = gpio_pir.motion_detected
55  print("Movimento Detectado: ", result_motion)
    return result_motion
57
# Metodo para coletar temperatura.
59 def collectTemperature(temperature):
    if temperature is None:
61      print("Falha ao ler o sensor")
        return -1
63  else:
        print("Temperatura (C): ", temperature)
65      return temperature

67 # Metodo para coletar umidade.
def collectHumidity(humidity):
69  if humidity is None:
        print("Falha ao ler o sensor")
71      return -1
    else:
73      print("Umidade (%): ", humidity)
        return humidity
75
# Metodo para coletar dos dados de temperatura e humidade diretamente do sensor (os
    dois dados sao coletados no mesmo instante).
77 def collectData(sensor_dht11, gpio_dht11):
    temperature, humidity = Adafruit_DHT.read_retry(sensor, gpio_dht11)
79  data_dht11 = ([collectTemperature(temperature), collectHumidity(humidity)])
    return data_dht11
81
# Metodo para publicar os dados dos sensores.
83 def publish(topic_key, topic_message):
    channel.basic_publish(exchange="topic_logs",
85                        routing_key=topic_key, body=topic_message)

87  print("Dado enviado %r:%r" % (topic_key, topic_message))

89 #
def main():
91  try:
        global channel
93      # Conexao com o servidor rabbit.
        connection = pika.BlockingConnection(pika.ConnectionParameters(
95          host=amqp_server, virtual_host="/", credentials=credentials,))
        channel = connection.channel()
97
        # Tipo de comunicacao que sera efetuada. Nesse caso sera uma comunicacao de
            topicos.
99      channel.exchange_declare(exchange="topic_logs", exchange_type="topic")

101     # Publicacao da mensagem.
        while(True):
103         # Coletando e enviando dados do sensor ldr (luminosidade).
            topic_message = str(collectLight(gpio_ldr))
105         publish(topic_ldr, topic_message)
```

```
107             # Coletando e enviando dados do sensor pir (movimento).
                topic_message = str(collectMotion(gpio_pir))
109             publish(topic_pir, topic_message)

111             # Coletando e enviando dados do sensor dht11 (temperatura e umidade).
                data_dht11 = collectData(sensor, gpio_dht11)
113             publish(topic_hum, str(data_dht11[0]))
                publish(topic_temp, str(data_dht11[1]))
115
                print("  ")
117             time.sleep(timer_publisher)

119     except KeyboardInterrupt:
            print("\nCtrl+C Encerrando a conexao.")
121         # Fechar a conexao.
            connection.close()
123
            # Limpar a porta GPIO utilizada.
125         GPIO.cleanup(17)
            GPIO.cleanup(18)
127         GPIO.cleanup(4)

129
#
131 if __name__ == "__main__":
        main()
```

Listing A.2 – Código em Python para o protocolo AMQP no Experimento I (Assinante).

```
1
# Codigo subscriber.
3 # Codigo baseado no tutoriais do rabbitmq.
import pika
5 import sys
from pymongo import MongoClient
7
client = MongoClient("localhost", 27017)
9 database = client.sensors

11 # IP do servidor.
amqp_server = "10.0.0.106"
13
# Porta de acesso ao servidor.
15 amqp_port = 5672

17 # Conexao com o servidor rabbit.
credentials = pika.PlainCredentials('sensor', 'sensor')
19 connection = pika.BlockingConnection(pika.ConnectionParameters(
    host=amqp_server, port=amqp_port, virtual_host='/', credentials=credentials,
        heartbeat=180))
21 # connection = pika.ConnectionParameters(host = amqp_server, amqp_port, credentials)
channel = connection.channel()
23
# Tipo de comunicacao que sera efetuada. Nesse caso sera uma comunicacao de topicos.
```

```
25 channel.exchange_declare(exchange='topic_logs', exchange_type='topic')

27 # Criacao da fila vazia e assim que a conexao se encerrar a fila sera excluida.
   result = channel.queue_declare(exclusive=True)
29 # Nome da fila que foi criaca aleatoriamente.
   queue_name = result.method.queue
31
   # Verifica se tem topico e qual o topico para se inscrever.
33 binding_keys = sys.argv[1:]
   if not binding_keys:
35     print('E necessario que tenha um topico para se inscrever.')
       sys.exit(1)
37
   # Identificacao de qual fila sera inscrita.
39 for binding_key in binding_keys:
       channel.queue_bind(exchange='topic_logs',
41                        queue=queue_name, routing_key=binding_key)

43 # Aguarda o recebimento dos dados enviados pelo publish.
   def callback(ch, method, properties, body):
45     # .decode() converte o body para utf-8.

47    # insercao no banco de dados
       sensor = method.routing_key.replace("sensor.", "")
49     if sensor != "#":
           insert_data({sensor: body.decode(), "protocol": "AMQP"}, sensor)
51
       print('Topico: %r - dado: %r' % (method.routing_key, body.decode()))
53
   # Metodo de insercao do dado no MongoDB.
55 def insert_data(data, sensor):
       collection = database[sensor]
57     collection.insert_one(data).inserted_id

59
   channel.basic_consume(callback, queue=queue_name, no_ack=True)
61
   # Sempre ira consumir o conteudo publicado (loop).
63 channel.start_consuming()
```

Listing A.3 – Código em Python para o protocolo AMQP no Experimento II (Publicador).

```
2  # Codigo publisher.
   # Codigo baseado no tutoriais do rabbitmq.
4  import pika
   import asyncio
6  import time
   import Adafruit_DHT
8  from gpiozero import MotionSensor
   import RPi.GPIO as GPIO
10 import os

12 #global channel
```

```
14 # IP do servidor.
   amqp_primary = "10.0.0.106"
16 amqp_secundary = "192.168.0.103"
   amqp_server = amqp_primary
18
   # Porta de conexao.
20 amqp_port = 5672

22 # Conexao com o servidor rabbit.
   credentials = pika.PlainCredentials("sensor", "sensor")
24
   # Tempo entre as mensagens.
26 timer_publisher = 5

28 # Topicos.
   topic_pub = "sensor.#"
30 topic_temp = "sensor.temperature"
   topic_hum = "sensor.humidity"
32 topic_pir = "sensor.motion"
   topic_ldr = "sensor.light"
34
   # Definindo o tipo de sensor que esta sendo utilizado.
36 sensor = Adafruit_DHT.DHT11

38 # Dados da temperatura e umidade.
   data_dht11 = []
40
   # GPIO o qual sensor esta conectado.
42 gpio_dht11 = 17
   gpio_ldr = 18
44 gpio_pir = MotionSensor(4)

46 # Configuracao para capturar os dados do sensor.
   GPIO.setmode(GPIO.BCM)
48 GPIO.setup(gpio_ldr, GPIO.IN)

50 ## Coletando dados dos sensores ##

52 # Metodo para coletar se o local tem luminosidade.
   def collectLight(gpio_ldr):
54     result_ldr = GPIO.input(gpio_ldr)
       if result_ldr == 0:
56         print("Luminosidade")
       else:
58         print("Sem Luminosidade")
       return result_ldr
60
   # Metodo para coletar se existe movimento.
62 def collectMotion(gpio_pir):
       result_motion = gpio_pir.motion_detected
64     print("Movimento Detectado: ", result_motion)
       return result_motion
66
   # Metodo para coletar temperatura.
68 def collectTemperature(temperature):
       if temperature is None:
70         print("Falha ao ler o sensor")
```

```
            return -1
72      else:
            print("Temperatura (C): ", temperature)
74          return temperature

76  # Metodo para coletar umidade.
    def collectHumidity(humidity):
78      if humidity is None:
            print("Falha ao ler o sensor")
80          return -1
        else:
82          print("Umidade (%): ", humidity)
            return humidity
84
    # Metodo para coletar dos dados de temperatura e humidade diretamente do sensor (os
        dois dados sao coletados no mesmo instante).
86  def collectData(sensor_dht11, gpio_dht11):
        humidity, temperature = Adafruit_DHT.read_retry(sensor, gpio_dht11)
88      data_dht11 = ([collectTemperature(temperature), collectHumidity(humidity)])
        return data_dht11
90
    # Metodo para publicar os dados dos sensores.
92  def publish(topic_key, topic_message):
        channel.basic_publish(exchange="topic_logs",
94                            routing_key=topic_key, body=topic_message)

96      print("Dado enviado %r:%r" % (topic_key, topic_message))

98  # Verifica se tem conexao com o servidor.
    def verify_connection(ip):
100     # c e a quantidade de pacotes enviados para o ping
        # W e a o tempo de espera.
102     # > /dev/null direciona a saida do ping para null.
        rep = os.system("ping -c 1 -W 1 %s > /dev/null" % ip)
104     if rep == 0:
            print("Okay")
106         return True
        else:
108         print("Erro")
            return False
110
    #
112 def main():
        try:
114         global amqp_server
            global channel
116
            connection = pika.BlockingConnection(pika.ConnectionParameters(
118             host=amqp_server, port=amqp_port, virtual_host="/", credentials=
                    credentials, heartbeat=30, connection_attempts=3))
            channel = connection.channel()
120
            # Tipo de comunicacao que sera efetuada. Nesse caso sera uma comunicacao de
                topicos.
122         channel.exchange_declare(exchange="topic_logs", exchange_type="topic")

124         # Iniciando a coleta de dados.
```

```
        publish(topic_pub, "Sensor Conectado")
126
        # Publicacao da mensagem.
128     while(True):
            # Vefifica a conexao.
130         if not verify_connection(amqp_server):
                if amqp_server == amqp_primary:
132                 amqp_server = amqp_secundary
                else:
134                 amqp_server = amqp_primary
                connection = pika.BlockingConnection(pika.ConnectionParameters(
136                 host=amqp_server, port=amqp_port, virtual_host="/", credentials=
                        credentials, heartbeat=30, connection_attempts=3))
                channel = connection.channel()
138             # Tipo de comunicacao que sera efetuada. Nesse caso sera uma
                    comunicacao de topicos.
                channel.exchange_declare(
140                 exchange="topic_logs", exchange_type="topic")
                # Iniciando a coleta de dados.
142             publish(topic_pub, "Sensor Conectado")

144         print(amqp_server)
            print(" _________ \n")
146
            # Coletando e enviando dados do sensor ldr (luminosidade).
148         topic_message = str(collectLight(gpio_ldr))
            publish(topic_ldr, topic_message)
150
            # Coletando e enviando dados do sensor pir (movimento).
152         topic_message = str(collectMotion(gpio_pir))
            publish(topic_pir, topic_message)
154
            # Coletando e enviando dados do sensor dht11 (temperatura e umidade).
156         data_dht11 = collectData(sensor, gpio_dht11)
            publish(topic_hum, str(data_dht11[1]))
158         publish(topic_temp, str(data_dht11[0]))

160         print("  ")

162         time.sleep(timer_publisher)

164 except KeyboardInterrupt:
        print("\nCtrl+C Encerrando a conexao.")
166     # Fechar a conexao.
        connection.close()
168
        # Limpar a porta GPIO utilizada.
170     GPIO.cleanup(17)
        GPIO.cleanup(18)
172     GPIO.cleanup(4)

174
#
176 if __name__ == "__main__":
    main()
```

Listing A.4 – Código em Python para o protocolo AMQP no Experimento II (Assinante).

```
# Codigo subscriber.

# Codigo baseado no tutoriais do rabbitmq.
import pika
import sys

from pymongo import MongoClient

client = MongoClient("localhost", 27017)
database = client.sensors

# IP do servidor.
amqp_server = "192.168.0.103"

# Porta de acesso ao servidor.
amqp_port = 5672

# Conexao com o servidor rabbit.
credentials = pika.PlainCredentials("sensor", "sensor")
connection = pika.BlockingConnection(pika.ConnectionParameters(
    host=amqp_server, port=amqp_port, virtual_host="/", credentials=credentials,
        heartbeat=180))
# connection = pika.ConnectionParameters(host = amqp_server, amqp_port, credentials)
channel = connection.channel()

# Tipo de comunicacao que sera efetuada. Nesse caso sera uma comunicacao de topicos.
channel.exchange_declare(exchange="topic_logs", exchange_type="topic")

# Criacao da fila vazia e assim que a conexao se encerrar a fila sera excluida.
result = channel.queue_declare(exclusive=True)
# Nome da fila que foi criada aleatoriamente.
queue_name = result.method.queue

# Verifica se tem topico e qual o topico para se inscrever.
binding_keys = sys.argv[1:]
if not binding_keys:
    print("E necessario que tenha um topico para se inscrever.")
    sys.exit(1)

# Identificacao de qual fila sera inscrita.
for binding_key in binding_keys:
    channel.queue_bind(exchange="topic_logs",
                       queue=queue_name, routing_key=binding_key)

# Aguarda o recebimento dos dados enviados pelo publish.
def callback(ch, method, properties, body):
    # .decode() converte o body para utf-8.

    # insercao no banco de dados
    sensor = method.routing_key.replace("sensor.", "")
    if sensor != "#":
        insert_data({sensor: body.decode(), "protocol": "AMQP"}, sensor)

    print("Topico: %r / dado: %r" % (method.routing_key, body.decode()))
```

```
# Metodo de insercao do dado no MongoDB.
def insert_data(data, sensor):
    collection = database[sensor]
    collection.insert_one(data).inserted_id


channel.basic_consume(callback, queue=queue_name, no_ack=True)

# Sempre ira consumir o conteudo publicado (loop).
channel.start_consuming()

#####

# Codigo baseado no tutoriais do rabbitmq.
import pika
import sys
from pymongo import MongoClient

client = MongoClient("localhost", 27017)
database = client.sensors

# IP do servidor.
amqp_server = "10.0.0.106"

# Porta de acesso ao servidor.
amqp_port = 5672

# Conexao com o servidor rabbit.
credentials = pika.PlainCredentials('sensor', 'sensor')
connection = pika.BlockingConnection(pika.ConnectionParameters(
    host=amqp_server, port=amqp_port, virtual_host='/', credentials=credentials,
        heartbeat=180))
# connection = pika.ConnectionParameters(host = amqp_server, amqp_port, credentials)
channel = connection.channel()

# Tipo de comunicacao que sera efetuada. Nesse caso sera uma comunicacao de topicos.
channel.exchange_declare(exchange='topic_logs', exchange_type='topic')

# Criacao da fila vazia e assim que a conexao se encerrar a fila sera excluida.
result = channel.queue_declare(exclusive=True)
# Nome da fila que foi criaca aleatoriamente.
queue_name = result.method.queue

# Verifica se tem topico e qual o topico para se inscrever.
binding_keys = sys.argv[1:]
if not binding_keys:
    print('E necessario que tenha um topico para se inscrever.')
    sys.exit(1)

# Identificacao de qual fila sera inscrita.
for binding_key in binding_keys:
    channel.queue_bind(exchange='topic_logs',
                       queue=queue_name, routing_key=binding_key)

# Aguarda o recebimento dos dados enviados pelo publish.
def callback(ch, method, properties, body):
```

```
    # .decode() converte o body para utf-8.

   # insercao no banco de dados
    sensor = method.routing_key.replace("sensor.", "")
    if sensor != "#":
        insert_data({sensor: body.decode(), "protocol": "AMQP"}, sensor)

    print('Topico: %r - dado: %r' % (method.routing_key, body.decode()))

# Metodo de insercao do dado no MongoDB.
def insert_data(data, sensor):
    collection = database[sensor]
    collection.insert_one(data).inserted_id


channel.basic_consume(callback, queue=queue_name, no_ack=True)

# Sempre ira consumir o conteudo publicado (loop).
channel.start_consuming()
```

Listing A.5 – Código em Python para o protocolo CoAP nos Experimentos I e II (Servidor).

```

# Servidor
# Codigo CoAP. Este codigo foi retirado da biblioteca CoAPthon e modificado.
from coapthon.server.coap import CoAP
from resourcescoap import Separate
from resourcescoapbasic import BasicResource

class CoAPServer(CoAP):
    def __init__(self, host, port):
        CoAP.__init__(self, (host, port))
        # Separate e o tipo de resposta.
        # sensors/ e o path.
        self.add_resource("sensors/", BasicResource())
        self.add_resource("temperature", BasicResource())
        self.add_resource("humidity", BasicResource())
        self.add_resource("motion", BasicResource())
        self.add_resource("light", BasicResource())
        #self.add_resource("sensors/", Separate())

def main():
    server = CoAPServer("0.0.0.0", 5683)
    try:
        server.listen(120)
    except KeyboardInterrupt:
        print ("Desligando...")
        server.close()

if __name__ == "__main__":
    main()

%%%%%%%%%
```

Listing A.6 – Código em linguagem de programação Python para o protocolo CoAP no experimento 1 (Cliente).

```
1
# Codigo CoAP. Este codigo foi retirado da biblioteca CoAPthon e modificado.
3 from coapthon.client.helperclient import HelperClient
from gpiozero import MotionSensor
5 import Adafruit_DHT
import RPi.GPIO as GPIO
7 import time
import sys
9
# IP do servidor.
11 host = "192.168.0.100"

13 # Porta de conexao.
port = 5683
15
# Path para conexao.
17 path = "sensors"
path_temperature = "temperature"
19 path_humidity = "humidity"
path_motion = "motion"
21 path_light = "light"

23 # Conteudo a ser enviado assim que o sensor e conectado ao servidor.
payload = "Sensor conectado"
25
# Definindo o tipo de sensor que esta sendo utilizado.
27 sensor = Adafruit_DHT.DHT11

29 # Tempo entre cada coleta.
timer_publisher = 5
31
# Dados da temperatura e umidade.
33 data_dht11 = []

35 # GPIO o qual sensor esta conectado.
gpio_dht11 = 17
37 gpio_ldr = 18
gpio_pir = MotionSensor(4)
39
# Configuracao para capturar os dados do sensor.
41 GPIO.setmode(GPIO.BCM)
GPIO.setup(gpio_ldr, GPIO.IN)
43
## Coletando dados dos sensores ##
45
# Metodo para coletar se o local esta ou com com luz.
47 def collectLight(gpio_ldr):
    result_ldr = GPIO.input(gpio_ldr)
49     if result_ldr == 0:
        print("Luminosidade")
51     else:
        print("Sem Luminosidade")
53     return result_ldr
```

```
55 # Metodo para coletar se existe movimento.
   def collectMotion(gpio_pir):
57     result_motion = gpio_pir.motion_detected
       print("Movimento Detectado: ", result_motion)
59     return result_motion

61 # Metodo para coletar temperatura.
   def collectTemperature(temperature):
63     if temperature is None:
           print("Falha ao ler o sensor")
65         return -1
       else:
67         print("Temperatura (C): ", temperature)
           return temperature
69
   # Metodo para coletar umidade.
71 def collectHumidity(humidity):
       if humidity is None:
73         print("Falha ao ler o sensor")
           return -1
75     else:
           print("Umidade (%): ", humidity)
77         return humidity

79 # Metodo para coletar dos dados de temperatura e humidade diretamente do sensor (os
       dois dados sao coletados no mesmo instante).
   def collectDataTH(sensor_dht11, gpio_dht11):
81     humidity, temperature = Adafruit_DHT.read_retry(sensor, gpio_dht11)
       data_dht11 = ([collectTemperature(temperature), collectHumidity(humidity)])
83     return data_dht11

85 #


87
   def main():
89     try:
           # Conexao com servidor.
91         client = HelperClient(server=(host, port))

93         while True:
               # Coletando e enviando dados do sensor ldr (luminosidade).
95             #client.post(ppath_light, str(collectLight(gpio_ldr)))
               response = client.post(path_light, str(collectLight(gpio_ldr)))
97
               print(response.pretty_print())
99
               # Coletando e enviando dados do sensor pir (movimento).
101            #client.post(path_motion, str(collectMotion(gpio_pir)))
               response = client.post(path_motion, str(collectMotion(gpio_pir)))
103            print(response.pretty_print())

105            # Coletando e enviando dados do sensor dht11 (temperatura e umidade).
               data_dht11 = collectDataTH(sensor, gpio_dht11)
107            #client.post(path_temperature, str(data_dht11[0]))
               response = client.post(path_temperature, str(data_dht11[0]))
109            print(response.pretty_print())
```

```
111             #client.post(path_humidity, str(data_dht11[1]))
                response = client.post(path_humidity, str(data_dht11[1]))
113             print(response.pretty_print())
                print("  ")
115
                time.sleep(timer_publisher)
117
        except KeyboardInterrupt:
119         print("\nCtrl+C Encerrando a conexao.")
            client.stop()
121         # Limpar a porta GPIO utilizada.
            GPIO.cleanup(17)
123         GPIO.cleanup(18)
            GPIO.cleanup(4)
125
            sys.exit(0)
127

129 #
    if __name__ == "__main__":
131     main()
```

Listing A.7 – Código em Python para o protocolo CoAP no Experimento II (Cliente).

```
1
  # Codigo CoAP. Este codigo foi retirado da biblioteca CoAPthon e modificado.
3 from coapthon.client.helperclient import HelperClient
  from gpiozero import MotionSensor
5 import Adafruit_DHT
  import RPi.GPIO as GPIO
7 import time
  import os
9
  # IP do servidor.
11 host_primary = "10.0.0.106"
  host_secundary = "192.168.0.100"
13 host = host_primary

15 # Porta de conexao.
  port = 5683
17
  # Path para conexao.
19 path = "sensors"
  path_temperature = "temperature"
21 path_humidity = "humidity"
  path_motion = "motion"
23 path_light = "light"

25 # Conteudo a ser enviado assim que o sensor e conectado ao servidor.
  payload = "Sensor conectado"
27
  # Definindo o tipo de sensor que esta sendo utilizado.
29 sensor = Adafruit_DHT.DHT11

31 # Tempo entre cada coleta.
```

```
timer_publisher = 5

# Dados da temperatura e umidade.
data_dht11 = []

# GPIO o qual sensor esta conectado.
gpio_dht11 = 17
gpio_ldr = 18
gpio_pir = MotionSensor(4)

# Configuracao para capturar os dados do sensor.
GPIO.setmode(GPIO.BCM)
GPIO.setup(gpio_ldr, GPIO.IN)

## Coletando dados dos sensores ##

# Metodo para coletar se o local esta ou com com luz.
def collectLight(gpio_ldr):
    result_ldr = GPIO.input(gpio_ldr)
    return result_ldr

# Metodo para coletar se existe movimento.
def collectMotion(gpio_pir):
    result_motion = gpio_pir.motion_detected
    return result_motion

# Metodo para coletar temperatura.
def collectTemperature(temperature):
    if temperature is None:
        return -1
    else:
        return temperature

# Metodo para coletar umidade.
def collectHumidity(humidity):
    if humidity is None:
        return -1
    else:
        return humidity

# Metodo para coletar dos dados de temperatura e humidade diretamente do sensor (os
    dois dados sao coletados no mesmo instante).
def collectDataTH(sensor_dht11, gpio_dht11):
    humidity, temperature = Adafruit_DHT.read_retry(sensor, gpio_dht11)
    data_dht11 = ([collectTemperature(temperature), collectHumidity(humidity)])
    return data_dht11


def verify_connection(ip):
    # c e a quantidade de pacotes enviados para o ping
    # W e a o tempo de espera.
    # > /dev/null direciona a saida do ping para null.
    rep = os.system("ping -c 1 -W 1 %s > /dev/null" % ip)
    if rep == 0:
        print("Okay")
        return True
    else:
```

```
        print("Erro")
89      return False
#
91

93  def main():
        global host
95      try:
            # Conexao com servidor.
97          client = HelperClient(server=(host, port))

99          while True:
                if not verify_connection(host):
101                 client.stop()
                    if host == host_primary:
103                     host = host_secundary
                    else:
105                     host = host_primary

107                 client = HelperClient(server=(host, port))

109             print(host)
                print(" ---------------------------------- ")
111
                # Coletando e enviando dados do sensor ldr.
113             #client.post(path_light, str(collectLight(gpio_ldr)))
                response = client.post(path_light, str(collectLight(gpio_ldr)))
115             print(response.pretty_print())

117             # Coletando e enviando dados do sensor pir.
                #client.post(path_motion, str(collectMotion(gpio_pir)))
119             response = client.post(path_motion, str(collectMotion(gpio_pir)))
                print(response.pretty_print())
121
                # Coletando e enviando dados do sensor dht11 (temperatura e umidade).
123             data_dht11 = collectDataTH(sensor, gpio_dht11)
                #client.post(path_temperature, str(data_dht11[0]))
125             response = client.post(path_temperature, str(data_dht11[0]))
                print(response.pretty_print())
127
                #client.post(path_humidity, str(data_dht11[1]))
129             response = client.post(path_humidity, str(data_dht11[1]))
                print(response.pretty_print())
131             print("  ")

133             time.sleep(timer_publisher)

135     except KeyboardInterrupt:
            print("\nCtrl+C Encerrando a conexao.")
137         client.stop()
            # Limpar a porta GPIO utilizada.
139         GPIO.cleanup(17)
            GPIO.cleanup(18)
141         GPIO.cleanup(4)

143
#
```

```
145 if __name__ == "__main__":
        main()
```

Listing A.8 – Código em Python para o protocolo MQTT no Experimento I (Publicador).

```
2  import paho.mqtt.client as mqtt
   import time
4  import sys
   import Adafruit_DHT
6  from gpiozero import MotionSensor
   import RPi.GPIO as GPIO
8
   # IP para o servidor.
10 mqtt_server = "192.168.0.100"

12 # Porta de conexao.
   mqtt_port = 1883
14
   # Tempo maximo de conexao.
16 mqtt_time_alive = 120

18 # Tempo entre cada publicacao.
   timer_publisher = 5
20
   # Definindo topicos.
22 # Definiu-se um topico para subscribe que engloba o topico sensor.
   mqtt_publisher = "sensor"
24 # Topicos de publishers.
   mqtt_subscribe = "sensor"
26 mqtt_dht11t = "sensor/temperatura"
   mqtt_dht11h = "sensor/umidade"
28 mqtt_ldr = "sensor/luminosidade"
   mqtt_pir = "sensor/movimento"
30
   # Definindo QoS.
32 #qos = 0
   qos = 1
34
   # Definindo o tipo de sensor que esta sendo utilizado.
36 sensor_dht11 = Adafruit_DHT.DHT11

38 # Dados da temperatura e umidade.
   data_dht11 = []
40
   # GPIO o qual sensor esta conectado.
42 gpio_dht11 = 17
   gpio_ldr = 18
44 gpio_pir = MotionSensor(4)

46 # Configuracao para capturar os dados do sensor.
   GPIO.setmode(GPIO.BCM)
48 GPIO.setup(gpio_ldr, GPIO.IN)

50 # Configuracao para capturar os dados do sensor.
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(gpio_ldr, GPIO.IN)

## Coletando dados dos sensores ##

# Metodo para coletar se o local esta ou com com luz.
def collectLight(gpio_ldr):
    result_ldr = GPIO.input(gpio_ldr)
    if result_ldr == 0:
        print("Luminosidade")
    else:
        print("Sem Luminosidade")
    return result_ldr

# Metodo para coletar se existe movimento.
def collectMotion(gpio_pir):
    result_motion = gpio_pir.motion_detected
    print("Movimento Detectado: ", result_motion)
    return result_motion

# Metodo para coletar temperatura.
def collectTemperature(temperature):
    if temperature is None:
        print("Falha ao ler o sensor")
        return -1
    else:
        print("Temperatura (C): ", temperature)
        return temperature

# Metodo para coletar umidade.
def collectHumidity(humidity):
    if humidity is None:
        print("Falha ao ler o sensor")
        return -1
    else:
        print("Umidade (%): ", humidity)
        return humidity

# Metodo para coletar dos dados de temperatura e humidade diretamente do sensor (os
    dois dados sao coletados no mesmo instante).
def collectData(sensor_dht11, gpio_dht11):
    humidity, temperature = Adafruit_DHT.read_retry(sensor_dht11, gpio_dht11)
    data_dht11 = ([collectTemperature(temperature), collectHumidity(humidity)])
    return data_dht11

## Conectando ao docker mosquitto. ##

# Chamado quando o corretor responde ao nosso pedido de conexao.
# Obs: metodo da biblioteca paho (github).
def on_connect(client, userdata, flags, rc):
    print("Conectado ao Broker, codigo: "+str(rc))
    # Inscricao no subscribe desejado. Neste caso sensor.
    client.subscribe(mqtt_subscribe)

# Mensagens recebidas do broken.
# Obs: metodo da biblioteca paho (github).
def on_message(client, userdata, msg):
```

```
    # Conteudo da mensagem.
108     receive_message = str(msg.payload)
    print("Mensagem Recebida. Topico: "+msg.topic+" Mensagem: "+receive_message)
110

112 #
def main():
114     try:
        print("Inicializando MQTT...")
116         # Inicializa o servico MQTT.
        client = mqtt.Client()
118         client.on_connect = on_connect
        client.on_message = on_message
120         #lient.on_publish = on_publish
        client.connect(mqtt_server, mqtt_port, mqtt_time_alive)
122         client.publish(mqtt_publisher, "Sensores ativos")

124         while(True):
            # Coletando e enviando dados do sensor ldr.
126             client.publish(mqtt_ldr, collectLight(gpio_ldr), qos=qos)

128             # Coletando e enviando dados do sensor pir.
            client.publish(mqtt_pir, collectMotion(gpio_pir), qos=qos)
130

            # Coletando e enviando dados do sensor dht11.
132             data_dht11 = collectData(sensor_dht11, gpio_dht11)
            client.publish(mqtt_dht11t, data_dht11[0], qos=qos)
134             client.publish(mqtt_dht11h, data_dht11[1], qos=qos)

136             time.sleep(timer_publisher)
            print("  ")
138             client.loop(2, 10)

140         # client.loop_forever()
    except KeyboardInterrupt:
142         print("\nCtrl+C encerrando a conexao")
        # Desconectar do mosquitto.
144         client.disconnect()

146         # Limpar a porta GPIO utilizada.
        GPIO.cleanup(17)
148         GPIO.cleanup(18)
        GPIO.cleanup(4)
150

        sys.exit(0)
152


154 #
if __name__ == "__main__":
156     main()
```

Listing A.9 – Código em Python para o protocolo MQTT no Experimento I (Assinante).

```
2 # Codigo baseado na bilioteca paho.
```

```
import paho.mqtt.client as mqtt
from pymongo import MongoClient

# Para conexao com o BD.
client = MongoClient("localhost", 27017)
database = client.sensors

# IP para o servidor.
mqtt_server = "10.0.0.106"

# Porta de conexao.
mqtt_port = 1883

# Tempo maximo de conexao.
mqtt_time_alive = 60

# Definindo topico.
mqtt_subscribe = "sensor/#"

# Mensagens recebidas do broken
# obs: metodo da biblioteca paho (github).
def on_message(client, userdata, msg):
    # Conteudo da mensagem.
    receive_message = str(msg.payload.decode("utf-8"))
    print("Mensagem Recebida. Topico %s Mensagem: %s" %
          (msg.topic, receive_message))

    # Insercao dos dados no MongoDB.
    mqtt_topic = msg.topic.replace("sensor/", "")
    insert_data({mqtt_topic: receive_message, "protocol": "MQTT"}, mqtt_topic)

# Insercao no MongoDB.
def insert_data(data, sensor):
    collection = database[sensor]
    collection.insert_one(data).inserted_id

#
def main():
    print("Inicializando...")
    client = mqtt.Client()
    # Conexao do cliente ao servidor.
    con = client.connect(mqtt_server, mqtt_port, mqtt_time_alive)

    # Verificando se ocorreu conexao.
    if con == 0:
        print("Conectado!")

    client.subscribe(mqtt_subscribe)
    client.on_message = on_message

    # Esta funcao faz com que a conexao so encerre caso o cliente se desconecte por
        meio da funcao especifica.
    client.loop_forever()


#
if __name__ == "__main__":
```

```
    main()
```

Listing A.10 – Código em Python para o protocolo MQTT no Experimento II (Publicador).

```
1
import paho.mqtt.client as mqtt
3 import time
import Adafruit_DHT
5 from gpiozero import MotionSensor
import RPi.GPIO as GPIO
7 import os

9 # IP do servidor.
mqtt_primary = "10.0.0.106"
11 mqtt_secundary = "192.168.0.103"
mqtt_server = mqtt_primary
13
# Porta de conexao.
15 mqtt_port = 1883

17 # Tempo maximo de conexao.
mqtt_time_alive = 120
19
# Tempo entre cada publicacao.
21 timer_publisher = 5

23 # Tempo de reconexao
timer_reconnection = 3
25
# Definindo topicos.
27 # Definiu-se um topico para subscribe que engloba o topico sensor.
mqtt_subscribe = "sensor"
29 # Topicos de publishers.
mqtt_publisher = "sensor"
31 mqtt_dht11t = "sensor/temperature"
mqtt_dht11h = "sensor/humidity"
33 mqtt_ldr = "sensor/light"
mqtt_pir = "sensor/motion"
35
# Definindo QoS.
37 #qos = 0
qos = 1
39
# Definindo o tipo de sensor que esta sendo utilizado.
41 sensor_dht11 = Adafruit_DHT.DHT11

43 # Dados da temperatura e umidade.
data_dht11 = []
45
# GPIO o qual sensor esta conectado.
47 gpio_dht11 = 17
gpio_ldr = 18
49 gpio_pir = MotionSensor(4)
```

```
51 # Configuracao para capturar os dados do sensor.
   GPIO.setmode(GPIO.BCM)
53 GPIO.setup(gpio_ldr, GPIO.IN)

55 # Configuracao para capturar os dados do sensor.
   GPIO.setmode(GPIO.BCM)
57 GPIO.setup(gpio_ldr, GPIO.IN)

59 ## Coletando dados dos sensores ##

61 # Metodo para coletar se o local esta ou com com luz.
   def collectLight(gpio_ldr):
63     result_ldr = GPIO.input(gpio_ldr)
       if result_ldr == 0:
65         print("Luminosidade")
       else:
67         print("Sem Luminosidade")
       return result_ldr
69
   # Metodo para coletar se existe movimento.
71 def collectMotion(gpio_pir):
       result_motion = gpio_pir.motion_detected
73     print("Movimento Detectado: ", result_motion)
       return result_motion
75
   # Metodo para coletar temperatura.
77 def collectTemperature(temperature):
       if temperature is None:
79         print("Falha ao ler o sensor")
           return -1
81     else:
           print("Temperatura (C): ", temperature)
83         return temperature

85 # Metodo para coletar umidade.
   def collectHumidity(humidity):
87     if humidity is None:
           print("Falha ao ler o sensor")
89         return -1
       else:
91         print("Umidade (%): ", humidity)
           return humidity
93
   # Metodo para coletar dos dados de temperatura e humidade diretamente do sensor (os
       dois dados sao coletados no mesmo instante).
95 def collectData(sensor_dht11, gpio_dht11):
       humidity, temperature = Adafruit_DHT.read_retry(sensor_dht11, gpio_dht11)
97     data_dht11 = ([collectTemperature(temperature), collectHumidity(humidity)])
       return data_dht11
99
   # Verifica se tem conexao com o servidor.
101 def verify_connection(ip):
       # c e a quantidade de pacotes enviados para o ping
103    # W e a o tempo de espera.
       # > /dev/null direciona a saida do ping para null.
105    rep = os.system("ping -c 1 -W 1 %s > /dev/null" % ip)
       if rep == 0:
```

```
107         print("Okay")
            return True
109     else:
            print("Erro")
111         return False

113 ## Conectando ao docker mosquitto. ##
    # Mensagens recebidas do broken.
115 # Obs: metodo da biblioteca paho (github).
    def on_message(client, userdata, msg):
117     # Conteudo da mensagem.
        receive_message = str(msg.payload)
119     #print("Mensagem Recebida. Topico: "+msg.topic+" Mensagem: "+receive_message)
        print("Mensagem Recebida. Topico %s Mensagem: %s" %
121           (msg.topic, receive_message))

123 # Metodo para conexao
    def connection():
125     client = mqtt.Client()
        client.on_message = on_message
127     #client.connect(mqtt_server, mqtt_port, mqtt_time_alive)
        con = client.connect(mqtt_server, mqtt_port, mqtt_time_alive)
129     if con == 0:
            print("Conectado!")
131
        client.publish(mqtt_publisher, "Sensores ativos", qos=qos)
133     return client

135 # Metodo para reconexao no caso de falha da conexao.
    def reconnection(rec=False):
137     while not rec:
            try:
139             client = mqtt.Client()
                #client.on_connect = on_connect
141             client.on_message = on_message
                #client.connect(mqtt_server, mqtt_port, mqtt_time_alive)
143             con = client.connect(mqtt_server, mqtt_port, mqtt_time_alive)
                if con == 0:
145                 print("Conectado!")

147             client.publish(mqtt_publisher, "Sensores ativos", qos=qos)
                rec = True
149         except Exception as ex:
                print("Reconectando...")
151             time.sleep(timer_reconnection)
        return client
153 #
    def main():
155     global mqtt_server
        try:
157         print("Inicializando MQTT...")
            # Inicializa o servico MQTT.
159         client = connection()

161         while(True):
                # Vefifica a conexao.
163             if not verify_connection(mqtt_server):
```

```
                client.disconnect()
165             if mqtt_server == mqtt_primary:
                    mqtt_server = mqtt_secundary
167             else:
                    mqtt_server = mqtt_primary
169             client = reconnection()
            # Mostra ao usuario quao IP esta conectado.
171         print(mqtt_server)
            print(" _________ \n")
173
            # Coletando e enviando dados do sensor ldr.
175         client.publish(mqtt_ldr, collectLight(gpio_ldr), qos=qos)

177         # Coletando e enviando dados do sensor pir.
            client.publish(mqtt_pir, collectMotion(gpio_pir), qos=qos)
179
            # Coletando e enviando dados do sensor dht11.
181         data_dht11 = collectData(sensor_dht11, gpio_dht11)
            # Temperatura.
183         client.publish(mqtt_dht11t, data_dht11[0], qos=qos)
            # Umidade.
185         client.publish(mqtt_dht11h, data_dht11[1], qos=qos)

187         time.sleep(timer_publisher)
            print("  ")
189         client.loop(2, 10)

191     client.loop_forever()
    except KeyboardInterrupt:
193     print("\nCtrl+C encerrando a conexao")
        # Desconectar do mosquitto.
195     client.disconnect()

197     # Limpar a porta GPIO utilizada.
        GPIO.cleanup(17)
199     GPIO.cleanup(18)
        GPIO.cleanup(4)
201


203 #
if __name__ == "__main__":
205     main()
```

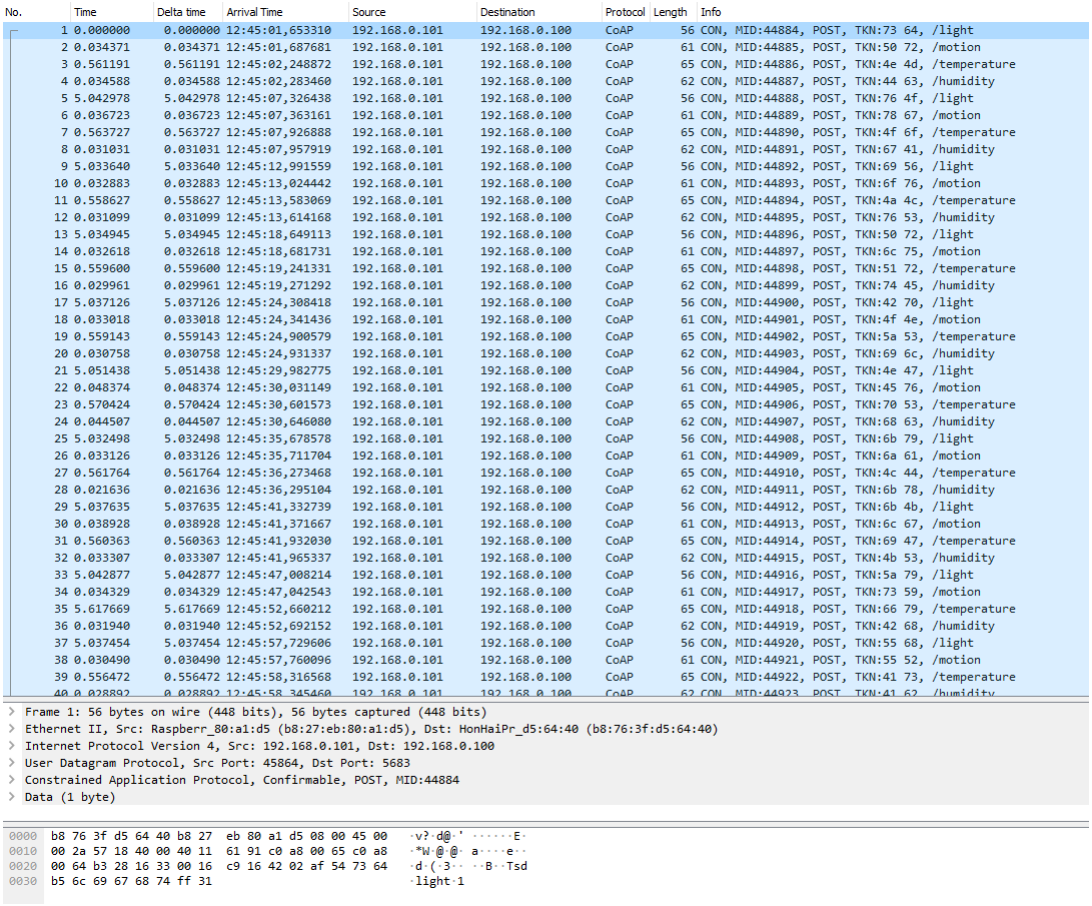
Listing A.11 – Código em Python para o protocolo MQTT no Experimento II (Assinante).

```
2 # Codigo baseado na bilioteca paho.
  import paho.mqtt.client as mqtt
4 from pymongo import MongoClient

6 # Para conexao com o BD.
  client = MongoClient("localhost", 27017)
8 database = client.sensors

10 # IP para o servidor.
```

```
mqtt_server = "10.0.0.106"

# Porta de conexao.
mqtt_port = 1883

# Tempo maximo de conexao.
mqtt_time_alive = 60

# Definindo topico.
mqtt_subscribe = "sensor/#"

# Mensagens recebidas do broken
# obs: metodo da biblioteca paho (github).
def on_message(client, userdata, msg):
    # Conteudo da mensagem.
    receive_message = str(msg.payload.decode("utf-8"))
    print("Mensagem Recebida. Topico %s Mensagem: %s" %
          (msg.topic, receive_message))

    # Insercao dos dados no MongoDB.
    mqtt_topic = msg.topic.replace("sensor/", "")
    insert_data({mqtt_topic: receive_message, "protocol": "MQTT"}, mqtt_topic)

# Insercao no MongoDB.
def insert_data(data, sensor):
    collection = database[sensor]
    collection.insert_one(data).inserted_id

#
def main():
    print("Inicializando...")
    client = mqtt.Client()
    # Conexao do cliente ao servidor.
    con = client.connect(mqtt_server, mqtt_port, mqtt_time_alive)

    # Verificando se ocorreu conexao.
    if con == 0:
        print("Conectado!")

    client.subscribe(mqtt_subscribe)
    client.on_message = on_message

    # Esta funcao faz com que a conexao so encerre caso o cliente se desconecte por
        meio da funcao especifica.
    client.loop_forever()


#
if __name__ == "__main__":
    main()

#####################

# Codigo baseado na bilioteca paho.
import paho.mqtt.client as mqtt
from pymongo import MongoClient

```

```
# Para conexao com o BD.
client = MongoClient("localhost", 27017)
database = client.sensors

# IP para o servidor.
mqtt_server = "192.168.0.103"

# Porta de conexao.
mqtt_port = 1883

# Tempo maximo de conexao.
mqtt_time_alive = 60

# Definindo topico.
mqtt_subscribe = "sensor/#"

# Mensagens recebidas do broken
# obs: metodo da biblioteca paho (github).
def on_message(client, userdata, msg):
    # Conteudo da mensagem.
    receive_message = str(msg.payload.decode("utf-8"))
    print("Mensagem Recebida. Topico %s Mensagem: %s" %
          (msg.topic, receive_message))

    # Insercao dos dados no MongoDB.
    mqtt_topic = msg.topic.replace("sensor/", "")
    insert_data({mqtt_topic: receive_message, "protocol": "MQTT"}, mqtt_topic)

# Insercao no MongoDB.
def insert_data(data, sensor):
    collection = database[sensor]
    collection.insert_one(data).inserted_id

#
def main():
    print("Inicializando...")
    client = mqtt.Client()
    # Conexao do cliente ao servidor.
    con = client.connect(mqtt_server, mqtt_port, mqtt_time_alive)

    # Verificando se ocorreu conexao.
    if con == 0:
        print("Conectado!")

    client.subscribe(mqtt_subscribe)
    client.on_message = on_message

    # Esta funcao faz com que a conexao so encerre caso o cliente se desconecte por
        meio da funcao especifica.
    client.loop_forever()


#
if __name__ == "__main__":
    main()
```

APÊNDICE B – AMOSTRAS OBTIDAS APÓS A SELEÇÃO DE PACOTES



No.	Time	Delta time	Arrival Time	Source	Destination	Protocol	Length	Info
1	0.000000	0.000000	12:45:01,653310	192.168.0.101	192.168.0.100	CoAP	56	CON, MID:44884, POST, TKN:73 64, /light
2	0.034371	0.034371	12:45:01,687681	192.168.0.101	192.168.0.100	CoAP	61	CON, MID:44885, POST, TKN:50 72, /motion
3	0.561191	0.561191	12:45:02,248872	192.168.0.101	192.168.0.100	CoAP	65	CON, MID:44886, POST, TKN:4e 4d, /temperature
4	0.034588	0.034588	12:45:02,283460	192.168.0.101	192.168.0.100	CoAP	62	CON, MID:44887, POST, TKN:44 63, /humidity
5	5.042978	5.042978	12:45:07,326438	192.168.0.101	192.168.0.100	CoAP	56	CON, MID:44888, POST, TKN:76 4f, /light
6	0.036723	0.036723	12:45:07,363161	192.168.0.101	192.168.0.100	CoAP	61	CON, MID:44889, POST, TKN:78 67, /motion
7	0.563727	0.563727	12:45:07,926888	192.168.0.101	192.168.0.100	CoAP	65	CON, MID:44890, POST, TKN:4f 6f, /temperature
8	0.031031	0.031031	12:45:07,957919	192.168.0.101	192.168.0.100	CoAP	62	CON, MID:44891, POST, TKN:67 41, /humidity
9	5.033640	5.033640	12:45:12,991559	192.168.0.101	192.168.0.100	CoAP	56	CON, MID:44892, POST, TKN:69 56, /light
10	0.032883	0.032883	12:45:13,024442	192.168.0.101	192.168.0.100	CoAP	61	CON, MID:44893, POST, TKN:6f 76, /motion
11	0.558627	0.558627	12:45:13,583069	192.168.0.101	192.168.0.100	CoAP	65	CON, MID:44894, POST, TKN:4a 4c, /temperature
12	0.031099	0.031099	12:45:13,614168	192.168.0.101	192.168.0.100	CoAP	62	CON, MID:44895, POST, TKN:76 53, /humidity
13	5.034945	5.034945	12:45:18,649113	192.168.0.101	192.168.0.100	CoAP	56	CON, MID:44896, POST, TKN:50 72, /light
14	0.032618	0.032618	12:45:18,681731	192.168.0.101	192.168.0.100	CoAP	61	CON, MID:44897, POST, TKN:6c 75, /motion
15	0.559600	0.559600	12:45:19,241331	192.168.0.101	192.168.0.100	CoAP	65	CON, MID:44898, POST, TKN:51 72, /temperature
16	0.029961	0.029961	12:45:19,271292	192.168.0.101	192.168.0.100	CoAP	62	CON, MID:44899, POST, TKN:74 45, /humidity
17	5.037126	5.037126	12:45:24,308418	192.168.0.101	192.168.0.100	CoAP	56	CON, MID:44900, POST, TKN:42 70, /light
18	0.033018	0.033018	12:45:24,341436	192.168.0.101	192.168.0.100	CoAP	61	CON, MID:44901, POST, TKN:4f 4e, /motion
19	0.559143	0.559143	12:45:24,900579	192.168.0.101	192.168.0.100	CoAP	65	CON, MID:44902, POST, TKN:5a 53, /temperature
20	0.030758	0.030758	12:45:24,931337	192.168.0.101	192.168.0.100	CoAP	62	CON, MID:44903, POST, TKN:69 6c, /humidity
21	5.051438	5.051438	12:45:29,982775	192.168.0.101	192.168.0.100	CoAP	56	CON, MID:44904, POST, TKN:4e 47, /light
22	0.048374	0.048374	12:45:30,031149	192.168.0.101	192.168.0.100	CoAP	61	CON, MID:44905, POST, TKN:45 76, /motion
23	0.570424	0.570424	12:45:30,601573	192.168.0.101	192.168.0.100	CoAP	65	CON, MID:44906, POST, TKN:70 53, /temperature
24	0.044507	0.044507	12:45:30,646080	192.168.0.101	192.168.0.100	CoAP	62	CON, MID:44907, POST, TKN:68 63, /humidity
25	5.032498	5.032498	12:45:35,678578	192.168.0.101	192.168.0.100	CoAP	56	CON, MID:44908, POST, TKN:6b 79, /light
26	0.033126	0.033126	12:45:35,711704	192.168.0.101	192.168.0.100	CoAP	61	CON, MID:44909, POST, TKN:6a 61, /motion
27	0.561764	0.561764	12:45:36,273468	192.168.0.101	192.168.0.100	CoAP	65	CON, MID:44910, POST, TKN:4c 44, /temperature
28	0.021636	0.021636	12:45:36,295104	192.168.0.101	192.168.0.100	CoAP	62	CON, MID:44911, POST, TKN:6b 78, /humidity
29	5.037635	5.037635	12:45:41,332739	192.168.0.101	192.168.0.100	CoAP	56	CON, MID:44912, POST, TKN:6b 4b, /light
30	0.038928	0.038928	12:45:41,371667	192.168.0.101	192.168.0.100	CoAP	61	CON, MID:44913, POST, TKN:6c 67, /motion
31	0.560363	0.560363	12:45:41,932030	192.168.0.101	192.168.0.100	CoAP	65	CON, MID:44914, POST, TKN:69 47, /temperature
32	0.033307	0.033307	12:45:41,965337	192.168.0.101	192.168.0.100	CoAP	62	CON, MID:44915, POST, TKN:4b 53, /humidity
33	5.042877	5.042877	12:45:47,008214	192.168.0.101	192.168.0.100	CoAP	56	CON, MID:44916, POST, TKN:5a 79, /light
34	0.034329	0.034329	12:45:47,042543	192.168.0.101	192.168.0.100	CoAP	61	CON, MID:44917, POST, TKN:73 59, /motion
35	5.617669	5.617669	12:45:52,660212	192.168.0.101	192.168.0.100	CoAP	65	CON, MID:44918, POST, TKN:66 79, /temperature
36	0.031940	0.031940	12:45:52,692152	192.168.0.101	192.168.0.100	CoAP	62	CON, MID:44919, POST, TKN:42 68, /humidity
37	5.037454	5.037454	12:45:57,729606	192.168.0.101	192.168.0.100	CoAP	56	CON, MID:44920, POST, TKN:55 68, /light
38	0.030490	0.030490	12:45:57,760096	192.168.0.101	192.168.0.100	CoAP	61	CON, MID:44921, POST, TKN:55 52, /motion
39	0.556472	0.556472	12:45:58,316568	192.168.0.101	192.168.0.100	CoAP	65	CON, MID:44922, POST, TKN:41 73, /temperature
40	0.028892	0.028892	12:45:58,345460	192.168.0.101	192.168.0.100	CoAP	62	CON, MID:44923, POST, TKN:41 62, /humidity

```
> Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits)
> Ethernet II, Src: Raspberr_80:a1:d5 (b8:27:eb:80:a1:d5), Dst: HonHaiPr_d5:64:40 (b8:76:3f:d5:64:40)
> Internet Protocol Version 4, Src: 192.168.0.101, Dst: 192.168.0.100
> User Datagram Protocol, Src Port: 45864, Dst Port: 5683
> Constrained Application Protocol, Confirmable, POST, MID:44884
> Data (1 byte)
```

```
0000  b8 76 3f d5 64 40 b8 27  eb 80 a1 d5 08 00 45 00   ·v?·d@·' ·····E·
0010  00 2a 57 18 40 00 40 11  61 91 c0 a8 00 65 c0 a8   ·*W·@·@· a····e··
0020  00 64 b3 28 16 33 00 16  c9 16 42 02 af 54 73 64   ·d·(·3·· ··B··Tsd
0030  b5 6c 69 67 68 74 ff 31                           ·light·1
```

Figura 36 – Amostras obtidas após a seleção de pacotes

Fonte: Elaborado pela autora