UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

Raphael Felipe Ramos Duarte Soares

Comparação entre protocolos da camada de aplicação para IoT

Recife

2019

Raphael Felipe Ramos Duarte Soares

Comparação entre protocolos da camada de aplicação para IoT

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Orientador: Glauco Estácio Gonçalves

Coorientador: Victor Wanderley Costa de Medeiros

Recife

2019

Agradecimentos

Agradeço à Deus por ter me dado a oportunidade de vivenciar este momento e por me dar muito mais do que mereço.

Â meus pais, por serem o suporte e alicerce que me sustentou e me elevou em todos os momentos da minha vida e me fez concluir mais esta etapa.

À minha queridíssima esposa, pela sua compreensão e ajuda nos momentos decisivos e pelo seu incentivo e cobrança constante, me impulsionando para a conclusão deste trabalho.

Ao meu professor e orientador Glauco Gonçalves pela gigantesca paciência, dedicação e encorajamento. Certamente sem você este trabalho não teria sido concluído. Muitíssimo obrigado.

“Não é a força do gotejar da água que fura a pedra, mas sim a persistência incansável desta ação.”

(Ivan Teorilang)

Resumo

A quantidade de objetos inteligentes, com capacidade computacional e de se comunicarem entrei si e se conetarem à Internet cresce cada vez mais. Nesse contexto, a forma como estes dispositivos enviam e recebem informações entre si é de fundamental relevância para alavancar a área da Internet das Coisas (IoT). Baseado nisto, o presente trabalho propõe avaliar os 3 protocolos de comunicação em rede, sendo eles o HTTP, o MQTT e o CoAP, e elencar o mais rápido, robusto e confiável. Para tal, foi elaborado um experimento utilizando uma placa Raspberry Pi, servindo como dispositivo IoT, e um notebook. O experimento consistiu no envio de uma certa quantidade de dados da placa Raspberry Pi para o notebook utilizando os protocolos mencionados anteriormente. Para deixar o experimento o mais próximo possível da realidade, foi utilizado um simulador de redes para emular uma rede de baixa qualidade. Os pacotes de dados enviados foram capturados e foi realizada uma análise estatística para comparar 2 fatores: latência e perda de pacotes. A partir do resultado dessa análise foi possível concluir que o protocolo CoAP se comportou melhor diante de uma rede de baixa qualidade, enquanto que o HTTP mostrou melhor resultado para redes com qualidade superior.

Palavras-chave: Internet das Coisas, Protocolos de Rede, Raspberry Pi, HTTP, MQTT, CoAP.

Abstract

The number of intelligent objects with computational capacity and communication with each other and connection to the Internet grows increasingly. In this context, the way these devices send and receive data is one of the fundamental issues to develop the area of the Internet of Things (IoT). Based on this, the present work analyze 3 network communication protocols in this area: HTTP, MQTT and CoAP to list the fastest and more reliable. To do this, an experiment was designed using a Raspberry Pi, serving as an IoT device, and a notebook. The experiment consists in sending a certain amount of data from the Raspberry Pi to notebooks using the previous protocols. To bring the experiment as close as possible to the reality, a network simulator was used to emulate a poor quality network. The sent data packets were captured and a statistical analysis was performed to compare 2 factors: latency and packet loss. From the result of this analysis, it was concluded that the CoAP protocol will behave better in a poor quality network, while HTTP performe the best result for superior quality network.

Keywords: Internet of Thing, Network Protocols, Raspberry Pi, HTTP, MQTT, CoAP.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IoT	Internet of Things (Internet das Coisas)
API	Application programming interface
HTTP	Hypertext Transfer Protocol
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
IP	Internet Protocol
MQTT	Message Queuing Telemetry Transport
CoAP	Constrained Application Protocol

Sumário

1 Introdução

1.1 Apresentação

No primeiro semestre de 2018 a humanidade alcançou a marca de 17 bilhões de dispositivos conectados (LUETH, 2018). Dentre estes 17 bilhões há um nicho de 7 bilhões que vem crescendo de forma quase exponencial nos últimos anos, com perspectiva de crescimento para 10 bilhões em 2020, 22 bilhões em 2025 e uma prospecção de movimentar até U$ 1,56 trilhão em neste mesmo ano (LUETH, 2018). Estes dispositivo são conhecidos como Internet das Coisas (Internet of Things, ou IoT). Dispositivos IoT são basicamente objetos que possuem alguma tecnologia embarcada, sensores e conexão com Internet, tornando-os capazes de coletar, transmitir e, em alguns casos, processar dados (SANTOS et al., 2016). Hoje já existe uma enorme variedade deles no mercado, tais como lâmpadas, geladeiras, micro-ondas inteligentes. A expectativa é que eles estejam cada vez mais inseridos no cotidiano das pessoas.

Atualmente já existem diversos exemplos de como estes dispositivos podem ajudar a sociedade em suas atividades diárias, no âmbito individual e coletivo. Na medicina, pode ser utilizado através do monitoramento dos sinais vitais e condições de saúde dos pacientes. Nas residências, através de monitoramento de câmeras de segurança à distância. No transporte, com sensores de proximidade nos carros ajudando a evitar colisões e controle de tráfego nas vias ajudando a melhorar o trânsito. Na agricultura, com sensores de temperatura, umidade e luminosidade nas fazendas e plantações ajudando a otimizar a produção de comida (SUNDMAEKER et al., 2010).

Apesar do progresso científico estar avançando consideravelmente neste setor, por ser um campo relativamente recente, a tecnologia para realizar de forma satisfatória e eficaz a comunicação entre estes objetos, também chamada comunicação Maquina-para-Maquina (em inglês: Machine to Machine, ou M2M) ainda é muito incipiente. Os problemas mais graves nesse sentido são a heterogeneidade, o desacoplamento, e a conectividade, ou seja, a comunicação entre dispositivos é amplamente divergente entre si, a solução dada para a comunicação entre dispositivos em um determinado cenário por vezes não se aplica a outros e os dispositivos que se encontram em um dado momento conectados a uma rede, podem não estar mais num momento posterior (SUNDMAEKER et al., 2010). O fato é que o aprimoramento e refinamento da troca de informações entre dispositivos IoT é um fator crucial para o avanço e solidificação dessa tecnologia e precisa ser estudado com mais atenção.

Com base nisto, este trabalho se propõe a analisar e comparar o desempenho

de 3 protocolos de rede da camada de aplicação utilizados em dispositivos IoT para transferência de dados: HTTP (*Hypertext Transfer Protocol* - Procolo de Transferência de hipertexto), MQTT (*Message Queuing Telemetry Transport* - Transporte Telemétrico de mensagens em filas) e CoAP (*Constrained Application Protocol* - Protocolo Restrita de Aplicação).

Para tal foi realizado um experimento seguindo as normas de projeto de experimentos em que foi utilizada uma placa Raspberry Pi como dispositivo IoT cuja responsabilidade foi o envio de dados, utilizando cada um dos 3 protocolos mencionados, capturados previamente de uma estação meteorológica, e um notebook como servidor de aplicação que foi responsável pelo recebimento destes dados enviados. Os pacotes de dados transferidos da placa para o notebook foram capturados a nível de rede, tratados, analisados e comparados. Através desta comparação foi possível chegar ao objetivo do trabalho.

1.2 Objetivos

O presente trabalho tem por objetivo principal avaliar a eficiência de alguns protocolos de comunicação de nível de aplicação para dispositivos de Internet das Coisas. Através deste estudo pretende-se atingir os seguintes objetivos específicos:

- Avaliar o desempenho dos protocolos quanto a quantidade de dados trafegada por segundo;
- Verificar o impacto do atraso da rede e da perda de pacotes sobre o desempenho de cada protocolo.

1.3 Estrutura do Trabalho

O presente trabalho está organizado em seis capítulos. O capítulo atual que visa introduzir o contexto geral em que problema se encontra inserido, além da motivação, justificativa e objetivos que nortearam sua execução. O demais capítulos se encontram divididos da seguinte maneira:

Capítulo 2: Trata da fundamentação teórica, onde são apresentados os conceitos envolvidos no trabalho, bem como a arquitetura e o *modus operandi* de cada protocolo.

Capítulo 3: São discutidos alguns trabalhos relacionados cujo objeto de estudo também é a comparação entre protocolos de comunicação.

Capítulo 4: Apresenta a metodologia e arquitetura empregada no experimento, juntamente com os métodos de captura dos dados para análise.

Capítulo 5: Expõe e discute os resultado a partir análise e interpretação dos dados obtidos.

Capítulo 6: Traz as considerações finais do trabalho, junto com suas contribuições, objetivos alcançados e propõe possíveis trabalhos futuros.

2 Fundamentação teórica

2.1 Considerações iniciais

Neste capítulo introduzimos os conceitos que serão abordados durante o desenvolvimento deste trabalho. A seção 2.2 faz uma introdução sobre os tipos de mensagens enviadas e recebidas pelos computadores. A seção 2.3 discorre sobre as arquiteturas que os protocolos abordados neste trabalho fazem uso. A seção 2.4 apresenta uma visão geral sobre cada um dos protocolos utilizados bem como seu funcionamento e características.

2.2 Tipos de mensagens

O intercâmbio de dados entre computadores pode se dar de duas maneiras distintas: Síncronas ou assíncronas. Mensagens enviadas de forma síncrona são mensagens em que o emissor, após o envio da mensagem, permanece esperando até obter uma resposta do receptor para então dar por concluído o envio. Já em mensagens assíncronas, não há necessidade de espera por parte do emissor. Mensagens assíncronas são uma solução pragmática para os problemas de sistemas distribuídos, pois torna-se dispensável que os dois sistemas, emissor e receptor da mensagem, estejam ativos no mesmo momento (HOHPE; WOOLF et al., 2003).

2.3 Arquiteturas

Quando uma aplicação de um computador deseja compartilhar dados com outras aplicações em outros computadores em rede, torna-se necessário definir como essa aplicação transmitirá esses dados.

De modo geral existem dois principais padrões para propagação de mensagens entre computadores conectados em rede, são eles: requisição-resposta (do inglês, request-response) e publicação-assinatura (do inglês, publish-subscribe) (HOHPE; WOOLF et al., 2003).

2.3.1 Requisição-Resposta

O padrão requisição-resposta é uma forma de comunicação entre computadores que funciona da seguinte forma: um computador envia uma requisição de algum dado através de um canal, nesse caso a rede, e um outro computador recebe essa

requisição, processa esse recebimento e envia uma resposta de volta (HOHPE; WOOLF et al., 2003). Esse procedimento é análogo a uma ligação telefônica, em que uma pessoa, requisitante, liga solicitando alguma informação através de um canal, nesse caso a rede de telefonia, e outra pessoa responde, atendendo essa solicitação.

Esse padrão é extremamente comum na Internet hoje, pois a grande maioria dos navegadores, servidores e aplicações web se comunicam uns com os outros utilizando o protocolo HTTP, que por sua vez é baseado neste padrão. Quando uma pessoa digita uma URL num navegador, este faz uma requisição utilizando o protocolo HTTP, que será detalhado mais adiante, a um servidor e este por sua vez, ao receber a requisição, processa e geralmente devolve como resposta um texto no formato HTML que é interpretado pelo navegador e utilizado para renderizar a página em questão (GOURLEY et al., 2002).

Na extensa maioria dos casos este padrão é implementado de forma síncrona, ou seja, ao realizar uma requisição, o requisitante permanece esperando até obter uma resposta do destinatário ou até o período máximo de espera (timeout) ser atingido. Entretanto, em alguns casos, este padrão pode ser implementado de forma assíncrona, ou seja, não há necessidade de espera por parte do solicitante após a realização da requisição. Neste caso cabe ao destinatário da solicitação entrar em contato com o requerente para devolver-lhe o resultado do seu requerimento (GOURLEY et al., 2002).

2.3.2 Publicação-Assinatura

No padrão publicação-assinatura, ao contrário do padrão requisição-resposta, a troca de mensagens entre o remetente e o destinatário não ocorre de forma direta. Neste padrão existem 3 agentes, o publicador, o assinante e a central (HOHPE; WOOLF et al., 2003).

A comunicação entre os agentes ocorre da seguinte forma: o publicador envia sua mensagem para a central contendo o conteúdo da mensagem e o tópico ou categoria da mensagem. O assinante, por sua vez, envia para a central uma solicitação de assinatura para um determinado tópico ou categoria. É permitido ao assinante possuir quantas assinaturas quiser. A central funciona como um intermediário e é responsável por receber mensagens de publicadores e por aceitar e registrar solicitações de assinaturas. Ao receber uma mensagem de um publicador, a central verifica quantos e quais são os assinantes do tópico daquela mensagem e então envia um cópia da referida mensagem à todos estes. É de responsabilidade da central armazenar todas as mensagens recebidas, garantir que todos os assinantes recebam todas as mensagens trafegadas nos tópicos dos quais assinaram, e remover as mensagens armazenadas uma vez que a mesma foi transmitida com sucesso para o todos os assinantes (HOHPE; WOOLF et al., 2003).

As duas principais vantagens desse padrão são a independência e a escalabilidade. Independência pois não existe nenhuma correlação entre publicadores e assinantes, eles não precisam ter conhecimento da existência um do outro, inclusive é factível a presença de publicadores de um determinado tópico sem que hajam assinantes e vice-versa. A escalabilidade refere-se a potencial de crescimento da rede através da inclusão nós, seja elem publicador ou assinante, neste tipo de rede. Não se faz necessário alterações na topologia da rede para adição ou remoção de um nó, pois o padrão já dá suporte a este tipo de operação (LI; JACOBSEN, 2005).

2.4 Protocolos

Esta seção irá dissertar sobre a história, as características e como funcionam os 3 protocolos abordados neste trabalho. São eles:

1. HTTP
2. CoAP
3. MQTT

2.4.1 HTTP

O que é ?

O Hypertext Transfer Protocol (HTTP), ou Protocolo de Transferência Hipertexto em português, é um protocolo de comunicação, localizado na camada de aplicação (a última camada no modelo OSI). Ele é utilizado em sistemas de informação com o objetivo de transmitir documentos em hipermídia, distribuídos e colaborativos. Hipermídia é a reunião de vários tipos de mídias, tais como imagens, vídeos, sons e texto, em um documento no ambiente computacional, que são navegáveis, interativos e cuja leitura se dá de forma não linear. Qualquer página na Web é um ótimo exemplo de um documento de hipermídia (GOURLEY et al., 2002).

O desenvolvimento do HTTP é coordenado pela World Wide Web Consortium (W3C) e pela Internet Engineering Task Force (IETF) e tem sido a principal forma de comunicação na World Wide Web desde que foi criado, em 1990. Na sua primeira versão HTTP/0.9, seu único objetivo era a transferência simples de dados através da internet. Na sua segunda versão, a 1.0, o procotolo foi melhorado, foi incluído os headers, que continham informações sobre o que estava sendo trafegado. Também foi incluído nesta versão o modelo request-response, ou requisição-resposta, para informar se a comunicação foi realizada com sucesso ou não. Atualmente o HTTP está

na versão 2.0 e diversas melhorias foram acrescidas, como compressão de headers, definição de prioridade nas requisições, entre outas (GOURLEY et al., 2002).

Características

De acordo com (GOURLEY et al., 2002), com relação as características do HTTP, pode-se dizer que ele é:

- Simples: as mensagens em HTTP foram desenhadas para serem fáceis de ler e entender por humanos sem necessitar de nenhum tipo de aparato extra;
- Extensível: os cabeçalhos das mensagens HTTP foram criados de forma a propiciar a adição de novas propriedades de maneira fácil e intuitíva;
- *Stateless*: o protocolo foi criado para não manter estado, ou seja, cada requisição é totalmente independente uma da outra. Não há compartilhamento de dados entre duas requisições diferentes.
- Confiabilidade: Como HTTP envia seus dados através da camada de transporte TCP/IP, ele é um protocolo considerado seguro, pois os dados trafegados não se perdem ”silenciosamente”.

Como funciona ?

O HTTP funciona no modelo request-response utilizando uma arquitetura cliente-servidor. Quando se abre um Web Browser, como Google Chrome, Mozilla Firefox ou Internet Explorer e se digita uma URL, esse navegador é o cliente da nossa arquitetura e ele está fazendo uma requisição HTTP a uma aplicação hospedada em um computador em outra parte do globo, que é o servidor. Esse servidor, por sua vez, ao receber essa requisição, realiza o processamento desta e retorna uma resposta ao cliente. Essa resposta contém informações sobre este processamento e pode ou não conter o conteúdo solicitado no corpo da mensagem.

As mensagens trocadas entre o cliente e o servidor HTTP possuem uma estrutura predefinida. Essa estrutura é constituída por 3 partes, a Request-Line, ou linha inicial, o Header, ou cabeçalho, e o Body, ou corpo da mensagem. A Request-line é a primeira linha da requisição. Caso esta seja uma solicitação enviada pelo cliente, tem-se o tipo de método da solicitação e a versão do protocolo que está sendo utilizada. Caso seja uma resposta do servidor, tem-se a versão do protocolo seguido pelo código da resposta da mensagem (FIELDING et al., 2006). Os códigos mais comuns são: 200 - indica que a solicitação foi processada com sucesso, 400 - indica que a requisição está com algum erro de sintaxe, 401 - indica que está tentando acessar um recurso cuja permissão é restrita e é necessário informar alguma credencial, 403 - indica que

a requisição é válida, e que o usuário foi autenticado, mas não possui permissão para acessar o recurso, 404 - indica que o recurso que está sendo solicitado não foi encontrado (GOURLEY et al., 2002). Da segunda linha em diante vem o Header, que contém uma série de informações sobre a requisição em si, tais como data/hora, endereço e tipo do servidor, tamanho e tipo do conteúdo de retorno da resposta, entre outros. Após o término do Header, vem uma linha em branco e em seguida vem o body, que é opcional, com o conteúdo da mensagem em si. Ao acessar uma URL através de um navegador web, é no body que o servidor manda o arquivo html que será renderizado na tela do browser (GOURLEY et al., 2002).

De acordo com (GOURLEY et al., 2002), o HTTP utiliza uma série de métodos para indicar qual é a ação desejada pelo cliente ao servidor. São ao todo 9 métodos, porém 4 destes são os mais utilizados, são eles: GET, POST, PUT e DELETE. Cada um deles possui uma semântica própria, todavia compartilham algumas características em comum. Abaixo segue uma breve descrição do que cada método faz.

- GET: é um método responsável por buscar informações. Esse método pode ou não ter conteúdo no corpo da resposta;
- HEAD: é um método idêntico ao GET, porém não poussui conteúdo no corpo da resposta;
- POST: é um método responsável por submeter informações ao servidor. Normalmente é utilizado para causar mudanças de estados no mesmo;
- PUT: também é um método de submissão, porém é comumente utilizado para atualizar informações já presentes no servidor;
- DELETE: é responsável por deletar recursos específicos;
- CONNECT: é responsável por criar um túnel de conversasão entre o cliente e o servidor. Normalmente requer algum tipo de autenticação por parte do cliente;
- OPTIONS: é usado para obter as opções de requisição permitidas para um determinado recurso no servidor;
- TRACE: é utilizado para obter o caminho que a requisição faz até o servidor. É retornado para o cliente quais proxys e máquinas a sua requisição passa até chegar ao destino;
- PATCH: é um método utilizado para realizar alterações parciais em um derterminado recurso;

2.4.2 CoAP

O que é ?

O *Constrained Application Protocol* (CoAP) é um protocolo criado pela *Internet Engineering Task Force* (IETF) que visa disponibilizar uma estrutura de troca de mensagens orientada a recursos. Ele foi planejado e otimizado para funcionar com dispositivos limitados, seja por disporem de uma fonte de energia restrita ou possuírem baixo poder de processamento ou armazenamento, ou estejam imersos em redes limitadas ou que possuam restrições de alguma forma, seja por uma alta taxa de perda de pacotes, ou por uma baixa velocidade de banda ou por qualquer outro motivo (CASTELLANI et al., 2011).

Características

O CoAP foi criado baseado no modelo REST, por isso uma de suas maiores vantagens é a integração facilitada com o protocolo HTTP, que é o protocolo base da *World Wide Web*. Outra característica importante deste protocolo é possuir um cabeçalho pequeno, limitado a 4 bytes, resultando numa diminuição do tamanho do pacote como um todo e na priorização do conteúdo da mensagem em si (CASTELLANI et al., 2011). Este cabeçalho é constituído de, entre outras coisas, um identificador da mensagem, um token e um tipo da mensagem. O identificador serve para o CoAP detectar mensagens em duplicidade. O Token por sua vez, permite o CoAP relacionar as mensagens de requisição com a sua resposta. O tipo de mensagem pode variar entre confirmable ou confirmável (CON), Non-confirmable ou não-confirmável (NON), acknowledgement ou recebido (ACK) e *reset* ou reiniciar (RST) (SHELBY; HARTKE; BORMANN, 2014).

A tupla formada pelo identificado, token e tipo tornam possível uma outra característica fundamental do CoAP que é o suporte a mensagens assíncronas. Desta forma, o cliente pode fazer uma requisição ao servidor e este, por sua vez, não responder imediatamente e sim depois de um certo tempo, quando o processamento daquele pedido estiver terminado (KOVATSCH; LANTER; SHELBY, 2014). Algumas outras características do CoAP incluem o suporte à descoberta de recursos, suporte multicast e suporte a *content-type*, do mesmo modo que o HTTP (SHELBY; HARTKE; BORMANN, 2014).

Como funciona ?

Na troca de mensagens existe um servidor que disponibiliza recursos através de uma URL, e um cliente, que acessa esses recursos fazendo uma requisição a esta URL utilizando um subconjunto dos mesmos métodos do HTTP: GET, POST, PUT

e DELETE. Porém ao contrário do HTTP, o CoAP foi desenvolvido utilizando UDP na camada de transporte, e por isso necessitou implementar seus próprios métodos de detecção de duplicidade, retransmissão de pacotes e confiabilidade (KOVATSCH; LANTER; SHELBY, 2014). Para tratar a confiabilidade o CoAP se utiliza do tipo de mensagem CON. Quando um nó (cliente ou servidor) envia uma requisição deste tipo, ele espera receber um outra requisição do tipo ACK como resposta, informando que o outro lado recebeu a requisição enviada. A Figura 1 retirada de (SHELBY; HARTKE; BORMANN, 2014) ilustra uma requisição CoAP do tipo CON feita por um cliente utilizando o método GET solicitando o valor da temperatura. Na requisição da esquerda a temperatura é retornada com sucesso pelo servidor no conteúdo mensagem ACK. Na requisição da direita, é retornado o erro 404 informando que o recurso solicitado não foi encontrado.

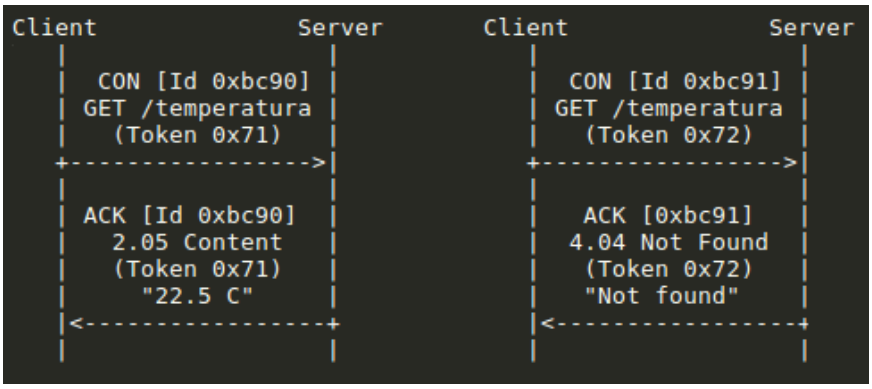

Figura 1 – Exemplo de requisição CoAP do tipo CON e da resposta ACK

Caso o nó receptor não devolva esta mensagem esperada, o nó solicitante aguarda o tempo de timeout definido e então envia novamente a mensagem CON, e continua reenviando até receber a resposta ACK esperada. A Figura 2 ilustra o reenvio da requisição por parte do cliente após timeout até o recebimento da resposta ACK do servidor. Por outro lado, mensagens do tipo NON são não confirmáveis, ou seja, não necessita receber uma mensagem ACK como resposta (SHELBY; HARTKE; BORMANN, 2014).

Como já mencionado anteriormente, o CoAP faz uso de um token no cabeçalho para possibilitar o envio de mensagens assíncronas. Este token é utilizado para relacionar as mensagens. Quando um cliente envia uma requisição para um servidor requisitando um recurso, seja ela do tipo CON ou NON, nesta requisição há um token, e o servidor, ao responder, faz uso deste mesmo token na mensagem de resposta. Caso a mensagem solicitada seja CON e o servidor não puder enviar o conteúdo da resposta imediatamente, ele envia uma mensagem ACK vazia e, após algum tempo, envia a resposta também utilizando o mesmo token inicial. A figura 3 ilustra bem a situação mencionada acima (SHELBY; HARTKE; BORMANN, 2014).

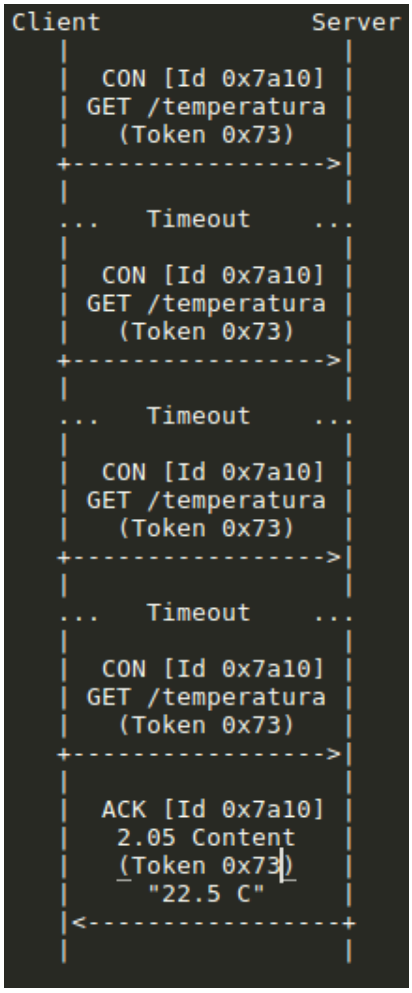

Figura 2 – Exemplo de reenvio da requisição CoAP do tipo CON após o timeout

2.4.3 MQTT

O *Message Queuing Telemetry Transport* (MQTT) é um protocolo de comunicação entre máquinas (M2M ou Machine-to-Machine) desenvolvido pela IMB e foi desenhado em cima de certos princípios que o torna ideal para ser utilizado em aplicações para Internet das Coisas (TANG et al., 2013). Os princípios que o MQTT se baseia são de minimizar a quantidade de banda consumida e de recursos utilizados do dispositivo em que está sendo executado, tentando manter a confiabilidade e assegurar que a mensagens serão entregues com sucesso. Ele é um protocolo extremamente leve que funciona usando uma arquitetura publicação-assinatura para enviar e receber mensagens. Ele é bastante eficiente quando utilizado em ambientes onde a rede é de baixa qualidade, ou quando torna-se necessário restringir a quantidade de meta-dados gerados nas mensagens transmitidas para que cada pacote de dados enviados possa ser aproveitado ao máximo (STANFORD-CLARK; TRUONG, 2013).

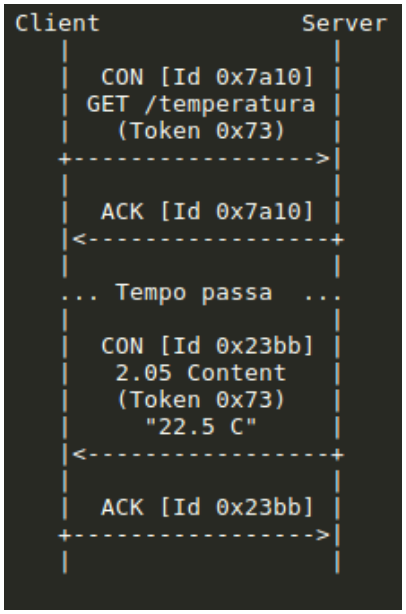

Figura 3 – Exemplo de uso do token relacionando diferentes requisições CoAP

Características

Dentre as características do MQTT estão o fato dele utilizar os protocolos TCP/IP para transmitir suas mensagens, garantindo desse modo a não perda de pacotes durante a transmissão dos dados. Além disso ele utiliza a arquitetura publicação-assinatura, que provê uma distribuição de mensagens de um para muitos, ou seja, um publica e vários recebem. Ademais o protocolo minimiza a sobrecarga durante a troca de mensagens no intento de reduzir a quantidade de dados enviados na rede (BANKS; GUPTA, 2014).

Outra característica importante do MQTT refere-se a qualidade do serviço durante a entrega das mensagens aos assinantes. Ele dispões de 3 tipos de serviços de entrega de mensagens: ”no máximo 1”, ”pelo menos 1”e ”exatamente 1”. ”No máximo 1”que dizer que a mensagem tentará ser entregue 1 vez apenas e não haverá tentativa de reenvio caso haja perda. ”Pelo menos 1”informa que a mensagem será entregue, porém pode haver duplicidade, ou seja, a mesma mensagem enviada mais de uma vez. ”Exatamente 1”quer dizer que há garantia que a mensagem será entregue e que não haverá duplicidade (TANG et al., 2013).

O MQTT também da suporte para comunicação assíncrona entre as partes, desse modo o emissor e o receptor são desacoplados entre si, tanto no espaço quanto no tempo. Essa característica lhe garante escalabilidade e o torna ideal para uso em redes não confiáveis (MANANDHAR, 2017).

Como funciona

O protocolo MQTT define dois tipos de entidades na rede: um *message broker* e vários clientes. O broker o responsável por receber todas as mensagens enviadas pelos clientes e, em seguida, roteia essas mensagens para os clientes de destino relevantes. Um cliente é qualquer coisa que possa interagir com o *broker* e receber mensagens. Um cliente pode ser um sensor de IoT em campo ou um aplicativo que processa dados de IoT (MANANDHAR, 2017).

A permuta de mensagens ocorre através de tópicos. Primeiramente o cliente conecta-se ao broker e se inscreve em um tópico. Em seguida este cliente publica as mensagens no tópico inscrito, enviando a mensagem e o tópico ao broker. Logo após o broker encaminha a mensagem enviada à todos os clientes que assinaram este tópico. A Figura 4 retirada de (TANG et al., 2013) ilustra como ocorre a troca de mensagens no MQTT. É válido salientar que não há exclusividade na publicação de mensagens e assinatura dos tópicos, ou seja, o cliente que publica uma mensagem num determinado tópico pode ser assinante em um outro tópico, como também pode ser assinante no mesmo tópico em que publicou (TANG et al., 2013).

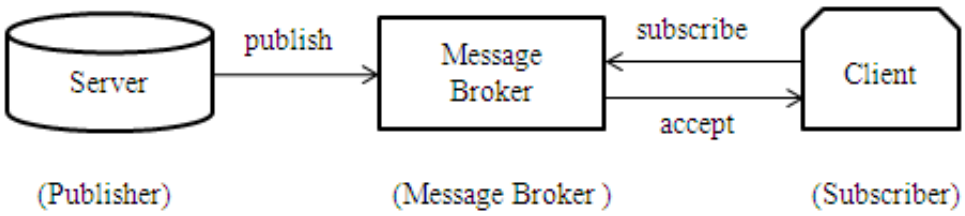

Figura 4 – Diagrama da arquitetura da troca de mensagens no MQTT

3 Trabalhos relacionados

Neste capítulo são apresentados alguns trabalhos similares que realizam análise de protocolos para comunicação M2M (Machine-to-Machine).

(CHEN; KUNZ, 2016) avaliaram a aplicabilidade da IoT na área médica. Em seu trabalho, foi proposto o seguinte cenário: um paciente utiliza diversos tipos sensores pelo corpo, que captam informações diversas tais como nível de oxigênio, pressão arterial, atividade cerebral, atividade cardíaca, atividade muscular e movimentação. Os dados destes sensores são enviados para um dispositivo central, denominado *Patient Gateway*, que os agrega e envia, via rede sem fio, para um servidor. Esse servidor fica então responsável pelo processamento dos dados e envio de alertas para o *smartphone* do médico.

Neste estudo foram utilizados 4 tipos de protocolos IoT para o envio dos dados do *Patient Gateway* para o servidor, comparados através de 3 métricas de rede. Os protocolo foram CoAP, MQTT, DDS e um 4º protocolo personalizado, criado especificamente para este trabalho, que envia dados no formado JSON através do protocolo UDP. Já os parâmetros de avaliação foram consumo de banda, latência e taxa de perda de pacotes.

Os resultados de (CHEN; KUNZ, 2016) apontaram que protocolos que utilizam TCP superam os protocolos UDP diante de uma rede de baixa qualidade devido a sua perda de pacotes quase nula, porém o consumo de banda e a latência aumentam. Também concluíram que o protocolo DDS é mais efetivo que o protocolo MQTT devido a sua latência ter sido menor. Já os protocolos UDP, por possuir uma taxa de perda de pacotes imprevisível, podem ser utilizados em aplicações onde o envio e a captação de dados não são críticos ao sistema, o que não é o caso da área médica.

O trabalho de (CARO et al., 2013) considera uma aplicação de sensoriamento baseado em multidões. Neste caso, sensores (GPS, microfones, câmeras, luminosidade etc) instalados em dispositivos pessoais (*smartphones*) são considerados parte de um sistema de monitoramento. Neste trabalho, os autores comparam, tanto quantitativa quanto qualitativamente, os protocolos MQTT e CoAP para este tipo de aplicação. O experimento considerou o envio de dados a partir de um *smartphone* Android para um notebook, via rede 802.11. Nos testes foram avaliados o consumo de banda, latência (através do *Round Trip Time* - RTT) e taxa de perda de pacotes dos protocolos, utilizando tanto mensagens com confiabilidade quanto sem confiabilidade.

Os autores concluíram que o CoAP apresentou melhores resultado nos quesitos de consumo de banda e latência, o que o torna mais eficiente quando o objetivo é na

redução da utilização dos recursos do dispositivo em questão e do consumo de dados da rede. Já o MQTT, por conta da sua confiabilidade, se torna mais indicado para aplicações mais sofisticadas, que requisitam controle de congestionamento e garantia que os dados irão chegar ao destinatário. Porém, em aplicações que não requerem uma alta frequência na transmissão dos dados, a taxa de confiabilidade de ambos os protocolos se mantém praticamente a mesma.

(BANDYOPADHYAY; BHATTACHARYYA, 2013) também realizaram uma análise entre os protocolos CoAP e MQTT. Neste trabalho foram considerados os aspectos consumo de energia, consumo de banda, confiabilidade, entre outros. É importante salientar que neste trabalho foram abordadas as possíveis arquiteturas utilizando o CoAP, tanto no seu aspecto requisição-resposta, quanto no modo recurso-observador, onde seu funcionamento se assemelha ao MQTT. Neste trabalho foram utilizados computadores Linux tanto para o Cliente quando para o Servidor, em conjunto com o software WANEM para fazer o controle da rede e simular uma rede de baixa qualidade e, por último, o software Wireshark para analisar o tráfego na rede. O experimento consistiu na análise dos dados trafegados na rede e do consumo de energia em kWh com taxa de perda de pacotes em 0% e em 20%. A principal conclusão foi que o CoAP é mais eficiente tanto no consumo de energia quanto no consumo de banda.

(THANGAVEL et al., 2014) realizaram a análise entre os protocolos CoAP e MQTT através de um intermediário customizado. Este intermediário consistiu numa API própria, que serviu de ponte entre a captação dos dados pelos sensores e o envio desses dados na rede IP. A performance foi medida em termos de latência e total de dados (bytes) transferidos por mensagem. O total de dados transferidos foi considerado como o indicador do consumo de banda. Já a latência foi mensurada pela diferença do tempo em que o dado foi recebido pelo servidor e que foi enviado. No experimento, os autores fizeram uso de um notebook que serviu como servidor tanto para o CoAP quanto para o MQTT, uma placa BeagleBoard-xM que foi responsável tanto pela captação dos dados dos sensores (onde foi implementado a API customizada) e envio destes para o servidor, quanto pela recepção dos dados processados. Por fim um Netbook rodando o software Wanem, cujo propósito foi emular uma rede de baixa qualidade.

Os autores observaram que a performance dos protocolos variam de acordo com a qualidade da rede, pois o MQTT experimentou um atraso menor que o CoAP, para pequenos níveis de perda dos pacotes, porém seu atraso foi superior ao CoAP para perda de pacotes alta. Também concluíram que, para tamanhos de mensagens pequenos e perda de pacotes igual ou inferior à 25%, o CoAP gera menos tráfego extra na rede do que o MQTT para obter uma conexão confiável.

(GLÓRIA; CERCAS; SOUTO, 2017) focaram seu estudo na camada de enlace.

Eles avaliaram 4 tipos de protocolos, sendo 2 deles com fio e 2 sem fio, foram eles: I^2C, RS232, ZigBee e LoRa. Para esta análise foram utilizadas 3 placas: Arduino, Raspberry Pi e ESP8266 para avaliar se a diferença entre as placas ou a distância em que elas estão uma da outra interfere no atraso, na taxa de dados trafegadas e na eficiência dos protocolos. Eles concluíram que o RS232 é a melhor escolha quando se trata de um protocolo com fio e LoRa por sua vez é a melhor nos protocolos sem fio. Isso se deu por conta da baixa complexidade e baixo custo envolvidos e pelo fato de não gerarem interferência com redes WiFi. Com relação às placas, a que obteve o melhor resultado foi a Arduino conectada com a ESP8266 tanto por conta da confiabilidade quanto por conta da Raspberry Pi, que por ser uma placa que roda em cima de um sistema operacional, possui características extras por vezes não desejadas, tais como gerenciamento de processos, sistemas de arquivos entre outras, que acabam requisitando um consumo de energia mais elevado.

4 Materiais e Método

O presente capítulo tem o objetivo de descrever os materiais utilizados, as ferramentas empregadas e a metodologia aplicada à análise dos protocolos abordados, tanto na elaboração e condução do experimento pelo qual se fundamenta este trabalho, como na captura, tratamento e análise dos dados coletados a partir do mesmo.

O experimento consistiu em escolher um conjunto fixo de dados e, utilizando uma placa IoT como cliente de aplicação, enviá-los repetidas vezes empregando alguns fatores preestabelecidos através de uma rede local para um notebook que funcionou como servidor de aplicação, fazendo uso de cada um dos protocolos estudados neste trabalho. Para a realização deste experimento foi seguido o seguinte processo:

- Escolha dos fatores a serem analisados;
- Delimitação dos níveis de cada fator;
- Escolha dos conjunto de dados a serem enviados;
- Escolha da placa IoT a ser utilizada;
- Definição da linguagem de programação e implementação do algoritmo responsável pelo envio do conjunto de dados;
- Determinação da arquitetura do experimento e implementação dos fatores;
- Fixação da quantidade de repetições de cada experimento;
- Definição do design de experimento utilizado;
- Execução dos experimentos e coleta dos meta-dados;
- Análise e comparação dos meta-dados coletados;

4.1 Fatores e níveis

Para determinar qual protocolo se comporta melhor em diferentes situações, foram selecionados 2 fatores preponderantes na comunicação de redes, sendo eles a latência e a perda de pacotes. Foram escolhidos 3 níveis para cada fator. Na latência os níveis selecionados foram 400 ms, 100 ms e 10 ms variando em 20% para mais ou menos seguindo uma distribuição normal. Para a perda de pacotes foram estabelecidos os valores 25%, 15% e 0% de perda. A velocidade máxima de tráfego na rede foi

fixada em 21kbps. A escolha destes níveis baseou-se no trabalho de (CHEN; KUNZ, 2016), no qual utilizou fatores e níveis similares para comparar os protocolos CoAP, MQTT, DDS e um protocolo UDP customizado. Já a escolha da taxa máxima de tráfego na rede foi definida tendo por base o trabalho de (EKO; PASTIMA et al., 2017) no qual um dos objetivos foi avaliar a taxa média de download utilizando a tecnologia GSM, chegando a um resultado similar a taxa utilizada no presente trabalho.

4.2 Conjunto de dados

A escolha do conjunto de dados que foram transmitidos pela placa IoT, que será tratado a partir de agora como dados de envio, foi feita visando obter informações o mais próximo possível da realidade. Para tal, foram utilizados dados reais previamente coletados a partir de uma estação meteorológica pelo Instituto Nacional de Meteorologia (INMET) capturados de minuto a minuto e agrupados por hora, durante o período de dezembro de 2016 à dezembro de 2017. Estes dados, que podem ser acessados através deste link, foram utilizados no formato JSON e contemplam um total de 9168 objetos, onde cada objeto corresponde a uma hora de um dia do período supracitado. Cada um destes registros contém, além da data, hora e timestamp daquele momento, as seguintes informações climáticas armazenadas durante toda aquela hora: precipitação, radiação, rajada de vento, velocidade do vento, direção do vento, temperatura mínima, máxima e naquele instante, umidade mínima, máxima e naquele instante, pressão mínima, máxima e naquele instante e ponto do orvalho mínimo, máximo e naquele instante. Abaixo encontra-se um exemplo de um dos JSON extraído do conjunto de dados enviados.

```
{
    "values": {
        "hora": "03",
        "radiacao": "-3.54",
        "vento_rajada": "1.2",
        "temp_max": "20.0",
        "umid_max": "90",
        "umid_min": "82",
        "data": "24/04/2017",
        "pressao": "888.5",
        "pressao_min": "888.5",
        "vento_direcao": "0.2",
        "pto_orvalho_inst": "17.3",
        "pto_orvalho_max": "17.3",
        "vento_vel": "213",
```

```
        "temp_min": "18.6",
        "pressao_max": "888.6",
        "codigo_estacao": "A001",
        "pto_orvalho_min": "16.6",
        "temp_inst": "19.2",
        "umid_inst": "89",
        "precipitacao": "0.0"
    },
    "ts": 1493002800
}
```


4.3 Escolha dos materiais

Um dos fatores preponderantes para a realização do experimento foi a escolha da placa IoT responsável pelo envio dos dados, que será tratada a partir de agora como placa de envio. Foi levado em conta fatores como facilidade de aquisição, preço, possibilidade de conexão Wireless e Ethernet, API's disponíveis para os protocolos escolhidos, facilidade de implementação e manutenção. As placas analisadas foram a ESP-8266 e a Raspberry Pi. A placa escolhida para o experimento foi a Raspberry Pi 3, modelo B, devido a facilidade de aquisição, de implementação e o fato da mesma já vir de fábrica pronta pra uso sem necessidade de peças adicionais, além de possuir conexão Wireless e Ethernet por padrão.

O notebook utilizado como servidor de aplicação para receber os dados enviados foi um Lenovo Thinkpad T430u, com 8GB de memória RAM e processador Intel Core i5 3º geração, rodando o sistema operacional Ubuntu versão 16.04.

Durante os testes prévios à execução do experimento, identificou-se a necessidade de inclusão de uma outra placa, Raspberry Pi 2 Modelo B (512MB), para funcionar como um roteador, intermediando a transmissão dos dados da placa de envio para o servidor de aplicação, e que será tratada a partir de agora como placa roteadora.

A inclusão da placa roteadora na arquitetura do experimento também teve um segundo propósito, que foi de executar um script elaborado especificamente para implementar a emulação de rede utilizando o sistema de controle de tráfego do Linux - NETEM. Este script, que pode ser visto neste link, teve o proposito de aplicar os fatores e níveis selecionados para o experimento, limitando a velocidade, aumentando o delay e gerando uma perda de pacotes à transmissão de dados na rede.

4.4 Linguagem de programação e algoritmo de envio

Para enviar o conjunto de dados utilizando os 3 protocolos escolhidos foi necessário a criação de um script para organizar e transmitir os dados de forma sistemática, fazendo uso das API's de cada protocolo. Como forma de evitar interferência por conta da linguagem de programação, todos os script foram desenvolvidos utilizando a linguagem Python. Para o protocolo HTTP, foi feito uso da biblioteca padrão que vem na versão 3.x do Python, tanto para cliente, chamada http.client, quanto para o servidor, chamada http.server. No protocolo MQTT foi utilizado a biblioteca Paho MQTT na versão 1.4.0. Por fim o script do CoAP fez uso da biblioteca CoAPthon na versão 4.0.2.

O script para o envio dos dados foi o mesmo para todos os protocolos, variando apenas a instanciação do cliente de cada protocolo e estão disponíveis neste link. O código a seguir, utilizado para o protocolo HTTP, demonstra como foi realizado o experimento tanto para este, como para os demais protocolos. O script basicamente consistiu primeiramente em identificar e estabelecer uma conexão com o servidor de aplicação enviando uma requisição de teste e esperando até receber uma confirmação de recebimento do servidor. Em seguida, é iniciada a transmissão dos dados de envio. A transmissão é feita de forma sequencial e ordenada, de registro em registro, com um intervalo de 1 segundo entre eles até que todos os objetos tenham sido enviados. O intervalo entre os envios só começa a ser contabilizado a partir do momento em que o servidor retorna uma mensagem de confirmação do recebimento do pacote anterior, exceto se esta confirmação demorar mais que 10 segundo, daí então é cancelado o envio do pacote e é feito uma nova tentativa de envio. Foi fixado um limite máximo de 3 tentativas de envio por pacote, caso não a confirmação de recebimento não ocorra até a 3ª tentativa, a transmissão daquele pacote é abortada e o fluxo segue para o próximo registro.

```
1  class Client():
2      _dados = Data()
3      conn = ''
4      startExperimentTS = ''
5      lostPkgs = []
6      delayPkgs = []
7      address = ""
8      port = ""
9      TIMEOUT = 10
10     MAX_SEND_ATTEMPT_NUMBER = 3
11
12     def timeout_handler(self, num, stack):
13         print("\n!! Timeout nível de aplicação !!")
14         raise Exception("timeout-aplicação")
```

```
15
16      def establishConnection(self, address='localhost', port=8080):
17          self.address = address
18          self.port = port
19          print('\nEstabelecendo conexão com ' + address + '...')
20          self.conn = http.client.HTTPConnection(self.address,
21              self.port)
22          self.conn.request("HEAD", "/")
23          res = self.conn.getresponse()
24          if res.status == 200:
25              print('conexão estebelecida!')
26
27      def reestablishConnection(self):
28          try:
29              print("Reestabelecendo conexão")
30              self.conn.close()
31              self.conn = http.client.HTTPConnection(self.address,
32                  self.port)
33              self.conn.connect()
34          except Exception as ex:
35              print(ex)
36              self.reestablishConnection()
37
38      def startExperiment(self, reps=-1, fileName,
39          timePerRep=1):
40
41          print("\nIniciando o experimento às {0}".format(
42              util.getFormattedDatetimeWithMillisec()))
43
44          if (reps <= 0):
45              reps = self._dados.length()
46          print("Quantidade de pacotes que serão enviados: {}".format(reps))
47          duracao = timePerRep * reps
48          print("Tempo estimado de duração do experimento: {} ({})\n".format(
49              util.getFormattedDateTimeFromSeconds(duracao),
50              str(datetime.timedelta(seconds=duracao))))
51
52          for i in range(0, reps):
53              print("iniciando repetição {} às {}".format(
54                  i + 1, str(util.getFormattedDatetimeWithMillisec())))
55              self.sendPackage(i)
56              time.sleep(timePerRep)
```

```
57          self.generateDelayAndLostFiles(fileName)
58
59      def generateDelayAndLostFiles(self, fileName):
60          if (len(self.delayPkgs) > 0):
61              fileDelay = open(fileName + "-delay.txt", 'w')
62              for i in self.delayPkgs:
63                  fileDelay.write("{},{},{}\n".format(i[0], i[1], i[2]))
64
65          if len(self.lostPkgs) > 0:
66              fileLost = open(fileName + "-perda.txt", 'w')
67              for i in self.lostPkgs:
68                  fileLost.write("{}\n".format(i))
69
70      def sendPackage(self, index):
71          print(TAB_1 + "Obtendo os dados p/ envio ...")
72          dados = self._dados.getByIndex(index)
73          sentPkgCount = 0
74          self.reestablishConnection()
75          while sentPkgCount < self.MAX_SEND_ATTEMPT_NUMBER:
76              sentPkgTimestamp = time.time()
77              signal.signal(signal.SIGALRM, self.timeout_handler)
78              signal.alarm(self.TIMEOUT)
79              try:
80                  #id = uuid.uuid4().time_mid
81                  id = "{}.{}".format(str(index + 1), str(sentPkgCount))
82                  headers = {
83                      'Content-type': 'application/json',
84                      'X-Timestamp': str(sentPkgTimestamp),
85                      'id': id,
86                  }
87                  sentPkgTimestamp = time.time()
88                  print(TAB_1 + "Enviando pacote...")
89                  self.conn.request("POST", "/markdown", dados, headers)
90                  print(TAB_1 + "Obtendo resposta...")
91                  response = self.conn.getresponse()
92                  print(TAB_1 + "Pacote id {} enviado: {} {}".format(
93                      id, response.status, response.reason))
94                  responseTimestamp = time.time()
95                  self.delayPkgs.append((id, sentPkgTimestamp,
96                                         responseTimestamp))
97                  sentPkgCount = self.MAX_SEND_ATTEMPT_NUMBER
98
```

```
99              except Exception as e:
100                 self.lostPkgs.append(sentPkgTimestamp)
101                 print("!! Pacote dropado !! - Erro: {}!".format(str(e)))
102                 self.reestablishConnection()
103                 sentPkgCount += 1
104             finally:
105                 signal.alarm(0)
```

4.5 Arquitetura e design do experimento

A arquitetura final do experimento consistiu nas duas placas Raspberry Pi e no notebook conectados via cabos ethernet a um switch. Devido a todos os equipamentos estarem conectados ao mesmo switch, o sistema operacional de cada um deles automaticamente cria uma rota de rede para todos os outros. Para evitar que o fluxo de dados seguisse a uma rota incorreta, enviando os dados direto da placa de envio para o notebook sem passar pela placa roteadora, onde os fatores e níveis do experimento foram implementados, foram criados dois script com objetivo definir a rota correta do fluxo dos dados. O primeiro foi executado na placa de envio, com a finalidade excluir o IP do notebook da sua lista de rotas e direcionar os dados enviados para placa roteadora. O segundo foi executado na placa roteadora, com o objetivo de receber os dados e encaminhá-los ao servidor de aplicação.

O design de experimento escolhido para a realização dos experimentos destes trabalho foi o *General Full Factorial Design with k Factors* (JAIN, 1990), onde a quantidade de experimentos realizados é igual ao produto entre a quantidade de fatores analisados pela quantidade de níveis de cada fator. Aplicando esta formula ao experimento proposto tem-se: 3 protocolos analisados x 3 fatores escolhidos x 3 níveis para cada fator, totalizando 27 experimentos.

Para que pudesse ser feita a análise do quanto cada nível de cada fator impactou em cada experimento, foi necessário capturar os metadados de todos os pacotes enviados pela Raspeberry Pi e recebidos pelo notebook durante a realização de cada experimento. A ferramenta TCPDump foi utilizada tanto no notebook quanto na placa de envio para realizar a captura e armazenamento dos pacotes trafegados a nível de rede. Antes de cada experimento era executado o script como o inserido abaixo para iniciar a captura de pacotes pelo TCPDump

```
1
2 $ sudo tcpdump -w coap-factor-l1-p1.pcap -i enp9s0 udp and dst 10.42.0.1 and
      port 8080
```

O comando *sudo* é para executar o restante do comando como administrador do sistema. O *tcpdump* serve para informar que será iniciado o uso da ferramenta. O *-w* serve para informar que os pacotes capturados devem ser salvos em um arquivo, cujo nome é informado logo em seguida pelo *coap-factor-l1-p1.pcap*. O *-i* informa qual a interface de rede a captura deve ser efetuada, seguido pelo *enp9s0*, que é a interface da rede cabeada. Posteriormente é selecionado o protocolo de rede, que no exemplo acima é o UDP, e por fim é informado o endereço e a porta de destino dos pacotes pelo *dst 10.42.0.1 and port 8080*. Dependendo do tipo de protocolo do experimento em questão, o protocolo de rede selecionado e a porta destino poderiam variar.

4.6 Execução do experimento e coleta dos meta-dados

A condução dos experimentos procedeu da seguinte forma: em primeiro lugar a arquitetura do experimento foi montada, em seguida foram executados os scripts para definir a rota do fluxo de dados, logo após foi escolhido os níveis de cada fator daquele experimento no script do NETEM na placa roteadora. A seguir a rota do fluxo foi testada enviando um único pacote da placa de envio para o servidor de aplicação para garantir que o fluxo de dados seguisse o caminho correto. Posteriormente foram executados os scripts de captura de dados tanto no notebook quanto na placa de envio. Por fim foi executado o código elaborado para realizar o envio dos dados utilizando o protocolo escolhido.

Tendo um total de 9168 objetos a serem enviados e um intervalo de 1 segundo entre os envios, o melhor dos casos resultou num tempo de execução total aproximado de 2 horas e 32 minutos. Com a finalidade de manter uma padronização entre os experimentos, foi definido que uma repetição equivaleu a um intervalo de 5 minutos dentro da duração de cada experimento. Desse modo, dentre todos os experimentos, o mais curto foi um que utilizou o protocolo CoAP, uma latência de 10 ms e perda de pacotes em 0%, gerando um total de 32 repetições, e o mais longo foi um que utilizou o protocolo HTTP, uma latência de 400 ms e perda de pacotes em 25%, produzindo um total de 195 repetições.

Após a realização dos 27 experimentos, foi realizada uma pré-análise dos dois arquivos gerados pelo TCPDump, um na placa e um no notebook, para cada experimento, com objetivo de agrupar e sintetizar os dados do respectivo experimento em uma única coleção, identificando, por exemplo, a quantidade de repetições do experimento, a quantidade de pacotes enviados e recebidos do experimento como um todo e de cada repetição, quantidade de pacotes perdidos e o atraso médio dos pacotes em cada repetição, entre outras métricas. A partir dos dados gerados da pré-análise, foi possível obter um arquivo CSV para cada experimento realizado, arquivos estes que

serviram de matéria-prima para a análise comparativa entre os protocolos.

4.7 Análise comparativa

Para a análise comparativa, foi utilizado a linguagem R, versão 3.2.3, juntamente com a IDE RStudio, versão 1.1.453.

De forma a padronizar a análise dos experimentos, tendo por base que a duração mais curta fez uso de apenas 32 repetições, foram consideradas apenas 30 repetições de cada experimento, sendo descartada a primeira repetição e selecionando da segunda à trigésima primeira.

A métrica escolhida para análise foi a taxa média do tamanho do pacote por segundo. A mesma foi calculada por repetição, a partir da soma do tamanho de todos os pacote presentes na repetição dividido pela quantidade de segundos definida para cada repetição, que assumiu o valor constante de 300. Para o tamanho do pacote, foi considerado o valor do campo *total length* presente no cabeçalho do protocolo da camada de transporte IP (POSTEL et al., 1981).

Foi projetado o script abaixo para unificar, em um único arquivo CSV, todos os dados coletados de todos os experimentos a partir dos arquivos concebidos na pré-análise. Este script primeiramente define uma lista do tipo chave-valor dos protocolos analisados e dos fatores e níveis escolhidos. Daí então é percorrido cada uma destas listas e, para cada iteração, o arquivo gerado na pré-análise daquele protocolo, fator e nível do experimento daquela iteração é importado e as 30 repetições acimas referidas são extraídas. Para cada uma destas repetições é gerada uma string no estilo CSV com o tipo do protocolo e valor do nível de cada fator, juntamente com o valor da métrica escolhida, reproduzindo o seguinte exemplo: ”HTTP,400,25,533.1100”. Cada uma destas strings é então armazenada em uma lista de resumo e, ao final de todas as iterações, esta lista é gravada em um arquivo CSV.

```
1
2     protocolos = {'http': "HTTP", 'coap': "COAP", 'mqtt': "MQTT"}
3     latencias = {'l1': 400, 'l2': 100, 'l3': 10}
4     perdas = {'p1': 25, 'p2': 15, 'p3': 0}
5     resumo = []
6     repeticoesConsiderar = 30
7
8     for protocolo in protocolos.keys():
9         for latencia in latencias.keys():
10            for perda in perdas.keys():
11                nomeArquivo = '{}_factor_{}_{}_v3_result.csv'.format(
```

```
12                  protocolo, latencia, perda)
13              caminhoArquivo = "Resultados/{}/analise/{}".format(
14                  protocolo, nomeArquivo)
15              file = open(caminhoArquivo, 'r')
16              linhas = file.readlines()
17              for index in range(2, repeticoesConsiderar):
18                  taxa = linhas[index].split(',')[3]
19                  resumoValue = "{},{},{},{}".format(
20                      taxa, latencias.get(latencia), perdas.get(perda),
21                      protocolos.get(protocolo))
22                  print(resumoValue)
23                  resumo.append(resumoValue)
24  resumoFileName = "./Resultados/resumo-analises.csv"
25  try:
26      os.remove(resumoFileName)
27  except OSError:
28      pass
29  resumoFile = open(resumoFileName, 'w')
30  resumoFile.write("Taxa,Lat,Perd,Proto")
31  for resumoLine in resumo:
32      resumoFile.write('\n' + resumoLine)
33  resumoFile.close()
```

De posse do arquivo final com o resumo de todas as repetições de todos os experimentos, este foi então importando utilizando o RStudio para enfim ser realizada a análise comparativa.

5 Resultados

Este capítulo tem por finalidade discorrer sobre os resultados obtidos a partir da análise comparativa dos protocolos estudados no presente trabalho.

5.1 Análise gráfica

Na Figura 5 pode-se observar que os diagramas de caixa dos protocolos baseados em TCP apresentam uma variabilidade maior que o protocolo baseado em UDP a medida que a rede varia. Isto ocorre por conta do controle de congestionamento e retransmissão do protocolo TCP. Ainda sobre a Figura 5 pode-se concluir também que o protocolo HTTP é o que apresenta maior variabilidade na taxa dentre todos os protocolos estudados.

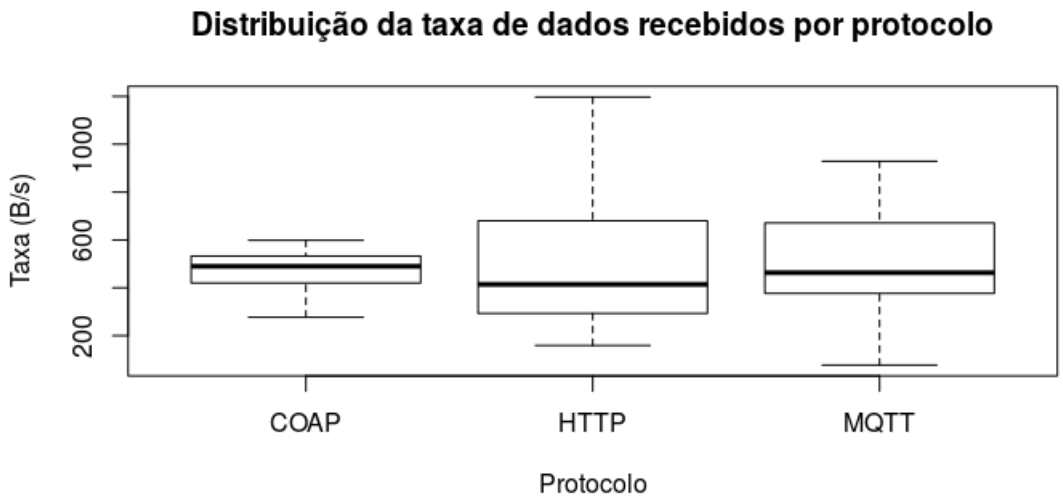

Figura 5 – Distribuição da taxa de dados recebidos (em bytes por segundo) em todos os protocolos.

Na Figura 6 é possível deduzir que a variabilidade da taxa apresentada quando a perda de pacotes encontra-se em 0% é muito alta. Também é possível observar um decaimento da taxa a medida que a perda de pacotes aumenta e que o terceiro quartil em 25% ficou próximo do primeiro quartil em 0%

Na Figura 7 ainda é possível observar uma variabilidade grande quando a latência está no melhor nível analisado, no entanto esta é menor que a variabilidade apresentada no melhor nível da perda de pacotes, observado na Figura 6. Ainda sobre a Figura 7, torna-se notável o decaimento da taxa, acentuado-se no nível 400 ms

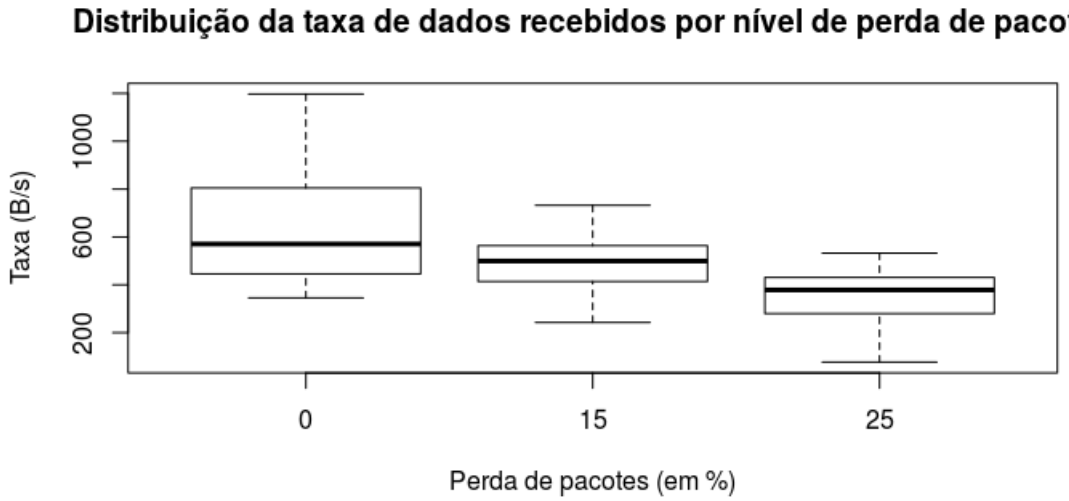

Figura 6 – Distribuição da taxa de dados recebidos (em bytes por segundo) em todos os níveis de perda de pacotes.

de latência, onde seu terceiro quartil encontra-se abaixo primeiro quartil da latência em 10 ms e seu limite superior está próximo a mediana da latência em 10 ms.

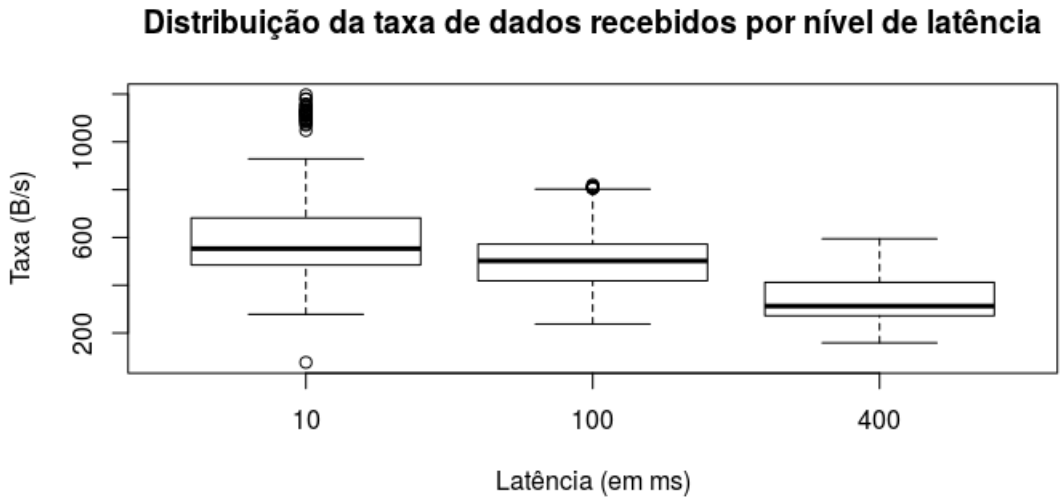

Figura 7 – Distribuição da taxa de dados recebidos (em bytes por segundo) em todos os níveis de latência.

Na figura 8 mostra que taxa decresce para todos os níveis de latência estudados a medida que a perda de pacotes aumenta. A taxa em 0% também apresenta uma grande variabilidade, corroborando com o resultado apresentado na Figura 6.

Na figura 9 é possível notar o declínio da taxa no aumento da perda de pacotes. É válido notar que a variabilidade na taxa apresentada pelos protocolos quando a perda encontra-se em 0% deve-se principalmente por conta do tamanho do cabeçalho de

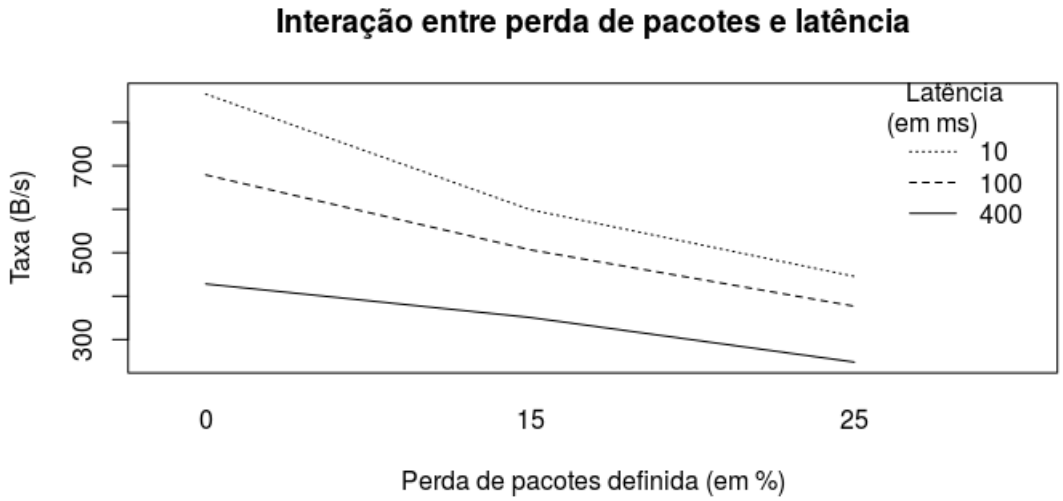

Figura 8 – Interação entre as perdas de pacotes (em porcentagem) e as latências (em milissegundos) para todos os protocolos.

cada protocolo. A isso deve-se também o fato do protocolo HTTP ter apresentado uma taxa mais alta que a dos outros protocolos analisados.

Ainda sobre a Figura 9, torna-se importante ressaltar que a sensibilidade dos protocolos à perda não é a mesma. Os protocolos que se baseiam em TCP são mais sensíveis às diferenças na perda da pacotes que os protocolos baseados em UDP. Ademais, pode-se observar que em cenários que apresentam perdas de pacotes o MQTT mantém uma taxa melhor que o HTTP, e o CoAP mostra uma taxa melhor que ambos.

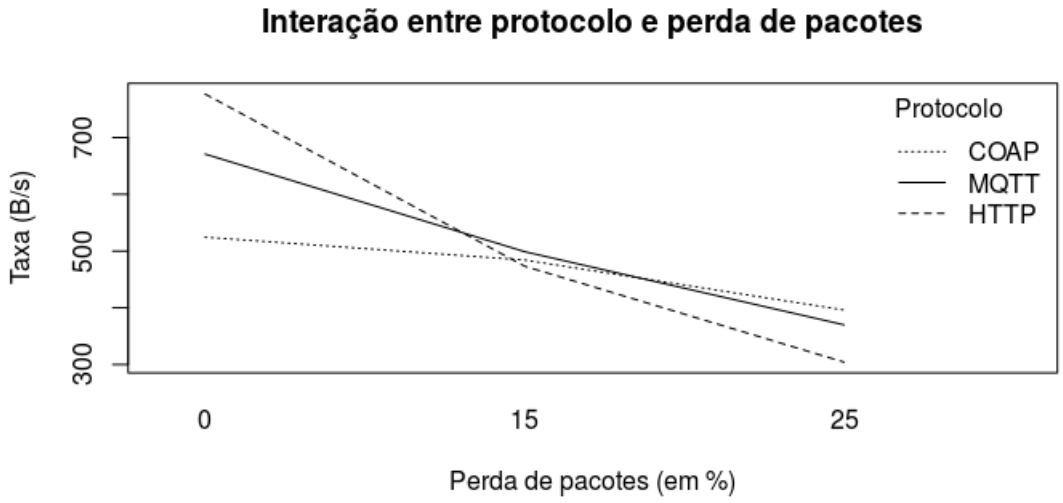

Figura 9 – Interação entre os protocolos e as perdas de pacotes (em porcentagem) para todas as latências.

Na Figura 10 constata-se o mesmo que na Figura 10, com o adendo que nos cenários em que a latência atinge o valor de 400 ms, o CoAP apresenta uma taxa significativamente maior que os protocolos MQTT e CoAP, e que, apesar do MQTT superar o HTTP em cenários com latência acentuada, ambos mantém uma taxa muito próxima.

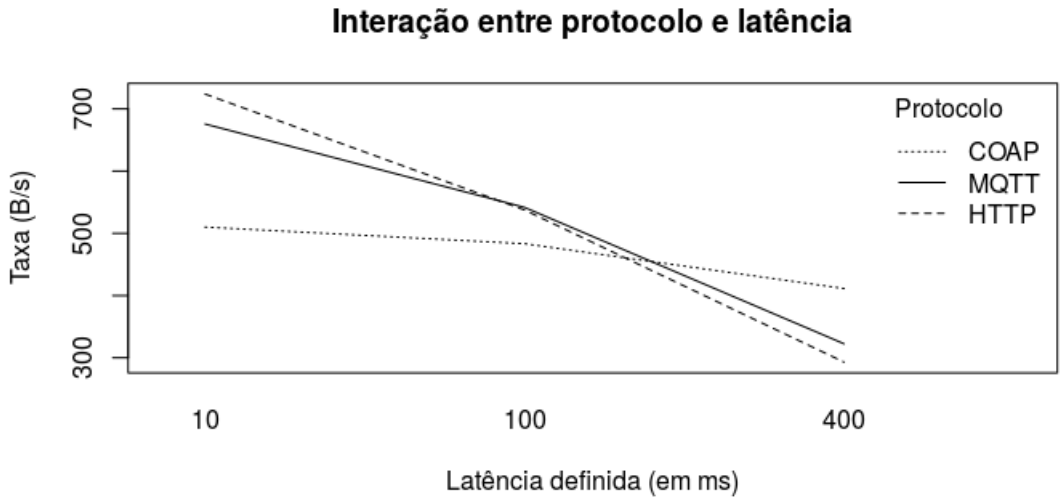

Figura 10 – Interação entre os protocolos e as latências (em milissegundos) para todas as perdas de pacotes.

5.2 Análise de variância

Ao aplicar o método de Análise de Variância sobre os resultados obtidos, chegou-se ao seguinte resultado apresentado na Tabela 1. A partir da análise desta tabela, pode-se inferir que a contribuição dos fatores escolhidos para o experimento, perda de pacotes e latência, na taxa observada foi de aproximadamente 70%. Também pode-se concluir que, como os valores obtidos para o Valor P são menores que 0,5 (ou 5%), existe uma diferença estatisticamente significativa dos fatores escolhidos no experimento para a taxa analisada.

Fator	**Grau de liberdade**	**Soma dos quadrados**	**Contribuição**	**Valor P**
Latência	2	11.860.080	34,61 %	< 2e-16
Perda de pacotes	2	12.278.044	35,83 %	< 2e-16
Protocolo	2	408.068	1,19 %	6.74e-08
Resíduo	803	9.719.168	28,36 %	

Tabela 1 – Resultados obtidos através da Análise de Variância.

6 Conclusão

O presente trabalho se propôs a avaliar e comparar a eficiência dos protocolos de comunicação de redes HTTP, CoAP e MQTT de uma maneira sistemática num ambiente controlado utilizando dispositivos de Internet das Coisas.

De uma maneira geral, os resultados obtidos mostraram que, para cenários em que a perda de pacotes se mantenha em torno de 25%, a escolha do protocolo CoAP é a mais indicada para a taxa analisada. Também é possível afirmar que o protocolo HTTP é mais impactado pela variação dos fatores escolhidos, tornando-o menos indicado para uso em dispositivos IoT. Entre os protocolos que fazem uso do TCP, o MQTT apresentou melhores taxas que o HTTP nos cenários de precariedade da rede.

Sobre os fatores e níveis analisados, observou-se que a perda de pacotes foi o fator que mais influenciou a taxa, independente de qual protocolo foi utilizado. A partir dos resultados da análise de variância, pode-se concluir que a escolha do protocolo tem um impacto menos relevante na taxa do que às condições da rede;

6.1 Trabalhos futuros

Para trabalhos futuros é sugerida a repetição do mesmo experimento empregando uma quantidade maior de protocolos, cujas arquiteturas ou diferem dos estudados no presente trabalho ou as implementam de uma forma diferente, tais como o Advanced Message Queuing Protocol (AMQP), que implementa um sistema de mensageiria utilizando o padrão publicação-assinatura semelhante ao MQTT, o Extensible Messaging and Presence Protocol (XMPP), que é um protocolo construido tendo por base o padrão XML e que suporta tanto a arquitetura publicação-assinatura quanto a requisição-resposta, o Streaming Text Orientated Messaging Protocol (STOMP) que também implementa o padrão publicação-assinatura porém de uma forma simplificada utilizando os princípios derivados do HTTP, e o Data Distribution Service (DDS) que faz uso do padrão publicação-assinatura valendo-se do protocolo UDP na camada de transporte.

Por fim, também é proposto a avaliação de diferentes métricas, tais quais atraso no envio dos pacotes e taxa de perda de pacotes por protocolo, além da escolha de níveis mais variados para os fatores selecionados, objetivando desse modo encontrar o protocolo mais adequado para o uso nos dispositivos de Internet das Coisas.

Referências

BANDYOPADHYAY, S.; BHATTACHARYYA, A. Lightweight internet protocols for web enablement of sensors using constrained gateway devices. In: IEEE. *Computing, Networking and Communications (ICNC), 2013 International Conference on*. [S.l.], 2013. p. 334–340. Citado na página 24.

BANKS, A.; GUPTA, R. Mqtt version 3.1. 1. *OASIS standard*, v. 29, 2014. Citado na página 21.

CARO, N. D. et al. Comparison of two lightweight protocols for smartphone-based sensing. In: IEEE. *Communications and Vehicular Technology in the Benelux (SCVT), 2013 IEEE 20th Symposium on*. [S.l.], 2013. p. 1–6. Citado na página 23.

CASTELLANI, A. P. et al. Web services for the internet of things through coap and exi. In: IEEE. *2011 IEEE International Conference on Communications Workshops (ICC)*. [S.l.], 2011. p. 1–6. Citado na página 18.

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. Citado 2 vezes nas páginas 23 e 27.

EKO, S. C.; PASTIMA, S. et al. Quality of service of gsm. a comparative internet access analysis of provider in batam. *International Journal of Open Information Technologies*, Лаборатория Открытых Информационных Технологий факультета ВМК МГУ им. МВ …, v. 5, n. 6, 2017. Citado na página 27.

FIELDING, R. et al. Hypertext transfer protocol–http/1.1, 1999. *RFC2616*, 2006. Citado na página 16.

GLÓRIA, A.; CERCAS, F.; SOUTO, N. Comparison of communication protocols for low cost internet of things devices. In: IEEE. *Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), 2017 South Eastern European*. [S.l.], 2017. p. 1–6. Citado na página 24.

GOURLEY, D. et al. *HTTP: the definitive guide*. [S.l.]: ”O’Reilly Media, Inc.”, 2002. Citado 4 vezes nas páginas 14, 15, 16 e 17.

HOHPE, G.; WOOLF, B. et al. Enterprise integration patterns. Addison-Wesley Reading, 2003. Citado 2 vezes nas páginas 13 e 14.

JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. [S.l.]: John Wiley & Sons, 1990. Citado na página 32.

KOVATSCH, M.; LANTER, M.; SHELBY, Z. Californium: Scalable cloud services for the internet of things with coap. In: IEEE. *2014 International Conference on the Internet of Things (IOT)*. [S.l.], 2014. p. 1–6. Citado 2 vezes nas páginas 18 e 19.

LI, G.; JACOBSEN, H.-A. Composite subscriptions in content-based publish/subscribe systems. In: SPRINGER. *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. [S.l.], 2005. p. 249–269. Citado na página 15.

LUETH, K. L. *State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating*. 2018. <https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/>. [Online; Acesso em: 07 de Novembro de 2018]. Citado na página 10.

MANANDHAR, S. Mqtt based communication in iot. 2017. Citado 2 vezes nas páginas 21 e 22.

POSTEL, J. et al. Rfc 791: Internet protocol. September, 1981. Citado na página 34.

SANTOS, B. P. et al. Internet das coisas: da teoriaa prática. *Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos*, 2016. Citado na página 10.

SHELBY, Z.; HARTKE, K.; BORMANN, C. The constrained application protocol (coap). 2014. Citado 2 vezes nas páginas 18 e 19.

STANFORD-CLARK, A.; TRUONG, H. L. Mqtt for sensor networks (mqtt-sn) protocol specification. *International business machines (IBM) Corporation version*, v. 1, 2013. Citado na página 20.

SUNDMAEKER, H. et al. Vision and challenges for realising the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commision*, v. 3, n. 3, p. 34–36, 2010. Citado na página 10.

TANG, K. et al. Design and implementation of push notification system based on the mqtt protocol. In: ATLANTIS PRESS. *2013 International Conference on Information Science and Computer Applications (ISCA 2013)*. [S.l.], 2013. Citado 3 vezes nas páginas 20, 21 e 22.

THANGAVEL, D. et al. 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. Citado na página 24.