Universidade do Estado do Rio de Janeiro

Centro de Tecnologia e Ciências

Instituto de Matemática e Estatística

Roberto da Silva Macedo

Um estudo e avaliação do desempenho de Protocolos de Aplicação para a Internet das Coisas

Rio de Janeiro

2019

Roberto da Silva Macedo

Um estudo e avaliação do desempenho de Protocolos de Aplicação para a Internet das Coisas

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Orientador: Prof. Dr. Alexandre Sztajnberg

Rio de Janeiro

2019

CATALOGAÇÃO NA FONTE

UERJ/REDE SIRIUS/BIBLIOTECA CTC/A

M141 Macedo, Roberto da Silva.
Um estudo e avaliação do desempenho de protocolos de aplicação para a internet das coisas/Roberto da Silva Macedo. – 2019.
130 f.: il.

Orientador: Alexandre Sztajnberg
Dissertação (Mestrado em Ciências Computacionais) - Universidade do Estado do Rio de Janeiro, Instituto de Matemática e Estatística.

1. Internet das coisas - Teses. 2. Internet - Medidas de segurança - Teses. 3. Proteção de dados - Teses. I. Sztajnberg, Alexandre. II. Universidade do Estado do Rio de Janeiro. Instituto de Matemática e Estatística. III.Título.

CDU 004.738.5

Patricia Bello Meijinhos CRB7/5217 - Bibliotecária responsável pela elaboração da ficha catalográfica

_____ _____

Assinatura Data

Roberto da Silva Macedo

Um estudo e avaliação do desempenho de Protocolos de Aplicação para a Internet das Coisas

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Ciências, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Aprovado em: 22 de Outubro de 2019

Banca Examinadora:

Prof. Dr. Alexandre Sztajnberg (Orientador)

Instituto de Matemática e Estatística - UERJ

Prof. Dr. Francisco Sant'Anna

Instituto de Matemática e Estatística - UERJ

Prof. Dr. Célio Vinícius Neves de Albuquerque

Universidade Federal Fluminense

Rio de Janeiro

2019

AGRADECIMENTO

Agradeço ao meu orientador, professor Alexandre Sztajnberg pela paciência e como conduziu a orientação em minha pesquisa. Durante este percurso foi quem cobrou qualidade e dedicação, o que me fez crescer como pessoa e profissional. Agradeço também por me proporcionar a experiência de preparar e apresentar minicurso no 37º JAI do CSBC de 2018. Foi um período de aprendizado, persistência e dedicação no qual levarei para o resto de minha vida.

Agradeço também a VALID S.A pelo apoio e tempo flexível disponibilizado para a realização das minhas atividades de pesquisa.

Agradeço a minha família e em especial a minha esposa Luciana, pelo apoio incondicional, principalmente nos meus momentos de ausência.

A Deus, o que seria de mim sem a fé que eu tenho nele.

"Talvez não tenha conseguido fazer o melhor, mas lutei para que o melhor fosse feito. Não sou o que deveria ser, mas Graças a Deus, não sou o que era antes".
(Marthin Luther King)

Roberto da Silva Macedo

RESUMO

MACEDO, Roberto da Silva. *Um estudo e avaliação do desempenho de Protocolos de Aplicação para a Internet das Coisas*. 2019. 130f. Dissertação (Mestrado em Ciências Computacionais) - Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, 2019.

A Internet das Coisas promete facilitar a interação automática e o acesso à dispositivos industriais, de automação e domésticos, como sensores e atuadores. Um dos desafios para o desenvolvimento da IoT é o número crescente e a heterogeneidade de dispositivos. *Frameworks* de software propostos para IoT, tais como o FIWARE [1], DOJOT [2], SOFIA [3], OpenIoT [4], geralmente contemplam serviços de suporte, e integração de dispositivos, com módulos-cliente das aplicações. Estes *frameworks* apresentam uma camada de Aplicação, que inclui Protocolos de Aplicação, orientados à interação entre processos e dispositivos. Vários Protocolos de Aplicação estão estabelecidos e alguns padronizados, com destaque para o HTTP [5], MQTT [6], CoAP [7], AMQP [8], XMPP [9] e o WebSocket [10]. Neste trabalho é avaliado o desempenho de cada um destes protocolos em uma rede Wi-Fi. Para os testes de todos os protocolos selecionados, quatro "dimensões" foram combinadas: os 6 protocolos; 1 métrica quantitativa (*Round Trip Time*); 3 métricas de escalabilidade (tamanho de mensagens, número de sensores e largura de banda disponíveis) e 2 métricas qualitativas (confiabilidade / QoS) e uso sobre a camada de segurança. Os testes foram realizados em um ambiente isolado. Aplicações de teste foram desenvolvidas para envio e recebimento de mensagens. Com isto, também foi possível se verificar a viabilidade de uso de algumas bibliotecasdisponíveis para os dispositivos NodeMCU e aparelhos móveis Android. Os resultados dos testes de desempenho são apresentados e discutidos. Estes resultados permitem identificar as diferenças de desempenho e de características de operação entre os Protocolos de Aplicação que precisam ser considerados de acordo com a aplicação quer irá utilizá-los. Alguns cenários de aplicação são associados aos parâmetros avaliados de tamanho das mensagens, banda e número de dispositivos utilizados.

Palavras-chave: Internet das Coisas. Protocolos de Aplicação. Middleware. HTTP. MQTT. WebSocket. CoAP. AMQP. XMPP.

ABSTRACT

MACEDO, Roberto da Silva. *A study and performance evaluation of Application Protocols for the Internet of Things*. 2019. 130f. Dissertação (Mestrado em Ciências Computacionais) - Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, 2019.

The Internet of Things aims to facilitate automatic interaction and access to industrial, automation and home devices such as sensors and actuators. One of the challenges for IoT development is the growing number and heterogeneity of devices. Proposed software frameworks for IoT, such as FIWARE [1], DOJOT [2], SOFIA [3], OpenIoT [4], usually include support services, and device integration with client application modules. These frameworks feature an Application layer, which includes Application Protocols, oriented to the interaction between processes and devices. Several Application Protocols are established and some standardized, with emphasis on HTTP [5], MQTT [6], CoAP [7], AMQP [8], XMPP [9], and WebSocket [10]. This work evaluates the performance of each of those protocols in a Wi-Fi network. For testing of all selected protocols, four "dimensions" were combined: the 6 protocols; 1 quantitative metric (*Round Trip Time*); 3 scalability metrics (message size, number of sensors and bandwidth available) and 2 qualitative metrics (reliability/QoS) and security layer usage. The tests were performed in an isolated environment. Test applications were developed for sending and receiving messages. This also made it possible to verify the feasibility and viability of using some libraries available for NodeMCU devices and Android mobile devices. The results of the performance tests are presented and discussed. Those results allow us to identify the differences in performance and operating characteristics between the Application Protocols that need to be considered according to the application and will use them. Some application scenarios are associated with evaluated parameters of message size, bandwidth and number of devices used.

Keywords: Internet of Things. Application Protocols. Middleware. HTTP. MQTT. WebSocket. CoAP. AMQP. XMPP.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
CoAP	Constrained Application Protocol
AMQP	Advanced Message Queuing Protocol
MQTT	Message Queuing Telemetry Transport
XMPP	Extensible Messaging and Presence Protocol
IoT	Internet of Things
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
MIT	Massachussetts Institute of Technology
OMG	Object Management Group
W3C	World Wide Web Consortium
ISO	International Standards Organization
ITU	International Telecommunication Union
EPC	Eletronic Product Code Network
NAT	Network Address Translation
RFID	Radio-Frequency IDentification
SOA	Service Oriented Architecture
SO	Sistema Operacional
TCP	Transport Control Protocol
SSL	Secure Socket Layer
JSON	JavaScript Object Notation
SOAP	Simple Object Access Protocol
MTU	Maximum Transmission Unit
IDE	Interface Development Environment
LCC	Laboratório de Ciência da Computação
UDP	User Datagram Protocol

SUMÁRIO

INTRODUÇÃO

A Internet das Coisas (*Internet of Things*, IoT) promete facilitar a interação automática e o acesso a dispositivos industriais, de automação e domésticos, como câmeras de vigilância, dispositivos sensores, atuadores ou monitores. Isso permitirá o desenvolvimento de aplicações que podem explorar uma quantidade enorme de, pequenos blocos de dados gerados por esses dispositivos [11], [12]. Estas aplicações podem surgir de domínios como automação industrial e residencial, gestão inteligente de ambientes, saúde e cidades inteligentes ([13], [11], [14], [15], [16]).

Um dos desafios para o desenvolvimento da IoT é o número crescente e a heterogeneidade de dispositivos. Em 2010, o número de dispositivos conectados à Internet já estava na ordem de 10^9 a e espera-se que ele alcance 10^{11} até 2020 [17].

A diversidade de tecnologias usadas na IoT requer o suporte de serviços de *middleware* e estruturas de desenvolvimento com abstrações bem definidas. O *middleware* pode abstrair a complexidade tecnológica e um *framework* pode agilizar o desenvolvimento das aplicações.

Por exemplo, é possível usar uma abordagem orientada a serviços, considerando os recursos dos dispositivos como serviços de software. Isso pode facilitar a integração destes dispositivos e incorporar padrões de interação conhecidos, como *register-lookup-use* [18]. No entanto, as técnicas usadas para encapsular dispositivos em artefatos de software precisam lidar com elementos típicos de dispositivos, como sensores/atuadores e atributos específicos, como localização, estado, frequência ou intervalo de medição [18]. Além disso, mais importante, alguns dispositivos podem requerer o uso de estilos de interação diferentes, *request-response* ou *publish-subscribe*, o que pode aumentar a complexidade de implementação [19], [20] [21], [22].

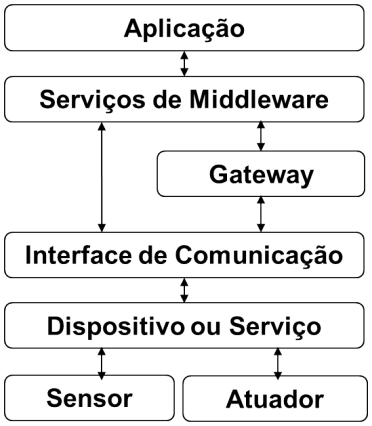
Iniciativas e soluções como FIWARE [1], SOFIA [3], OpenIoT [4], Particle [23], KNOT [24] e Eclipse IoT [25], de um modo geral, oferecem plataformas que permitem o desenvolvimento de aplicações para a IoT em alto nível de abstração. *Frameworks* propostos para IoT geralmente contemplam serviços de suporte, formas de integração e interação destes serviços, e de dispositivos, com módulos-cliente das aplicações [18], [19], [26]. Também são geralmente oferecidas opções de persistência de dados entre aplicações, serviços e dispositivos, em repositórios de dados locais ou na nuvem [12]. Com isso podem ser utilizadas técnicas inteligentes atuando sobre grandes massas de dados coletados.

Órgãos de padronização como ITU-T, IEEE e consórcios como o oneM2M e OASIS trabalham na padronização de uma arquitetura para a Internet das Coisas, em suas camadas de infraestrutura, segurança e privacidade, rede, comunicação máquina-máquina (*machine-to-machine*) e serviços para as aplicações [27], [28], [29], [30], [31], [32], [33]. Um dos objetivos é orientar a concepção de produtos e sistemas, para que sejam interoperáveis.

Aplicações desenvolvidas para utilizar sistemas de suporte para IoT podem integrar e fazer uso de diversos dispositivos, de fabricantes diferentes com recursos heterogêneos e limitações específicas. Estes dispositivos podem oferecer serviços através de interfaces de software específicas e se conectar por diferentes interfaces físicas, como USB, Bluetooth, ZigBee ou Ethernet. Mesmo em um nível mais alto de abstração, cada dispositivo pode utilizar protocolos de comunicação específicos e formatos de dados diferentes [34].

A Figura 1 - apresenta uma organização geral de elementos para a Internet das Coisas. Uma aplicação pode utilizar serviços de *middleware*, quando conveniente, para acessar serviços e dispositivos em nível alto de abstração. A comunicação entre os serviços de *middleware* com os dispositivos e serviços pode ser feita diretamente ou através de um *gateway*[1]. Neste contexto, as arquiteturas propostas para IoT incluem Protocolos de Aplicação, orientados à interação distriuída entre processos e dispositivos ou entre dispositivos. Estes protocolos podem ser utilizados tanto pelos serviços de *middleware* — que encapsulam o código necessário — como diretamente pelas aplicações — que podem fazer um uso específico dos mesmos.

Figura 1 - Organização de elementos na IoT.

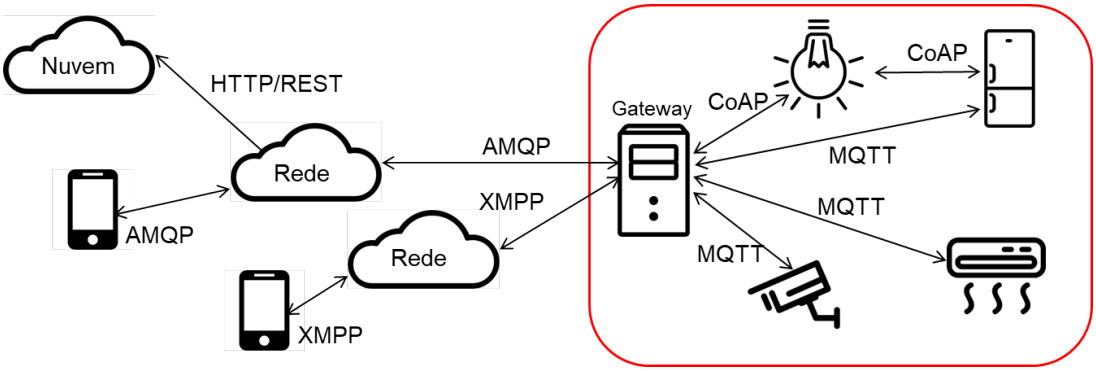


Vários Protocolos de Aplicação estão estabelecidos e alguns padronizados, com

[1] *Gateways* para IoT poderiam ser discutidos num capítulo à parte. Soluções em hardware e software podem ser consultadas em [35].

destaque para o MQTT [6], CoAP [7], AMQP [8] e XMPP [9]. Cada um tem características indicadas para determinadas aplicações em IoT, dependendo dos recursos e modos de operação dos dispositivos ou serviços a serem integrados. A Figura 2 - ilustra as possibilidades de uso de protocolos diferentes em várias situações.

Figura 2 - Aplicação com uso de múltiplos protocolos de aplicação.



Pode-se observar, por exemplo, a utilização dos protocolos CoAP e MQTT em um ambiente restrito com sensores e atuadores com poucos recursos computacionais, como geladeiras, lâmpadas, câmeras de monitoramento, ar-condicionados e outros. Saindo de um contexto local, observa-se a utilização de um *gateway* com o objetivo de permitir a interoperabilidade com outros protocolos como o AMQP, XMPP e HTTP, sendo estes mais apropriados para sincronização dos dados coletados dos sensores com banco de dados e serviços *SOA* (*Service-Oriented Architecture*) na nuvem. Estes protocolos podem ser utilizados indiretamente através de *API's* (Application Programming Interface) IoT disponibilizadas por grandes provedores como a Microsoft [36] e Amazon [37], por exemplo.

Alguns destes protocolos contemplam aspectos não-funcionais como segurança, autenticação, QoS e escalabilidade (por exemplo, [20], [38], [21]), que podem não ser integralmente aproveitados quando um *framework* é empregado. Além disso, fabricantes podem fornecer dispositivos com algum Protocolo de Aplicação embarcado ou, ainda, um novo dispositivo pode requerer justamente o provisionamento de um Protocolo de Aplicação para realizar a comunicação com o ambiente de IoT não contemplado em um *framework*. Assim, mesmo utilizando (ou até desenvolvendo) um *framework* para IoT é importante dominar os conceitos e mecanismos destes protocolos.

Trabalhos relacionados, discutidos na primeira seção do Capítulo 3, apresentam avaliações comparando alguns dos protocolos mencionados, como por exemplo, [22] e [39].

As avaliações e os cenários de uso destes trabalhos, entretanto, são limitados à dispositivos com poucos recursos computacionais, como por exemplo o Raspberry Pi e Arduino (dispositivo-dispositivo (MQTT x CoAP)), porém incompletos nos aspectos de segurança, desempenho, escalabilidade.

Dentro do contexto delineado, na primeira parte do presente trabalho foi realizado um amplo estudo sobre os principais Protocolos de Aplicação para IoT, mais especificamente o HTTP, MQTT, WebSocket, CoAP, AMQP e XMPP. O resultado deste estudo foi apresentado na forma de um mini-curso durante a Jornada de Atualização em Informática no CSBC 2018 [40].

Na segunda parte do trabalho foram avaliados o desempenho, características funcionais e não funcionais de cada protocolo, de forma a se comparar o funcionamento e os melhores cenários de uso para cada um deles. Para isso foram definidas a infraestrutura de hardware e software necessárias e desenvolvidas rotinas para cada protocolo com o objetivo de de se verificar o impacto na camada de aplicação, de configurações diferentes do tamanho de mensagens, banda disponível, número de dispositivos e a habilitação de caraterísticas de segurança. O desempenho de cada protocolo foi avaliado em relação ao tempo para uma interação de requisição e resposta (*Round Trip Time*, RTT), variando-se as configurações. Observa-se que alguns protocolos incluem mensagens de reconhecimento ou a possibilidade de transmitir mensagens criptografadas. Neste caso, avaliamos a sobrecarga destas características.

Os resultados obtidos oferecem informações importantes sobre o desempenho de cada protocolo e as possibilidades de uso. Para o MQTT e o CoAP, dado o seu uso recorrente em aplicações atuais para IoT, alguns testes complementares e comparações mais detalhadas foram realizadas. Também investigamos um pouco mais o protocolo AMQP, dado o desempenho muito distante em relação aos demais protocolos. Além disso, consolidamos as lições aprendidas no uso das bibliotecas e serviços empregados na avaliação. Por último, a partir dos resultados obtidos e das lições aprendidas, são feitas recomendações de uso para os protocolos avaliados.

Os capítulos desta dissertação estão estruturados da seguinte forma. No Capítulo 1 são resumidos os conceitos básicos e mecanismos utilizados em aplicações para Internet das Coisas. Os estilos de interação *request-response* e *publish-subscribe*, base para os protocolos avaliados, são descritos e discutidos. As tecnologias de comunicação mais uti-

lizadas, tais como *Wi-Fi* e *Bluetooth* em IoT, são também resumidas neste capítulo. No Capítulo2, são apresentados os conceitos básicos de cada Protocolo de Aplicação considerados neste trabalho, destacando aspectos específicos, que distinguam os protocolos, como, por exemplo, aspectos de QoS, diferentes esquemas de filas, facilidade para transpor redes que empregam NAT e *firewall*, registro, autenticação e mecanismos de segurança. No Capítulo 3 são apresentadas as métricas e os testes que permitiram comparar o desempenho dos Protocolos de Aplicação para IoT. Dentro deste capítulo também apresentamos as bibliotecas e sistemas de suporte utilizados, bem como as rotinas de software desenvolvidas para realizar os testes, ilustrando as diferenças e facilidades de cada um, também expondo informações que permitam verificar que os testes realizados em protocolos distintos são comparáveis entre si. O Capítulo 4 apresenta os resultados obtidos nos experimentos descritos no Capítulo 3. O desempenho medido e as características funcionais e não-funcionais elencadas no capítulo anterior são discutidas e comparadas. Destacamos também neste capítulo as lições aprendidas e fechamos o mesmo com recomendações de uso de cada protocolo. A Conclusão apresenta as últimas considerações e aponta trabalhos futuros.

1 CONCEITOS BÁSICOS E MECANISMOS UTILIZADOS

1.1 A Internet das Coisas

A Internet das Coisas (em inglês, *Internet of Things* (IoT)) é considerada como um próximo passo na evolução da Internet. Existe a expectativa de que a Internet das Coisas IoT mudará a forma como interagimos com o ambiente que nos envolve por meio de objetos físicos e virtuais que vão interagir de forma transparente e inteligente, para proporcionar mais comodidade, conforto, segurança e economia de energia.

Os conceitos e tecnologias relacionados à IoT no estágio atual são inspirados em propostas e iniciativas anteriores das quais listamos os mais relevantes:

- Em 1990, John Romkey criou o primeiro dispositivo IoT. Foi uma torradeira que era ligada e desligada pela internet.

- Em 1991, Weiser apresentou o artigo The Computer for the 21st Century, que disserta sobre o futuro do IoT, citando o termo "Computação Ubíqua". No artigo, o autor afirma que os dispositivos serão conectados em todos os lugares de forma tão transparente, que se tornará imperceptível, viabilizando de forma natural, a instalação, configuração e manutenção dos recursos computacionais [41].

- Em 1996, Venkatesh previu que as tarefas de casa, como por exemplo, a preparação de alimentos ou compras seriam automaticamente gerenciados.

- Em 1999, o termo Internet das Coisas (IoT), foi primeiramente usado em por Kevin Ashton e sua equipe no centro do Auto-ID do MIT (*Massachussetts Institute of Technology*). O termo surgiu em uma apresentação para a Procter & Gamble, com o objetivo de utilizar um sistema RFID para rastrear o produto em uma cadeia de suprimentos.

- Em 2003, o projeto foi apresentado com o nome de a rede EPC (*Eletronic Product Code Network*), uma infraestrutura aberta que permitia que computadores identificassem e seguissem objetos desde o início de uma linha de produção até a sua distribuição, através do uso do RFID [42].

- Em 2005, a Internet das Coisas começa a ganhar popularidade e o ITU (*International Telecommunication Union*) publica um relatório com o título "*The Internet of Things*". Inicia-se também a discussão por padrões internacionais.

- Em 2009, Kevin Ashton publica um artigo na revista RFID (RFID Journal), onde ratifica o significado do termo "Internet das Coisas", afirmando que se referia à adição da capacidade de produção de dados e informação nos computadores com o mínimo de interferência humana. "Nós precisamos capacitar os computadores com seus próprios meios de aquisição de informação, para que eles vejam, sintam e ouçam o mundo por si mesmos, com toda glória da aleatoriedade".

Como a Internet das Coisas ainda é uma área recente de pesquisa, várias definições são encontradas na literatura.

"A Internet das Coisas permite que as pessoas e coisas estejam conectadas a qualquer hora, em qualquer lugar, com qualquer coisa, com qualquer outra pessoa e idealmente usando qualquer rede e qualquer serviço [43]."

"Uma infraestrutura de rede global, interligando objetos físicos e virtuais por meio da exploração de captura e comunicação de dados e capacidades de comunicação. Essa infraestrutura inclui a internet existente e em evolução, bem como os desenvolvimentos de rede. Ela oferecerá identificação de objetos específica e capacidade de sensoriamento e de conexão como base para o desenvolvimento de aplicações e serviços independentes cooperativos. Estes serão caracterizados por elevado grau de captura autônoma de dados, transferência de eventos, conectividade e interoperabilidade de rede [44]."

A Comissão Europeia apresenta: "Internet e Coisa, onde *Internet* pode ser definida como a rede global de computadores interconectados baseados no protocolo de comunicação IP, enquanto *Coisa* é um objeto não necessariamente identificado. Assim, semanticamente, Internet das Coisas significa uma rede global de objetos interconectados, unicamente identificáveis baseados em um protocolo padronizado de comunicação [45]."

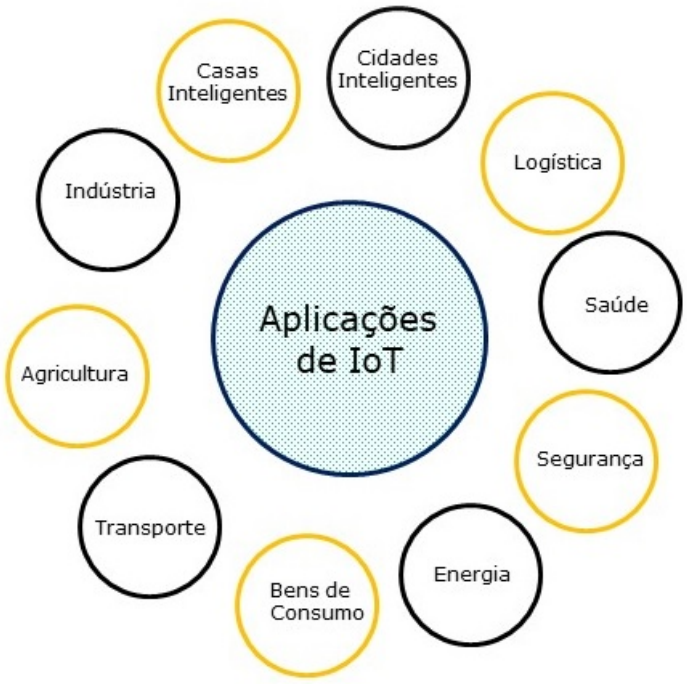
Atualmente, já podemos observar que vários equipamentos presentes em nossa vida cotidiana, possuem conexão com a Internet. Em 2010, o número de dispositivos conectados à Internet já estava na ordem de 10^9 e espera-se que ele alcance 10^{11} até 2020 [17]. Por isso, a Comissão Europeia por meio do Programa Horizon 2020, o maior programa de Pesquisa e Inovação da União Europeia (EU), tem feito investimentos consideráveis no desenvolvimento da IoT [46].

A Internet das Coisas faz com que objetos do dia a dia, se conectem à Internet. Esta conexão viabiliza o controle remoto dos objetos, além de os tornarem provedores de serviços que poderão ser compartilhados entre pessoas-pessoas, pessoas-objetos e objetos-objetos. As definições dos parágrafos anteriores estão corretas e possuem as mesmas definições, porém expostas com termos diferentes.

1.2 **Aplicações**

Atualmente, mais da metade da população mundial vive em ambientes urbanos. Em 2050, a previsão da ONU [47] é que a proporção suba para dois terços. Por isso, é fundamental cuidar para que o planeta seja um lugar sustentável e bem organizado. Neste cenário, o desenvolvimento de aplicações IoT, tem papel importante no suporte à soluções de problemas já conhecidos e no aumento da qualidade de vida das pessoas. A Figura 3 - ilustra os domínios IoT que já são explorados e estão em constante expansão.

Figura 3 - Aplicações de IoT.



- Casas Inteligentes - Algumas das motivações para o desenvolvimento de aplicações para casas inteligentes são: segurança (vigilância e monitoramento, alertas e chamadas automatizadas para autoridade policial), eficiência energética e conforto (ligar-desligar luzes de acordo com a luz do dia, abrir-fechar as cortinas de acordo com o

dia luz, ligar-desligar ou ajustar a temperatura do ar-condicionado de acordo com parâmetros previamente customizados e ligar-desligar de equipamentos automaticamente), saúde e bem-estar (contagem dos passos, batimentos cardíacos, níveis de insulina, padrões de sono), ajuda a idosos ou deficientes (utilização de recursos sonoros ou visuais para fornecer alertas, uso de dispositivos vestíveis para alertar familiares ou autoridades em caso de incidente).

- Cidades Inteligentes - motivações para o uso de IoT em uma cidade são: Gerenciamento de tráfego (indicações das melhores rotas, gerenciamento de congestionamentos e semáforos com sensores), monitoramento do consumo de energia (eficiência e transparência dos gastos públicos), monitoramento da qualidade do ar e ruído (uso de sensores para o monitoramento do ar e ruído). Nos cenários descritos se encaixam o transporte público e estacionamento, iluminação, vigilância, hospitais e escolas, manutenção de áreas públicas, preservação do patrimônio cultural, manipulação do lixo entre outras necessidades de uma cidade.

- Logística - a logística tem uma grande quantidade de atividades realizadas por automação, tais como: gerenciamento de frota (características de estado dos veículos, por exemplo, posição geográfica, se estão ligados, desligados, controle da emissão de poluentes, consumo de combustível), carga (controle do peso, quantidade, temperatura, pressão, acústica, vibração), armazéns (controle do inventário, entradas e saídas de itens), cadeia de suprimentos (controle total e em tempo real do fluxo de produção até o seu destino final). O crescimento do *E-Commerce*, tem tornado o uso de soluções IoT, não mais uma opção e sim uma necessidade.

- Saúde - muitos são os exemplos de aplicações IoT para a saúde. Alguns destes servem de motivação para uso do IoT, tais como: manter os pacientes mais conectados aos médicos por meio de monitoramento remoto e visitas virtuais, rastrear funcionários e pacientes, monitoramento e controle de exercícios físicos, alimentação saudável, automatizar o fluxo de de atendimento ao paciente, diminuir os custos de assistência médica simplificando o processo como um todo.

- Segurança - este item refere-se ao desafio de se proteger e gerenciar as informações que são trafegadas em um eco-sistema IoT. Bilhões de “coisas” conversando umas com as outras, como TVs, geladeiras, carros, medidores inteligentes, monitores de

saúde e dispositivos móveis. Além disto, os dados armazenados local ou remotamente, também precisam estar seguros, no que se refere a privacidade e perdas de informação.

- Energia - a utilização de energia tem sido intensiva e crescente, uma vez que sua adoção é essencial para o funcionamento dos mais diversos setores e atividades da sociedade. Por exemplo, a energia elétrica é responsável pelo funcionamento de grande parte dos equipamentos, sejam eles eletro-eletrônicos, automóveis e outros. Com a intensificação do uso da tecnologia pelas pessoas, o aumenta-se também o o consumo de energia. O uso de energia renovável, como por exemplo, a energia solar, também é considerada. A motivação para o uso energético em um eco-sistema IoT, é o gerenciamento sustentável, eficaz e eficiente de seu consumo.

- Bens de Consumo - os bens de consumos estão relacionados à itens que ou dispositivos que tragam experiências que satisfaçam a certos interesses pessoais ou coletivos, como por exemplo, *smart car*, *smartphone*, *smart tv*, *smart watch*, entre outros. Algumas das motivações para uso do IoT em bens de consumo são: a possibilidade de customização, automatizada ou não, de cada destes itens. Um exemplo é o ajuste automático de configurações de uma *smart tv*, ou a regulagem da temperatura do ar-condicionado, sugestões de compras a partir da localização do seu *smart phone* ou *smart watch*. Isto mostra a motivação pelo crescente uso de aplicações cientes de contexto [48] que auxiliem na tomada de decisões, seja automatizada ou por decisão final de um ser humano.

- Transporte - o transporte está relacionado à automatização e gerenciamento remoto, por exemplo do uso em caminhões, onde sua carga é monitorada remotamente (peso durante o trajeto), a rota (trajeto está dentro do esperado), consumo do combustível, recolhimento automatizado de tributos (diminuição da burocracia), entre outros itens relevantes.

- Agricultura - A internet das coisas (IoT), caracterizada pela comunicação entre máquinas (*M2M*), têm grande potencial de melhorar os processos produtivos e trazer oportunidades, tanto sob o aspecto da pesquisa agropecuária como das aplicações no campo. Há uma enorme variedade de soluções computacionais e automatizadas,

como aplicação de defensivos, irrigação, plantio e colheita conforme condições meteorológicas de cada local e sistema de gerenciamento e monitoramento da frota de tratores e outros maquinários utilizados no campo. Entretanto, interpretar essa quantidade imensa de dados para extrair informações relevantes e integrar tudo isso em soluções que garantam o aumento da qualidade da produção agrícola é um dos principais desafios para o IoT no setor de agricultura [49].

- Indústria - segundo [50], aplicações de IoT aplicados à indústria, trata de melhorar o monitoramento, controle e segurança de tudo ao nosso redor, reduzindo o risco e melhorando a confiabilidade dos processos industriais. Estamos vivendo uma nova revolução industrial (Indústria 4.0) [51], ou IIoT (*Industry Internet of Things*), onde temos – enfim – a oportunidade de mensurar o que está realmente acontecendo em uma planta, seja em assuntos relacionados aos maquinários, movimentação de ativos, produtividade de colaboradores e controle sobre os processos produtivos, sendo estes itens, razão de motivação para inovação e desenvolvimento de novas soluções tecnológicas.

Aplicações para IoT podem demandar estilos de interação diferentes e protocolos de aplicação com características diferentes, como mostrado na Seção 1.3. Além disto, aplicações para IoT também pressupõem infra-estrutura diferentes, o que inclui redes com bandas distintas. Nesta dissertação, as bandas de 20, 50 e 1000 kbps, foram escolhidas para os testes porque estas características de restrição de banda de rede estão presentes na maioria das aplicações para IoT.

1.3 **Estilos de Interação**

Sistemas de acionamento e monitoramento normalmente utilizam dois estilos de interação: requisição-resposta (*request-response*) e publica-subscreve (*publish-subscribe*) [52]. Aplicações na Internet das Coisas têm este perfil: ou a aplicação tem interesse em monitorar algum sensor — ou conjunto de sensores, ou interesse em acionar algum dispositivo atuador, ou os dois. Estas aplicações geralmente incorporam uma estrutura de laço com os seguintes passos: monitoramento, avaliação, acionamento, reconfiguração e novo monitoramento. Neste laço, ou de forma concorrente, podem estar incluídas nas atividades da aplicação a interação com o usuário, exibição de informações, etc.

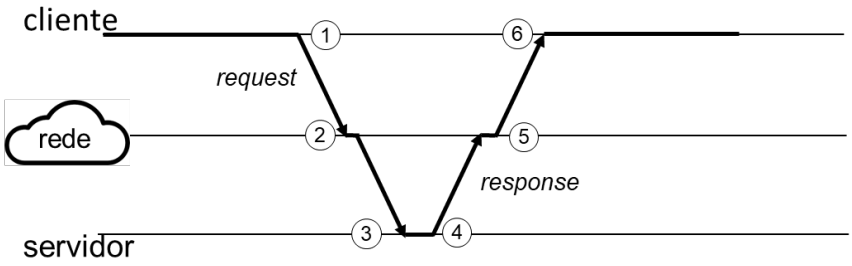
Considerando os passos de monitoramento e acionamento, de uma forma geral, aplicações de IoT podem precisar interagir de forma síncrona com um processo, agente ou dispositivo remoto. Isto é, a aplicação envia uma mensagem com uma requisição e só pode continuar suas atividades depois de receber uma resposta, ou uma recusa.

Outra possibilidade é a aplicação IoT não poder, ou não querer, aguardar uma resposta, mas receber uma notificação de forma assíncrona. Isto é, a aplicação registra sua requisição junto ao seu alvo, e continua sua execução logo em seguida. Num momento posterior, uma notificação pode chegar, e a aplicação deve estar preparada para tratar a mesma. Este estilo de interação é útil quando um evento é gerado, um dado estado é atingido ou uma computação concluída por uma entidade remota e, tão logo ocorra, isso deva ser informado à aplicação.

1.3.1 Direta - Requisição-Resposta (*Request-Response*)

Protocolos de Aplicação que utilizam o estilo *request-response* são estruturados para oferecer interação síncrona e direta entre dois processos (Figura 4 -).

Figura 4 - Estilo de interação *request-response*.



1. O processo cliente, ou enviante, envia uma mensagem contendo uma requisição para o processo servidor, ou recebedor e em seguida é bloqueado para aguardar a resposta;

2. A mensagem contendo a requisição precisar ser transportada pela rede, passando pelas barreiras de segurança, até que seja entregue ao servidor;

3. O processo servidor, que estava bloqueado, aguardando uma requisição, recebe a mensagem, interpreta o seu conteúdo e realiza o serviço solicitado na requisição;

4. A requisição pode, por exemplo, solicitar algum cálculo/processamento do servidor, a leitura de uma variável de um sensor ou o acionamento de um atuador. Se uma

resposta é necessária, uma nova mensagem é preparada contendo esta resposta e retornada para o cliente;

5. A mensagem de resposta também precisa ser transportada pela rede, passando pelas barreiras de segurança no caminho de volta;

6. Finalmente, é recebida pelo processo cliente, requisitante, que se desbloqueia, avalia resposta e prossegue sua execução.

É importante observar que as duas trocas de mensagens ocorrem de forma direta: os dois elementos cliente e servidor precisam estar envolvidos diretamente. A requisição só é recebida pelo servidor depois que o cliente envia a mensagem, e esta é recebida com algum atraso devido à latência da rede e dos sistemas de suporte à troca de mensagens. Se o servidor se prepara para receber a requisição antes da mensagem ser transmitida, este é bloqueado até que a mesma esteja disponível.

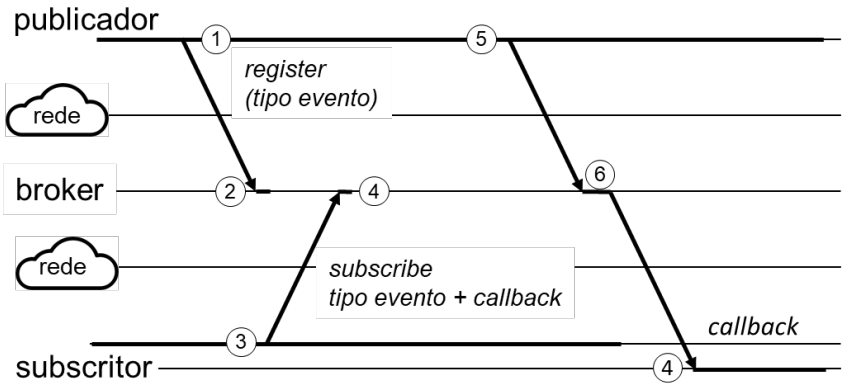
Por outro lado, se uma resposta é necessária, os papéis são invertidos. Logo após o envio da requisição o cliente se coloca pronto para receber uma mensagem contendo a resposta. O servidor prepara uma mensagem com a resposta e a envia para o cliente. Este é desbloqueado apenas quando a mensagem é efetivamente entregue.

Uma característica inerente à comunicação direta, o processo cliente precisa conhecer a referência ou endereço do processo servidor, geralmente o número IP, a porta e o protocolo de transporte utilizado (no caso do presente capítulo, UDP ou TCP). O servidor, por sua vez, deve expor uma referência ou endereço conhecido e possível de ser obtido pelo processo cliente. Além disso, elementos de segurança, como *firewall* e NAT, precisam ser configurados para o endereço do servidor.

1.3.2 Indireta - Publica-Subscreve (*Publish-Subscribe*)

O estilo de interação indireta, publica-subscreve (*publish-subscribe*), também chamado de interação baseada em eventos (*event-based*), é apropriado para comunicação assíncrona, quando os elementos que precisam interagir, não podem ou não querem estabelecer uma comunicação direta para trocar mensagens, ou não podem aguardar bloqueados a interação ocorrer. Ainda, também é a opção adequada quando a informação a ser comunicada pode ser produzida à qualquer momento, como por exemplo, uma variável medida por um sensor atinge um valor ou limiar.

Figura 5 - Estilo de interação *Publish-Subscribe*.



O suporte para este estilo de interação é mais complexo, se comparado com o estilo *request-response*. Seguindo a Figura 5 - , observamos os seguintes elementos:

- Publicador (*publisher*) ou fonte de evento (*event source*), é o elemento que pode gerar eventos. A aplicação pode interagir com vários publicadores.
- Evento, é uma abstração que pode ser representada por uma estrutura de dados ou objeto. Os eventos podem ter um tipo, identificação ou atributos, que indicam a sua natureza ou origem.
- Subscritor (*subscriber*) ou sorvedor de evento (*event sync*), é o elemento que está interessado em receber notificações quando eventos de determinado tipo ocorrem. A aplicação pode trabalhar com vários subscritores concorrentemente.
- O subscritor interessado precisa informar uma referência de *callback* ou tratador (*handler*), para onde a notificação deve ser enviada e o evento entregue. Essa referência pode ser padronizada, pode ser um endereço, ou um nome de procedimento/método.
- Sistema de eventos (*event engine*) ou *event broker* ou simplesmente *broker* (termo mais comum nos sistemas de IoT), intermedeia as interações entre publicadores e subscritores, incluindo os pedidos de registro e interesse em eventos, e o encaminhamento das notificações.

Como existem mais elementos e mais trocas de mensagens entre os elementos, os atarsos de comunicação poderiam interferir mais, se comparado ao *request-response*. Por

outro lado, observe-se também na Figura 5 - que as linhas de interação entre os elementos ocorrem de forma independente ao longo do tempo, reforçando a ideia de uma interação assíncrona. Assim, os atrasos da rede podem ter importância menor.

1. O publicador primeiro envia uma mensagem para o *broker* pedindo o registro de um tipo de evento;

2. O *broker* recebe a mensagem e registra o tipo de evento.

   Em alguns casos, esta sequência pode ser feita por configuração administrativa.

3. O processo subscritor envia uma mensagem para o *broker* registrando interesse em um tipo de evento, e passa a referência de *callback*;

4. O *broker* verifica os aspectos de segurança – quando isso fizer parte de sua operação – e, então, adiciona a referência informada pelo subscritor na lista de interessados pelo evento;

   Depois de registrar o interesse em um evento, o processo subscritor pode continuar seu processamento.

5. Em algum momento qualquer, o publicador gera um evento e envia uma mensagem ao *broker* solicitando sua publicação;

6. Recebendo a publicação, o *broker* percorre a lista relacionada ao tipo de evento e envia uma mensagem de notificação para cada subscritor interessado, em sua referência da *callback*;

7. A notificação é recebida na referência de *callback* pela rotina de tratamento, *handler*, sem que a rotina original seja interrompida. A arquitetura de software necessária para o tratamento concorrente das atividades "normais" e de notificações não é aprofundada aqui.

A necessidade de um elemento intermediário, como o *broker*, introduz um ponto de complexidade: temos mais um elemento, que precisa executar como um serviço contínuo, que por sua vez precisa ser mantido/administrado. Entretanto, como ressaltado nas próximas seções, este elemento pode ser responsável por procedimentos de segurança e controle de acesso, o que é positivo. Pode também oferecer opções de enfileiramento,

filas com prioridade diferenciada e persistência de eventos, permitindo que um subscritor que estava *offline* receba notificações passadas. Um elemento central como o *broker* também pode facilitar a comunicação de elementos atraz de redes com NAT e *firewall*, sem a necessidade de esforços de configuração nos elementos de borda da rede.

Por último, uma dica que pode ajudar o entendimento do uso dos protocolos. Os mecanismos relacionados ao estilo de interação *publish-subcribe* podem ser usados em uma aplicação de uma maneira um pouco "invertida" no caso de atuadores. Por exemplo, digamos que a aplicação precise acionar um relé controlado por um dispositivo Arduino. Digamos também, que uma aplicação Android permita ao usuário controlar este relé remotamente, enviando um "comando" de liga-desliga. O Arduino controlando o relé poderia ter o papel de subscritor, registrando o interesse em "eventos de liga-desliga". A aplicação Android, por sua vez poderia fazer o papel de fonte de evento do tipo "liga-desliga", enviando um evento quando o usuário assim acionasse, que por sua vez levaria ao *broker* notificar o dispositivo Arduino, que então atuaria ligando ou desligado o relé.

1.4 **Tecnologias de Comunicação**

Smartphones utilizados em aplicações na IoT podem, por exemplo, controlar iluminação, ligar ou desligar dispositivos eletrônicos e se comunicar com outros dispositivos na rede (também chamado de comunicação máquina-a-máquina, M2M). A mesma necessidade de comunicação se dá com outros dispositivos com menos recursos ou serviços. Para que isto seja possível, é preciso que os dispositivos possuam alguma interface de comunicação. Cada dispositivo na IoT pode oferecer interfaces diferentes, o que pode dificultar a integração destes dispositivos. É esperado que desenvolvimento de software possa contar com níveis de abstração maiores, independentes de interfaces específicas. Ainda assim, é importante que se conheça as principais opções de comunicação disponíveis. Nas seções seguintes são descritas as principais características de 4 padrões de interface de comunicação mais para IoT.

1.4.1 Wi-Fi

O Wi-Fi (*Wireless Fidelity*) é uma tecnologia de rede sem fio que permite que dispositivos como computadores (laptops e desktops), dispositivos móveis (*smartphones* e

wearables) e outros equipamentos se comuniquem, trocando informações entre si [53], [54].

O padrão IEEE 802.11 define os protocolos que permitem a comunicação com os dispositivos sem fio habilitados para Wi-Fi,o que inclui roteadores sem fio e pontos de acesso sem fio. O Wi-Fi é um termo registrado da Wi-Fi Alliance [53]. É utilizada por produtos certificados que pertencem à classe de dispositivos de rede local sem fios que implementam o IEEE 802.11.

Existem vários fatores que podem influenciar o alcance do sinal Wi-Fi, como o exemplo paredes, barreiras, frequências, equipamentos elétricos, e etc [55]. A potência e o ponto de instalação do equipamento também influenciam na distribuição do sinal.

O padrão mais atual da família IEEE 802.11 é 802.11ac, que pode atingir taxas de 1.300 mpbs.

Em 2017, foi publicado pelo IEEE uma extensão do padrão de rede sem fio, chamada de 802.11ah [56], também conhecido como Wi-Fi HaLow, operando em frequências próximas de 900MHz. Tem por objetivo prover conectividade com menor consumo de energia e maior raio de abrangência, algo relevante para aplicações da Internet das Coisas.

Algumas características justificam a popularidade do Wi-Fi, são elas: compatibilidade entre dispositivos e baixo custo de aquisição de equipamentos certificados.

1.4.2 Bluetooth

O Bluetooth [57] é uma tecnologia de comunicação sem fio de curto alcance que permite que dispositivos como telefones celulares, computadores e periféricos transmitam dados ou voz sem fio a uma curta distância. O objetivo do Bluetooth é substituir os cabos que normalmente conectam os dispositivos, mantendo as comunicações entre eles seguras.

O nome “Bluetooth” tem origem de um rei dinamarquês do século X chamado Harald Bluetooth, que dizia unir facções regionais diferentes. Como seu homônimo, a tecnologia “Bluetooth” reúne uma ampla gama de dispositivos em muitos setores diferentes por meio de um padrão de comunicação unificado.

Desenvolvido em 1994 pela *Bluetooth Special Interest Group* [57], o Bluetooth foi concebido como um substituto sem fio para os cabos. Ele usa de 2,4 GHz, a mesma que algumas outras tecnologias sem fio, como por exemplo telefones sem fio e roteadores Wi-Fi. Ele cria uma rede sem fio de raio de 10 metros, conhecida como rede de área pessoal

(PAN), que pode fazer a rede entre dois e oito dispositivos.

O Bluetooth consome menos energia e custa menos para implementar do que o Wi-Fi. Sua menor potência também torna muito menos propenso a sofrer ou causar interferência em outros dispositivos sem fio na mesma banda de rádio de 2.4GHz.

O alcance do Bluetooth e as velocidades de transmissão são tipicamente inferiores ao Wi-Fi. O Bluetooth em sua versão v3.0, os dispositivos podem fornecer até 24 Mbps de dados, o que é mais rápido que o padrão 802.11b Wi-Fi, poréms mais lento que os padrões wireless-a ou wireless-g. À medida que a tecnologia evoluiu, no entanto, as velocidades do Bluetooth aumentaram.

A especificação Bluetooth 4.0 foi oficialmente adotada em 6 de julho de 2010. Os recursos da versão 4.0 do Bluetooth incluem baixo consumo de energia, baixo custo, interoperabilidade de vários fornecedores e alcance aprimorado. Atualmente, o Bluetooth já se encontra na versão 5.0.

1.4.3 ZigBee

O ZigBee [58] é uma tecnologia sem fio baseada em padrões desenvolvida para redes sem fio de baixo custo e baixo consumo de energia (M2M) e Internet das coisas (IoT).

O Zigbee foi padronizado e projetado para aplicações de baixa taxa de dados e baixo consumo de energia. Este é baseado na especificação 802.15 da Institute of Electrical and Electronics Engineers (IEEE) Standards Association. O Zigbee foi construído para redes de controle e sensores no padrão sem fio IEEE 802.15.4 para redes de área pessoal sem fio (WPANs), operando em frequências de 2,4 GHz, 900 MHz e 868 MHz.

Existem três especificações ZigBee: ZigBee PRO, ZigBee RF4CE e ZigBee IP.

O ZigBee PRO visa fornecer a base para a IoT com recursos para suportar redes altamente confiáveis e de baixo custo para comunicação de dispositivo para dispositivo. O Zigbee PRO também oferece o Green Power, um novo recurso que suporta a coleta de energia ou dispositivos autoalimentados que não exigem baterias ou fonte de alimentação.

O Zigbee RF4CE foi projetado para aplicativos de controle de dispositivo para dispositivo simples, bidirecionais, que não precisam das funcionalidades de rede *mesh* com todos os recursos oferecidas pela especificação ZigBee.

O Zigbee IP otimiza o padrão para redes *mesh* sem fio completas baseadas em

IPv6, oferecendo conexões à Internet para controlar dispositivos de baixo consumo de energia e baixo custo.

Em resumo, uma rede *mesh* é composta de vários nós/roteadores, que passam a se comportar como uma única e grande rede, possibilitando que o cliente se conecte em qualquer um destes nós. Os nós têm a função de repetidores e cada nó está conectado a um ou mais dos outros nós. Desta maneira é possível transmitir mensagens de um nó a outro por diferentes caminhos.

1.4.4 6LoWPAN

O 6LoWPAN [59] (*IPv6 over Low power Wireless Personal Area Networks*) é um padrão aberto definido pelo IETF, RFC 6282 [60]. Esta tecnologia utiliza o IEEE 802.15.4 para fornecer as camadas inferiores para este sistema de rede sem fio de baixa potência.

O grupo de trabalho 6LoWPAN da IETF une esforços no sentido que seja possível a comunicação pelo protocolo IPv6 nas redes IEEE 802.15.4. As questões mais pertinentes passam pela fragmentação/desfragmentação e a compressão dos cabeçalhos IPv6. A MTU máximo do IPv6 é de 1280 octetos, enquanto no IEE 802.15.4 é de 127 octetos, tornando-se importante o uso de fragmentação e desfragmentação. Normalmente nas redes LoWPAN o cabeçalho IP e UDP contêm muitos campos desnecessários e outros que também podem ser comprimidos. Portanto é importante e indispensável a compressão dos cabeçalhos IPv6 para o sucesso e futuro do uso de IPv6 nas RSSF. O 6LOWPAN cria uma camada de adaptação entre o IEEE 802.15.4 e o IPv6, com cabeçalhos específicos que podem ser adicionados ou removidos mediante a sua necessidade, permitindo que seja apenas enviado o que realmente é útil.

Neste trabalho só usamos o Wi-Fi para as avaliações de desempenho, devido a seu baixo custo e disponibilidade de equipamentos compatíveis. Foram feitas configurações para limitação de banda no Wi-Fi, assim pôde-se simular características presentes em todos as tecnologias de comunicação utilizadas para IoT. Em trabalhos futuros, outras interfaces podem ser testadas.

2 PROTOCOLOS DE APLICAÇÃO

2.1 Introdução

Neste capítulo, serão apresentados os conceitos básicos, mecanismos/estruturas principais e os estilos de interação suportados para cada protocolo de aplicação para IoT objeto de estudo deste trabalho. Também serão abordados aspectos específicos, que distinguam os protocolos como, por exemplo aspectos de QoS, diferentes esquemas de filas, facilidade para interação com elementos em redes protegidas por NAT e *firewall*, registro, autenticação e mecanismos de segurança disponíveis. Também importante, serão apresentadas as soluções para identificação de referências, endereços, eventos e filas, quando pertinente.

2.2 HTTP

O *Hypertext Transfer Protocol*, HTTP [5], é um protocolo com estilo *request-response*, descrito na RFC 2616, base para a comunicação de dados para a *World Wide Web*, WWW, desde 1990. Requisições do navegador Web e a resposta do servidor são transportadas por HTTP. Como outros protocolos de aplicação da Internet, o HTTP tem as informações de controle transmitidas como cadeias de caractere.

O HTTP é *media independent*, ou seja, qualquer tipo de dado pode ser enviado, desde que o cliente e o servidor saibam como lidar com os conteúdos. A informação efetivamente sendo transportada pode ter um dos tipos previstos no padrão, usando o tipo *MIME*, descrito em um campo chamado *content-type*, como por exemplo, imagens, arquivos compactados, textos em HTML ou XML.

O TCP é utilizado como protocolo de transporte. Nas primeiras versões do HTTP a conexão era desfeita logo após uma resposta ser retornada para o cliente. Na versão 1.1, descrita na RFC 2616 [5], o protocolo passa a manter a conexão aberta entre cliente e servidor, permitindo que sequências de requisições e respostas sejam tratadas, melhorando o desempenho.

Também é possível prover segurança na comunicação entre um cliente e servidor HTTP utilizando a versão segura, HTTP *Secure*, que executa sobre uma camada SSL/-TLS.

O HTTP faz uso do Identificador Uniforme de Recursos, *Uniform Resource Identifier*, URI [61], como referência para o servidor com o qual a conexão será estabelecida, e para determinado recurso hospedado neste servidor. No seguinte exemplo, (i) *http* é chamado de esquema, e indica o protocolo ou tecnologia utilizada; (ii) *www.lcc.uerj.br* é chamado de domínio, e será convertido para um número IP após consulta ao DNS; (iii) *8080* é a especificação da porta onde o servidor espera receber pedidos de conexão. Caso não seja informado, o esquema vincula a porta a um padrão (80, no caso do HTTP); (iv) *laboratorio* indica a rota ou caminho para o diretório onde o recurso está disponível; e (v) *recurso* é a referência ao recurso desejado.

`http://www.lcc.uerj.br:8080/laboratorio/recurso`

Uma vez estabelecida a conexão, mensagens HTTP de requisição e reposta são transmitidas. São padronizados alguns tipos de mensagem, chamados de métodos no HTTP, que podem provocar reações diferentes do servidor. Nos exemplos deste capítulo utilizaremos:

GET. Recupera um *recurso* do *servidor*, dados fornecidos na URI.

POST. Envia dados para o servidor no corpo da mensagem HTTP, como por exemplo, informações de texto, *upload* de arquivos, ou informações contidas em formulários HTML. O servidor precisa estar configurado e preparado para realizar alguma ação com as informações enviadas. O exemplo "clássico" são os programas acionados pelo mecanismo de CGI, disparados quando o servidor identifica o parâmetro *action* dentro de um formulário HTML.

Outros métodos disponíveis são: *HEAD*, *PUT*, *DELETE*, *CONNECT*, *OPTIONS* e *TRACE*.

Por padrão, as mensagens de resposta contém um código de retorno, entre os quais: (i) 200, indica sucesso; (ii) 404, indica recurso não encontrado e (iii) 500, erro interno do servidor.

A razão central para considerar o HTTP como suporte à interação entre processos, dispositivos e serviços em uma aplicação para a IoT é utilizar um protocolo de aplicação provadamente funcional, disseminado, que executa sobre uma infraestrutura de comunicação já estabelecida, com segurança geralmente configurada.

Os métodos disponíveis *PUT*, *GET*, *POST* e *DELETE*, e a abordagem de campos e meta-informações do HTTP podem ser relacionados a uma interface CRUD (*Create*

(Criar), *Retrieve* (Consultar), *Update* (Atualizar) e *Delete* (Apagar)), amplamente usada em sistemas de informação. Estes poderiam ser utilizados para acionar atuadores ou obter informações de sensores, por exemplo. O problema, em princípio, seria: (i) mudar a interpretação padrão dos servidores Web para estas operações e (ii) prover um mecanismo para executar ações fora do contexto do processo do servidor Web. Por exemplo, o método GET, por padrão, implica em se percorrer o *caminho* até o diretório onde o *recurso* se encontra e em seguida recuperar o arquivo cujo nome é "recurso" e enviar o mesmo como resposta para o cliente. Outro aspecto desta interpretação é que o conteúdo de *recurso* é estático, à menos que seja alterado manualmente. No caso de IoT queremos chamar um procedimento para acionar um atuador, ler um sensor ou solicitar um serviço, por exemplo. Neste caso, a execução e a resposta podem ser diferentes a cada chamada.

Ao longo dos anos de operação da Web, alguns mecanismos foram propostos para utilizar a sua infraestrutura para executar procedimentos remotos. O *Common Gateway Interface*, CGI, foi um dos primeiros mecanismos propostos [62]. A cada solicitação, um ambiente de execução é criado e um programa carregado e executado. Parâmetros de entrada podem ser passados como variáveis de ambiente para o programa a ser executado. A resposta é, geralmente, um texto em HTML criado dinamicamente pelo programa. Várias evoluções foram propostas.

Os Serviços Web (*Web Services*) são uma outra abordagem, mais eficiente que o CGI, que permite que um processo cliente envie uma requisição para execução remota de procedimento. Para isso, é enviando para o servidor o nome do procedimento e uma lista de parâmetros de entrada. O serviço é realizado e uma resposta é retornada para o cliente. Conceitualmente, funciona como o mecanismo *Remote Procedure Call*, RPC, utilizando também o conceito de interface, descritas em WSDL, e é aderente ao padrão SOA (*Service Oriented Architecture*). Requisições e respostas são representadas em XML. Um esquema XML, chamado SOAP, regulamenta como os elementos das requisições e respostas devem ser estruturados em arranjos e *tags* XML específicos. *Web Services* são amplamente suportados por bibliotecas, servidores HTTP e servidores de aplicação especializados. A abordagem é bem completa e tem um padrão estabelecido [63].

Sistemas e aplicações para IoT podem ser baseadas em *Web Services*. Entretanto, a abordagem tem custo computacional notável. Este custo pode ser atribuído às rotinas de *parsing* das estruturas XML, pela necessidade de ser manter os elementos de suporte

para descrição de interfaces em WSDL e pelo custo dos mecanismos que encaminham as informações de uma chamada para a entidade que vai executar a mesma.

REST sobre HTTP. A abordagem *REpresentational State Transfer*, REST, tem se mostrado um alternativa interessante para IoT [64], [65]. Independente de aspectos conceituais discutidos em [66], REST permite o uso da infraestrutura do HTTP para acionar ou obter recursos, apenas dando uma interpretação diferente para os métodos *GET*, *PUT*, *POST* e *DELETE*, e valendo-se da possibilidade de enviar informações adicionais numa mensagem HTTP (que podem ser interpretados como parâmetros de entrada). O conceito em torno de REST discute que a abordagem não é um RPC, que o melhor uso não é fazer chamadas remotas de procedimentos com uma possível resposta como retorno. Entretanto, algumas infraestruturas para IoT utilizam REST para ler e acionar dispositivos transferindo estruturas JSON nos dois sentidos, como por exemplo [1], com o objetivo de se ter um mecanismo de interação distribuído, com custo aceitável executando sobre HTTP.

O mecanismo REST não precisa considerar detalhes da implementação do serviço RESTful. Por outro lado, o mecanismo pode usar as informações de controle do HTTP para complementar a definição de como o serviço será executado. Campos customizados podem ser usados para transferir parâmetros para o serviço. Mesmo não sendo recomendado, o próprio campo com a informação útil sendo transportada pode transferir dados em notação XML ou JSON. A configuração definida no servidor para tratar a requisição REST e a própria implementação do serviço é que vão definir como estas informações são utilizadas e como o serviço será acionado [67].

O uso de HTTP não é obrigatório para acionamento de serviços com abordagem REST. Um exemplo é a adoção desta abordagem no protocolo CoAP.

2.3 CoAP

O *Constrained Application Protocol*, CoAP [7], é projetado para permitir a troca direta de mensagens entre dispositivos com poucos recursos computacionais, interconectados por redes com baixas taxas de transferência, com a 6LowPAN [59], por exemplo. Os dispositivos considerados geralmente são construídos com microcontroladores de baixo custo, como por exemplo o Arduino, NodeMCU e outros. O CoAP é descrito na RFC 7252 do IETF CoRE (*Constrained RESTful Enviroments*) Working Group.

O estilo de interação oferecido pelo CoAP é o *request-response*, com características semelhantes ao HTTP/REST. Como na *Web*, os dispositivos são referenciados usando o endereço IP e o número da porta. Entretanto, por estratégia de projeto, o CoAP é executado sobre UDP e não TCP. Isso pode dificultar a implantação de uma aplicação cujos elementos estejam distribuídos em redes atrás de NAT e firewall. A configuração, neste caso, deve considerar tanto os clientes como os servidores dado que não existe uma conexão. Assim, o CoAP é mais facilmente usado em uma rede local.

O CoAP não foi desenhado para substituir o HTTP, mas ele implementa um pequeno subconjunto de práticas HTTP amplamente aceitas, e as otimiza para a troca de mensagens entre dispositivos (*machine-to-machine*, M2M) [30].

O acesso aos serviços expostos pelo dispositivo se dá por URIs REST. Também de forma semelhante ao HTTP, são previstos tipos de requisição, por exemplo *GET*, *PUT*, *POST* e *DELETE*, com códigos de resposta, como por exemplo, 404, 500 e a especificação de um tipo de conteúdo usados para transmitir informações.

As mensagens transportadas pelo CoAP tem formato binário, permitindo o transporte de qualquer formato de dados, como por exemplo XML, JSON ou qualquer outro desejado. Possui um cabeçalho fixo de 4 bytes e um *payload* máximo que deve caber em um único datagrama UDP, ou no *frame* IEEE 802.15.4 [60].

O CoAP pode também usar o DTLS como protocolo de transporte para comunicação segura. Além disso, para contornar a falta de garantia de entrega de pacotes do UDP, o CoAP oferece dois níveis de qualidade de serviço: *Confirmado*, que adiciona um pacote *ACK* de confirmação de recebimento, e *Não-Confirmado*, sem mensagem de confirmação.

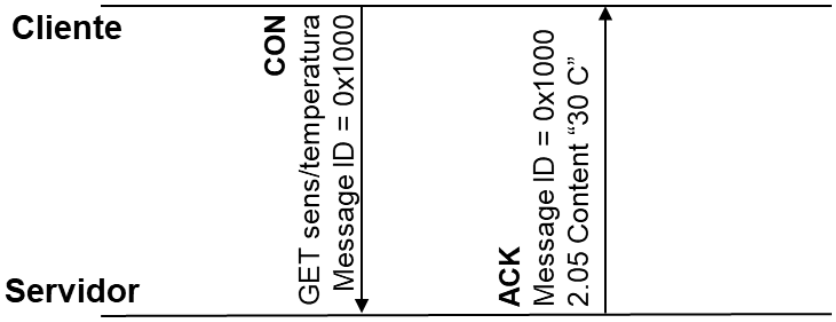
As opções na troca de mensagens dão flexibilidade ao CoAP. Por isso, discutimos algumas opções a seguir.

Confirmado com *piggbacking*. O cliente envia uma mensagem *CON* para o servidor solicitando a temperatura (Figura 6 -). O tipo da requisição é *GET*, a URL do caminho é "sensores/temperatura". O *ID* da mensagem é um número de 16 bits usado para identificar exclusivamente uma mensagem e ajudar o servidor na detecção de mensagens duplicadas. Assim que o servidor recebe a mensagem *CON*, a temperatura é obtida e uma mensagem de confirmação *ACK* é retornada, com o mesmo *ID*. Junto com a confirmação, a mensagem também leva "de carona" os dados de temperatura solicitada, no caso *30C*.

Por fim, a resposta também tem um código de mensagem, neste caso, "2.05 Content". Estes são semelhantes aos códigos de status HTTP.

Se uma confirmação não for necessária, ou se for aceitável para a aplicação eventualmente não receber uma resposta, uma mensagem do tipo *NON* pode ser utilizada.

Figura 6 - Mensagem CoAP confirmada com resposta *piggybacked*.



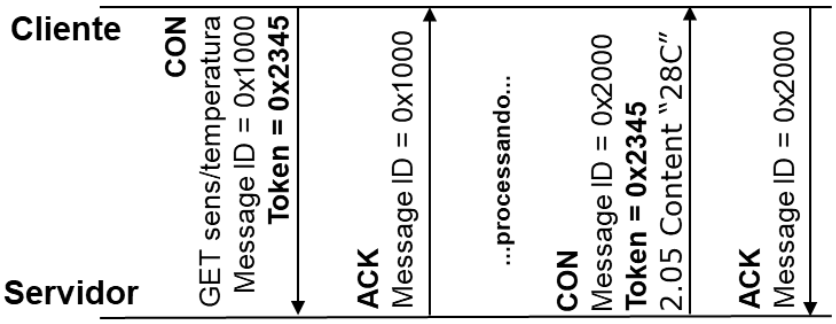
Confirmado sem *piggybacking* (correlação de mensagens). No caso em que a resposta para uma mensagem confirmada não possa ser levada de " carona", *piggybacked*, o ACK é enviado rapidamente, por exemplo, no caso do servidor precisar de um tempo maior para processar a mensagem. Posteriormente, uma resposta separada é enviada de volta ao cliente.

Como o CoAP é transportado por UDP, não existe uma conexão estabelecida: cada mensagem é um datagrama independente (por isso não pode ser usado com WebSocket). Assim, mensagens de requisição e resposta precisam ser correlacionadas pelo protocolo, dado que a interação deve ser coerente. A solução é o uso de um *token* para identificar o par de mensagens requisição-resposta (Figura 7 -).

Depois que o servidor tiver concluído o serviço, ele enviará uma outra mensagem *CON* contendo a resposta. Esta resposta possui um novo *ID* de mensagem e deverá ser confirmada posteriormente por um ACK correspondente. Entretanto, o valor do *token* desta resposta é o mesmo da requisição, permitindo ao cliente correlacionar as mensagens.

De uma forma geral, o *token* é usado para correlacionar mensagens, simulando uma conexão, mesmo que a mensagem seja não-confirmada e os IDs diferentes.

Figura 7 - Mensagem CoAP confirmada, resposta separada (correlação).



Além da troca de mensagens *request-response*, o CoAP prevê duas outras funcionalidades e opções:

Observador. Com esta opção, a mensagem de requisição do cliente pode conter o registro do interesse por um recurso. Qualquer mudança de estado no recurso, por exemplo, pode ser notificada com o envio de uma mensagem. Neste caso, o *token* é usado para identificar o evento ou origem da notificação. Com isso simula-se o estilo *publish-subscribe*, chamado de *resource-observe* no CoAP [68].

Descoberta de Recursos. O CoAP também padroniza uma forma para um cliente descobrir os recursos disponíveis em um servidor. Para isso, uma mensagem *GET* deve ser enviada com a URL do servidor finalizada com `/.well-known/core`. Neste caso, o servidor retorna os recursos disponíveis no formato CoRE Link [69].

O CoAP oferece outras opções de configuração e interação, além das discutidas nesta seção. Por exemplo, o conceito *block-wise* que facilita a transmissão de informações longas. A RFC 7252, já mencionada, apresenta estas outras opções.

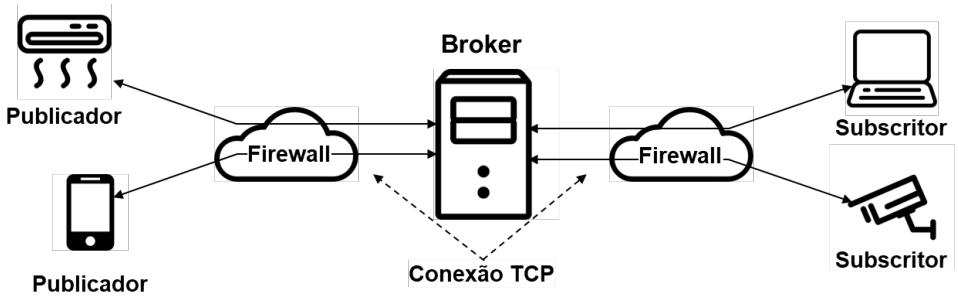
2.4 MQTT

O *Message Queuing Telemetry Transport*, MQTT [6], foi concebido por Andy Stanford-Clark (IBM) e Arlen Nipper (Arcom, agora Cirrus Link) em 1999, como um protocolo leve para troca de mensagens entre dispositivos, que tivesse baixo consumo de bateria e pouco uso de banda de rede. Atualmente o MQTT é um padrão ISO/IEC, 20922:2016 e OASIS [70].

Embora o seu nome inclua "enfileiramento de mensagens" o MQTT não é orientado à fila de mensagens como o AMQP e o XMPP. Essencialmente o MQTT é um sistema gerenciador de eventos, que suporta o estilo de interação *publish-subscribe*, e utiliza o

TCP para todas as trocas de mensagens entre publicadores e subscritores com o *broker*. A Figura 8 - apresenta os elementos principais do MQTT.

Figura 8 - Elementos do MQTT.



A arquitetura centralizada do *broker*, facilita a introdução de mecanismos de autenticação e a implementação de características de confiabilidade ou garantia de entrega (chamadas de QoS no MQTT). As credenciais do usuário devem ser enviadas ao se estabelecer uma conexão. Além disso, a comunicação pode fazer uso da camada de segurança TLS/SSL.

No MQTT um tipo de evento é chamado de *tópico*. Um tópico é criado quando uma mensagem *PUBLISH* ou *SUBSCRIBE* é enviada (detalhes adiante). A estratégia para nomear ou referenciar um tópico segue a da formação de nomes de diretórios, que pode ter níveis separados pelo caractere "`/`" e usar *wildcards*.

Exemplo 1: `minhacasa/terreo/sala/temperatura`. O subscritor recebe o tópico *temperatura* de *sala*.

Exemplo 2: `minhacasa/terreo/$+$/temperatura`. O subscritor recebe o tópico *temperatura* de qualquer "parte" do *terreo*.

Exemplo 3: `minhacasa/terreo/\#`. O subscritor recebe qualquer tópico de qualquer "parte" do *terreo*.

Os clientes MQTT — publicadores e subscritores — e o *broker* comunicam-se através de mensagens de tipos específicos. As principais: *CONNECT*, *SUBSCRIBE*, *UNSUBSCRIBE* e *PUBLISH*, oferecem suporte para a interação no estilo *publish-subscribe*. Existem também mensagens do tipo *ACK* para cada ação, dependendo do nível de QoS selecionado. Para utilizar o MQTT o programador deve enviar e receber mensagens destes tipos, contendo alguns dados inerentes à informação de interesse, como, por exemplo, o

tópico, e informações de controle, como, por exemplo, a senha do usuário. Na sequência discutimos as principais.

CONNECT. Enviada pelo cliente quando este deseja conectar-se ao *broker*. Como o MQTT utiliza o TCP, uma conexão é efetivamente estabelecida entre cada cliente e o *broker*. Isso facilita a administração da rede pois, a configuração de *firewall* e NAT precisa apenas ser realizada no nó onde o *broker* executa, uma vez que a conexão TCP permite troca de mensagens bi-direcional. Além disso, a conexão é também reconhecida como sendo o endereço de *callback* para todos os *tópicos* subscritos por cada cliente.

- clientId. Identificador único de cada cliente. A aplicação deve gerenciar esta informação.
- cleanSession. Especifica se uma conexão vai iniciar uma nova sessão ou vai reestabelecer uma sessão prévia. Se um dado cliente subscrever um tópico e indicar o uso de QoS 1 ou 2, isso fica registrado no *broker*. Caso seja necessário reestabelecer uma conexão e *cleanSession false*, a sessão recupera as subscrições e parâmetros daquele *clientID* e as mensagens possivelmente perdidas são recebidas.
- username. Credenciais de autenticação e autorização junto ao broker. Observa-se que o usuário deve registrar-se previamente no *broker*.
- password. Credenciais de autenticação e autorização junto ao broker.
- lastWillTopic. Quando a conexão for encerrada inesperadamente, o *broker* publicará automaticamente uma mensagem de "último desejo" em um tópico.
- lastWillMessage. A própria mensagem de "último desejo"enviada para o *lastWillTopic*.
- keepAlive. Intervalo de tempo em que o cliente precisa efetuar *ping* no *broker* para manter a conexão ativa.

SUBSCRIBE. Enviada para informar ao *broker* os tópicos de interesses para recebimento de notificações.

- tópico. Referência do tópico para subscrever.
- qos. Indica a semântica de entrega e confirmação com que as mensagens neste tópico precisam ser entregues aos clientes, segundo a Tabela Tabela 1 - .

Tabela 1 - Níveis de QoS no MQTT.

Nível	Semântica	Descrição
0	no máximo uma vez (*at most once*)	A mensagem de confirmação não é enviada pelo receptor. É conhecido como "Fire and Forget". Pode ser mais eficiente.
1	pelo menos uma vez (*at least once*)	A mensagem será entregue ao receptor pelo menos uma vez, porém, a tentativa de entrega pode ocorrer mais de uma vez. O publicador precisa guardar a mensagem em um *buffer* local até o recebimento da confirmação pelo receptor.
2	exatamente uma vez (*exactly once*)	A mensagem será recebida, e apenas uma vez. É o nível mais seguro, mas também pode ser menos eficiente. Neste caso, tanto o publicador quanto o receptor, tem a certeza do recebimento da mensagem pelo destinatário.

PUBLISH. Este tipo de mensagem é usado em dois momentos. Primeiro, pelo cliente ao *broker* para a publicação de algum conteúdo em algum tópico. Em seguida, o *broker* realiza as verificações necessárias e retransmite esta mensagens a cada um dos clientes subscritos.

- topicName. Referência do tópico no qual a mensagem é publicada. Se o tópico ainda não existe, ele é criado.
- qos. Nível de qualidade de serviço da entrega da mensagem. Observe-se que o nível que QoS configurados nas mensagens *PUBLISH* e *SUBSCRIBE* são independentes, pois dizem respeito à comunicação entre o cliente o *broker*.
- topicMessage. Informação que dever ser transportada na mensagem. Pode ser uma sequência de caracteres ou um *blob* binário de dados.

2.5 AMQP

O *Advanced Message Queuing Protocol*, AMQP [8], é um sistema de comunicação orientado à filas de mensagens, projetado e desenvolvido pela JPMorgan Chase em 2003. O AMQP é, atualmente, um padrão ISO/IEC (versão 1.0), OASIS. A norma está descrita no documento ISO/IEC 19464:2014 [71].

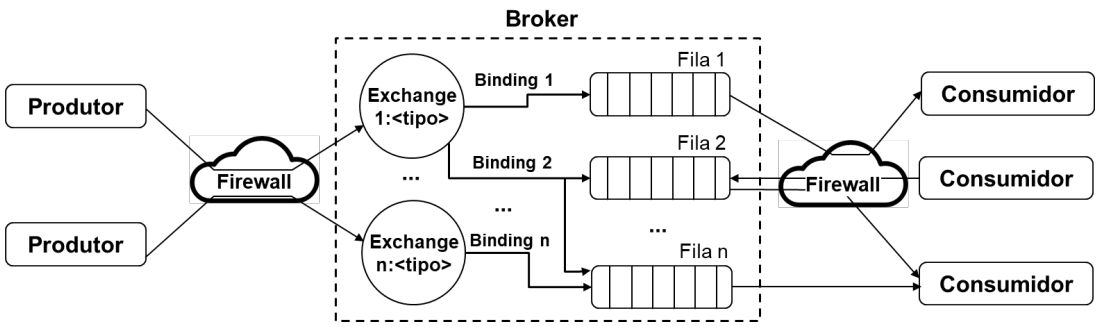
Como todo sistema orientado a filas, o AMQP tem em sua arquitetura um gerenciador de filas e um roteador de mensagens, além do protocolo de envio, recebimento e enfileiramento de mensagens. Este conjunto de elementos é geralmente chamado de *middleware* orientado a mensagens, MOM (*message oriented middleware*).

Para situar este tipo de infraestrutura, temos a analogia com sistemas de correio eletrônico que utiliza elementos chamados Agentes de Troca de Mensagens (*Message Transfer Agents*, MTA), que poderiam ser baseados no AMQP.

A arquitetura do AMQP versão 0-9-1, utilizada neste capítulo, apresenta os seguintes elementos, coletivamente chamados de *entidades* do *broker* AMQP (Figura 9 -)[2]:

- Produtores ou Publicadores. Clientes que enviam mensagens a um ou mais destinatários. Ao publicar uma mensagem o produtor pode especificar atributos de mensagens (meta-dados), que podem ser usados pelo *broker*.
- Consumidores ou Subscritores. Clientes que recebem mensagens de uma fila.
- *Exchange*. Recebe mensagens dos produtores e as encaminha para uma fila de mensagem. Este encaminhamento pode utilizar políticas parametrizadas por propriedades extraídas da própria mensagem ou de seu conteúdo, ou segundo regras chamadas *Bindings*.
- *Binding* (ligação). Regra que define o relacionamento entre um *Exchage* e uma e Fila de Mensagens, incluindo critérios de roteamento/encaminhamento.
- Fila de mensagens. Armazena mensagens até que elas possam ser processadas com segurança por um consumidor ou múltiplos consumidores. Cada fila é criada com um nome, que pode ser usado como chave de roteamento pelo *Exchange*.

Figura 9 - Elementos e entidades do *Broker* AMQP.



As mensagens são transportadas de forma binária, e o TCP é utilizado como protocolo de transporte. A camada SSL/TLS pode ser usada para comunicação segura entre

[2]A versão 1.0 apresenta um arquitetura e nomenclatura de elementos um pouco diferentes.

os elementos. Para ter acesso ao servidor, aplicações produtoras e consumidoras precisam estabelecer uma conexão e se autenticar no *broker*. A conexão é mantida pela aplicação enquanto houver interesse pelos serviços do AMQP. Esta característica também facilita o acesso ao serviço mesmo que o cliente esteja em redes com NAT e protegida por *firewall*.

O padrão inclui características configuráveis de confiabilidade, segurança e interoperabilidade. O AMQP pode emular o estilo de comunicação *request-response* e também o *publish-subscribe*. Isso é facilitado pelo uso de um elemento central, o *broker* AMQP, para intermediar a troca de mensagens e, também, pela flexibilidade dos elementos como o *Exchange* e o gerenciamento das Filas de Mensagens. Também é possível oferecer combinações de características não-funcionais como prioridade de filas ou filtros, com o uso de *Bindings*. A confiabilidade da entrega de mensagens, baseada em mensagens de controle *ACK* é um outro ponto positivo do AMQP.

O protocolo para troca de mensagens do AMQP é neutro e aberto, facilitando o envio de qualquer informação e a interoperabilidade entre diferentes plataformas. Além disso, o protocolo é programável, contribuindo para a facilidade de configuração de características oferecidas pelo *broker*. A configuração das entidades, regas e políticas é realizada pela própria aplicação, através de mensagens específicas, e não pelo administrador do servidor. Assim, uma aplicação pode declarar as entidades necessárias, esquemas de roteamento e o comportamento de filas. Uma das configurações possíveis para a aplicação é selecionar o tipo *Exchange* a ser utilizado:

- Direto, *amq.direct* ou sem nome, tipo mais simples, padrão para toda fila criada, utilizado para comunicação ponto a ponto ou *unicasting*. Roteia as mensagens segundo a chave de roteamento (na verdade, um nome associado ao *Exchange* e a uma *Fila*).

- Fanout, *amq.fanout*, utilizado para comunicação *1*-para-*N* ou comunicação *broadcast*. A chave de roteamento é ignorada e uma cópia de mensagem é colocada em todas as filas associadas ao *Exchange*. Este tipo de *Exchange* pode ser usado para aumentar a escalabilidade.

- Tópico, *amq.topic*, utilizando emular o estilo *publish-subscribe*, ou comunicação *1*-para-*N* ou *N*-para-*N*, ou comunicação *multicasting*. As mensagens são roteadas para uma ou mais filas, com base na chave de roteamento da mensagem e em um

padrão usado para associar a *Fila* ao *Exchange*.

Além do tipo, é possível selecionar atributos do *Exchange* como: o nome específico (uma cadeia de caracteres), se serão duráveis ou não duráveis (são restaurados ou não após uma reinicialização do servidor), e se têm metadados associados a eles na criação (detalhes em [72]).

Depois que as mensagens são recebidas pelo *Exchange* e roteadas de acordo com os *Bindings*, estas seguem para suas respectivas filas. As mensagens são armazenadas em uma base de dados interna e gerenciada pelo *broker* para posterior recuperação de clientes interessados. As Filas de Mensagens no AMQP também têm atributos como o nome e durabilidade, pode ser exclusiva ou compartilhada, apagada automaticamente, além de metadados associados na declaração. Selecionando propriedades adequadas, as Filas de Mensagens podem oferecer suporte para a interação entre produtores e consumidores com características específicas:

- Armazena e encaminha (*Store and forward*). Armazena temporariamente as mensagens encaminhadas pelos produtores e as distribui para consumidores registrados de forma circular. Geralmente são criadas para longos períodos e compartilhadas por vários consumidores.

- Resposta privada. Recebe as mensagens dos produtores, e as retém até encaminhá-las para um único consumidor. Geralmente são temporárias, com nome atribuído pelo servidor e privativas para um único consumidor.

- Subscrição privada. Coleta mensagens de vários produtores, subscritos previamente, e as encaminha para um único consumidor, responsável pelas subscrições.

As possibilidades de configuração e a API do AMQP são amplas. Conforme mencionado, a criação de *Filas*, seleção de *Exchanges* e declaração de *Bindings*, além de várias outras ações, têm suporte da API. O leitor é orientado a consultar as referências para avaliar as opções. Combinações de *Exchanges*, Filas de Mensagens e configurações de roteamento e mensagens de confirmação *ACK* também são possíveis. Na documentação do *broker* RabbitMQ [72], signatário da versão 0-9-1 do AMQP, são apresentadas algumas destas combinações.

Clientes consumidores utilizam a família de métodos *consume* e *deliver* da API do AMQP. Podem também utilizar métodos do tipo *push*. Para usar esta API o consumidor

deve se registrar na fila de interesse. Quando uma mensagem for colocada na fila, o *broker* entrega a mesma diretamente para o consumidor. Uma fila pode ter mais de um consumidor interessado. Cada registro ou subscrição tem uma identificação chamada *consumer tag*. Também é possível buscar uma mensagem usando métodos do tipo *pull*. Clientes produtores utilizam a família de métodos *put* e *publish*. Variações de operação para estas primitivas são previstas para que os clientes usufruam das diferentes configurações do *broker*.

O AMQP, como acontece vários outros MOMs, é flexível e robusto. Para isso, é necessária a execução de muitos elementos de suporte no servidor, e o consumo de recursos computacionais é inevitável. O AMQP pode ser usado de forma federada para oferecer escalabilidade, adicionando mais complexidade ao suporte. Este tipo de de suporte é geralmente necessário em grandes sistemas de informação, orientado à negócios, *enterprise*. O IBM MQ, utilizado em sistemas corporativos e o Java Message Service, JMS, que pode ser integrado a sistemas com suporte do Enterprise Java Beans, são alguns exemplos de MOM.

No contexto de IoT, o AMQP não vai ser utilizado diretamente no acesso à dispositivos com poucos recursos ou com interfaces de comunicação de baixa capacidade, como ocorre com o CoAP ou o MQTT. Bibliotecas AMQP ainda não estão amplamente disponíveis para o Arduino, por exemplo. Por outro lado, a confiabilidade e robustez do AMQP pode ser empregada com vantagens para trazer ou levar requisições, respostas e informações para dispositivos com mais recursos, como *smartphones*, ou ainda para organizar o registro de eventos e demais informações coletadas de sensores em um serviço na nuvem. O cenário da Figura 2 - ilustra este ponto.

2.6 **XMPP**

O *Extensible Messaging and Presence Protocol*, XMPP [9] é também um sistema orientado a fila de mensagens. Ele oferece um protocolo para troca de mensagens que usa XML para comunicação, o que inclui mensagens instantâneas, como o AMQP, e também suporte para os conceitos de presença e colaboração.

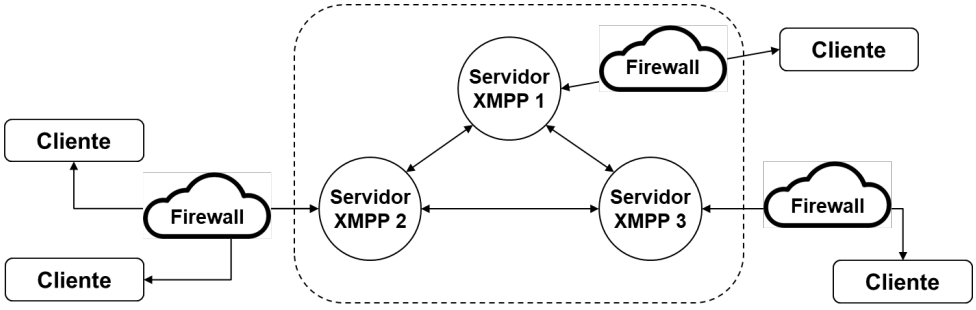
O XMPP utiliza o TCP como protocolo de transporte e suporta vários estilos de comunicação, sendo os principais o *request-response* e o *publish-subscribe*. A camada SSL/TLS pode ser usada para comunicação segura entre os elementos. Para ter acesso

ao servidor, os clientes precisam estabelecer uma conexão e se autenticar.

O projeto original do XMPP era chamado de *Jabber*, com sua primeira versão lançada em 1999. O *Jabber* foi originalmente projetado para uso em aplicativos de mensagens instantâneas, sendo adaptado aos poucos, para utilização em outros cenários. O protocolo original teve seu nome modificado para XMPP. Atualmente, o XMPP tem sido utilizado para atender a diferentes requisitos não funcionais, como funcionamento em navegadores, utilizando extensões WebSocket 2.7, além de aplicações para IoT. É um protocolo aberto e padronizado desde 2011 pela Internet Engineering Task Force [73].

O XMPP é utilizado por aplicativos de Mensagens Instantâneas, *Chat* em Grupo, Jogos, Geolocalização e voz sobre IP. O WhatsApp, por exemplo, utilizava o XMPP em suas primeiras versões. O Google Talk também utiliza o suporte do XMPP.

Figura 10 - Servidores XMPP formando uma federação.



Sendo um sistema de mensagens, o XMPP também requer elementos gerenciadores de filas, chamados servidores de mensagens XMPP. O XMPP pressupõe uma rede federada de servidores, como mediadores no encaminhamento das mensagens. Isso permite que clientes distribuídos em redes protegidos por *firewalls* separados comuniquem-se uns com os outros. Cada servidor controla seu próprio domínio e autentica usuários nesse domínio. Os clientes de um domínio podem se comunicar com clientes em outros domínios através da federação. Os servidores XMPP estabelecem uma malha de conexões, de maneira segura, para trocar mensagens entre seus domínios. A Figura 10 - ilustra este aspecto.

A autenticação de um cliente é feita usando uma arquitetura baseada em autenticação simples e camada de segurança (SASL) ou TLS/SSL.

A identidade de cada cliente é chamada de endereço XMPP ou *Jabber ID* (JID). Este identificador é semelhante à combinação de um endereço de e-mail e uma URL. Por exemplo, um usuário com o nome “aluno” executando a aplicação “protocolosIoT” em “lcc.uerj.br” terá o seguinte JID:

`aluno@lcc.uerj.br/protocolosiot`.

Da mesma forma, recursos compartilhados, como salas de discussão, também são representados por JIDs. Por exemplo, “reuniao@lcc.uerj.br” representa uma discussão “reuniao” que está disponível no endereço “lcc.uerj.br”.

A comunicação XMPP consiste de fluxos bidirecionais de fragmentos XML chamados de *stream* e *stanza*, respectivamente. O *stream* é um contêiner para o troca de elementos, ou *stanzas* XML entre duas entidades quaisquer em uma rede. São especificadas três tipos de *stanza*:

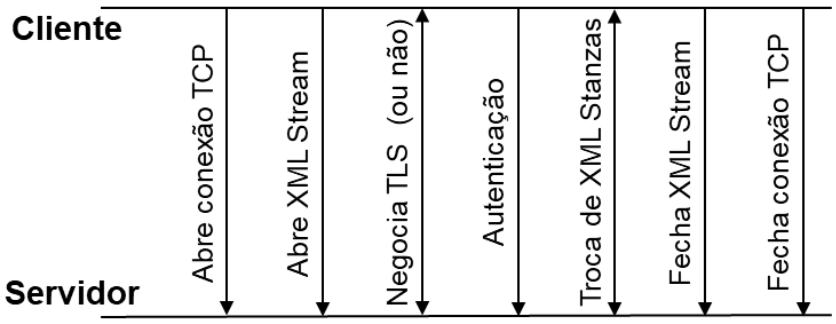
- *Presence*. Utilizado para enviar informações sobre o próprio usuário para outras partes autorizadas e interessadas. Um exemplo é o anúncio para estas entidades sobre sua disponibilidade ou presença (conectado ou desconectado). Este é um diferencial em relação ao modelo de serviços do AMQP.

- *Message*. Utilizado para enviar mensagens assíncronas (sem aguardar resposta) para um determinado receptor.

- *IQ* (*info-query*). Utilizada para enviar uma requisição que requer uma resposta da outra entidade, interação *request-response*. Existem tipos de *stanza* IQ pré-definidos: *get*, *set*, *result* e *error*. É obrigatório o uso de um campo *id*, para permitir a correlação de requisições e respostas, como ocorre com o *token* do CoAP.

Um fluxo típico de *stanzas* é apresentado na Figura 11 - . Um *stanza* é enviado para um destinatário e este a responde de forma apropriada. O processo de comunicação se dá da seguinte forma: (i) um cliente abre uma conexão com o servidor XMPP e inicia um *stream* XML; (ii) em seguida, existe a negociação de segurança e a autenticação; (iii) *stanzas* do tipo *Message* são, então, trocados e (iv) por último, o *stream* XML e a conexão são fechadas.

Figura 11 - Troca de mensagens em um fluxo XMPP.



As bibliotecas de desenvolvimento XMPP, de forma geral, encapsulam a criação e manipulação dos *stanzas* XML, necessários para comunicação entre as entidades. Os *stanzas* gerados são simples. O Código 2.1 apresenta um *stanza* do tipo *Message*, com os elementos necessários.

```
1 <message xml:lang='en'
2   to='aluno@lcc.uerj.br'
3   from='alexszt@ime.uerj.br/dicc'
4   type='chat'>
5   <body>Podemos trabalhar no texto hoje?</body>
6 </message>
```

Código 2.1*Stanza* XMPPdo tipo *Message*.

O exemplo no Código 2.2, mostra um *stanza IQ* com uma requisição, e um *stanza IQ* com a resposta retornada pelo servidor XMPP. A requisição solicita ao servidor uma lista de recursos suportados. Esta funcionalidade é útil para o cliente realizar verificações antes de tentar utilizar algum serviço do servidor XMPP.

O *stanza* de requisição tem tipo *get* (linha 5) e precisa de um identificador único (linha 3). O identificador é usado para correlacionar a resposta/confirmação com a requisição. A solicitação específica é descrita em uma *query* qualificada por um *namespace*, neste caso, referenciado por *disco#info*, indicando um pedido de informações ao serviço de descoberta (linha 6). Outros tipos de solicitação e extensões estão disponíveis.

```
1 <!--Requisição-->
2  <iq from="roberto@lcc.uerj.br"
3    id="123456"
4    to="lcc.uerj.br"
5    type="get">
6    <query xmlns="http://jabber.org/protocol/disco#info"/>
7  </iq>
8  <!--Resposta-->
9  <iq from="lcc.uerj.br"
10   id="123456"
11   to="roberto@lcc.uerj.br/6bcfui1lpj"
12   type="result">
13   <query xmlns="http://jabber.org/protocol/disco#info">
14     <identity name="Openfire Server" category="server" type="im"/>
15     <identity category="pubsub" type="pep"/>
16     [...]
17     <feature var="http://jabber.org/protocol/pubsub#subscribe"/>
18   </query>
19  </iq>
```

Código 2.2*Stanza* XMPP *IQ* dos tipos *get* e *result*.

A resposta é retornada pelo servidor XMPP em um *stanza IQ* do tipo *result* (linha 12), com informações de suas capacidades. O mesmo identificador deve ser usado, linha 10, para permitir que o cliente correlacione a resposta com a requisição, dado que as mensagens em um fluxo XML são independentes. Observamos na resposta que a extensão *publish-subscribe* está disponível no servidor (categoria *pubsub*, linha 15), permitindo ao cliente tomar a decisão de usar este estilo de interação.

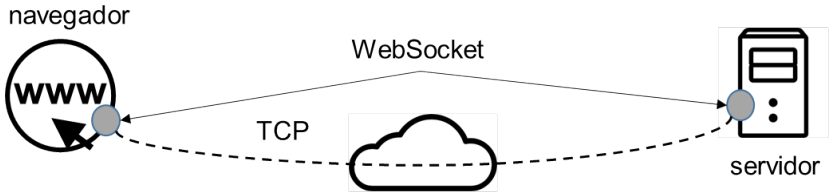
O XMPP fornece um protocolo aberto, fácil de usar, flexível e extensível. Isso faz com que a comunidade que utiliza o protocolo, crie uma série de extensões. Existe um fórum chamado XMPP Standards Foundation (XSF), que publica um conjunto de extensões, que são revisadas e discutidas, garantindo a interoperabilidade. Essas extensões são chamadas de XMPP Extension Protocols (XEPs) [74]. O XMPP incentiva e reúne as extensões relacionadas à IoT em `http://www.xmpp-iot.org/`.

2.7 **WebSocket**

O WebSocket [10] é um mecanismo similar ao *socket* "tradicional", proposto para oferecer suporte para troca de mensagens bidirecionais, *full-duplex*, sobre TCP, entre um navegador Web e um servidor HTTP remoto, que suportem HTML5 (Figura 12 -). Navegadores como, por exemplo, o Internet Explorer, Mozilla Firefox, Google Chrome,

Safari e Opera já possuem ou estão viabilizando suporte ao WebSocket.

Figura 12 - Comunicação com WebSocket.



O mecanismo WebSocket cria uma conexão que pode ser mantida aberta por um longo tempo, com pouco impacto no desempenho do navegador. É possível abrir e fechar conexões a qualquer momento. Com isso passa a ser também possível receber notificações, sem a necessidade de uso de "truques" como o *polling*, com vantagens na estrutura dos programas, melhor desempenho e latências menores.

O WebSocket tem características específicas em relação ao "socket" tradicional, dado que o contexto de uso é, em princípio, específico ao ambiente Web. Existe a preocupação de compatibilidade com o HTTP. Por isso, existe um protocolo WebSocket, padronizado pelo IETF [75] na RFC 6455, que gerencia o ciclo de vida de uma conexão WebSocket e permite, também, que uma conexão iniciada por HTTP chaveie para uma conexão WebSocket. Isso é feito através de uma mensagem HTTP *UPGRADE*.

Para o desenvolvimento das aplicações é proposta uma API de programação, padronizada pelo W3C [76]. A API para WebSocket abre a possibilidade para que aplicações para IoT possam também utilizar os navegadores como parte da solução para acionar diretamente um atuador ou obter uma informação de algum sensor.

O programador deve utilizar uma linguagem de programação Web, como JavaScript, e bibliotecas de suporte ao mecanismo WebSocket para desenvolver uma aplicação cliente executando em um navegador. Após abrir uma conexão TCP utilizando WebSocket, o programa pode enviar ou receber mensagens contendo qualquer tipo de informação. É possível criar um protocolo customizado para uma aplicação específica, mas também é possível transportar protocolos padronizados, como os Protocolos de Aplicação pesquisados nesta dissertação de mestrado.[3].

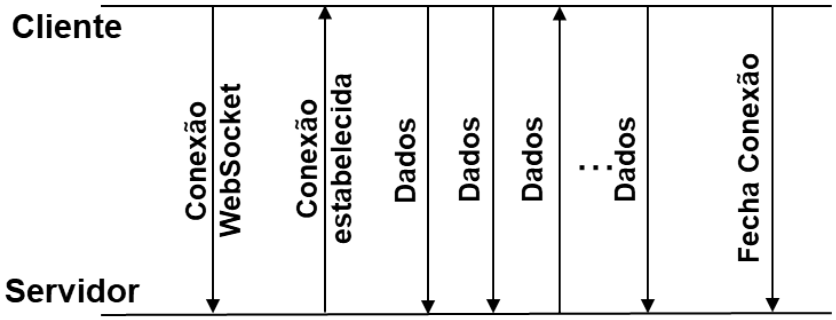
A API WebSocket requer que o programador descreva a referência, ou endereço, do servidor remoto através de uma URL, e informe um esquema específico *ws* ou *wss*

[3]O CoAP é baseado em UDP e, em princípio não pode ser transportado por TCP (Seção 2.3)

(para a versão segura), por exemplo: `ws://lcc.uerj.br:8000/`. A versão segura utiliza a camada SSL/TLS como no HTTPS.

O fluxo de interação entre o navegador e o servidor Web segue àquele utilizado por *sockets* TCP (Figura 13 -).

Figura 13 - Conexão, troca de mensagens, desconexão com WebSocket.



A API do mecanismo WebSocket oferece primitivas para a criação, conexão, envio-recebimento de mensagens e término da conexão. Também oferece alternativas para recebimento de eventos e integração com o HTTP e mecanismos do navegador.

2.8 **Comparação**

Concluindo a apresentação dos protocolos sob estudo, nesta seção, comparamos os protocolos apresentados de forma consolidada e complementamos o conjunto com algumas informações qualitativas, que serão utilizadas na avaliação de desempenho.

Na Tabela 2 - reunimos características gerais. Como curiosidade, temos o ano em que cada protocolo foi proposto. Ainda que o uso no contexto de IoT seja recente, as propostas não são tão novas. Todas os protocolos estão atualmente padronizados por algum órgão ou entidade ligada à computação, telecomunicações ou sistemas distribuídos. Também listamos os principais sistemas de suporte para os protocolos. Na seção seguinte utilizaremos alguns deles para construir os exemplos. Destaque para o RabbitMQ, que oferece suporte ao MQTT e ao AMQP. Na última coluna consolidamos os principais estilos de interação suportados por cada protocolo.

Tabela 2 - Características Gerais.

Protocolo	**Ano**	**Padrão**	**Bibliotecas**	**Interação**
HTTP	1997	IETF, W3C	Disponíveis para Java, CSharp, Javascript e outras	request-response
CoAP	2010	IETF, Eclipse Foundation	Esp-CoAP, Californium	request-response
MQTT	1999	OASIS, Eclipse Foundation	PubSubClient, RabbitMQ, Eclipse MQTT JS	publish-subscribe
AMQP	2003	OASIS, ISO/IEC	RabbitMQ	request-response, publish-subscribe
XMPP	1997	IETF, W3C	Jaxmpp2, OpenFire, Outras	request-response, publish-subscribe

A Tabela 3 - reúne algumas características ligadas à comunicação. Observa-se que o único protocolo que não utiliza o TCP é o CoAP. O CoAP foi projetado para ter pouco *overhead*, ou seja o menor impacto possível na comunicação. Entretanto por utilizar o UDP, é necessário um esforço maior para a configuração de *firewall* e *NAT*, por isso, a indicação direta do seu uso é em redes locais. Listamos, para rápida referência, as portas usadas por padrão em cada protocolo.

A coluna *Confiabilidade* indica como cada protocolo trata o requisito não-funcional de garantia de entrega das mensagens. Em alguns casos este requisito é tratado como QoS (o MQTT coloca explicitamente desta forma). Preferimos usar o termo *confiabilidade*, pois o termo *QoS* é geralmente usado para várias características não-funcionais ligadas à garantias de tempo de resposta, latência, prioridade e variação de atraso. O AMQP, por exemplo, pode oferecer tratamento prioritário para filas específicas — e isso poderia ser considerado suporte à QoS. Mas, isso não está incluído na coluna. A confiabilidade de entrega é apoiada em dois elementos: o uso do TCP, que tem características de garantia de entrega, controle de erro e controle de congestionamento, por exemplo, e na transmissão de mensagens de confirmação (tipo *ACK*), que devem ser verificadas pela aplicação.

Tabela 3 - Características de Comunicação.

Protocolo	**Transp.**	**Porta**	**Confiabilidade**	**Autent.**	**Segur.**
HTTP	TCP	80, 443 (TLS/SSL)	Confb. TCP	Sim	TLS/SSL
CoAP	UDP	5683, 5684 (DTLS)	Confirmado, Non-confirmable	Não	DTLS
MQTT	TCP [4]	1883, 8883 (TLS/SSL)	QoS 0 - At most once, QoS 1 - At least once, QoS 2 - Exactly once	Sim	TLS/SSL
AMQP	TCP	5672, 5671 (TLS/SSL)	Confb. TCP, mensagens ACK (versão 0.9.1)	Sim	TLS/SSL
XMPP	TCP	5222 (cliente), 5269 (servidor), 5223 (TLS/SSL)	Confb. TCP, mensagens ACK	Sim	TLS/SSL

Por último, características relacionadas ao controle de acesso e à privacidade. Com exceção do CoAP, os protocolos oferecem alguma forma de autenticação, restringindo o acesso à usuários (ou aplicações) que são autenticados. Os protocolos baseados em um elemento central como o *broker* permitem que serviços de terceiros para autenticação sejam utilizados. Para privacidade, opções de criptografia das mensagens são utilizadas.

Na Seção 4.4 do Capítulo 4 ampliamos esta discussão com observações a partir dos testes realizados.

[4] UDP com MQTT-SN

3 AVALIAÇÃO DE DESEMPENHO

Neste capítulo é apresentada a proposta de avaliação de desempenho dos Protocolos de Aplicação para IoT introduzidos no Capítulo 2. Os testes desenvolvidos permitiram comparar o desempenho dos Protocolos de Aplicação para IoT em relação às métricas propostas e discutir recomendações de melhor uso.

Nas próximas seções apresentamos os trabalhos relacionados, para justificar a avaliação de desempenho proposta, e em seguida apresentamos o cenário de teste. Com o estes pontos estabelecidos, discutimos as métricas, os testes realizados. Os resultados obtidos são apresentados no capítulo seguinte.

3.1 Trabalhos relacionados

Antes de desenhar as rotinas de teste e métricas utilizadas na avaliação realizada em nosso trabalho, investigamos trabalhos correlatos que também realizaram avaliações de desempenho de Protocolos de Aplicação para IoT em cenários controlados, destacando-se [77], [78] e [79]. Além de constatar os resultados obtidos, era de interesse conhecer os cenários utilizados e as respectivas justificativas de uso (por exemplo, o tipo de rede, número de dispositivos, domínio das aplicações que inspiraram o cenário de testes, etc.). Um último ponto de interesse era identificar se os trabalhos também avaliavam aspectos como segurança e mecanismos de confirmação, tais como os disponíveis no MQTT e no CoAP.

Em "*Performance evaluation of RESTful web services and AMQP protocol*" [77] realiza-se uma avaliação de desempenho entre uma solução baseada em arquitetura REST comparada com outra baseada em um MOM (*Message Oriented Middleware*), utilizando o protocolo AMQP. A métrica utilizada foi a quantidade de mensagens enviadas entre um cliente e servidor, em um determinado período de tempo. Além disso, o artigo apresenta uma descrição dos serviços *Web* e expõe seus problemas, sendo o mais relevante, o desempenho atrelado à grande quantidade de mensagens que são recebidas e tratadas por um *endpoint* de um serviço *web*. Assim, o autor propõe como solução, o uso do protocolo AMQP. O autor conclui ao final da avaliação de desempenho entre o HTTP e AMQP que, em um cenário com grandes quantidades de troca de mensagens, os melhores resultados são obtidos com a utilização do protocolo AMQP. Este trabalho está diretamente relacio-

nado a esta dissertação, uma vez que a métrica utilizada para a quantidade de mensagens, com a diferença da inclusão do uso de protocolos de segurança, neste caso o SSL/TLS.

No trabalho "*Comparison with HTTP and MQTT on required network resources for IoT*" [79], o autor faz comentários sobre a adoção do HTTP como o principal protocolo de aplicação utilizado para a transferência de dados, tanto em aplicações em geral como também para a Internet das Coisas. Em oposição ao HTTP, o autor escolheu o MQTT para a realização de avaliações de desempenho, uma vez que seu uso tem crescido em aplicações IoT. Como métrica, o autor utilizou de um número crescente de dispositivos simulados (10, 100 e 1000) e o tamanho das mensagem transmitidas, iniciando em 1 byte e crescente em intervalos 100000 até o máximo de 900000 bytes. O autor conclui que o uso do MQTT oferece vantagens em relação ao HTTP. Ao final, o autor propõe mudanças na estrutura do MQTT para melhorar ainda mais a eficiência na utilização dos recursos de rede. Esta dissertação também fará uma avaliação de desempenho entre o HTTP e o MQTT, em um ambiente não simulado, adicionando o uso de protocolos de segurança.

O artigo "*Performance evaluation of MQTT and CoAP via a common middleware*" [78] faz referência aos protocolos de aplicação MQTT (usando o protocolo TCP) e CoAP (usando o protocolo UDP), que segundo o artigo, são os protocolos mais usados em soluções IoT onde são empregados dispositivos com recursos computacionais limitados. O autor projetou e implementou um *middleware* que suporta ambos os protocolos de aplicação. Os resultados experimentais mostraram que o desempenho de diferentes protocolos depende de diferentes condições de rede. As mensagens do MQTT apresentaram baixa atrasos que o CoAP para menor perda de pacotes e maior atrasos que o CoAP para maior perda de pacotes. Além disso, quando o tamanho da mensagem é pequeno e a taxa de perda é igual a menos de 25%, o CoAP gera menos tráfego extra do que o MQTT para garantir uma transmissão confiável.

Até onde foi possível verificar nos trabalhos pesquisados, nenhum deles faz uma avaliação consolidada dos seis protocolos considerados nesta dissertação — o HTTP, CoAP, MQTT, WebSocket, AMQP e XMPP — com e sem segurança, com a verificação do *Round Trip Time*, variando-se a largura de banda disponível (Kbps) e o tamanho da mensagem. Também, é comum encontrar estudos que segmentam os estilos de interação, desassociando os protocolos com estilo *request-response* dos com estilo *publish-subscribe*. Nosso trabalho aborda pontos pouco explorados por outros trabalhos tratando-se de avaliação

de desempenho, realizando assim, um estudo amplo e ao mesmo tempo complementar dos protocolos da camada de aplicação para a Internet das Coisas.

3.2 Ambientes de avaliação e simulação para IoT

Para aplicações reais de IoT nos domínios de cidades inteligentes, saúde, automação residencial e outros, pressupõem-se o uso de milhares de sensores e atuadores. Para a realização de testes de desempenho, numa configuração similar, é necessária a utilização de simuladores, tendo em vista a grande dificuldade de se montar um ambiente controlado com centenas e/ou milhares de dispositivos.

Alguns trabalhos como [79] e [80] fazem o uso de simuladores *off-line* IoT, porém são restritos quanto ao suporte de protocolos que podem ser testados, geralmente contemplando apenas o HTTP, MQTT ou CoAP.

Algumas ferramentas como o Iotify [81] e o *Amazon device simulator* [82] oferecem simuladores de dispositivos IoT integrados à nuvem e que também fazem o uso dos protocolos de aplicação mais populares como o HTTP, MQTT e CoAP. O limite máximo de dispositivos simulados está diretamente associado à capacidade do serviço de nuvem contratado. Esta plataforma de software não dá suporte para o provisionamento de outros protocolos úteis em aplicações IoT, tais como o WebSocket, AMQP e XMPP.

Podemos constatar a ausência de testes e avaliações de desempenho em ambiente controlado com um conjunto mais amplo de protocolos de aplicação como os selecionados no presente trabalho (HTTP, CoAP, MQTT, AMQP, XMPP e WebSocket), o que diminuí o domínio de opções e recomendações de uso disponíveis para orientação a desenvolvedores e integradores de soluções IoT.

No artigo [83], os protocolos de aplicação SOAP, HTTP e MQTT são avaliados, medindo também o RTT (*Round Trip Time*) com 5, 10, 20 e 50 dispositivos enviando dados ao servidor em paralelo, com mensagens variando em 25, 100, 250, 500 e 1000 bytes. Não se aplicou restrição de banda de rede. O artigo em questão não testou os protocolos com a camada extra de segurança (SSL/TLS).

Inúmeras aplicações e soluções baseadas em IoT, atualmente implantadas e em operação, fazem o uso de redes 2G. É estimado que 30% dos dispositivos IoT conectados à internet, ainda utilizem esta rede [84]. Ambientes de alta latência e baixa largura de banda disponível são problemas esperados e devem ser tratados pelo provedor da solução

IoT. No presente trabalho, apesar de usarmos o Wi-Fi, foi possível o provisionamento de configurações de rede com baixa banda disponível, simulando assim, ambientes 2G.

Como ressaltado anteriormente, em consequência do avanço tecnológico e da ubiquidade de soluções IoT no cotidiano das pessoas, novos desafios também emergem. Por exemplo, segurança e privacidade são aspectos que requerem tratamento especial em IoT [85] [86]. Assim, foram previstos neste trabalho testes de desempenho acionando os mecanismos de segurança, quando disponíveis no protocolo (SSL/TLS, DTLS).

3.3 **Cenário dos Testes**

A infraestrutura da Internet é usada para prover a troca de informações e comunicação para realizar diferentes tipos de serviços, como monitoramento remoto, rastreamento, localização e outros, que podem estar associados a domínios diferentes, entre os quais podemos citar a gestão de cidades inteligentes, saúde, educação, automação industrial e residencial [87]. O cenário proposto por este trabalho, tem por objetivo chegar a uma configuração de ambiente próxima do real.

Inicialmente foi cogitado o uso de um simulador para a realização dos testes de desempenho, uma vez que no contexto de IoT é recorrente considerar milhares de sensores e atuadores comunicando-se entre si [88], [89],] [90] e [91]. Isso seria factível em um ambiente simulado.

Foram analisadas as ferramentas como o Packet Tracer, GNS3 (Graphical Network Simulator-3). O empecilho geral encontrado foi a indisponibilidade de módulos simulado uma boa parte dos Protocolos de Aplicação que objetivávamos avaliar. Por exemplo, o Packet Tracer, oferece suporte somente ao HTTP, enquanto o Network Simulator-3, apenas ao HTTP e MQTT. Apesar de os fornecedores oferecerem uma *API* para a criação de *plugins* e a possibilidade de se adicionar suporte a outros protocolos ao software de simulação, esta opção ficou inviável no momento, em função da complexidade da arquitetura e do tempo de desenvolvimento/adaptação que seria despendido. Assim, optamos por realizar os testes em ambiente real, com dispositivos reais.

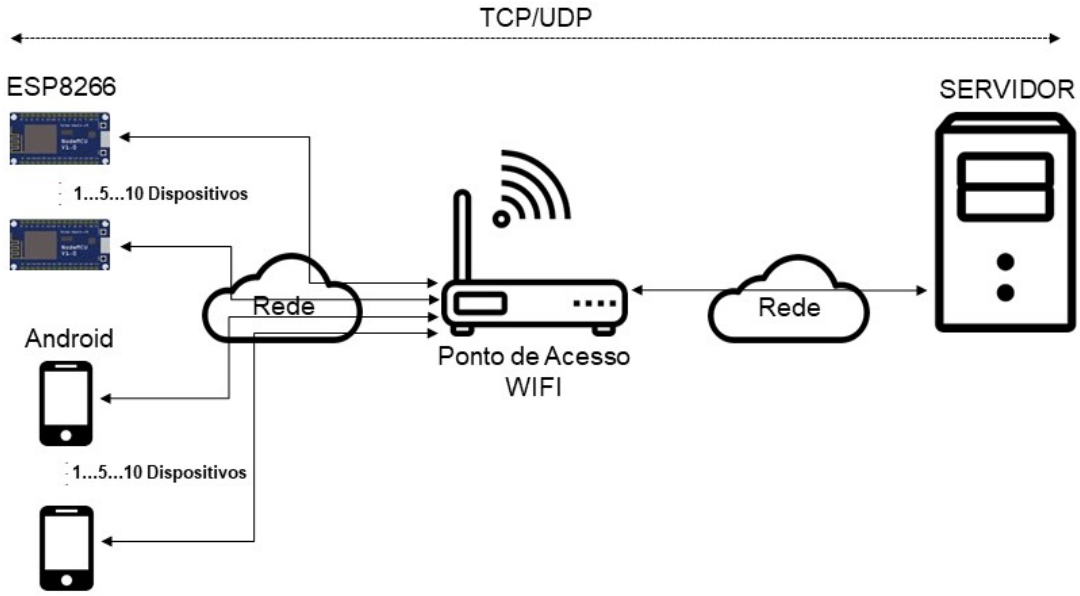
Para o ambiente de testes, o *Wi-Fi* foi escolhido por seu baixo custo de aquisição e implantação [92] [93] [94]. Para chegar em um ambiente mais próximo de redes 2G, foi utilizado o software NetLimiter [95] para o gerenciamento da banda de rede disponível.

Dispositivos e placas de prototipagem como o Arduino, NodeMCU ESP8266, Par-

ticle Photon, bem como dispositivos móveis baseados em Android, são, em sua maioria, compatíveis com este tipo de rede. Redes do tipo *Ethernet* foram descartadas do teste, uma vez que são poucos os dispositivos que o possuem o hardware integrado, o que requer na grande maioria dos casos, a aquisição de um componente em separado para que a comunicação via *Ethernet* seja possível de ser realizada.

A Figura 14 - ilustra a inter-relação dos componentes e o fluxo de comunicação nos experimentos realizados com os servidores locais da rede. Os dispositivos NodeMCU e os dispositivos móvel com o sistema operacional Android, transmitem mensagens aos servidores através de uma rede sem fio local. Os detalhes do equipamentos utilizados estão descrito na Seção 3.4.

Figura 14 - Cenário.



Não está no escopo do presente trabalho testes com outros tipos de rede sem fio como *Bluetooth*, *ZigBee* e *6LoWPAN*. Estas tecnologias de comunicação, apesar de viáveis para uso em aplicações para IoT, no que refere-se ao alcance da rede e velocidade máxima de transmissão de dados, possuem um alto custo de aquisição e implantação quando comparado ao Wi-Fi. Por esta razão, este trabalho de mestrado utilizou o Wi-Fi para a realização dos testes de desempenho.

Para tornar possível a realização dos testes com todos os protocolos de aplicação propostos neste trabalho, em um ambiente controlado, foi necessário o desenvolvimento de módulos de software específicos que pudessem ser executados nos dispositivos NodeMCU ESP8266 e nos dispositivos móveis com Android. Foram usados software já estabelecidos e

amplamente utilizados para o preparo de parte do ambiente para os testes de desempenho. Algumas destas soluções, como o RabbitMQ, Mosquitto e OpenFire, seguem os padrões do IETF, W3C e OASIS. Neste aspecto, adotamos a mesma estratégica utilizada em [79] e [80], que não utiliza ambiente simulados, focando em ambientes reais.

3.4 **Infraestrutura e Ambiente**

O ambiente para os testes foi estruturado com hardware e software (bibliotecas, *brokers* e ferramentas) específicos.

- *Hardware*
  - Servidor: Processador Inter Core i7, 16GB de Memoria RAM 1333 MHz SDRAM DDR3, Disco Rígido Western Digital 2TB Blue 5400 Rpm SataIII modelo WD20EZEX, Interface de rede Intel Ethernet 10/100/1000 Mbit/s.
  - Ponto de Acesso: TP-LINK modelo TL-WR741ND Wireless N 150Mbps.
  - Placa de Prototipagem: NodeMCU família ESP8266, CPU 32-bit RISC: Tensilica Xtensa LX106 rodando a 80 MHz, 64 KB de memória RAM de instruções e 96 KB de dados, Interface de rede IEEE 802.11 b/g/n Wi-Fi.
  - Dispositivo móvel: Motorola modelo Moto E Dual XT1022, Processador 1.2Ghz Dual-Core ARM Cortex-A7, 1GB LPDDR2 de memória RAM, Interface de rede IEEE 802.11 b/g/n Wi-Fi.
- *Software*
  - Bibliotecas
    - Californium [96] - Kit de desenvolvimento para uso do protocolo CoAP. O Framework é uma implementação de código aberto do CoAP escrito em Java. É mantido pela Eclipse Foundation. Este componente foi usado para o desenvolvimento do servidor CoAP. Atualmente,o Californium é a única biblioteca disponível para o CoAP desenvolvida em Java, sendo este, um dos motivos para sua escolha.
    - ESP8266WiFi [97] - Biblioteca desenvolvida em C de código aberto para o gerenciamento de conexões entre a placa NodeMCU e a rede Wi-Fi. Esta

implementação também possui classes nativas para a interação entre dispositivos usando o protocolo HTTP, sendo a mais importante, *ESP8266HTTPClient*.

  * WebSocket [98] - Biblioteca desenvolvida em C de código aberto para o protocolo/mecanismo WebSocket.
  * libcoap [99] - Biblioteca desenvolvida em C de código aberto do COAP. Entre outras disponíveis, o libcoap foi escolhido pela sua popularidade entre os desenvolvedores, além da documentação bem elaborada disponível para consulta.
  * XMPPNodeMCU [100] - Biblioteca desenvolvida em C de código aberto para o protocolo XMPP.

- Servidor/*Broker*
  * Sistema Operacional Windows 2019 Server *Enterprise Edition Trial Version* - O *Windows Server* [101] é uma família de sistemas operacionais da Microsoft baseado na arquitetura NT, com versões direcionadas para uso em servidores. Este sistema operacional para servidor permite que um computador possa operar serviços de rede, tais como servidor de impressão, controlador de domínio, servidor de web, servidor de arquivos, entre outros.
  * RabbitMQ [72] - É um servidor de código aberto que foi implementado pela Pivotal Software para suportar um protocolo de mensagens denominado AMQP. Também possui suporte nativo ao protocolo MQTT. Segundo Valeriu [102], este servidor tem melhor desempenho entre outros equivalentes, como o ActiveMQ e ApacheMQ. Estas características, motivaram a escolha do RabbitMQ para os testes do protocolo MQTT e AMQP.
  * OpenFire [103] - É um servidor de código aberto escrito em Java para uso do protocolo XMPP. Por ser o mais popular entre outros disponíveis [104], esta implementacao foi escolhida para os testes de desempenho desta dissertação.
  * Mosquitto [105] - É um servidor de código aberto que foi implementado pela Eclipse Foundation para suportar, exclusivamente, o protocolo MQTT. Este *broker* é leve e é considerado adequado para uso em todos os dispositivos, desde computadores de placa com baixo poder de processamento

e armazenamento até servidores mais modernos.

* OpenFire [103] - É um servidor de código aberto escrito em Java para uso do protocolo XMPP. Por ser o mais popular entre outros disponíveis [104], esta implementacao foi escolhida para os testes de desempenho desta dissertação.
* IIS [106] - É um servidor *web* desenvolvido pela Microsoft para diferentes sistemas operacionais. Tem por objetivo, prover serviços de hospedagem de *web sites*, *web services* e outras solucoes que podem ser disponibilizadas como serviços (*SaS*). Segundo o sitio na internet *netcraft* [107], o IIS é o segundo servidor mais usado na atualidade. Para este trabalho, o Internet Information Services foi escolhido para ser utilizado na avaliação de desempenho dos protocolos HTTP e WebSocket.

– *Ferramentas*

* Iperf [108] - Software desenvolvido no início dos anos 2000, pelo *National Laboratory for Applied Network Research/Distributed Application Support Team NLANDR/DAST* para medição da largura banda de redes de computadores cabeadas e sem fio, com suporte aos protocolos TCP e UDP através de injeção de tráfego. Ferramenta executada através da CLI (do Inglês, *Command Line Interface*), não possuindo interface gráfica. Tal aplicação funciona com base no modelo cliente/servidor, com o cliente iniciando as sessões e o servidor respondendo às solicitações de testes. O Iperf está disponível para diversos sistemas operacionais, tais como Windows, MacOS, Linux, Solaris, FreeBSD e OpenBSD.
* NetLimiter [95] - é uma ferramenta de controle e monitoramento de tráfego de redes desenvolvida pela Yours Locktime Software. Este software, executado somente através de interface gráfica, está disponível apenas para plataforma Windows. Entre as principais características do *NetLimiter* estão: controle total sobre a largura de banda de computadores e aplicações;estatísticas de longo prazo sobre o tráfego de rede; e bloqueio sobre conexões selecionadas.
* Clumsy [109] - é uma ferramenta de controle dos pacotes que chegam a uma interface de rede. Entre algumas funcionalidades, podemos citar as

mais comuns para ambientes de testes controlados, tais como controle do atraso (ms) e perda de pacotes (%).

* Wireshark [110] - é uma aplicação de controle do tráfego de uma rede e monitoramento de entrada e saída dedados do computador, com diferentes protocolos da rede à qual o computador está conectado. O Wireshark é a ferramenta mais utilizada para análise do tráfego de redes de computadores [111], por este motivo foi escolhido para o suporte da análise dos protocolos utilizados nos testes de desempenho e consumo de banda.
* Android Studio [112] - O Android Studio é um ambiente de desenvolvimento integrado (IDE) da Google, para desenvolver para a plataforma Android. Tem suporte para diversas linguagens de programação, tais como C, C++, Java e Kotlin.
* Visual Studio [113] - O Microsoft Visual Studio é um ambiente de desenvolvimento integrado (IDE) da Microsoft para desenvolvimento de software, com suporte à diversas linguagens de programação, tais como C,C++,C#,F# e Python. Também é um produto de desenvolvimento para a *web*, usando a plataforma do ASP.NET e ASP.NET Core, como sítios da internet,serviços web (API) e aplicativos móveis.

3.5 **Aplicações desenvolvidas para os testes**

Utilizando as bibliotecas mencionadas anteriormente, construímos aplicações reais para executar os testes desejados. Em alguns casos, como o MQTT, apenas o lado do cliente foi usado, mas em outros, como exemplo o WebSocket, foi necessário desenvolver módulos de software para os dois agentes (cliente e servidor). Não usamos clientes de teste genéricos para poder explorar as opções de cada protocolo.

Observamos durante a implementação dos agentes de software, mesmo usando bibliotecas e linguagens diferentes, a compatibilidade de parâmetros e funcionalidades de forma que poderiam ser comparadas em termos de parametrização, operações, *timeout* e carga útil (tamanho da mensagem). Também, tivemos que observar nas implementações os aspectos não-funcionais avaliados, ativando e desativando os mecanismos relacionados (criptografia e níveis de QoS, ou mecanismos de confirmação).

Chamamos os programas de agentes de software, de forma genérica. Um agente

para IoT é amplamente explorada no artigo [34].

Foram implementados agentes de software para cada protocolo de aplicação testado. Para alguns protocolos, como o CoAP, também se fez necessário o desenvolvimento de um agente adicional, neste caso, para ser executado no servidor, com o objetivo de viabilizar o recebimento, processamento e o envio de volta aos clientes das respostas das requisições previamente realizadas.

Para todas as aplicações implementadas para o dispositivo NodeMCU ESP8266, foi utilizado o "dialeto" C como linguagem de programação. Para os outros exemplos foi usada a linguagem Java.

3.5.1 CoAP

O código 3.1 apresenta o programa do cliente, utilizando a biblioteca *libcoap* [99]. Inicialmente, são configurados os elementos específicos, como por exemplo o Wi-Fi, sensores, atuadores e etc. Na linha 1 e 2 são informados o endereço de IP e PORTA do destinatário das mensagens. A linha 4 atribui à variável *msg*, um conteúdo que pode ter o tamanho variável entre 500, 1000 e 1500 bytes. Entre as linhas 5 e 11, está definido o *loop* que tem por objetivo o envio de mensagens ao servidor CoAP. Podemos observar que a linha 9 **inicia** a contagem do tempo do início do envio da mensagem ao servidor. O método *millis*, retorna o número de milissegundos passados desde que a placa NodeMCU começou a executar o programa. Na linha 10, é feito a chamada do método *post* que envia mensagem *POST* ao servidor. Outras opções seriam *GET*, *PUT* e *DELETE*. Por padrão, a biblioteca *libcoap*, utiliza as mensagens do tipo *CON*, com confirmação. Assim, entre as linhas 13 e 18, é mostrado a função de *callback*, que faz o tratamento do *ack* enviado de volta pelo servidor. Pode-se observar nesta função que assim que o *ack* é recebido a contagem do tempo é **finalizada**. O tempo total é gravado no *log* da IDE do Arduino.

A biblioteca *libcoap* também possui parâmetros que podem ser modificados e assim possibilitar o envio de mensagens do tipo *NON*, porém mensagens com esta característica não garantem confiabilidade na entrega. São requisições do tipo *fire and forget.*

```
1 //endereço IP e porta do servidor CoAP
2 IPAddress ip(192,168,1,3);
3 int port = 5683;

4 //callback
5 void callback_response(coapPacket &packet, IPAddress ip, int port) {
6      contadorAckRecebido++;
7      Serial.print("Round Trip Time(ms): ");
8      long tempoRTT = millis() - startTime;
9      Serial.print(tempoRTT);
10     Serial.println("");
11     envia = 1;
12 }

13 void enviaMensagem()
14 {
15     String msg = var500bytes;
16     if(i < 500)
17     {
18         char __mensagem[msg.length()];
19         msg.toCharArray(__mensagem, msg.length());
20         //Início da contagem do tempo
21         startTime = millis();
22         //Envio da mensagem
23         int msgid = coap.post(ip,port,"Teste",__mensagem,strlen(__mensagem));
24         envia = 0;
25         }
26 }

27 void loop() {
28     if(envia == 1)
29     {
30         //Envia nova mensagem depois do recebimento do ACK
31         enviaMensagem();
32     }
33     coap.loop();
34 }
```

Código 3.1Exemplo de programa em loop cliente CoAP.

Para o agente do servidor CoAP, foi utilizada a biblioteca Californium [96], desenvolvida em JAVA. O código 3.2 apresenta de forma simplificada, o método responsável por iniciar um *endpoint* CoAP. Observa-se na linha 1 que um dos parâmetros do método é do tipo *boolean*, *tcp*. A biblioteca utilizada para este exemplo, em sua versão mais recente, já está preparada para utilizar o protocolo CoAP sobre o protocolo de transporte TCP, ao invés do UDP [114]. Este trabalho utilizou o CoAP com UDP em sua configuração padrão de operação. Entre as linhas 3 e 11, ocorre o *loop* que percorre todas interfaces de rede disponíveis, obtendo seus respectivos endereços IP e configura um *endpoint* CoAP,

na porta 5683.

Entre as linha 13 e 16, é definida a função que recebe a requisição *POST* do cliente, a processa e envia a confirmação de volta (linha 15). Existe a possibilidade, caso desejado pelo desenvolvedor, de utilizar uma técnica chamada de *piggybacking*, onde a confirmação é enviada juntamente com alguma outra informação. Neste caso, o método *respond*, poderia receber algum parâmetro. Para outros tipos de mensagens CoAP, como por exemplo GET, PUT ou DELETE, seriam implementados os respectivos métodos *handleGET*, *handlePUT* e *handleDELETE*.

```
1 private void addEndpoints(boolean udp, boolean tcp) {
2         NetworkConfig config = NetworkConfig.getStandard();
3         for (InetAddress addr : EndpointManager.getEndpointManager().getNetworkInterfaces()) {
4                 InetSocketAddress bindToAddress = new InetSocketAddress(addr, 5683);
5                 if (udp) {
6                         CoapEndpoint.Builder builder = new CoapEndpoint.Builder();
7                         builder.setInetSocketAddress(bindToAddress);
8                         builder.setNetworkConfig(config);
9                         addEndpoint(builder.build());
10                }
11        }
12 }

13 public void handlePOST(CoapExchange exchange) {
14        //respond to the request - Piggybacking if necessary
15        exchange.respond();
16 }
```

Código 3.2Exemplo de agente servidor CoAP.

3.5.2 MQTT

Para a implementação do agente de software cliente para o protocolo MQTT, foi usada a biblioteca Adafruit [115]. O Código 3.3 apresenta o programa em linguagem C para o NodeMCU como publicador MQTT.

Não foram detalhados trechos de importação de biblioteca, configuração do Wi-Fi e etc.

Entre as linhas 1 e 4 do programa, são informados o IP e porta do servidor MQTT. Na linha 5 é informado à biblioteca, os parâmetros configurados previamente, além do tópico onde serão publicadas as mensagens e o nível de confiabilidade da entrega da mensagem (QoS). Pode-se observar que o último parâmetro do método construtor é 1, o

que quer dizer que as mensagens enviadas por esta implementação de código, requer o recebimento da confirmação do recebimento da mensagem pelo servidor. Esta confirmação foi considerada como mensagem de *response* nos nossos testes. Existem outros tipo de confiabilidade para o MQTT, sendo eles o nível 0 e 2, que não foram utilizados neste trabalho e que são opções de trabalho futuro.

Entre as linhas 6 e 24, é realizado um *loop* de 500 envios de mensagens ao *broker* MQTT. Pode-se observar que entre as linhas 14 e 19, é feita uma checagem do recebimento da confirmação (*ack*) retornada pelo servidor. Em caso negativo a mensagem *failed* é registrada em um arquivo de *log*. Caso contrário, *OK*. Na linha 8 é atribuído à variável *msg* o conteúdo a ser transmitido ao servidor e na linha 9 é iniciado a contagem do tempo que resultará no RTT (*Round Time Trip*) da mensagem. Na linha 20, a contagem é finalizada e o resultado registrado no arquivo de *log* da IDE do Arduino.

Existem vários servidores/*brokers* que suportam o MQTT. O *broker* Mosquitto foi utilizado. Assim, não foi necessário o desenvolvimento de nenhum agente para atuar como servidor, nos testes deste protocolo.

```
#define AIO_SERVER      "192.168.1.3"
#define AIO_SERVERPORT  1883

WiFiClient client;

Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);

Adafruit_MQTT_Publish pub = Adafruit_MQTT_Publish(&mqtt, "/topic/test",1);

for(int i=0;i<500;i++)
{
    String msg = var500bytes;
    startTime = millis();
    Serial.print(i);
    Serial.print(" - ");

    char __mensagem[msg.length()];
    msg.toCharArray(__mensagem, msg.length());
    if (!pub.publish(__mensagem)) {
      Serial.println(F("Failed"));
    }

    elapsedTime =   millis() - startTime;
    Serial.print("Round Trip Time(ms): ");
    Serial.print(elapsedTime);
    Serial.println("");
}
```

Código 3.3Exemplo de programa em *loop* cliente MQTT.

3.5.3 HTTP

O código 3.4 apresenta o programa do cliente HTTP 1.1, utilizando a biblioteca ESP8266WIFI [97]. Inicialmente, são configurados os elementos específicos, como por exemplo o Wi-Fi, sensores, atuadores, que não mostrados no exemplo de código. A linha 9 atribui à variável *msg*, um conteúdo que pode ter o tamanho variável entre 500, 1000 e 1500 bytes. Entre as linhas 1 e 24, está definido o *loop* que tem por objetivo o envio de mensagens ao servidor HTTP ao servidor. Podemos observar que a linha 9, inicia a contagem do tempo do início do envio da mensagem ao servidor. Na linha 10, é feito a chamada do método *GET* que envia uma mensagem *GET* ao servidor. Outras opções seriam *POST*, *PUT*, *DELETE* e outras disponíveis no protocolo HTTP. Na linha 19, é calculado o RTT (Round Trip Time) e este é registrado no *log* da IDE do Arduino.

Não foi necessário o desenvolvimento de um agente servidor para o protocolo HTTP. Quando o servidor IIS é provisionado, um recurso HTTP vazio, já é disponi-

bilizado automaticamente na porta padrão (80), pronto para responder requisições do protocolo.

```
1 for(int i=0;i<500;i++)
2 {
3     startTime = millis();
4     Serial.print(contador);
5     Serial.print(" - ");
6     HTTPClient http;

7     http.begin("http://192.168.1.6");
8     http.addHeader("Content-Type", "text/plain");
9     String msg = var500bytes;

10    int httpCode = http.GET(msg);
11    if (!httpCode == 200)
12    {
13      Serial.print("Código do Erro:");
14      String payload = http.getString();
15      Serial.println(httpCode);
16    }
17    elapsedTime =   millis() - startTime;
18    Serial.print("Round Trip Time(ms): ");
19    Serial.print(elapsedTime);
20    Serial.println("");
21    http.end();
22 }
```

Código 3.4Exemplo de programa em *loop* cliente HTTP.

3.5.4 WebSocket

O código 3.5 apresenta o programa do cliente WebSocket. Inicialmente, são configurados os elementos específicos, como por exemplo o Wi-Fi, sensores, atuadores, que não mostrados no exemplo de código. Na linha 2 é informado o IP, porta e o recurso do servidor WebSocket. A linha 13 atribui à variável *msg*, atribui-se à variável *msg*, um conteúdo que pode ter o tamanho variável entre 500, 1000 e 1500 bytes. Entre as linhas 6 e 19, está definido o *loop* que tem por objetivo o envio de mensagens ao servidor HTTP ao servidor. Podemos observar que a linha 11, inicia a contagem do tempo do início do envio da mensagem ao servidor. Na linha 13, é feito a chamada do método *sendTXT* que envia a mensagem com o conteúdo da variável msg da linha 12 ao servidor. O processo é bloqueado até o recebimento da mensagem de retorno do servidor, confirmando o recebimento. Na linha 15, é calculado o RTT (Round Trip Time) e este é registrado no *log* da IDE do Arduino.

```
1 //IP e Porta do Servidor
2 webSocket.begin("192.168.1.5", 81, "/chat");

3 void loop() {
4     for(int i=0;i<500;i++)
5     {
6         Serial.print(i+1);
7         Serial.print(" - ");
8         //Início da contagem do tempo
9         startTime = millis();
10        String msg = 500bytes;
11        webSocket.sendTXT(msg);
12        //Fim da contagem
13        elapsedTime =   millis() - startTime;  do tempo
14        Serial.print("Round Trip Time(ms): ");
15        Serial.print(elapsedTime);
16        Serial.println("");
17    }
18 }
```

Código 3.5Exemplo de programa cliente WebSocket.

3.5.5 AMQP

O código 3.6 apresenta o programa do cliente AMQP. Inicialmente, são configurados os elementos específicos, como por exemplo o Wi-Fi, sensores, atuadores, que não mostrados no exemplo de código. Nas linhas 1 e 2 são atribuídas as suas respectivas constantes, os valores do IP e porta do *broker*. Na linha 5 é criada uma instância do método que realiza o envio de uma mensagem à uma fila no *broker*. Entre a linha 6 e 21, está definido o *loop* que tem por objetivo o envio de 500 mensagens ao servidor AMQP. Observa-se que a linha 9 inicia a contagem do tempo do início do envio da mensagem ao servidor. Na linha 14 é chamado o método que realiza o envio da mensagem ao servidor com o conteúdo da variável msg declarada na linha 8. O processo é bloqueado até que se receba a confirmação do recebimento da mensagem pelo servidor. Em seguida, na linha 17, é calculado o RTT (Round Trip Time) e este é registrado no *log* da IDE do Arduino.

```
1 #define AIO_SERVER      "192.168.1.3"
2 #define AIO_SERVERPORT  5672

3 WiFiClient client;

4 AMQP_Client amqp(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);

5 AMQP_Publish pub = AMQP_Publish(&amqp, "/fila/test");

6 for(int i=0;i<500;i++)
7 {
8     String msg = var500bytes;
9     startTime = millis();
10    Serial.print(i);
11    Serial.print(" - ");

12    char __mensagem[msg.length()];
13    msg.toCharArray(__mensagem, msg.length());
14    if (!pub.publish(__mensagem)) {
15      Serial.println(F("Failed"));
16    }

17    elapsedTime =   millis() - startTime;
18    Serial.print("Round Trip Time(ms): ");
19    Serial.print(elapsedTime);
20    Serial.println("");
21 }
```

Código 3.6Exemplo de programa cliente AMQP.

3.5.6 XMPP

O código 3.7 apresenta o programa do cliente XMPP. Inicialmente, são configurados os elementos específicos, como por exemplo o Wi-Fi, sensores, atuadores, que não mostrados no exemplo de código. Na linha é criada uma instância da classe XMPPClient, passando 2 os parâmetros de IP e porta do servidor ao construtor. Na linha 9 é chamado o método *connect*, onde são passados os parâmetros para autenticação no servidor, como o usuário, domínio, o recurso de interesse e a senha respectivamente. A linha 13 atribui à variável *msg*, um conteúdo que pode ter o tamanho variável entre 500, 1000 e 1500 bytes. Entre a linha 14 e 26, está definido o *loop* que tem por objetivo o envio de 500 mensagens ao servidor AMQP. Observa-se que a linha 19 inicia a contagem do tempo do início do envio da mensagem ao servidor. Na linha 20 é chamado o método que realiza o envio da mensagem ao servidor com o conteúdo da variável msg declarada na linha 13. O processo é bloqueado até que se receba a confirmação do recebimento da mensagem pelo servidor.

Em seguida, na linha 22, é calculado o RTT (Round Trip Time) e este é registrado no *log* da IDE do Arduino.

```
1 byte server[] = { 192,168,1,5 };
2 byte ip[]     = { 192,168,1,10 };

3 XMPPClient client(server, 5222);

4 void setup()
5 {
6     Serial.begin(115200);
7     //Conexão com o servidor XMPP
8     //Username, Server, Resource, Password
9     client.connect("roberto", "lcc.uerj.br", "labtestresource", "123456");
10 }

11 void loop()
12 {
13     String msg = var500bytes;
14     for(int i=0;i<500;i++)
15     {
16         Serial.print(i+1);
17         Serial.print(" - ");
18         //Início da contagem do tempo
19         startTime = millis();
20         client.sendMessage("roberto@lcc.uerj.br", msg);
21         //Fim da contagem do tempo
22         elapsedTime =   millis() - startTime;
23         Serial.print("Round Trip Time(ms): ");
24         Serial.print(elapsedTime);
25         Serial.println("");
26     }
27     client.close();
28 }
```

Código 3.7Exemplo de programa cliente XMPP.

Os códigos completos dos testes implementados para a avaliação dos protocolos estão disponíveis no repositório de código GitLab.[5].

3.6 **Métricas**

Com base nos trabalhos relacionados, foi selecionado um conjunto de métricas quantitativas, de escalabilidade e qualitativas para a avaliação dos protocolos de aplicação.

[5]https://gitlab.com/rmac737/codigodissertacaomestrado

3.6.1 Métricas Quantitativas

***Round Trip Time* (RTT):** O RTT, é a medida do tempo total iniciada pelo envio de uma mensagem um cliente - dispositivo IoT ou celular móvel - até o servidor/*broker*, o processamento nulo desta mensagem, a transmissão da mensagem de retorno para o dispositivo, até o recebimento desta mensagem. São incluídos na medida todos os atrasos de rede.

3.6.2 Métricas de Escalabilidade

Tamanho das mensagens: Os protocolos de aplicação foram avaliados quanto ao tamanho das mensagens, em bytes, enviadas entre um cliente o servidor. Os tamanhos de mensagens utilizados estão relacionados ao diversos cenários possíveis em aplicações relacionadas à internet das coisas, evitando-se assim, tamanhos que pudessem ultrapassar o MTU (*Maximum Transmission Unit*) de 1492 bytes padrão do TCP e Wi-Fi e assim, gerar mais de uma transmissão de pacotes para apenas uma mensagem.

Número de sensores: O número de sensores foi escalado até cinco, com o objetivo de se observar o comportamento dos protocolos em cenários distintos, o que envolveu a combinação com outras métricas, tais como o tamanho da mensagem e largura de banda disponível. Assim, foram montados ambientes típicos aplicados à Internet das Coisas. Os resultados obtidos das combinações das métricas são mostrados e exibidos no capítulo 4.

Largura de banda disponível: A Largura de Banda é a medida da capacidade de transmissão de um determinado meio, conexão ou rede, determinando a velocidade que os dados passam através desta rede. Neste trabalho, a escalabilidade do protocolo foi avaliada para algumas configurações de banda disponível, tendo como objetivo uma aproximação de ambientes IoT em um cenário real.

3.6.3 Métricas Qualitativas

Confiabilidade e QoS: Os protocolos de aplicação podem ter um aumento de confiabilidade usando alguma forma de confirmação de envio de mensagens, também chamado

de “nível de QoS” no MQTT, quando disponível, o que provê ou permite a construção de mecanismos para garantia a entrega dos dados e pode evitar duplicação de pacotes. Alguns dos protocolos estudados neste trabalho de mestrado possuem mecanismos de confirmação de entrega de mensagens de forma opcional. Nossa avaliação foi realizada com estes mecanismos sempre acionados, quando disponíveis nos protocolos, de forma a termos uma mensagem de resposta do servidor. Assim, as medidas de RTT e escalabilidade incorporam os impactos destes mecanismos.

Segurança: Nesta métrica avaliamos o impacto do acionamento da criptografia como mecanismo opcional dos protocolos, quando disponível. Para isso, medimos o RTT e a escalabilidade com os mecanismos de criptografia acionados.

3.7 Testes Realizados

Os testes foram realizados em um ambiente isolado, sem conexão com a Internet ou outros dispositivos de rede, para reduzir possíveis interferências no resultado. Para se chegar mais próximo de um ambiente real, onde o processamento local e a comunicação entre os dispositivos IoT são afetados por vários fatores, como por exemplo, a perda de pacotes e latência, algumas características de comunicação foram configuradas manualmente durante os experimentos. Por exemplo, os testes foram realizados com larguras de banda variando entre 20, 50 e 100 Kbps. Outro parâmetro manualmente alterado foi o tamanho das mensagens, variando entre 500, 1000 e 1500 bytes. Também foram feitos testes sem a restrição de banda para os protocolos MQTT e CoAP, que são os mais utilizados em dispositivos como Arduino e NodeMCU em aplicações IoT.

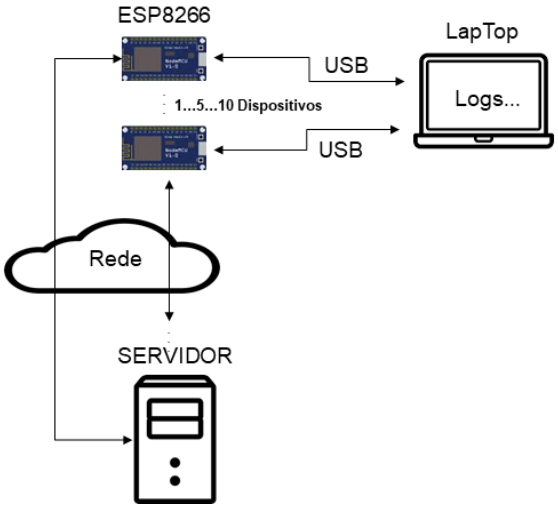
A ferramenta de análise de redes *Wireshark* foi usada no servidor, para permitir a visualização dos pacotes capturados na rede e confirmar informações acerca do tamanho, quantidade e sequência de pacotes trafegando e interações não esperadas. O software *IPerf* foi utilizado para se verificar a largura banda utilizada. Para isto, foi provisionado um agente no cliente e outro no servidor, para em seguida iniciar o sincronismo entre estes agentes para obtenção do valor de banda de rede disponível.

Nos testes sem a utilização da camada de segurança SSL/TLS, foram empregados dispositivos NodeMCU conectados em portas USB de um *notebook* atuando como cliente, como ilustrado na Figura 15 - . A IDE do Arduino foi utilizada para a coleta dos *logs*.

Na medida em que foram adicionados novos microcontroladores para a avaliação de escalabilidade, outras instâncias da IDE foram abertas e outras portas COM configuradas para obtenção dos *logs*. Foi a forma utilizada para realizar o experimento com mais de um microcontrolador ao mesmo tempo.

A Figura 16 - mostra o ambiente de testes montado no laboratório de ciência da computação do IME/UERJ. Nos testes com segurança, foram utilizados dispositivos móveis com o sistema operacional Android conectados à uma rede sem fio.

Figura 15 - Visão da coleta dos *logs* das placas NodeMCU ESP8266.



Em cada experimento, o número de 500 requisições foram feitas do dispositivo NodeMCU e do dispositivo móvel para o servidor. Para cada uma destas requisições, uma confirmação de recebimento da mensagem pelo servidor foi retornada ao cliente, para que só então este estivesse apto a enviar a próxima mensagem. Destas 500 mensagens, foram aproveitadas 100 requisições, sendo o índice com início em 250 e fim no 350. Desta forma, garantimos que todos os dispositivos, quando operando em paralelo, estavam transmitindo ao mesmo tempo. O campo de carga útil da mensagem dos protocolos de aplicação testados, foi carregado com uma cadeia de caracteres com uma representação *JSON*, simulando dados reais de sensores, como por exemplo, velocidade, dados de GPS (*Global Positioning System*), Altitude e Temperatura, como exemplificado no código *JSON* 3.8.

O tempo máximo de *timeout* foi estipulado em 30 segundos como padrão para todas as requisições com todos os protocolos. Ou seja, se a conexão não for estabelecida (protocolos que usam o TCP) ou a confirmação não seja recebida dentro da janela de tempo (30s), um erro deve ser registrado. Este valor foi definido com o objetivo de evitar

erros na transmissão das mensagens do cliente ao servidor, uma vez que o objetivo desta trabalho, não foi de realizar uma avaliação da taxa de erro na camada de aplicação.

Além disto, foi observado que, durante os testes, quando foram ativados 10 dispositivos enviando mensagens em paralelo, com banda restrita de 20Kbps, tamanho de mensagem em 1500 bytes e um tempo de *timeout* de 20 segundos, ocorria o erro de *timeout* na transmissão para todos os dispositivos. O tempo foi ajustado para 4 minutos, e com este cenário de 10 dispositivos a transmissão se tornava possível e cada mensagem levava cerca de quase 4 minutos para transmissão. Assim, ajustes foram feitos com o objetivo de tornar possível a transmissão de 100% das mensagens e sem erro de *timeout* em um tempo razoável.

```
1 [{'Id':1,'InfoSensor':'Temp:10C,Vel:90kmh,Alt:100m,CoordGPS:-10.0201020102012
2 ,-20.0201020102012'}, {'Id':1,'InfoSensor':'Temp:10C,Vel:90kmh,Alt:100m,
3 CoordGPS:-10.0201020102012,-20.0201020102012'},{'Id':1,'InfoSensor':'Temp:10C,Vel:90kmh,
4 Alt:100m,CoordGPS:-10.0201020102012,-20.0201020102012'},{'Id':1,'InfoSensor':'Temp:10C,
5 Vel:90kmh,Alt:100m,CoordGPS:-10.0201020102012,-20.0201020102012'},{'Id':1,'InfoSensor':'
6 Temp:10C,Vel:90kmh,Alt:100m,CoordGPS:-10.0201020102012, -20.020102010201215151050515 15'}]
```

Código 3.8Dados de exemplo de um sensor em formato JSON

Os protocolos avaliados possuem diferenças fundamentais e características próprias conforme descrito na seção 2. Por exemplo, o protocolo HTTP envia ao remetente uma mensagem de confirmação códigos de resposta, tais como o 200 (sucesso), 500 (erro no servidor) ou outros códigos especificados do próprio protocolo. A confirmação no protocolo CoAP ocorre com as mensagens do tipo ACK posto que, o protocolo de transporte utilizado pelo CoAP é o UDP. Para o MQTT definiu-se um QoS de nível 1, sendo a confirmação de responsabilidade do próprio MQTT. No caso do MQTT, existem outras opções de confiabilidade de entrega das mensagens, como por exemplo, o QoS de nível 0, onde a confirmação fica a cargo da camada de transporte utilizando-se do protocolo TCP. Com estas configurações, chegou-se a uma configuração global que pudesse equiparar o nível de confiabilidade e controle da entrega das mensagens entre um cliente e o servidor usando os protocolos de aplicação estudados.

Os testes de desempenho foram focados na comunicação entre o cliente e o servidor apenas, não levando em consideração a interação entre um cliente e algum outro cliente interessado na informação produzida por algum agente publicador-requisitor.

O RTT (*Round Trip Time*) foi calculado com base no envio de uma mensagem ao servidor e a posterior mensagem de retorno sobre o sucesso no recebimento.

Figura 16 - Ambiente real dos testes com os protocolos.

Para se obter um quadro de testes completo para todos os protocolos selecionados, quatro "dimensões" foram combinadas:

1. 6 protocolos (HTTP, CoAP, WebSocket, MQTT, AMQP e XMPP);
2. 1 métrica quantitativa (*Round Trip Time*);
3. 3 métricas de escalabilidade (tamanho de mensagens, número de sensores e largura de banda disponíveis);
4. 2 métrica qualitativas (confiabilidade e QoS) e uso sobre a camada de segurança;

Observa-se que para a métrica de escalabilidade as variações do tamanho das mensagens (de 500, 1000 e 1500 bytes), o número de sensores (de 1 a 5, em passos de 1) e largura de banda (20, 50, 100 Kbps e sem restrição para os protocolos MQTT e CoAP sem segurança), ainda trazem novas dimensões aos testes.

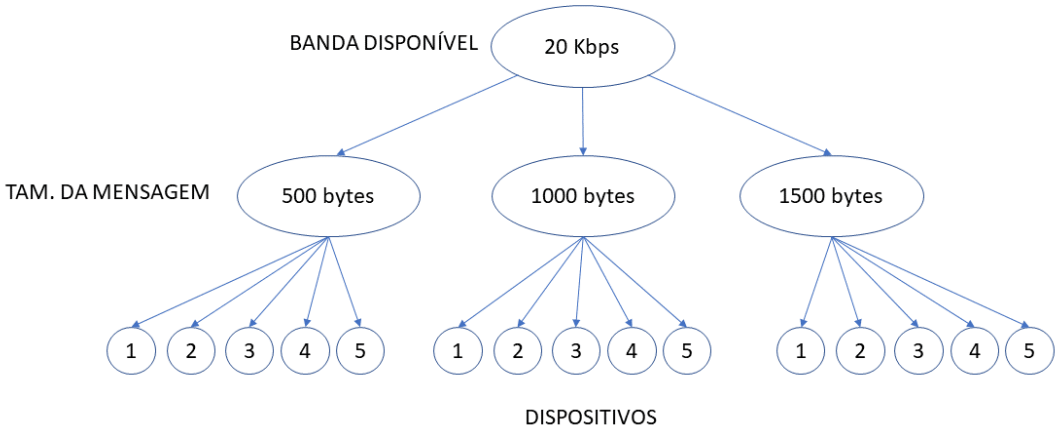
Combinar todas estas dimensões, levaria a realização de aproximadamente 13 mil testes, o que seria de gerência complexa e as informações obtidas poderia não ser relevantes para o conhecimento sobre os protocolos de aplicação ou para a tomada de decisão em relação ao seu uso. Neste contexto, para cada protocolo, realizamos uma sequência detalhada dos testes no seu melhor uso e depois uma série de testes complementares quando opções menos recorrentes são usadas, que ainda poderiam ser de claro interesse.

Cada teste foi realizado 500 vezes, sendo aproveitado os índices de 250 a 350 (100 medidas) e o resultado exibido apresenta a média e o intervalo de confiança de 95% considerando uma distribuição normal.

Foram realizadas várias combinações de testes que estão resumidos a seguir:

- *Largura de banda*: 20, 50 e 100 Kbps.
- *Tamanho da mensagem*: 500, 1000 e 1500 bytes.
- *Número de dispositivos*: 1, 2, 3, 4, 5.

Figura 17 - Quantidade de testes para um protocolo com banda de 20Kbps.



A Figura 17 - exibe uma ilustração da quantidade de testes realizados para uma configuração de banda de rede, neste caso de 20 Kbps.

Cada conjunto de testes foi repetido por mais duas vezes, para observar se as medidas estavam seguindo um padrão consistente.

3.7.1 Quantitativo

RTT (*Round Trip Time*). Uma mensagem foi preparada e enviada pelo agente de software. No momento do envio, inicia-se o contador de tempo do processamento. Este contador é parado, quando o agente de software recebe do servidor, a confirmação do recebimento da mensagem.

3.7.2 Escalabilidade

O teste do RTT da seção 3.7.1, para cada protocolo, foi realizado integrado ao teste de escalabilidade.

Tamanho de mensagens. A medida de latência realizada para mensagens com tamanho variável iniciando com 500, 1000 e 1500 bytes, em combinação com o número de dispositivos e a banda de rede. Com isso é observado como o protocolo se comporta em relação ao RTT (*Round Time Trip*) calculado para cada interação de cada dispositivo. O tamanho da mensagem foi estimado baseando-se em cenários reais de IoT, onde não ocorrem o tráfego de mensagens maiores do que 1500 bytes oriundas de dispositivos conectados à rede.

Número de dispositivos. Em combinação com o tamanho da mensagem e a banda de rede, o número de dispositivos foi escalado de um em um, até atingir cinco. Inicialmente, estava no planejamento, escalar os dispositivos de um em um até atingir o número máximo de 20. Porém, foi observado, durante os testes, que, em casos de transmissões simultâneas, com uma alta restrição de banda (ex., 20 Kbps) e mensagens de maior tamanho (ex., 1500 bytes), o tempo de *timeout* foi atingido para todas as mensagens dos clientes que estavam tentando o envio para o servidor. Desta forma, foi possível a realização dos testes com no máximo 5 dispositivos em concorrência no envio de dados ao servidor.

Largura de Banda da rede. A banda de foi fixada em 20, 50 e 100 Kbps. Os testes foram feitos em conjunto com o tamanho das mensagens e o número de dispositivos. Estas medidas se justificam, uma vez que estes se aproximam de um ambiente IoT em um cenário real, ou seja, redes com alta latência e vários dispositivos tentando a transmissão de dados concorrentemente.

3.7.3 Qualitativo

A análise qualitativa dos protocolos propostos neste trabalho de dissertação, se voltou para os testes da confiabilidade da entrega das mensagens e o desempenho sobre a camada de segurança, sendo o SSL para os protocolos que utilizam o TCP e o DTLS para os que usam o UDP, sendo este último, somente o protocolo de aplicação CoAP.

Confiabilidade e *QoS*. Algumas configurações de confiabilidade de alguns protocolos não foram explorados em sua totalidade. Um exemplo é o MQTT, que foi testado com o seu nível 1 de confiabilidade. O MQTT possui outros níveis (0 e 2) de confiabilidade que serão propostos para um trabalho futuro complementar à esta dissertação. Além disto, há também a confiabilidade na entrega dos protocolos que usam o TCP como protocolo de

transporte. A exceção ao TCP é o CoAP, que utiliza o UDP e a confiabilidade na entrega das mensagens fica a cargo da camada de aplicação. Nos testes, o protocolo CoAP não foi testado sem a opção da confirmação da entrega da mensagem ao destinatário. Os testes deste trabalho visaram uma taxa de entrega de 100% das mensagens em ambientes computacionais restritos (processamento e largura de banda de rede), por esta razão, configurações sem confirmação de entrega, não foram utilizados.

Segurança. Os testes dos protocolos sem o uso da camada de segurança, foram repetidos para todos os protocolos de aplicação sobre a camada de segurança. Os protocolos HTTP, WebSocket, MQTT, AMQP e XMPP utilizaram o SSL, e o CoAP usou o DTLS. Ainda não existem bibliotecas que implementam o uso de alguns protocolos sobre a camada de segurança, para execução em dispositivos, como Arduino e NodeMCU. Um exemplo é o protocolo CoAP, AMQP e XMPP. Assim, foi decidido usar um dispositivo móvel com o sistema operacional Android e bibliotecas já testadas e estabelecidas para a linguagem de programação Java foram usadas no desenvolvimento dos protótipos dos testes.

3.7.4 Testes Complementares

Os testes de desempenho foram feitos, para todos os protocolos, com largura de banda fixada em 20, 50 e 100kbps. Porém, alguns testes complementares se fizeram necessários em relação ao MQTT e CoAP, por estes serem os protocolos mais utilizados em ambientes com dispositivos com baixo poder computacional e rede de comunicação de alta latência. Assim, além do estrangulamento da banda de rede, testes adicionais foram feitos sem restrição de banda, com o objetivo de se observar o tempo de resposta do MQTT e CoAP diante de recursos ilimitados de rede. Posteriormente, com os resultados obtidos, média e intervalo de confiança calculados, foi feita uma comparação entre estes dois protocolos.

3.7.5 Ameaças à validade dos experimentos

Embora cuidados tenham sido tomados durante a avaliação, foram detectadas as seguintes ameaças à validade, segundo a taxonomia de:

Validade Interna. A validade interna diz respeito à influência da configuração do ambiente de testes nos resultados.

(i) Os testes dos protocolos com os dispositivos NodeMCU, foram feitos com estes conectados a USB de um computador com o Windows 10 e as rotinas para o envio do *log* à IDE do arduino, podem ter influenciado no tempo de processamento do dispositivo.

(ii) Todos os serviços de atribuição do servidor, como por exemplo o *broker* RabbitMQ, *OpenFire* e o *IIS* executaram localmente em um computador com o Windows Server 2019 e sem conexão com a internet. Assim, a implementação de cada um destes componentes por seus devidos fabricantes, pode ter influenciado nos resultados. É provável que a utilização de outros agentes de software, afetem o resultado do experimento.

(iii) A implementação de código utilizada para os testes com os protocolos de aplicação e os resultados do *RTT*, estão intimamente relacionados à implementação de cada biblioteca utilizada. Por exemplo, existem várias bibliotecas disponíveis para uso no NodeMCU que usam o MQTT como protocolo de comunicação, e que a escolha de alguma outra, que não seja a escolhida para uso nesta dissertação, pode afetar o resultado dos testes. Também, é possível que *bugs* no código ou nas bibliotecas usadas, também possa acarretar resultados distintos desta pesquisa.

Validade Externa. A validade externa define as restrições dos resultados quanto a sua generalização.

(i) os resultados apresentados estão relacionados as configurações e tecnologias escolhidas para a implementação do protótipo. Por exemplo, a linguagem de programação, sistema operacional, versão das bibliotecas e linguagens.

(ii) diferentes configurações de hardware teriam diferentes resultados. Foi utilizada a placa NodeMCU Esp8266. As bibliotecas utilizadas que implementam os protocolos de aplicação, utilização a memória RAM disponível para cache. Outras placa como por exemplo a Arduino UNO R3, poderiam afetar consos dispositivosos disositivoseravelmente os resultados. Em compensação, a placa NodeMCU Esp32, poderia obter medidas melhores, uma vez que possui mais memória RAM e processador de dois núcleos. A Tabela 4 - mostra em detalhes as diferenças.

Tabela 4 - Comparativos entre placas de prototipagem IoT.

	ESP32	**ESP8266**	**Arduino Uno**
Arquitetura	32 bits	32 bits	8 bits
Clock	160MHz	80MHz	16MHz
RAM	16MB	16MB	32KB
FLASH	512KB	160KB	2KB

Validade de Conclusão. Relacionada a fatores que podem levar um relacionamento, existente ou não, nas observações dos resultados do experimento.

(i) Foram gerados 100 amostras para cada cenário de teste. Além disto, para cada uma destas 100 amostrar foram repetidas 3 vezes para que se pudesse confirmar um padrão consistente. No entanto, erros humanos na manipulação dos dados na planilha excel utilizada para geração dos gráficos, podem tem influenciado os resultados.

Validade de Construção. Relacionado à precisão do objetivo das medições, considerando a construção do método utilizado para a a sua coleta.

(i) A função utilizada para a coleta do tempo de execução, pode ter sido influenciada pelo processamento interno e na persistência do dos dados na serial do computador, no caso das placas NodeMCU Esp8266 e na memória Flash, no caso dos dispositivos móveis.

4 RESULTADOS

Neste capítulo são apresentados e discutidos os resultados obtidos nos experimentos descritos no Capítulo 3. A fim de facilitar a comparação dos resultados, as avaliações quantitativas e qualitativas foram separadas em seções, e os protocolos foram separados em subseções.

Conforme mencionado, para cada protocolo apresentamos os resultados da métrica RTT *Round Trip Time* combinadas com as métricas qualitativas, considerando bandas de 20, 50 e 100 Kbps. O tamanho das mensagens variaram de 500, 1000 e 1500 bytes. Para cada largura de banda e tamanho de mensagem, foram adicionados 1 dispositivo até no máximo 5.

No trabalho [116] são apresentados diferentes tipos de tratamento estatístico com base no número de amostras coletadas, sendo elas:

(i) um número grande de amostras de medições de teste de desempenho quando n $\geq$ 30, e

(ii) pequeno quando n $<$ 30 onde n é o número de amostras.

Para este experimento foi utilizado o método estatístico para amostras classificadas como "grande", sendo usados a média do valor das amostras coletadas, o desvio padrão e um intervalo de confiança 95% para as amostras.

Para cada experimento, as aplicações desenvolvidas para a coleta das medidas foram iniciadas manualmente em cada dispositivo. Foram coletados 500 medições para cada dispositivo e utilizadas 100 amostras entre os índices 250 e 350, com o objetivo de garantir que a coleta de dados fosse feita com os dispositivos competindo entre si pelo acesso à rede Wi-Fi.

Durante o processamento de dados foi observado a existência de valores *outliers*, 100 ou mais milissegundos acima da média, que estavam deformando o desvio padrão, média e intervalo de confiança.

Os *outliers* não ocorriam de forma aleatória. Com apenas 1 dispositivo e com banda restrita de 20 Kbps, foi observado um intervalo de confiança próximo de zero, ou seja, poucos foram os casos de medidas com mais de 100 ms acima da média. A cada entrada de um novo dispositivo a concorrer pelo meio de transmissão, observa-se um

aumento dos *outliers*. Como o teste consistia de 500 medidas, e consideramos apenas as com índice de 250 a 350, possíveis *outliers* devidos a atividades de *warm-up* e *handshake* foram descartados. Além disto, com estas medidas, se pode garantir que 2 ou mais dispositivos estavam enviando mensagens ao mesmo tempo, uma vez que o envio em paralelo de mensagens pela rede, foi objeto de estudo deste trabalho.

Uma comparação entre o MQTT e CoAP é descrita com base no RTT. Além disto, são exemplificados cenários onde cada tem suas melhores aplicações em projetos IoT.

Os cenários foram repetidos para os testes com a camada de segurança habilitada (SSL/TLS e DTLS). Alguns gráficos são exibidos e o resultado descrito com bases nos resultados do RTT obtido.

Algumas particularidades foram observadas para alguns protocolos, como por exemplo o MQTT e o AMQP. Estas características são descritas nas seções 4.4.1 e 4.4.2.

4.1 ***Round Trip Time*** **e Escalabilidade**

Nas subseções seguintes são apresentados os resultados de RTT combinados com escalabilidade dos testes para cenários com largura de banda de 20, 50, 100 Kbps, tamanhos de mensagem de 500, 1000 e 1500 bytes e número de dispositivos crescentes de 1 a 5. Para facilitar a comparação entre os diferentes protocolos e explorar melhor o espaço dos gráficos, a escala do eixo Y foi padronizada diferentemente para cada banda e protocolo.

4.1.1 HTTP

A avaliação da métrica RTT para o HTTP foi feita sem a camada de segurança. No gráfico, ilustrado na Figura 18 - (a), observa-se a variação do número de dispositivos no eixo e o RTT medido para uma banda de rede de 20 Kbps, para os diferentes tamanhos de mensagens. As medidas do cenário com 1 dispositivo mostram uma diferença de tempo muito grande entre mensagens de tamanho diferente. Para 500 bytes, por exemplo, temos um tempo de 18ms e para mensagens de 1000 bytes, 291ms — mais de 1500%. Até 4 dispositivos concorrendo pela banda de rede, a diferença do RTT medido para bandas diferentes se manteve razoavelmente constante Foi observado também, para o intervalo de confiança considerado, a média não tem grande variação, o que mostra que as medidas estão próximas do valor médio obtido. Com 5 dispositivos enviando mensagens ao

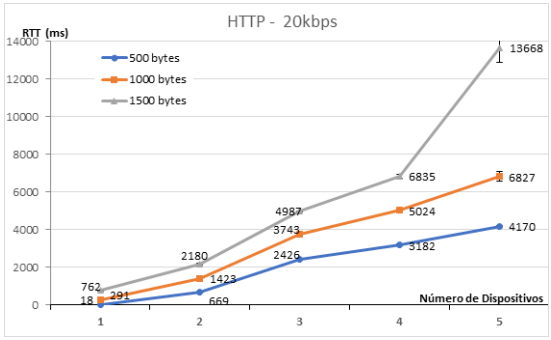
servidor concorrentemente, com tamanho de mensagens de 1000 e 1500 bytes, observa-se um aumento na variação da média. Isto mostra que as medidas de RTT estão um pouco distantes da média. Nota-se também que, especificamente para a mensagem de 1500 bytes, o RTT médio subiu consideravelmente, o que evidencia a grande concorrência dos dispositivos ao meio de transmissão, resultando em retransmissões e controles adicionais do protocolo TCP (camada de transporte).

O gráfico, exibido na Figura 18 - (b), mostra um cenário com banda de rede configurada em 50 Kbps. Observa-se que, com 1 dispositivo, foram obtidas medidas muito próximas. Com 2 dispositivos a porcentagem do aumento do RTT foi de cerca de 300% entre as médias. Com 3 a 5 dispositivos a porcentagem de aumento do RTT se manteve entre 34 e 55%, evidenciando um aumento próximo de linear de acordo com o número crescente de dispositivos para mensagens de 500, 1000 e 1500 bytes.

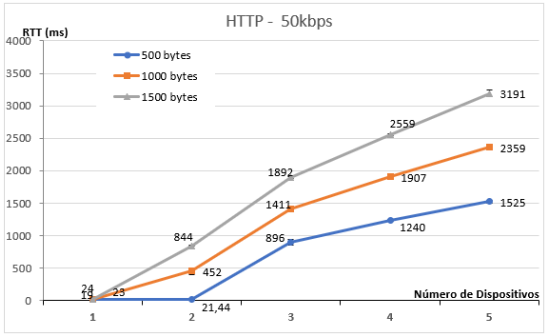
No gráfico da Figura 18 - (c), o comportamento dos valores medidos se aproxima da Figura 18 - (b), porém com interações mais rápidas, RTT menores, uma vez que a configuração da banda de rede foi menos restritiva.

Podemos observar que a restrição de banda de rede e a concorrência pelo meio de transmissão, pode levar a efeitos indesejáveis à aplicações IoT, como por exemplo lentidão e ocorrência de *timeout* na tentativa de envio de mensagens de um cliente ao servidor. Além disto, mais mensagens de controle do protocolo TCP, base para o protocolo HTTP, contribuem para piorar o desempenho.

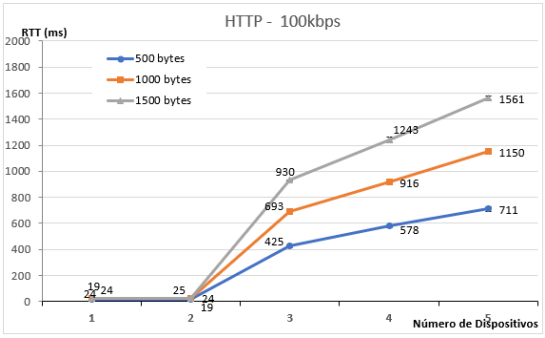
Figura 18 - RTT para o HTTP



(a) 20 Kbps



(b) 50 Kbps

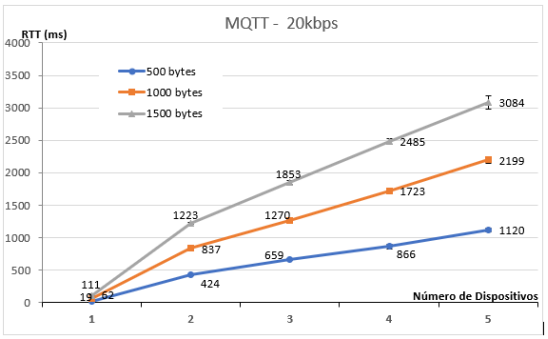


(c) 100 Kbps

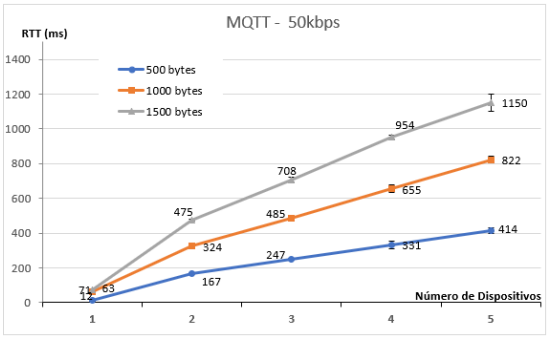
4.1.2 MQTT

Para protocolo de aplicação MQTT, sem a camada de segurança, e com a configuração de QoS 1 (confirmação de mensagem), os mesmos testes aplicados ao HTTP foram mantidos. Observa-se, o impacto do tamanho das mensagens e da quantidade de dispositivos no RTT calculado. Os gráficos ilustrados nas Figura 19 - (a), Figura 19 - (b) e Figura 19 - (c), mostram que, como esperado, que com uma banda de rede menos restrita, menores valores de RTT são obtidos. Observa-se também uma estabilidade nos valores medidos mesmo considerando-se bandas, tamanhos de mensagens diferentes e número de dispositivos operando.

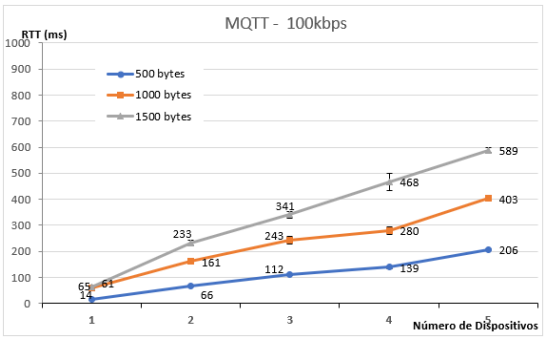
Figura 19 - RTT para o MQTT



(a) 20 Kbps



(b) 50 Kbps



(c) 100 Kbps

Comparado ao protocolo HTTP, o MQTT apresentou um RTT consideravelmente menor. Este resultado justifica-se, em parte, pelo fato de que o protocolo MQTT possui um cabeçalho menor. Entretanto, observa-se que a gerência de conexão do MQTT estabelece uma conexão com o *broker* uma única vez, e todas as interações são realizadas por este canal, ao passo que o HTTP para o NodeMCU estabelece uma conexão, com respectiva sequência de *handshake* entre cliente com o servidor, a cada interação.

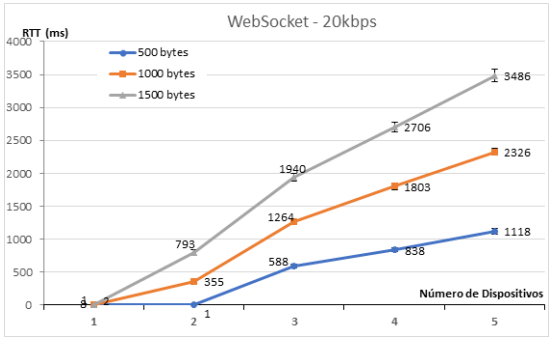
4.1.3 WebSocket

Como no protocolo MQTT, as medidas de WebSocket em relação ao HTTP também foram menores. Observa-se que o WebSocket não tem uma caeçalho específico de aplicação, sendo assim, mais eficiente. Os tempos médios de resposta do protocolo WebSocket, se aproximaram dos tempos do MQTT. Apesar de compartilharem características de desempenho semelhantes, eles possuem objetivos diferentes. O WebSocket foi projetado para funcionar como um mecanismo, semelhante ao *socket* tradicional. Por esta característica, este é um protocolo estabelecido e utilizado em soluções baseadas na Web e que preci-

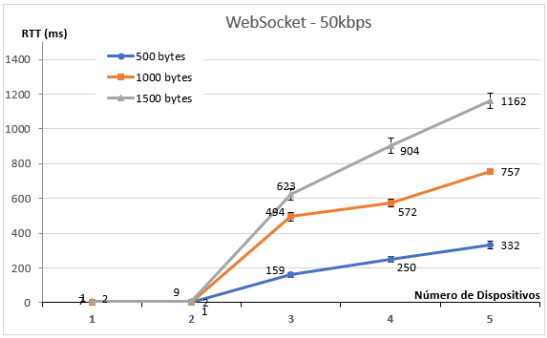
sam de mecanismos de comunicação eficientes, em tempo real no navegador, operando receptor de mensagens. É comum o emprego de WebSocket para transportar outros protocolos, como por exemplo *MQTT over WebSocket*, *XMPP over WebSocket* e outros. São inúmeras os projetos IoT que utilizam o protocolo/mecanismo *WebSocket* [117], [118], [119].

Verifica-se também que o cenário de 20 Kbps o WebSocket tem o desempenho ligeiramente pior do que o MQTT, excetuando-se o caso de apenas 1 dispositivo, o que não seria esperado, dado que os testes usaram diretamente o mecanismo (sem biblioteca relacionada à protocolo de aplicação). Por outro lado, para os cenários de 50 e 100 Kbps o WebSocket se mostrou ligeiramente mais eficiente, comparado ao MQTT. Esta diferença pode ser atribuída à gerência da conexão TCP, que poderia ser melhor parametrizada para o WebSocket, algo já consolidado pelas bibliotecas e *brokers* para o MQTT.

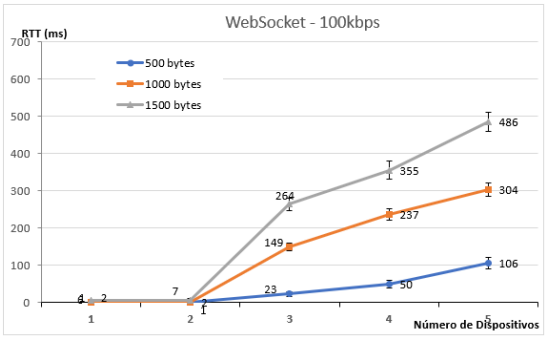
Figura 20 - RTT para o WebSocket



(a) 20 Kbps



(b) 50 Kbps



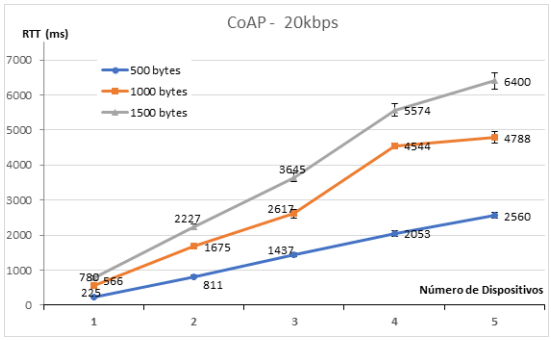
(c) 100 Kbps

4.1.4 CoAP

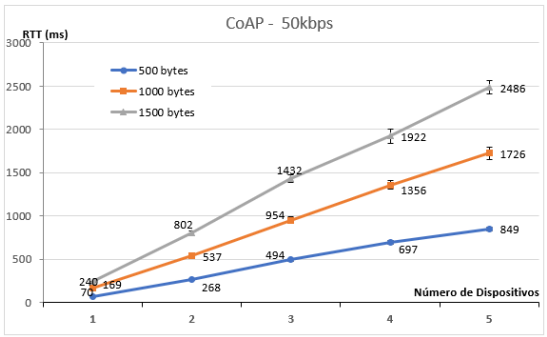
Para o CoAP, pode-se observar na Figura 21 - (a), que o protocolo, se comportou como o esperado apresentando um aumento do tempo de RTT medido quase linear com

o incremento do número de dispositivos e do tamanho da mensagem. Na Figura 21 - (b), já com banda de rede menos restrita, observa-se a diminuição do RTT. Na Figura 21 - (c), com banda de rede ainda menos restrita, 100 Kbps, o RTT obtido diminuiu consideravelmente em relação ao apresentado na Figura 21 - (a).

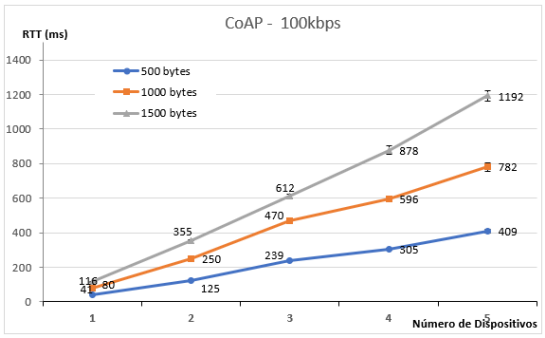
Figura 21 - RTT para o CoAP



(a) 20 Kbps



(b) 50 Kbps



(c) 100 Kbps

Em relação ao HTTP, o CoAP obteve melhor desempenho, o que se justifica, em parte, por seu cabeçalho menor. Além disto, o UDP também consome menos banda de rede por possuir um cabeçalho do segmento menor que o TCP e por não incorporar mecanismos de controle de erro e fluxo. Apesar da vantagem do CoAP em relação ao HTTP, o MQTT e o WebSocket obtiverem melhores medidas de RTT. O gráfico da Figura 24 - , que compara o desempenho de todos os protocolos avaliados para 20 Kbps e 1500 byes, os gráficos específicos das Figura 21 - (a), Figura 21 - (b) e Figura 21 - (c), evidencia que o CoAP possui um desempenho pior, mesmo em um ambiente restrito. Entretanto, destaca-se que para a realização das medidas de RTT seria necessário uma mensagem de resposta, para se fechar medida de tempo. Neste caso, utilizamos o mecanismo de confirmação do CoAP, que envia uma mensagem de confirmação ao cliente (mensagens CoAP do tipo *CON*) a cada mensagem recebida pelo servidor. Este mecanismo é parte

da biblioteca utilizada e está preparado para utilizar *piggybacking*. O que pode justificar o desempenho ruim.

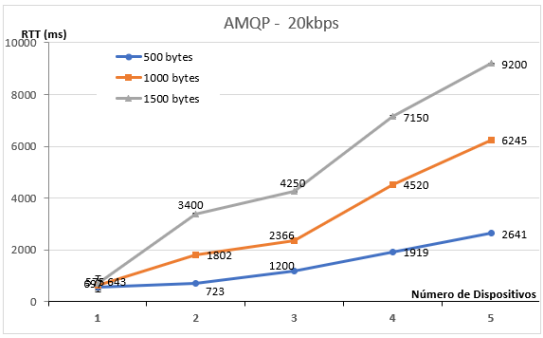
Utilizando-se a ferramenta Wireshark, verificou-se que para bandas restritas, 20 Kbps, por exemplo, módulo no papel de servidor, reenviava algumas mensagens *CON*, onerando a rede, fazendo com que o valores medidos de RTT aumentassem em média. Vale destacar que, caso os testes utilizassem mensagens CoAP do tipo *NON*, não seria gerado este tráfego adicional. Entretanto o perfil das medidas não seria o RTT.

Segundo [120], o uso do CoAP é preferível em relação a todos os outros protocolos, nos cenários onde não se faz necessária uma confirmação automática do recebimento das mensagens enviadas pelo cliente. Ou seja, quando interações assíncronas (*fire and forget*) são suficientes.

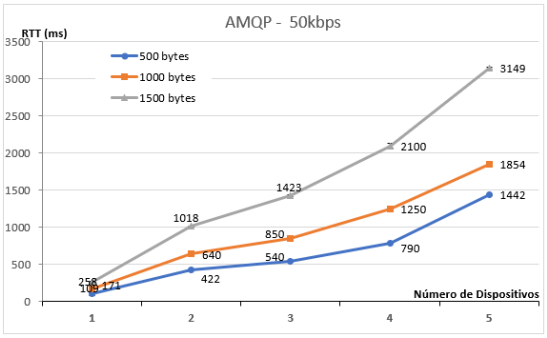
4.1.5 AMQP

Analisando os gráficos Figura 22 - (a), Figura 22 - (b) e Figura 22 - (c), observa-se que o RTT médio cresceu de acordo com a inclusão de novos dispositivos, enviando mensagens paralelamente ao servidor. Igualmente observado para os outros protocolos testados, chega-se a conclusão de que, quanto maior a banda de rede, menor é o RTT medido. O AMQP apresentou um intervalo de confiança próximo da média, o que sugere que o AMQP é um protocolo estável diante de ambientes de rede restritos. O RTT maior quando comparados aos protocolos HTTP, MQTT, WebSocket e CoAP, se justifica, pela quantidade de mensagens que são trocadas entre o cliente e o servidor, para o envio de uma mensagem. A Seção 4.4.2 detalha esta característica. O AMQP apresentou uma variação baixa na média dentro do intervalo de confiança com 5 dispositivos e banda de rede configurada em 20kbps. Esta observação sugere maior estabilidade e controle das mensagens trocadas entre o cliente e o servidor.

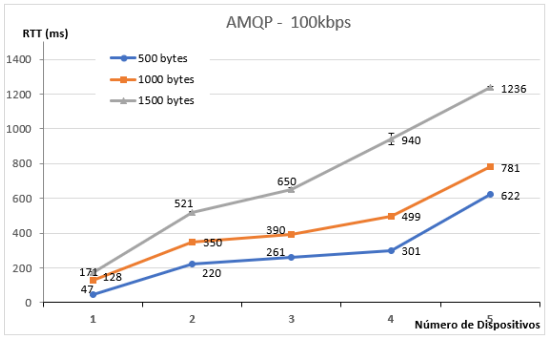
Figura 22 - RTT para o AMQP



(a) 20 Kbps



(b) 50 Kbps

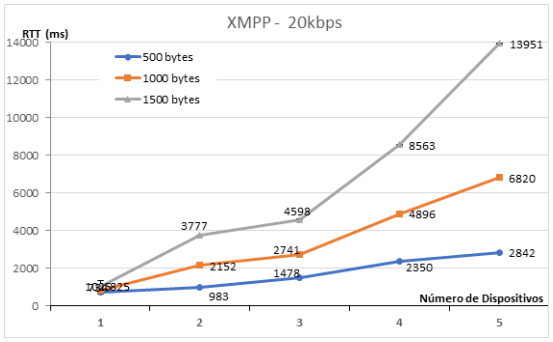


(c) 100 Kbps

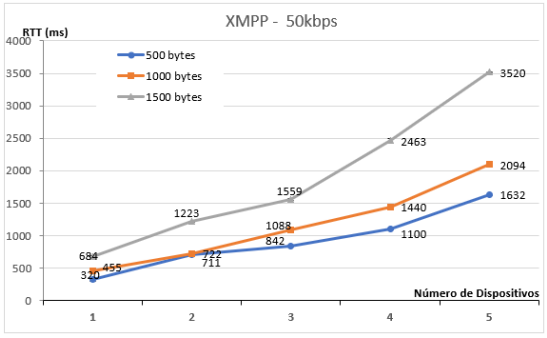
4.1.6 XMPP

Em análise ao gráfico Figura 23 - (a), podemos observar tendência no crescimento do tempo de resposta médio, de acordo com o aumento do tamanho da mensagem e inclusão de mais dispositivos. O XMPP foi o protocolo que mais apresentou variação no aumento do RTT para 4 e 5 dispositivos. Foi registrado um salto de de 140% entre o RTT de uma mensagem de 1000 bytes em relação a uma mensagem de 1500 bytes. Este aumento para o HTTP, foi de 63%. Esta medida, sugere que o XMPP, tal qual o HTTP, não é uma boa escolha para utilização em ambientes de rede com alta latência.

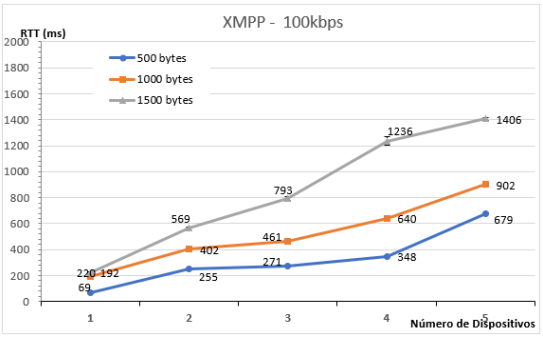
Figura 23 - RTT para o XMPP



(a) 20 Kbps



(b) 50 Kbps



(c) 100 Kbps

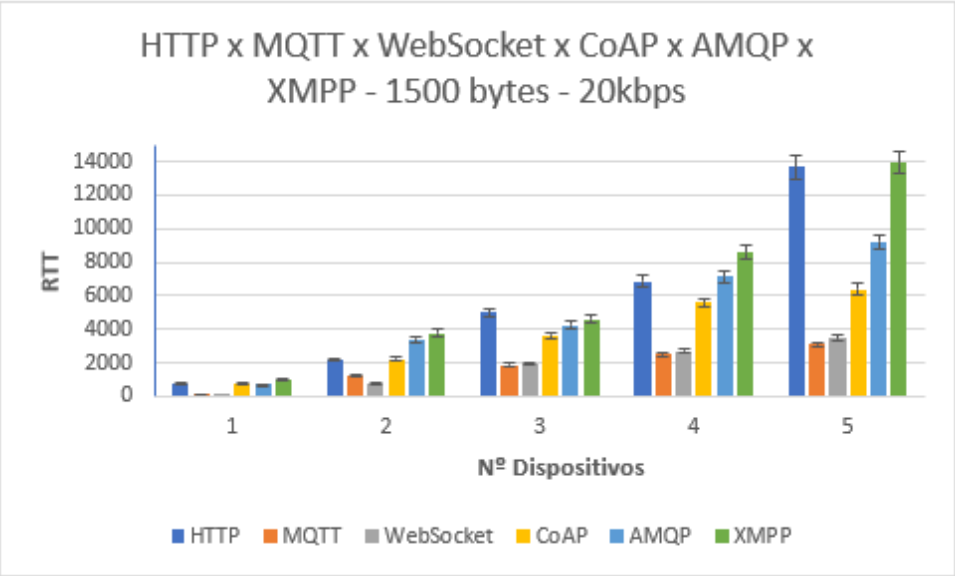
4.1.7 Resumo

Consolidando as medidas, a Figura 24 - , apresenta uma comparação do tempo de resposta entre o HTTP, MQTT, WebSocket, CoAP, AMQP e XMPP no cenário mais restrito dos testes, 20 Kbps de banda e mensagens de 1500 bytes. Observa-se que o XMPP obteve o maior RTT médio entre todos os protocolos testados. Este fato é justificado pelo tráfego constante de mensagens de controle em XML, além do uso de*tags*, também em XML, que não possuem valor, mas são enviadas para uma mensagem de confirmação (*ack*). Além disto, deve-se considerar o tempo de decodificação e processamento, tanto no cliente como no servidor.

O RTT alto para o protocolo HTTP, se justifica pela quantidade de *handshake* do TCP para cada requisição enviada pelo cliente ao servidor. O HTTP possui um *header*, chamado de *Keep-Alive*, que tem por objetivo, aproveitar conexão *socket* abertas para várias requisições HTTP. Porém, as bibliotecas disponibilizadas para placas de prototipagem, como o NodeMCU ESP8266, ainda não o suportam e-ou apresentam erros em sua utilização. Assim, o resultado do HTTP ficou em penúltimo lugar, atrás somente do

protocolo XMPP.

Figura 24 - RTT de todos os protocolos com 20 Kbps de banda de rede e 5 dispositivos



4.2 **Aplicando Confirmação**

Durante os experimentos realizados, todas as mensagens enviadas do cliente para o servidor, com e sem segurança, foram recebidas e respondidas pelo destinatário, independentemente do tamanho, do número de dispositivos e restrição da banda de rede. Ou seja, não ocorreram erros dentro do tempo *timeout* estabelecido de 30s, como descrito na Seção 3.7.

A razão para ausência de erros, é o uso das opções de confirmação habilitadas nos protocolos que oferecem tal característica. Além disso, a maioria dos protocolos utiliza TCP que possui mecanismos de controle de erro.

Por exemplo, os protocolos HTTP, MQTT, WebSocket, AMQP e XMPP usam o TCP como protocolo de transporte, o que garante confiabilidade na entrega da mensagem, independente do controle da aplicação.

A exceção é o protocolo CoAP, que usa o UDP como protocolo de transporte e não garante confiabilidade na entrega. Assim, o CoAP foi parametrizado para transmitir mensagens com controle de *ack*. O código da aplicação cliente precisa solicitar a confirmação ao destinatário. Na biblioteca utilizada o envio da mensagem de confirmação não requer a programação no lado servidor, mas está ligada ao nível de aplicação.

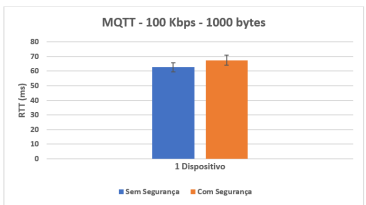
Como trabalhos futuros, podemos avaliar variações dos requisitos de confirmação do MQTT (QoS 0 e 2) e comparar o CoAP sem o mecanismo de confirmação, usado nos testes para manter-se a mesma rotina para todos os protocolos (ver a Conclusão).

4.3 **Aplicando Segurança**

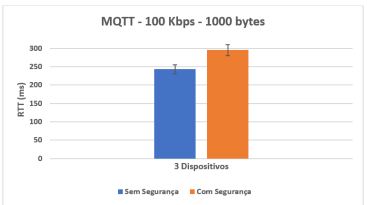
Os testes de RTT e escalabilidade foram repetidos com o uso da camada de segurança, sendo ativado a SSL, para os protocolos que usam o TCP, e o DTLS para os que usam o UDP (CoAP). Na sequência destacamos os resultados para o cenário com 100 Kbps de banda, a menos restritiva, e mensagens de 1000 bytes, tamanho médio, para configurações de 1, 3 e 5 dispositivos. O objetivo desta configuração foi avaliar a sobrecarga do mecanismo de segurança sem interferir muito nos sistemas de comunicação por conta do tamanho da mensagem ou da limitação de banda.

Os gráficos, ilustrados nas Figura 25 - (a), Figura 25 - (b) e Figura 25 - (c), mostram um aumento, esperado, nos valores de RTT no uso da criptografia no MQTT para a troca de mensagens na rede, em 7.51% para 1 dispositivo, 15% para 3 dispositivos e 24.81% para 5 dispositivos. O CoAP, Figura 26 - , registrou um aumento de 21.39%, 31.91% e 38.74%, para 1, 3 e 5 dispositivos respectivamente. Este resultado justifica-se pela natureza dos protocolos de transporte utilizados. O MQTT, com o uso do TCP, se mostrou mais estável, registrando um intervalo de confiança menor do que o registro pelo uso do DTLS pelo CoAP. O resultado sugere um número menor de retransmissões de informações criptografadas na rede, o que aumenta o tamanho da carga útil e torna mais onerosa a transmissão de informações em redes com banda restrita e como consequência, aumenta o RTT medido.

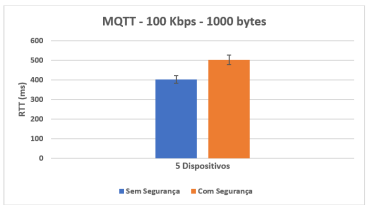
Figura 25 - MQTT - Sem Segurança x Com Segurança (SSL)



(a) 1 Dispositivo

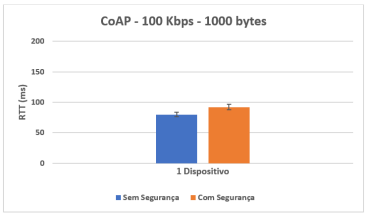


(b) 3 Dispositivos

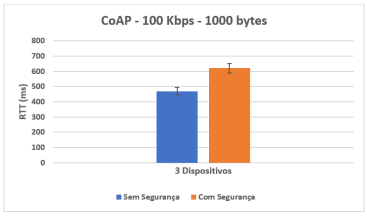


(c) 5 dispositivos

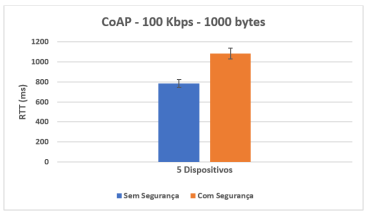
Figura 26 - CoAP - Sem Segurança x Com Segurança (DTLS)



(a) 1 Dispositivo



(b) 3 Dispositivos



(c) 5 Dispositivos

Os resultados para os outros protocolos testados sobre a camada de segurança (HTTP, WebSocket, AMQP e XMPP), não foram apresentados, pois ficariam muito repetitivos em relação às seções anteriores deste capítulo. Os resultados mostram estabilidade na diferença entre o RTT dos outros protocolos de aplicação testados sem segurança e com segurança. Foi registrado um aumento no RTT entre 6 e 8% para 1 dispositivo, entre 12 e 17% para 3 dispositivos e 23 e 26% para todos os protocolos que utilizam o TCP. Para o CoAP, o único testado que usa o UDP, o aumento foi entre 18 e 23%, 27 e 34% e 34 e 40%, para 1, 3 e 5 dispositivos respectivamente.

4.4 **Testes Complementares**

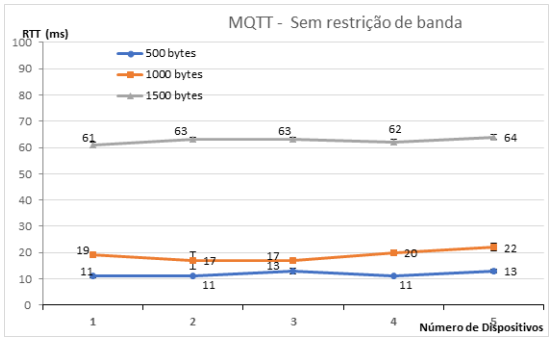
4.4.1 Comparando MQTT e CoAP

Segundo [121] [122] [123], o MQTT e o CoAP são, atualmente, os protocolos de aplicação mais utilizados para comunicação entre dispositivos com baixo poder de processamento e recursos limitados, sendo amplamente recomendados para emprego em projetos de Internet das Coisas.

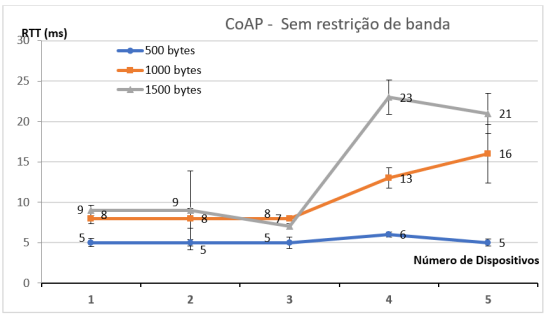
Assim, para estes protocolos, realizamos testes ainda mais detalhados e específicos. Primeiro, comparamos o desempenho dos dois, sem restrição de banda. Ou seja, a interface poderia utilizar toda a capacidade do ambiente de comunicação. Os gráficos da Figura 27 - (a) e da Figura 27 - (b) apresentam os resultados obtidos.

Observa-se que o CoAP apresentou um RTT menor em todos os cenários (número de dispositivos e tamanho de mensagem), quando comparado ao MQTT. Apesar disto, o CoAP apresentou não ser tão estável quanto o MQTT, na transmissão de mensagens de 1000 e 1500 bytes, com 4 e 5 dispositivos. Estes valores de RTT se justificam uma vez que o CoAP utiliza o UDP e o MQTT o TCP como protocolo de transporte.

Figura 27 - RTT para o MQTT e CoAP sem restrição de banda



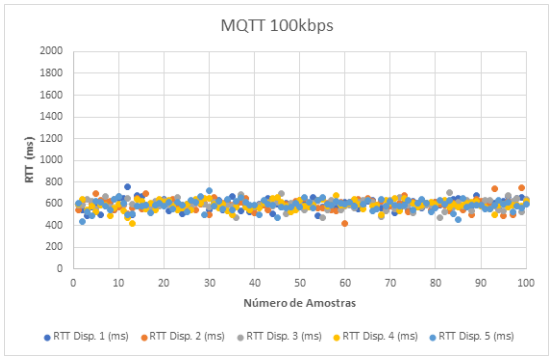
(a) RTT para o MQTT (QoS 1)



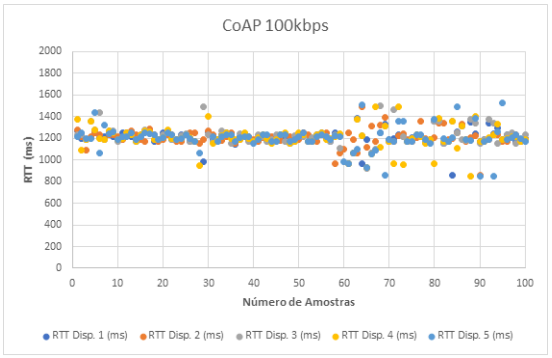
(b) RTT para o CoAP (CON *Messages*)

Além disso, também apresentamos os gráficos de dispersão, ilustrados nas Figura 28 - (a) e Figura 28 - (b), para o cenário de 100 Kbps e 1000 bytes de tamanho da mensagem. Os resultados mostram a estabilidade do MQTT diante do CoAP. O resultado justifica-se pelo controle do recebimento das mensagens que são feitas na camada de aplicação pelo CoAP. O CoAP, por utilizar o UDP (não orientado à conexão) envia uma mensagem e se um *ack*, não é recebido em um intervalo de tempo (30 segundos), uma nova mensagem é enviada. Assim, a dispersão mostrada no gráfico é justificada. Algumas medidas do CoAP tiveram alguma dispersão, o que resultou em um intervalo de confiança maior do que o obtido pelo MQTT.

Figura 28 - Comparação da dispersão entre o MQTT e o CoAP com banda de rede de 100Kbps



(a) RTT para o MQTT



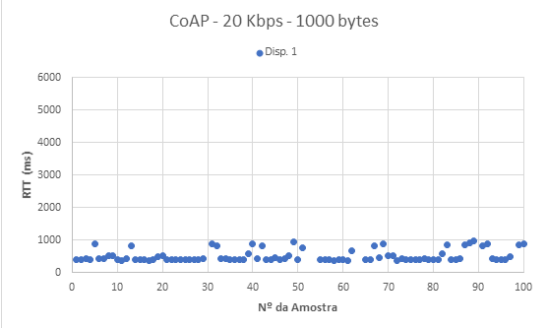
(b) RTT para o CoAP

4.4.1.1 CoAP

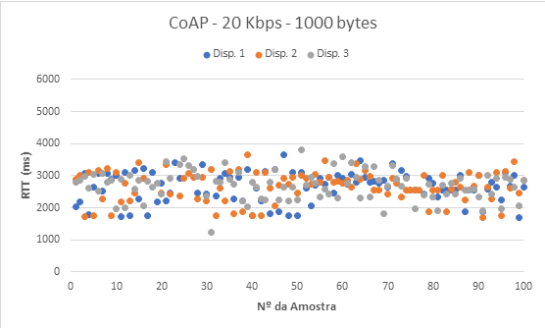
Durante as medidas com o CoAP, observa-se que os resultados deste protocolo, foram os que mais apresentaram dispersão. Os gráficos, ilustrados nas Figura 29 - (a),Figura

29 - (b) e Figura 29 - (c), evidenciam o comportamento do CoAP em cenários de ambientes de rede restritos. De acordo com o incremento de dispositivos na rede, mais os valores de RTT variaram em relação a média.

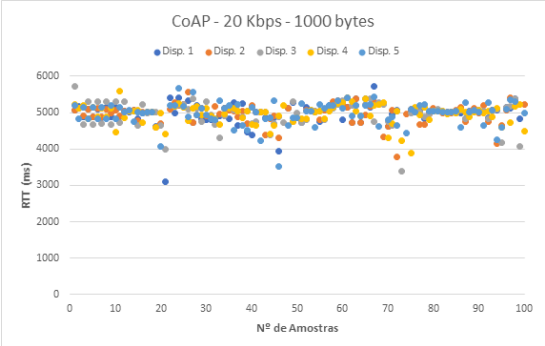
Figura 29 - CoAP - Dispersão - 20 Kbps



(a) 20 Kbps-1000 bytes-1 Disp.

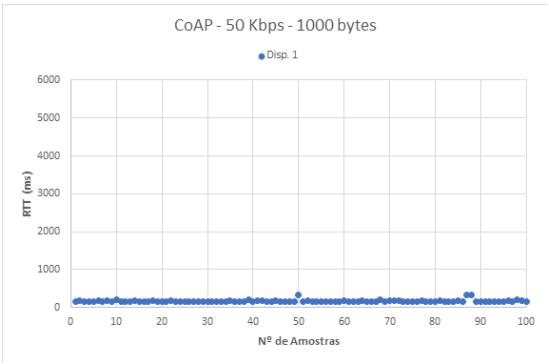


(b) 20 Kbps-1000 bytes-3 Disp.

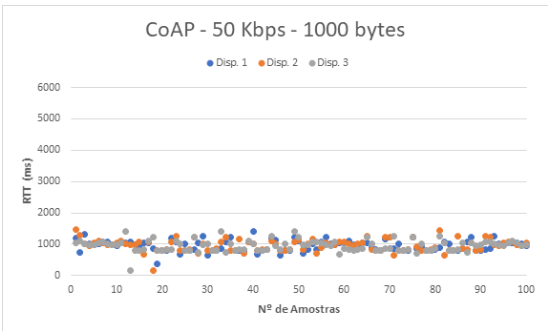


(c) 20 Kbps-1000 bytes-5 Disp.

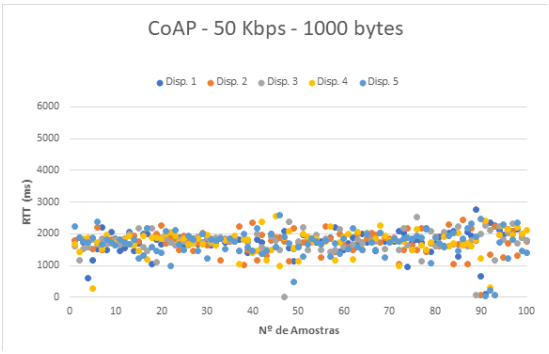
Figura 30 - CoAP - Dispersão - 50 Kbps



(a) 50 Kbps-1000 bytes-1 Disp.

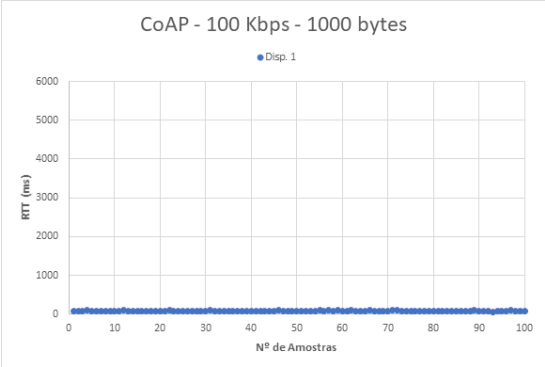


(b) 50 Kbps-1000 bytes-3 Disp.

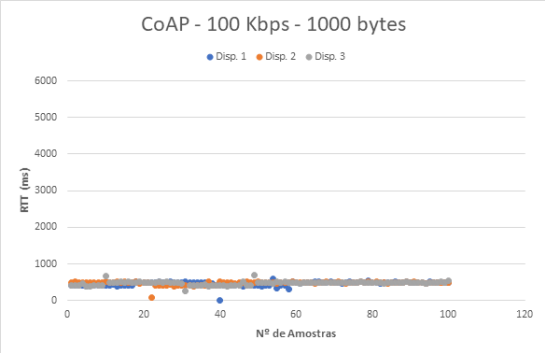


(c) 50 Kbps-1000 bytes-5 Disp.

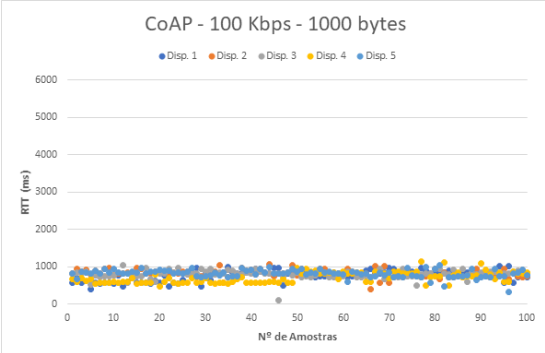
Figura 31 - CoAP - Dispersão - 100 Kbps



(a) 100 Kbps-1000 bytes-1 Disp.



(b) 100 Kbps-1000 bytes-3 Disp.



(c) 100 Kbps-1000 bytes-5 Disp.

4.4.2 Observações do AMQP

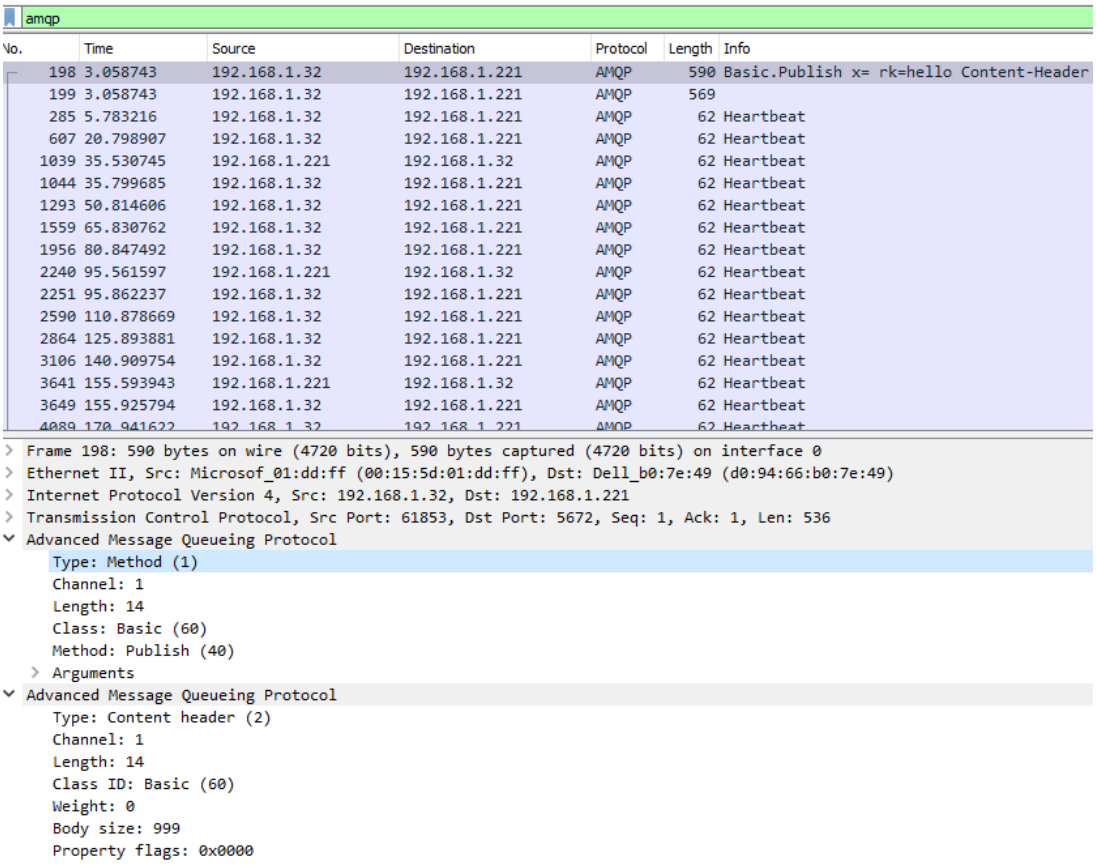
Foi observado que o AMQP, envia três pacotes separados para apenas uma mensagem. Estes são o *Basic.Publish*, *Content-Header* e *Content-Body*. O *Basic.Publish* informa o servidor sobre o processo de publicação da mensagem na fila, como por exemplo o tipo da fila, detalhes do canal de comunicação e outros. O *Content-Header* contém detalhes da mensagem publicada, sendo a principal o nome da fila de publicação. O *Content-Body* contém a carga útil.

A Figura 32 - mostra uma captura de tela, de um *log* obtido através do *Wireshark*, onde observa-se a interação cliente-servidor de uma mensagem, utilizando protocolo AMQP. Diante destas observações, sugere-se que o alto consumo de banda de rede, impacta negativamente o RTT calculado para o AMQP.

Apesar do alto custo para o tráfego de rede, o protocolo AMQP ainda leva vantagem em relação ao XMPP. O XMPP envia apenas uma mensagem para o servidor, porém em XML estruturado e com várias *tags* que são formatadas à mensagem, independente

do uso ou não.

Figura 32 - AMQP - Alto consumo de banda de rede.



4.5 **Discussão**

Os resultados obtidos nos testes mostraram-se comparáveis aos trabalhos relacionados 3.1, embora diferentes parâmetros de ambientes utilizados. Por exemplo, em [79], o autor também chega à conclusão de que o impacto na comunicação do protocolo HTTP em relação ao MQTT com QoS 1, é maior em todos os cenários (número de dispositivos e tamanho da mensagem). Além disto, como obtido nos resultados deste trabalho, o artigo conclui que, se o número de dispositivos, usando o HTTP e transmitindo mensagens em paralelo, aumentam, a sobrecarga da rede chega a um estado crítico, onde erros começaram a ser apresentados. O artigo [78] também indica que os dois protocolos testados, MQTT e CoAP, alcançaram 100% de taxa de entrega da mensagens, no cenário avaliado, com um *timeout* comparável ao utilizado nos nossos testes.

Além das medidas de desempenho, avaliamos, sob um olhar operacional e de utilização, algumas características de qualidade, sendo as principais a confiabilidade na en-

trega da mensagem e a segurança (criptografia e confidencialidade).

Confiabilidade e QoS. Todos os protocolos de aplicação podem ter um aumento de confiabilidade usando algum mecanismo de confirmação de entrega de mensagens, ou nível de QoS, quando disponível. Estes mecanismos garantem a entrega dos dados e, em alguns casos evitam duplicação de pacotes pelos mecanismos de reenvio.

O HTTP não possui opção para parametrização do nível de confiabilidade, sendo assim, utilizado o código de status retornado gerenciado pelo próprio protocolo (status 200, 400, 500). O MQTT possui níveis de confiabilidade, também conhecidos como QoS (*Quality of Service*), nível 0, 1 e 2, que podem ser selecionados na aplicação. No caso do *WebSocket*, após estabelecida comunicação entre um cliente e servidor, o envio de uma mensagem de destinatário para um remetente requer implementação de software para a confirmação (*ack*) do recebimento de uma mensagem pelo destinatário, via camada de aplicação. O AMQP e XMPP também possuem opção de se usar mensagens *ack* para garantir confiabilidade na entrega das mensagens.

O CoAP, apesar de usar o UDP como transporte, possui a opção para se enviar mensagens do tipo "CON"e assim garantir a confiabilidade na entrega de mensagens. Esta opção foi usada nos testes. Cenários de medição de temperaturas de um ambiente é um exemplo onde mensagens podem ser enviadas de tempo em tempo, onde a perda de algumas delas não será crítica para a solução IoT. Assim, o CoAP, sem a confirmação de mensagens ("NON") poderia ser utilizado. Em contrapartida, em cenários, onde por exemplo, um alerta de incêndio é gerado, o CoAP com mensagens de confirmação ("CON") é recomendado.

Segurança. A segurança é um dos problemas principais a serem resolvidos nos cenários de IoT, no que tange a confidencialidade e a autenticação. A utilização da camada de segurança (SSL/TLS e DTLS) satisfaz, em parte, estes requisitos de segurança, adicionando autenticidade e confidencialidade na troca de mensagens entre um dispositivo e o servidor/*broker*.

Todos os protocolos avaliados possuem opção para uso de uma camada de segurança. O CoAP é baseado no DTLS e os demais são baseados no SSL/TLS. Por outro lado, dispositivos com baixo recurso computacional, como Arduino e NodeMCU, ainda carecem de bibliotecas de software que implementem a camada de segurança e componentes de hardware para aplicações com segurança de criptografia assimétrica. Assim nestes

casos seria necessário incluir segurança nos agentes de software, e garantir de outra forma que a segurança entre agente e dispostivo é segura.

4.6 Lições aprendidas durante os testes

Durante o planejamento da avaliação, havia sido previsto que os testes de RTT e escalabilidade seriam realizados sem e com os mecanismos de segurança acionados nos protocolos HTTP, CoAP, MQTT, WebSocket, AMQP e XMPP em dispositivos de processamento restrito como por exemplo, placas Arduino e NodeMCU Esp8266.

Os testes sem o acionamento dos mecanismos de segurança foram feitos com sucesso. Porém, não foi possível realizar os testes com os mecanismos de segurança acionados nos dispositivos, uma vez que algumas bibliotecas que implementam os protocolos, ainda estão em desenvolvimento e/ou possuem muitos erros em sua implementação. Um exemplo disto foi a tentativa de uso da biblioteca TinyDtls para o protocolo CoAP [124]. Várias tentativas foram feitas sem êxito e chegou-se à conclusão de que a única biblioteca disponível no repositório do site oficial do protocolo, ainda precisa de ajustes. Assim, optamos por realizar os testes com segurança utilizando dispositivos móveis do mesmo modelo e configuração, com o sistema operacional Android.

Outra observação interessante é referente aos primeiros testes com o protocolo MQTT com e sem segurança utilizando o *broker* RabbitMQ. Embora este tenha boa reputação e seja usado em várias aplicações em produção, os testes apresentaram resultados fora do esperado. As interações com mensagens menores apresentavam valores medidos de RTT superiores ao de mensagens de tamanho maior. Como o protocolo não é nativo do *broker* e sim um *plugin* que foi desenvolvido e incorporado ao RabbitMQ, foi decidido que os testes deveriam ser refeitos em outro *broker*. O servidor Mosquitto foi escolhido para estas novas interações. Após a reedição dos testes para a avaliação, verificou-se que a anomalia não foi observada.

4.7 Recomendações de Uso

Ainda que o uso dos protocolos de aplicação HTTP, CoAP, WebSocket, MQTT, AMQP e XMPP seja recente no contexto de IoT, estes não são tão novos. Todas os protocolos estão atualmente padronizados por algum órgão ou entidade ligada à computação,

telecomunicações ou sistemas distribuídos.

Para cenários onde o emprego de IoT é interessante, mas poucos recursos computacionais e de energia estão disponíveis, tais como, soluções para agropecuária, monitoramento de linhas férreas extensas, controle e bilhetagem de transportes de cargas em estradas, recomendada-se o emprego de dispositivos com características que atendam aos requisitos de restrição relacionados. Pode-se, por exemplo, utilizar de sistemas tipicamente empregados para prototipagem (Arduino, NodeMCU e Particle Photon) em ambiente de produção. Neste caso, não se recomenda o uso da camada de segurança, uma vez que, o uso da criptografia, naturalmente levará a um aumento do consumo de energia, devido ao aumento do uso do processador na encriptação das mensagens. Além disto, haverá um aumento do tamanho do *payload* de cada mensagem, que apesar de não ser tão significativo, aumentará o tempo de transmissão em redes com banda restrita e/ou alta latência. Ainda, como constatado, atualmente não existem bibliotecas para estes dispositivos, para os protocolos avaliados, com implementações robustas destes mecanismos de segurança.

Caso o uso da camada de segurança seja um requisito obrigatório, recomenda-se o uso de dispositivos mais robustos, como o *Raspberry Pi* ou dispositivos de celulares móveis, pois possuem processadores, memória e outros itens, mais sofisticados.

Também é possível isolar a rede local através de mecanismo de NAT e *firewall* e garantir que o acesso remoto aos dados gerados pelos dispositivos será feito apenas por intermédio do IoT *Gateway*, este protegido por mecanismos mais robustos de segurança.

O tamanho das mensagens enviadas e recebidas pelos dispositivos e *gateway* também deve ser considerado. Seria recomendável fixar o tamanho da mansagem ou, pelo menos o seu tamanho máximo, de forma a evitar (i) uso excessivo de recursos dos dispositivos e comportamentos inesperados da aplicação na transmissão das mensagens pela rede como a fragmentação e consequente aumento do número de mensagens de controle (por exemplo, como constatado na Seção 4.4.1. Algumas bibliotecas que implementam protocolos de aplicação para placas de prototipagem, como o NodeMCU, fixam o tamanho máximo do *payload* por mensagem em alguns bytes, geralmente em 500 bytes. Para este trabalho, este parâmetro foi modificado diretamente nas constantes de código das bibliotecas, para permitir o envio de mensagens de até 1500 bytes.

As áreas de agropecuária, cidades e casas inteligentes, também são bons exemplos

de aplicações IoT. Por exemplo, câmeras de monitoramento e sensores de ambiente podem enviar imagens e dados coletados periodicamente para um agente remoto, com o objetivo de se detectar automaticamente, com o uso da inteligência artificial, condições de interesse ou possíveis ameaças à segurança do local e das pessoas. Em cenários onde existe a limitação de recursos de processamento e conexão de rede limitada ou instável, com banda restrita, é necessário levar em consideração informações de operação como as obtidas no presente trabalho. Assim, o projeto pode incluir, por exemplo, soluções de imagem que se encaixem nos limites de tamanho das mensagens, o número de sensores e a periodicidade adequada para a transmissão das informações coletadas, ou até a introdução de solução de *cache off-line*. De uma forma geral, estas decisões de projeto devem ser acompanahdas da seleção do protocolo de aplicação mais adequado.

Observa-se que o único protocolo que não utiliza o TCP é o CoAP. O CoAP foi projetado para ter pouco *overhead*, ou seja o menor impacto possível na comunicação. Entretanto por utilizar o UDP, é necessário um esforço maior para a configuração de *firewall* e *NAT*, caso seja necessário o acesso remoto. Por isso, a indicação direta do seu uso é em redes locais. A implementação também exige esforço, uma vez que tanto o agente cliente como a versão do servidor devem ser implementados por programação.

O MQTT também foi projetado para ter pouco *overhead*, porém, diferentemente do CoAP, usa o TCP. O MQTT é de fácil configuração e não requer esforço de configuração de *firewall* e *NAT* para sua utilização (o acesso ao *broker*). A implementação é simples e exige apenas o desenvolvimento da aplicação cliente. *Brokers*, estabelecidos e disponíveis, implementam o padrão do protocolo MQTT em sua totalidade. O número de sessões suportadas pelo *broker* deve ser considerado, pois pode ser uma limitação. Como o TCP mantém o estado da conexão, um grande número de clientes conectados a um *broker* MQTT, pode se tornar um problema. Soluções de *Load Balancing* e federação de *brokers* podem ser empregadas. O mesmo serve para todos os protocolos testados nesta dissertação que utilizam o TCP como protocolo de transporte.

O AMQP não foi projetado para ter pouco *overhead*. Ao contrário, como observado na Seção 4.1.5, ele gera um impacto maior na comunicação. Em contrapartida, agrega controle e confiabilidade na entrega das mensagens em nível de aplicação. O AMQP é amplamente utilizado e recomendado para uso sincronismo com entre ambientes locais e a nuvem (Azure, Amazon MQ, Cloud AMQP, entre outros), quando utilizado em redes

de baixa latência (estáveis).

Tal como o AMQP, o XMPP também não foi projetado considerando o impacto na comunicação. O XMPP possui muitas opções, extensões e mecanismos de controle. Estas opções fazem do XMPP um protocolo difícil de ser implementado e utilizado. Além disto, o tamanho das mensagens, que são transportadas em XML, adicionam ainda mais *overhead* na rede. Estas mensagens, contém *tags* XML nulas, porém se fazem necessárias para a estrutura da mensagem a ser interpretada pelo servidor. Em contrapartida, estas extensões podem facilitar o uso de funcionalidades que exigiriam um esforço grande para implementação usando os outros protocolos. Um exemplo é a obtenção do estado da conexão (*on-line*/*off-line*) de um cliente conectado no servidor, entre todos os outros clientes. Esta opção pode ser uma vantagem, em casos onde um cliente pode decidir assumir as tarefas de outro cliente (que ficou *off-line*), sem a interferência do servidor. O XMPP pode ser uma boa opção para cenários de comunicação onde se exige um grande número de funcionalidades na comunicação entre diferentes usuários e dispositivos.

O WebSocket trouxe para os navegadores Web, a possibilidade de comunicação *full-duplex* com os servidores, o que não era permitido antes do surgimento deste mecanismo. Devido ao baixo impacto na comunicação, o WebSocket é, em muitas aplicações, utilizado para transportar protocolos de aplicação, permitindo o uso de protocolos legados e já estabelecidos, como por exemplo o RTP (Real Time Protocol) em navegadores Web sem perda de desempenho. Aplicações de *dashboards* (Bancos, Bolsa de Valores) e mensagens instantâneas, também são cenários típicos para o uso do protocolo WebSocket.

O HTTP, apesar de utilizar o TCP, mantém o estado da conexão, somente enquanto uma mensagem é transmitida ao servidor, ou seja, será feito um *handshake* TCP com o servidor em cada momento que o cliente enviar uma mensagem. Para diminuir os efeitos de múltiplos *handshakes*, o HTTP 1.1 implementou o *keep-alive*, que é um *header*, que o cliente envia na mensagem ao servidor, informando-o da manutenção da conexão para múltiplas requisições. Se o *header* do HTTP, chamado de *Keep-Alive*, for utilizado incorretamente, pode-se gerar um alto impacto na comunicação. Em resumo, o HTTP não é recomendado para uso em dispositivos com pouco poder de processamento e em redes de alta latência. Uma vantagem relevante, é que o HTTP é um protocolo já estabelecido e muito utilizado. Isto agrega valor quanto trata-se do requisito de interoperabilidade entre diversas plataformas legadas. Também, já existem diversas bibliotecas que abstraem a

complexidade de implementação de aplicações que utilizam o HTTP, o que torna fácil a sua utilização. Na Conclusão, em Trabalhos Futuros, são propostos testes com a nova versão do HTTP (2.0), que promete ganhos de desempenho, tornando-o atraente para uso em soluções IoT.

A confiabilidade na entrega das mensagens, indica como cada protocolo trata o requisito não-funcional de garantia de entrega das mensagens. Em alguns casos este requisito é chamado QoS (o MQTT coloca explicitamente desta forma). Nesta dissertação, foi decidido o uso do termo *confiabilidade*, pois *QoS* é geralmente usado para várias características não-funcionais ligadas à garantias de tempo de resposta, latência, prioridade e variação de atraso. O AMQP, por exemplo, pode oferecer tratamento prioritário para filas específicas e isso poderia ser considerado suporte à QoS. A confiabilidade de entrega é apoiada em dois elementos: o uso do TCP, que tem características de garantia de entrega, controle de erro e controle de congestionamento, por exemplo, e na transmissão de mensagens de confirmação (tipo *ACK*), que devem ser verificadas pela aplicação.

Características relacionadas à privacidade e integridade da comunicação (criptografia) também são importantes em aplicações para o IoT. Para isto, se faz necessário a utilização de uma camada extra (segurança), sendo o SSL/TLS para o TCP e DTLS para o UDP. Todos os protocolos testados neste trabalho de mestrado, oferecem suporte para o uso de mecanismos de segurança e controle de acesso.

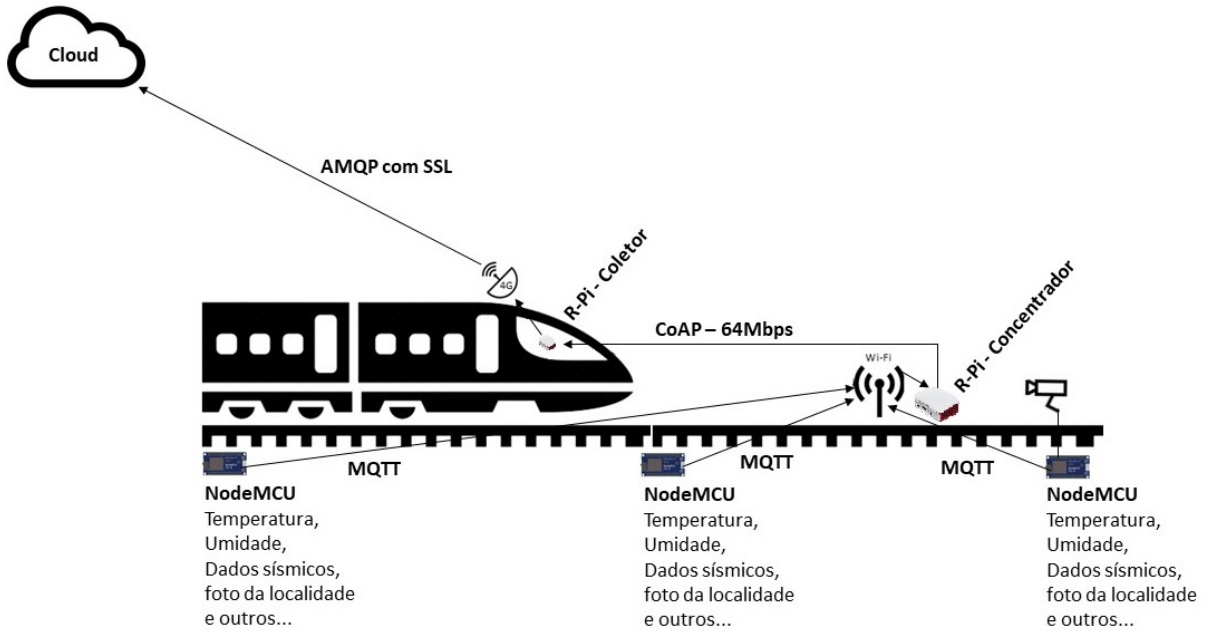
A utilização da camada de segurança para todos os protocolos testados, requer o provisionamento de um Certificado Digital (um arquivo ".cer") contendo um par de chaves assimétricas. Para fins de teste, a geração destas chaves e certificados auto-assinados, podem ser facilmente obtidas através de ferramentas como o "openssl". O certificado deve ser persistido no servidor e seu apontamento deve ser realizado no serviço servidor do protocolo (RabbitMQ, Mosquitto, IIS, OpenFire e Californium). As bibliotecas que foram usadas para a implementação dos testes, já se encarregam do *handshake* entre o cliente e o servidor para uma transmissão segura.

4.7.1 Caso de Uso

Para ilustrar como a avaliação de desempenho realizada e características observadas de cada protocolo podem ser empregadas no desenvolvimento de uma aplicação IoT, apresentamos um caso de uso.

A Figura 33 - ilustra um cenário de monitoramento de uma linha férrea, onde algumas características de ambiente testadas neste trabalho são encontrados, como por exemplo, a concorrência no envio de mensagens pela rede sem fio até o concentrador local das mensagens.

Figura 33 - Ilustração do cenário do caso de uso para emprego de dispositivos e protocolos de aplicação para IoT em um cenário real possível.



Observam-se três dispositivos NodeMCU que estão implantados em pontos distintos e remotos acompanhando o percurso do trilho. Cada dispositivo NodeMCU deve estar a uma distância máxima de 500 metros da estação de trem, onde se encontra o *Access Point* da rede Wi-Fi, considerando-se uma antena e potência apropriadas (Seção 1.4.1). Considera-se também, que num cenário deste tipo, à "céu aberto", variações de temperatura, condições climáticas, trepidações ou mesmo obstáculos naturais imprevistos podem causar impactos na transmissão das informações entre os dispositivos NodeMCU e o concentrador *Raspberry Pi*, o que por sua vez pode resultar em causar retransmissões de dados ou a perda das informações, dependendo da solução projetada. Para o caso de uso, a possibilidade de total desconexão entre um dispositivo e o *Access Point* é improvável dadas as características da rede adotada e as distâncias consideradas.

O concentrador (*Raspberry Pi*) é implantado nas premissas da estação de trem, em distância adequada do *Access Point* da rede Wi-Fi. O papel do concentrador é coletar e

armazenar as mensagens temporariamente, até que estas sejam sincronizadas (enviadas) com o coletor.

Dentro do trem um outro dispositivo (*Raspberry Pi*), o coletor, é implantado, com o objetivo de coletar as informações do concentrador e enviar estas informações para a nuvem. Isto deve ocorrer durante o período que se inicia com a aproximação do trem, sua parada para desembarque e embarque de passageiros. Ou seja, enquanto existir a conectividade com o *Access Point*.

Considera-se o uso da tecnologia Wi-Fi IEEE 802.11G, infraestruturada (ou seja, como uso de um *Access Point* — como já mencionado) que apresenta banda máxima teórica de 65Mbps. Entretanto, o uso desta banda pode variar muito entre os dispositivos (NodeMCU, concentrador e coletor), uma vez que a mobilidade, distância e condições do ambiente podem influenciar as taxas de transmissão na rede local. Observa-se também que esta rede Wi-Fi está isolada, sem acesso a outras redes.

Os dispositivos NodeMCU são "alimentados" por placas fotovoltaicas (energia solar) e não possuem qualquer tipo de conexão remota, apenas com o *Access Point* da rede local. Na Figura 33 - observamos apenas 3 dispositivos, porém, poderiam haver mais.

Na aplicação, todos os dados gerados pelos sensores são importantes e não podem ser perdidos. Neste cenário, onde existe a possibilidade de instabilidade na transmissão dos dados pela rede Wi-Fi, optou-se pelo uso do MQTT para a comunicação dos dispositivos NodeMCU e o concentrador, uma vez que este protocolo se mostrou estável e com ótimo desempenho nos testes realizados em redes com banda restrita (Seção 4.1.7). Observa-se que as imagens são persistidas no concentrador.

Cada dispositivo NodeMCU coleta dados dos respectivos sensores periodicamente, em um intervalo de 5 minutos, de forma a não sobrecarregar a rede, como resultado do envio de mensagens ao mesmo tempo por outros dispositivos conectado ao mesmo *Access Point*. Após a coleta, estes dados são encapsulados em mensagens que são em seguinda, enviadas ao concentrador através de mensagens MQTT. Os dispositivos NodeMCU não fazem *cache* local. A confiabilidade de que a imagem não será perdida é obtida pelo uso do MQTT (Seção 2.4 e Seção 3.6.3).

Entre os dados coletados, imagens do ambiente, ou do trilho são obtidas segundo a periodicidade definida ou quando se atinge algum limiar, como por exemplo, queda brusca da temperatura, chuvas fortes ou deslizamentos. As imagens são geradas em preto

e branco, com 8 bits por pixel, formato TIFF, dimensão de 500x500 pixels e não ultrapassa o tamanho de 1500 bytes. Com estas característica obtém-se uma imagem nítida e dentro dos limites de tamanho estabelecidos. Vale ressaltar que a aplicação hospedada nos dispositivos mantém um controle do tamanho das mensagens, não permitindo o envio de mensagens com mais de 1500 bytes.

Observa-se também que o tamanho máximo das mensagens e a periodicidade definida, em alinhamento com a tecnologia Wi-Fi selecionada, absorve os impactos verificados nas métricas avaliadas quando existe a concorrência vários dispositivos (Seção 4.4).

O coletor localizado no trem é o único dispositivo que possui conectividade com a Internet. Ao iniciar sua aproximação com a área coberta pela rede Wi-Fi da estação onde encontram-se os outros dispositivos, este inicia a tentativa de conexão com o *Access Point* Wi-Fi da estação e tão logo seja possível, o concentrador local inicia a sincronização utilizando o protocolo CoAP com mensagens de confirmação. Neste caso, o trem ficará a maior parte do tempo da conexão Wi-Fi, parado na estação e muito próximo ao *Access Point*, resultando em maior qualidade da conexão entre o coletor e o ponto de acesso (Seção 1.4.1). Assim, utilizamos o CoAP valendo-se do seu desempenho em redes sem restrição de banda (Seção 4.4.1).

As mensagens com informações cuja transmissão, porventura, não sejam confirmadas durante a passagem do trem, continuam persistidas. Uma nova tentativa será feita após a passagem de outro trem. O coletor implantado no trem, com conexão com a Internet e de posse das mensagens recebidas do concentrador local da estação, inicia a sincronização com um serviço de nuvem utilizando o protocolo AMQP com SSL/TLS, uma vez que é preciso garantir a integridade e confidencialidade das mensagens entre o trem a nuvem.

4.8 **Resumo**

A Tabela 5 - resume as principais caraterísticas consideradas positivas e negativas de cada protocolo avaliado. Também, são resumidas as principais recomendações de uso, com base dos resultados obtidos, correlacionando aplicações para cada protocolo.

Tabela 5 - Diferenças observadas entre os protocolos de aplicação testados.

Protocolo	**Características**
HTTP	**Prós.** Estabelecido e amplamente utilizado, o que favorece uma rápida implementação, implantação e integração. **Contras.** Cabeçalho variável, a depender da aplicação. Ex: cabeçalho de autenticação, criptografia e outros, aumenta o tamanho da mensagem. Problemas com os *flags keep-alive* e *connection* em bibliotecas para dispositivos NodeMCU. **Aplicações.** Amplamente utilizado para construção de *API's* para interoperar entre plataformas distintas na internet. Recomendado para aplicações de rede local e sem restrição de banda. Soluções IoT para casas inteligentes é um bom exemplo para utilização do HTTP.
CoAP	**Prós.** Cabeçalho fixo, o que torna previsível o tamanho da mensagem. Uso do UDP, que resulta em melhor *performance*, em mensagens sem confirmação. Fácil abstração, uma vez que os conceitos foram herdados do HTTP. **Contras.** *Frameworks* como JDK, .NET e outros, não possuem suporte nativo. Poucas bibliotecas disponíveis. Mensagens com confirmação e banda de rede restrita, resulta em RTT's dispersos e altos como pode ser observado no gráfico ilustrado na Figura 24 - e nas medidas obtidas e comentadas na Seção 4.4.1.1. Além disso, como já comentado, usando UPD, o acesso remoto dentro de redes protegidas por NAT é exige configuração complexa. **Aplicações.** Recomendado para aplicações em rede local e agentes de recursos com com capacidade de processamento restrito. Amplamente recomendado para ambientes com restrição de banda, tais como cidades inteligentes, transporte público e agropecuária.
MQTT	**Prós.** Cabeçalho fixo, o que torna previsível o tamanho da mensagem. Mantém estado da conexão, favorecendo cenários com "rajadas" de mensagens constantes, evitando *handshakes* desnecessários. Transpor barreiras de firewall e NAT é um ponto positivo.

	Contras. Requer um *broker*, um elemento centralizador, que representa um ponto de falha. **Aplicações.** Recomendado para aplicações em redes de alta latência. Ambientes restritos de cidades inteligentes, agronegócio, logística e transporte público, são exemplos de emprego do MQTT em soluções IoT.
WebSocket	**Prós.** Uso do navegador via mecanismo *socket*. Melhor performance em relação aos outros protocolos de aplicação testados. Mantém estado da conexão, favorecendo cenários com "rajadas" de mensagens constantes, evitando *handshakes* desnecessários. **Contras.** Apesar de ser apresentar um protocolo para interação com servidores Web ou Servidores de Aplicação, é muito utilizado como um mecanismo-base, tal como o *socket* tradicional. Assim, é comum ser utilizado com algum outro protocolo para prover a interação com dispositivos distintos (*MQTT over WebSocket, XMPP over WebSocket, HTTP over WebSocket*) ou optar-se pelo desenvolvimento de um protocolo próprio em detrimento da interoperabilidade. **Aplicações.** Recomendado para todas as áreas de aplicação de soluções IoT, seja para o uso em ambientes com restrição de banda e processamento, como também para o sincronismo dos dados para a nuvem. Além disso, é uma solução boa para permitir o desenvolvimento de módulos integrados aos navegadores Web.
AMQP	**Prós.** Estabilidade e confiabilidade na entrega das mensagens. Pode usar os estilos *Publish-Subscribe* e *Request-Response*. Oferece várias configurações de filas, ordenação e entrega. Permite a defeinição de prioridades na entrega de mensagens. **Contras.** Uso excessivo de banda de rede para a troca de mensagens entre o cliente e o servidor (Seção 4.4.2). Além disto, o uso do AMQP requer um elemento centralizador, que representa um ponto de falha. **Aplicações.** Recomendado para uso em redes sem restrição de banda. É comum que soluções de sincronização e integração entre clientes e servidores na nuvem façam uso do protocolo AMQP, com destaque para a ordenação das mensagens no nível da aplicação.

XMPP	**Prós.** Opções de controle (descoberta e interação) entre dispositivos, disponíveis em forma de *plugins*. Pode usar os estilos *Publish-Subscribe* e *Request-Response*. **Contras.** São poucas as bibliotecas disponíveis para o desenvolvimento de aplicações interessadas no uso do protocolo XMPP. As mensagens *XML* trocadas entre o cliente e servidor, possuem *tags* vazias, o que torna as mensagens XMPP muito grandes, resultando em mais uso de banda de rede quando comparado com outros protocolos. O gráfico 4.1.7 e a Seção 4.1.6 mostram o desempenho ruim do protocolo XMPP. Assim como o MQTT e o AMQP, o XMPP também requer um elemento centralizador, o que também representa um ponto de falha. **Aplicações.** Recomendado para uso em redes sem restrição de banda de rede. Soluções IoT que exijam monitoramento em tempo real entre os dispositivos (*on-line/off-line*) e/ou entre dispositivos e o servidor, o XMPP é a melhor opção. Aplicações que necessitem que dispositivos IoT se comuniquem (M2M) diretamente, sem restrição de banda, também recomenda-se a escolha do protocolo XMPP.

CONCLUSÃO

Nesta pesquisa de mestrado, realizou-se um estudo das características e uma análise de desempenho dos protocolos da camada de aplicação HTTP, CoAP, MQTT, WebSocket, AMQP e XMPP aplicados à Internet das Coisas. Os protocolos avaliados são padronizados por organizações como o ITU-T, IEEE, e o IETF ou por organizações como o OASIS e o W3C. Estes protocolos são suportados atualmente por bibliotecas disponíveis para, praticamente, todas as plataformas, incluindo dispositivos com poucos recursos. Com estas bibliotecas, foram implementados agentes de software para dispositivos NodeMCU Esp8266 e para dispositivos móveis com o sistema operacional Android. Além disto, foram feitos testes com os servidores/*brokers* que suportam cada um dos protocolos estudados.

Os testes de desempenho sem o uso da camada de segurança (SSL/TLS ou DTLS no caso do CoAP), foram concebidos considerando dispositivos com recursos limitados (NodeMCU Esp8266). Inicialmente, alguns protocolos como o AMQP e XMPP, não foram projetados para IoT, porém, com o avanço da tecnologia de armazenamento e processamento para dispositivos com poucos recurso (placas do tipo Arduino), foram desenvolvidas algumas bibliotecas para estes protocolos.

Ainda assim, são poucas as bibliotecas disponíveis para Arduino ou NodeMCU que implementem as funcionalidades de segurança e algumas das bilbiotecas testadas apresnetam a implementação com erros. Por isso, os testes sobre a camada de segurança, foram feitos utilizando dispositivos móveis com sistema operacional Android, cujas bibliotecas funcionaram adequadamente.

Outro ponto está relacionado ao aspecto de confirmação. Todos os testes consideraram uma confirmação da mensagem, pelo servidor, de cada mensagem enviada por um cliente. Assim, níveis os níveis de QoS do MQTT — termo utilizao para expressar o modelo de confirmação de envio — 0 (*fire and forget*) e 2 (*exactly once* - garantia de recebimento apenas uma vez, tanto pelo cliente como pelo *broker*) e mensagens *NON* do CoAP (sem confirmação), foram desconsiderados.

Com base nos resultados dos testes, observa-se que o protocolo XMPP é o menos indicado para uso em dispositivos IoT. O RTT (*Round Trip Time*) calculado e o tamanho da mensagem com o mesmo *payload*, foi maior do que todos os outros protocolos. Muito próximo ao XMPP, também é constatado que o HTTP não é indicado para ambientes IoT.

A versão mais atual do HTTP (2.0), promete ganhos consideráveis de desempenho 4.8. O AMQP apresentou um RTT menor do que o XMPP e HTTP, porém, consideravelmente mais alto em relação ao MQTT, CoAP e WebSocket. O AMQP se mostrou estável, com intervalo de confiança baixo, porém, o preço desta estabilidade, se reflete na quantidade de mensagens de controle, trocadas entre o cliente e o servidor.

Dentre todos os protocolos, o AMQP é o que possui maior possibilidade de parametrização, tais como, controle da conexão com *acks* de *heartbeat*, lotes de confirmação de *acks*, exemplo: um mensagem de *ack* para 10 mensagens recebidas. Esta manipulação de parâmetros, pode trazer ganho de desempenho, porém a personalização destes parâmetros deve ser seguida de estudos técnicos prévios. Sugere-se, a partir dos resultados dos testes, que os protocolos XMPP, HTTP e AMQP são mais adequados para ambientes onde a economia de recursos (energia, rede e processamento) não são itens a serem considerados.

O CoAP, mostrou resultado (RTT) intermediário comparado a todos os protocolos. Este resultado se justifica pelo fato de que o remetente precisa ser notificado do recebimento da mensagem pelo servidor. O CoAP é um protocolo indicado para ambientes IoT de poucos recursos computacionais, porém a troca de mensagens do tipo *CON*, deve ser previamente avaliada antes de sua utilização.

O WebSocket obteve um bom resultado em relação aos protocolos XMPP, HTTP, HTTP e CoAP. Isto mostra o porquê de sua utilização como um mecanismo ao invés de um "protocolo". É recorrente o uso de outros protocolos como, HTTP, MQTT, AMQP, XMPP, sobre o WebSocket, atuando como "transportador". É também uma opção para utilização como protocolo de aplicação, porém, requer implementação personalizada de controle, como por exemplo, as mensagens de confirmação.

O MQTT obteve o melhor tempo de RTT diante de todos os outros protocolos de aplicação testados, mesmo com a configuração de *QoS* 1. Este resultado mostra que o MQTT é recomendado para uso em projeto IoT em ambientes com recursos de rede e computacionais restritos.

Os testes foram repetidos sobre a camada de segurança e os resultados do RTT apresentaram variação, no caso do TCP, entre 6% para 1 dispositivo (WebSocket, 500 bytes de tamanho da mensagem e 100Kbps de banda de rede) e 26% para 5 dispositivos (XMPP, 1500 bytes de tamanho da mensagem e 20Kbps de banda de rede), e para o UDP, 18% e 40% para 1 e 5 dispositivos respectivamente. Este resultado reflete o aumento

do tamanho da mensagens e sua respectiva transmissão com banda de rede restrita. A utilização ou não dos recursos de criptografia, devem ser avaliados antes da implementação de cada projeto.

Trabalhos futuros

O protocolo HTTP está em constante evolução. Desde a primeira versão (0.9), de 1991, a simplicidade de sua implementação foi seu ponto forte. Com este foco, o HTTP ganhou popularidade e se tornou o protocolo mais utilizado da internet. Evoluiu para a versão 1.0 e 1.1, sendo esta última, usada nos testes deste trabalho de mestrado. O HTTP, em sua versão mais recente, está a 2.0, que está especificada na RFC 7540 [125]. É sabido que o uso da Web e os variados dispositivos conectados, crescem exponencialmente. O HTTP versão 2 é uma resposta a este novo cenário em especial às aplicações da Internet das Coisas. Esta última versão do protocolo, permite o uso mais eficiente da rede, de seus recursos e redução da latência, introduzindo algoritmos eficientes de compressão e permitindo várias trocas simultâneas na mesma conexão entre o cliente e servidor. Ainda são necessárias avaliações que explorem as opções disponíveis no HTTP 2.0 em aplicações para IoT.

O protocolo AMQP também evoluiu ao decorrer do tempo. Os testes realizados neste trabalho utilizou a versão 0.9.1 do protocolo, e a versão mais atual é a 1.0.0 [126]. Esta versão mais recente, tem muitas diferenças em relação a versão 0.9.1, porém, manteve as mesmas opções de fila já disponíveis em versões anteriores. Destaca-se a melhoria de desempenho prometida na especificação. Assim, devido ao seu potencial de utilização e às suas características descritas no padrão OASIS, o AMQP 1.0 precisa de estudos mais aprofundados e do desenvolvimento de bibliotecas que possam o incorporar a novas e/ou *frameworks* já existentes.

O STOMP (*Streaming Text Oriented Messaging Protocol*) é um protocolo simples baseado em texto, projetado para trabalhar com MOM (*middleware* orientado a mensagens). Podemos citar alguns MOM's suportam o protocolo, tais como o RabbitMQ, ApacheMQ e o OpenMQ. Ele fornece um formato de conexão interoperável que permite que os clientes STOMP conversem com qualquer intermediário de mensagens que suporte o protocolo. Sua última versão é a 1.2 e mais uma vez, o desempenho em redes de alta latência é o foco, uma vez que o desempenho se destaca em relação a outros protocolos

similares como o AMQP e XMPP. Isto, torna o STOMP um protocolo com potencial de uso em soluções IoT, sugerindo assim que pesquisas sejam feitas com este protocolo de aplicação. O STOMP ainda não é um padrão de direito, porém, devido ao seu amplo uso, é um forte candidato a ser em breve, padronizado.

Testes de desempenho e escalabilidade em ambientes simulados poderiam ser realizados com os mesmos Protocolos de Aplicação avaliados, principalmente para se avaliar escalabilidade, empregando centenas ou milhares de dispositivos. Provavelmente, resultados que não foram observados nos testes de bancada desta dissertação poderiam ser estudados. Conforme comentamos na Seção 3.2 nem todos os Protocolos de Aplicação para IoT avaliados possuem implementação nos simuladores mais populares, como o NS-3. Uma alternativa seria realizar simulações empregando tecnologia de micro-serviço, como o *Docker*, que pode facilitar a implementação de testes deste tipo, devido sua facilidade de carregamento, execução e orquestração de muitas instâncias em paralelo.

Muitos *frameworks* IoT como o ThingsBoard, FIWARE e DOJOT, já incorporam alguns Protocolos de aplicação, tais como o HTTP e MQTT e disponibilizam agentes de software para facilitar seu uso. Porém, outros já padronizados e passíveis de utilização em diversos cenários IoT, não foram provisionados nestes sistemas. Desta forma, existe uma oportunidade de investigação para permitir a oferta de agentes "plugáveis" e interoperabilidade ou conciliação de Protocolos de Aplicação para IoT de forma simples.

Por último, como discutido ao longo da dissertação, os vários Protocolos de Aplicação disponíveis para uso em soluções IoT, tem características específicas. Por exemplo, alguns que usam o estilo de interação *request-response*, *publish-subscribe* ou os dois. Alguns projetos da Internet das Coisas podem ser abrangentes o suficiente para atender uma ampla área geográfica, como por exemplo, uma cidade, estado ou até um país. Assim, podem existir diversos cenários onde um protocolo pode ter seu uso mais adequado do que outro, ou o melhor estilo de interação pode necessitar de reconfiguração, em determinado período do dia, devido ao aumento da latência na rede ou sobrecarga do servidor/*broker* no processamento das mensagens. Como trabalhos futuros, poderíamos propor uma solução para conciliação automática de protocolos. Um dos objetivos seria permitir que um sistema de *middleware* para IoT pudesse dinamicamente mudar ou reconfigurar o protocolo em uso, de acordo com o contexto do ambiente e da aplicação.

REFERÊNCIAS

[1] CPSE Labs. *FIWARE*. 2017. Web page. European Union for the development and global deployment of applications for Future Internet, `https://www.fiware.org/`.

[2] http://www.dojot.com.br/. *DOJOT*. 2018. Web page. Framework DOJOT, `http://www.dojot.com.br/`.

[3] cpse-labs.eu. *SOFIA - Smart Objects For Intelligent Applications*. 2012. Web page. ARTEMIS project, `http://www.cpse-labs.eu/sp_sofia.php`.

[4] SOLDATOS, J. et al. Openiot: Open source internet-of-things in the cloud. In: _____. *Interoperability and Open-Source Solutions for the Internet of Things: International Workshop, FP7 OpenIoT Project, Held in Conjunction with SoftCOM 2014, Split, Croatia, September 18, 2014, Invited Papers*. Cham: Springer International Publishing, 2015. p. 13–25.

[5] FIELDING, R. et al. *Hypertext Transfer Protocol – HTTP/1.1*. 6 1999. RFC 2616. IETF. Web page. `https://tools.ietf.org/html/rfc2616`.

[6] Mqtt.org. *Message Queuing Telemetry Transport (MQTT)*. 2017. Web page. `http://mqtt.org/`.

[7] SHELBY, Z.; HARTHE, K.; BORMANN, C. *RFC 7252 Constrained Application Protocol (CoAP)*. 2017. Web page. `http://coap.technology/`.

[8] AMQP.org. *Advanced Message Queuing Protocol (AMQP)*. 2014. Web Page. `https://www.amqp.org/`.

[9] XMPP Standards Foundation (XSF). *Extensible Messaging and Presence Protocol (XMPP)*. 2011. Web page. `https://xmpp.org/`.

[10] websocket.org. *About HTML5 WebSocket*. 2017. Web page. `https://websocket.org/aboutwebsocket.html`.

[11] ZANELLA, A. et al. Internet of things for smart cities. *IEEE Internet of Things Journal*, v. 1, n. 1, p. 22–32, Feb 2014. ISSN 2327-4662.

[12] DíAZ, M.; MARTíN, C.; RUBIO, B. State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing. *Journal of Network and Computer Applications*, v. 67, p. 99 – 117, 2016. ISSN 1084-8045.

[13] BELLAVISTA, P. et al. Convergence of manet and wsn in iot urban scenarios. *IEEE Sensors Journal*, v. 13, n. 10, p. 3558–3567, Oct 2013. ISSN 1530-437X.

[14] KON, F.; ZAMBOM, E. Cidades inteligentes: Tecnologias, aplicações, iniciativas e desafios. In: *CSBC 2016. JAI 1. Porto Alegre, RS*. [S.l.: s.n.], 2016.

[15] SANTOS, B. P. et al. Internet das coisas: da teoria à prática. In: *SBRC 2016. Minicurso 1 (MC-1). Salvador, BA*. [S.l.: s.n.], 2016. p. 256–315.

[16] KON, F.; SANTANA, E. F. Z. Computação aplicada a cidades inteligentes: Como dados, serviços e aplicações podem melhorar a qualidade de vida nas cidades. In: *CSBC 2017. JAI 4. São Paulo, SP.* [S.l.: s.n.], 2017. p. 2536.

[17] SUNDMAEKER, H. et al. *Vision and Challenges for Realising the Internet of things.* The address of the publisher, 3 2010.

[18] TEIXEIRA, T. et al. Service oriented middleware for the internet of things: A perspective. In: _____. *Towards a Service-Based Internet: 4th European Conference, ServiceWave 2011, Poznan, Poland, October 26-28, 2011. Proceedings.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 220–229. ISBN 978-3-642-24755-2.

[19] AL-FUQAHA, A. et al. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, v. 17, n. 4, p. 2347–2376, Fourthquarter 2015. ISSN 1553-877X.

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

[21] NASTASE, L. Security in the internet of things: A survey on application layer protocols. In: *2017 21st International Conference on Control Systems and Computer Science (CSCS).* [S.l.: s.n.], 2017. p. 659–666.

[22] RAMIREZ, J.; PEDRAZA, C. Performance analysis of communication protocols for internet of things platforms. In: *2017 IEEE Colombian Conference on Communications and Computing (COLCOM).* [S.l.: s.n.], 2017. p. 1–7.

[23] PARTICLE. *The Particle IoT Platform.* 2017. Web page. `https://www.particle.io/`.

[24] FILHO, P. S.; NASCIMENTO, R. *KNoT: Integrando Plataformas e Aplicações de Internet das Coisas.* 2017. X Escola Potiguar de Computação e suas Aplicações. Tutorial 2. Recife-PE. `https://www.knot.cesar.org.br/`.

[25] Eclipse IoT Working Group. *IoT Standards.* 2012. Web page. `https://iot.eclipse.org/standards/`.

[26] PEREIRA, C. et al. Iot interoperability for actuating applications through standardised m2m communications. In: *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM).* [S.l.: s.n.], 2016. p. 1–6.

[27] ONEM2M. *Standards for M2M and the Internet of Things.* 2017. Web page. `http://www.onem2m.org/`.

[28] P2413, I. *Standard for an Architectural Framework for the Internet of Things (IoT).* 2016. `http://grouper.ieee.org/groups/2413/`. IEEE-SA Project. Chair: Oleg Logvinov (STMicroelectronics), Manager: Brenda Mancuso.

[29] OASIS. *Organization for the Advancement of Structured Information Standards.* 2017. Web page. `https://www.oasis-open.org/`.

[30] ITU-T. *M2M service layer: APIs and protocols overview*. 4 2014. ITU-T Focus Group on M2M Service Layer. https://www.itu.int/dms_pub/itu-t/opb/fg/T-FG-M2M-2014-D3.1-PDF-E.pdf.

[31] ITU-T. *Framework of constrained device networking in the IoT environments*. 9 2016. Global Information InfraStructure, Internet Protocol Aspects, Next-generation Networks, Internet of Things and Smart Cities. https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Y.4451-201609-I!!PDF-E&type=items.

[32] IETF. *The Internet of Things*. 2017. Web page. https://ietf.org/topics/iot/.

[33] IEEE. *The Internet of Things*. 2017. Web page. https://iot.ieee.org/.

[34] PöTTER, H. A. B.; ALEXANDRE. Adapting heterogeneous devices into an iot context-aware infrastructure. In: *SEAMS '16*. New York, NY, USA: ACM, 2016. p. 64–74. ISBN 978-1-4503-4187-5. Disponível em: <http://doi.acm.org/10.1145/2897053.2897072>.

[35] POSTSCAPES. *IoT Gateways*. 2017. Web page. https://www.postscapes.com/iot-gateways/.

[36] MICROSOFT. *Hub IoT - Microsoft Azure*. 2018. Web Site. Web page. https://azure.microsoft.com/pt-br/services/iot-hub/.

[37] AMAZON. *AWS IoT*. 2018. Web Site. Web page. https://aws.amazon.com/pt/iot/.

[38] KRISHNAPUR, P. M.; M, S.; Y., C. A. Fast realtime data transfer using xmpp. In: *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*. [S.l.: s.n.], 2016. p. 477–479.

[39] THANGAVEL, X. M. D.; VALERA, A. Performance evaluation of mqtt and coap via a common middleware. In: *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. [S.l.: s.n.], 2014.

[40] SZTAJNBERG, A.; STUTZEL, M.; MACEDO, R. *Protocolos de Aplicação para a Internet das Coisas: conceitos e aspectos práticos*. [S.l.]: CSBC - JAI, 2018. http://natal.uern.br/eventos/csbc2018/wp-content/uploads/2018/08/Anais-Final-JAI-2018.pdf.

[41] WEISER, M. The computer for the 21st century. In: . [S.l.: s.n.], 1991.

[42] SUNDMAEKER, H. The computer for the 21st century. In: . [S.l.: s.n.], 1991.

[43] Harald Sundmaeker,Patrick Guillemin, Peter Friess, Sylvie Woelfflé . *Vision and Challenges for Realising the Internet of Things*. 2010. Web page. http://www.internet-of-things-research.eu/pdf/IoT_Clusterbook_March_2010.pdf.

[44] CASAGRAS, E.F.P. *Casagras final report: rfid and the inclusive model for the internet of things*. 2009. Web page. https://docbox.etsi.org/zArchive/TISPAN/Open/IoT/low20resolution/www.rfidglobal.eu%20CASAGRAS%20IoT%20Final%20Report%20low%20resolution.pdf.

[45] Perera, C. et al. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys Tutorials*, v. 16, n. 1, p. 414–454, First 2014.

[46] European Plataforms Initiative. *Casagras final report: rfid and the inclusive model for the internet of things*. 2019. Web page. `https://iot-epi.eu/`.

[47] Previsao ONU. *Previsao ONU*. 2014. Web page. Https://www.un.org/en/development/desa/news/population/world-urbanization-prospects-2014.html.

[48] Song, R. et al. Context-aware bpm using iot-integrated context ontologies and iot-enhanced decision models. In: *2019 IEEE 21st Conference on Business Informatics (CBI)*. [S.l.: s.n.], 2019. v. 01, p. 541–550.

[49] Dholu, M.; Ghodinde, K. A. Internet of things (iot) for precision agriculture application. In: *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*. [S.l.: s.n.], 2018. p. 339–342.

[50] Industrial IoT. *Industrial IoT*. 2019. Web page. Https://www.wired.com/insights/2014/11/industrial-internet-of-things-software/.

[51] Petrasch, R.; Hentschke, R. Process modeling for industry 4.0 applications: Towards an industry 4.0 process modeling language and method. In: *2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. [S.l.: s.n.], 2016. p. 1–5.

[52] COULOURIS, G. et al. *Distributed Systems: Concepts and Design*. 5th. ed. USA: Addison-Wesley Publishing Company, 2011. ISBN 0132143011, 9780132143011.

[53] Wi-Fi Alliance. *Wi-Fi Alliance*. 2019. Web. `https://www.wi-fi.org/`.

[54] What is Wi-Fi by Cisco. *What is Wi-Fi by Cisco*. 2017. Web page. `https://www.cisco.com/c/en/us/products/wireless/what-is-wifi.html`.

[55] Khan, M. F. et al. Wi-fi radar placement for coverage in collapsed structures. In: *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*. [S.l.: s.n.], 2018. p. 423–430.

[56] IEEE. *IEEE 802.11ah-2016 - IEEE Standard*. 2016. Web page. `https://standards.ieee.org/standard/802_11ah-2016.html`.

[57] GROUP, B. S. I. *Bluetooth Special Interest Group*. 2019. Web Site. Web page. `https://www.bluetooth.com/bluetooth-technology`.

[58] ALLIANCE, Z. *ZigBee*. 2017. Https://www.zigbee.org/. Web page. `https://www.zigbee.org/`.

[59] MONTENEGRO, G. et al. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. 7 2007. RFC 4944. IETF. Web page. `https://datatracker.ietf.org/doc/rfc4944/`.

[60] IETF. *6LowPAN*. 2017. IETF RFC6282. Web page. https://tools.ietf.org/html/rfc6282.

[61] BERNERS-LEE, T.; FIELDING, R.; MASINTER, L. *Uniform Resource Identifier (URI): Generic Syntax*. 1 2005. RFC 3986. IETF. Web page. https://tools.ietf.org/html/rfc6690.

[62] ROBINSON, D.; COAR, K. *The Common Gateway Interface (CGI) Version 1.1*. 10 2004. RFC 3876. IETF. Web page. https://tools.ietf.org/html/rfc3875.

[63] W3C. *Web Services Activity*. 4 2011. Web page. https://www.w3.org/2002/ws/.

[64] FERREIRA, H. G. C.; CANEDO, E. D.; SOUSA, R. T. de. Iot architecture to enable intercommunication through rest api and upnp using ip, zigbee and arduino. In: *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. [S.l.: s.n.], 2013. p. 53–60. ISSN 2160-4886.

[65] VáSQUEZ, F. Prototype for monitoring patients at rest with wearable and iot technology (june 2017). In: *2017 IEEE Central America and Panama Student Conference (CONESCAPAN)*. [S.l.: s.n.], 2017. p. 1–6.

[66] IETF. *RESTful Design for Internet of Things Systems*. 2017. Web page. https://tools.ietf.org/id/draft-keranen-t2trg-rest-iot-04.html.

[67] GOMES, G. G. R.; SAMPAIO, P. N. M. A specification and tool for the configuration of rest applications. In: *2009 International Conference on Advanced Information Networking and Applications Workshops*. [S.l.: s.n.], 2009. p. 500–505.

[68] BANDYOPADHYAY, S. Lightweight internet protocols for web enablement of sensors using constrained gateway devices. In: *2013 International Conference on Computing, Networking and Communications, Workshops Cyber Physical System*. [S.l.: s.n.], 2013.

[69] IETF. *Constrained RESTful Environments (CoRE) Link Format*. 8 2012. https://tools.ietf.org/html/rfc6690.

[70] ISO. *Message Queuing Telemetry Transport (MQTT) v3.1.1*. 2016. OASIS Standard. ISO/IEC 20922:2016. Web page. Edited by Andrew Banks and Rahul Gupta. https://www.iso.org/standard/69466.html.

[71] ISO/IEC JTC 1 Information technology. *Advanced Message Queuing Protocol (AMQP) v0-9-1 specification*. 2014. ISO/IEC 19464:2014. https://www.iso.org/standard/64955.html.

[72] Pivotal Software, Inc. *Rabbit MQ*. 2017. Web page. Rabbit MQ Messaging Server, https://www.rabbitmq.com/.

[73] IETF. *Extensible Messaging and Presence Protocol (XMPP): Core*. 2011. Web page. https://tools.ietf.org/html/rfc6120.

[74] XMPP Extensions. *Extensible Messaging and Presence Protocol (XMPP)*. 2011. Web page. https://xmpp.org/extensions/. XMPP Extensions.

[75] IETF. *The WebSocket Protocol*. 2017. IETF RFC6455. Web page. https://tools.ietf.org/html/rfc6455.

[76] W3C. *The WebSocket API*. 2015. https://www.w3.org/TR/websockets/.

[77] FERNANDES, J. L. et al. Performance evaluation of restful web services and amqp protocol. In: *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*. [S.l.: s.n.], 2013. p. 810–815. ISSN 2165-8528.

[78] THANGAVEL, D. et al. Performance evaluation of mqtt and coap via a common middleware. In: *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. [S.l.: s.n.], 2014. p. 1–6.

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

[80] S, S.; KUMAR, G. S. Performance of iot protocols under constrained network, a use case based approach. In: *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*. [S.l.: s.n.], 2017. p. 1–6.

[81] IOTIFY. *Iotify*. 2019. Web Site. Web page. https://iotify.help/network/.

[82] AMAZON. *Amazon device simulator*. 2019. Web Site. Web page. https://aws.amazon.com/answers/iot/iot-device-simulator/.

[83] KOKKONIS, A. C. G.; KONTOGIANNIS, S. Middleware iot protocols performance evaluation for carrying out clustered data. In: *2018 South-Eastern European Design Automation, Computer Engineering, Computer Networks and Society Media Conference (SEEDA_CECNSM)*. [S.l.: s.n.], 2018. p. 1–6.

[84] Finley, B.; Vesselkov, A. Cellular iot traffic characterization and evolution. In: *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. [S.l.: s.n.], 2019. p. 622–627.

[85] ROMAN C. ALCARAZ, J. L. R.; SKLAVOS, N. The exploration in the education of professionals in applied internet of things engineering. In: *2010 4th International Conference on Distance Learning and Education*. [S.l.: s.n.], 2011.

[86] NING, H.; LIU, H. Cyber-physical-social based security architecture for future internet of things. In: . [S.l.: s.n.], 2012.

[87] YU, J. W. Y.; ZHOU, G. The exploration in the education of professionals in applied internet of things engineering. In: *2010 4th International Conference on Distance Learning and Education*. [S.l.: s.n.], 2010.

[88] EMILIANO, R.; ANTUNES, M. Automatic network configuration in virtualized environment using gns3. In: *2015 10th International Conference on Computer Science Education (ICCSE)*. [S.l.: s.n.], 2015. p. 25–30.

[89] VELIEVA, T. R.; KOROLKOVA, A. V.; KULYABOV, D. S. Designing installations for verification of the model of active queue management discipline red in the gns3. In: *2014 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. [S.l.: s.n.], 2014. p. 570–577. ISSN 2157-023X.

[90] VIJAYALAKSHMI, M.; DESAI, P.; RAIKAR, M. M. Packet tracer simulation tool as pedagogy to enhance learning of computer network concepts. In: *2016 IEEE 4th International Conference on MOOCs, Innovation and Technology in Education (MITE)*. [S.l.: s.n.], 2016. p. 71–76.

[91] SMITH, A.; BLUCK, C. Multiuser collaborative practical learning using packet tracer. In: *2010 Sixth International Conference on Networking and Services*. [S.l.: s.n.], 2010. p. 356–362.

[92] SALONI, S.; HEGDE, A. Wifi-aware as a connectivity solution for iot pairing iot with wifi aware technology: Enabling new proximity based services. In: *2016 International Conference on Internet of Things and Applications (IOTA)*. [S.l.: s.n.], 2016. p. 137–142.

[93] ACER, U. G. et al. Sensing wifi network for personal iot analytics. In: *2015 5th International Conference on the Internet of Things (IOT)*. [S.l.: s.n.], 2015. p. 104–111.

[94] ANG, C. Vehicle positioning using wifi fingerprinting in urban environment. In: *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*. [S.l.: s.n.], 2018. p. 652–657.

[95] Yours Locktime Software. *NetLimiter Website*. 2019. Web page. Https://www.netlimiter.com.

[96] Eclipse Foundation. *Californium*. 2019. Web page. Https://www.eclipse.org/californium/.

[97] Adafruit. *ESP8266WiFi library*. 2017. Web page. `https://github.com/esp8266/Arduino`.

[98] LINKS2004. *WebSocket Server and Client for Arduino*. 2019. `https://github.com/Links2004/arduinoWebSockets`.

[99] https://libcoap.net/. *C-Implementation of CoAP*. 2019. Web page. `https://libcoap.net/`.

[100] FabLab RUC, Roskilde University. *XMPP library for Arduino*. 2019. Web page. `http://fablab.ruc.dk/get-xmpp-protocol-on-arduino/`.

[101] Microsoft. *The Windows Server 2019*. 2019. Web page. `https://www.microsoft.com/pt-br/cloud-platform/windows-server`.

[102] IONESCU, V. M. The analysis of the performance of rabbitmq and activemq. In: *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*. [S.l.: s.n.], 2013.

[103] Ignite Realtime. *Openfire XMPP Server*. 2017. Web page. `https://www.igniterealtime.org/projects/openfire/`.

[104] XMPP Org. *Most Popular Servers for XMPP Protocol*. 2018. Web page. `https://xmpp.org/software/servers.html`.

[105] The Eclipse Foundation. *Eclipse Mosquitto MQTT Server*. 2017. Web page. `https://projects.eclipse.org/projects/technology.mosquitto`.

[106] Microsoft. *Internet Information Services*. 2019. Web page. Https://www.iis.net.

[107] NetCraft. *Web Server Survey*. 2018. Web page. Https://news.netcraft.com/archives/2018/08/24/august-2018-web-server-survey.html.

[108] JúNIOR, P. H. D. da Silva e N. A. Ferramenta iperf: geração e medição de tráfego tcp e udp. In: *Ferramenta IPERF: geração e medição de Tráfego TCP e UDP*. [S.l.: s.n.], 2014.

[109] jagttt@gmail.com. *Clumsy*. 2019. Web page. Https://jagt.github.io/clumsy/.

[110] wireshark.org. *Wireshark*. 2019. Web page. Https://www.wireshark.org/.

[111] Goyal, P.; Goyal, A. Comparative study of two most popular packet sniffing tools-tcpdump and wireshark. In: *2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)*. [S.l.: s.n.], 2017. p. 77–81. ISSN 2472-7555.

[112] The Eclipse Foundation. *Eclipse Mosquitto MQTT Server*. 2017. Web page. `https://projects.eclipse.org/projects/technology.mosquitto`.

[113] The Eclipse Foundation. *Eclipse Mosquitto MQTT Server*. 2017. Web page. `https://projects.eclipse.org/projects/technology.mosquitto`.

[114] IETF. *CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets*. 2018. IETF RFC8323. Web page. `https://datatracker.ietf.org/doc/rfc8323/`.

[115] Adafruit. *Adafruit library*. 2019. Web page. `https://github.com/esp8266/Arduino`.

[116] GEORGES, A.; BUYTAERT, D.; EECKHOUT, L. Statiscally rigorous java performance evaluation. p. 57–76, 2007. Disponível em: <https://dri.es/files/oopsla07-georges.pdf>.

[117] Carro Fernandez, G. et al. From rgb led laboratory to servomotor control with websockets and iot as educational tool. In: *Proceedings of 2015 12th International Conference on Remote Engineering and Virtual Instrumentation (REV)*. [S.l.: s.n.], 2015. p. 32–36.

[118] Vujović, M. et al. Usage of nginx and websocket in iot. In: *2015 23rd Telecommunications Forum Telfor (TELFOR)*. [S.l.: s.n.], 2015. p. 289–292.

[119] Oliveira, G. M. B. et al. Comparison between mqtt and websocket protocols for iot applications using esp8266. In: *2018 Workshop on Metrology for Industry 4.0 and IoT*. [S.l.: s.n.], 2018. p. 236–241.

[120] Slabicki, M.; Grochla, K. Performance evaluation of coap, snmp and netconf protocols in fog computing architecture. In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. [S.l.: s.n.], 2016. p. 1315–1319.

[121] FATHIA, O.; SAID, R. A comparative study of mqtt and coap application layer protocols via. performances evaluation. *Medwell Journals*, 2018. ISSN 1816-949X.

[122] van der Westhuizen, H. W.; Hancke, G. P. Practical comparison between coap and mqtt - sensor to server level. In: *2018 Wireless Advanced (WiAd)*. [S.l.: s.n.], 2018. p. 1–6.

[123] van der Westhuizen, H. W.; Hancke, G. P. Comparison between coap and mqtt - server to business system level. In: *2018 Wireless Advanced (WiAd)*. [S.l.: s.n.], 2018. p. 1–5.

[124] https://projects.eclipse.org/projects/iot.tinydtls. *Eclipse tinydtls - CoAP*. 2019. Web page. `https://projects.eclipse.org/projects/iot.tinydtls`.

[125] BELSHE, M. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. 5 2015. RFC 7540. IETF. Web page. `https://tools.ietf.org/html/rfc7540`.

[126] OASIS. *AMQP version 1.0*. 10 2012. OASIS Web page. `http://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf`.

[127] ANTON-HARO, C.; DOHLER, M. *Machine-to-Machine (M2M) Communications: Architecture, Performance and Applications*. 1st. ed. [S.l.]: Woodhead Publishing, Limited, 2015. ISBN 1782421025, 9781782421023.

[128] Sumthane, M. Y.; Vatti, R. A. Wi-fi interference detection and control mechanism in ieee 802.15.4 wireless sensor networks. In: *2017 Fourth International Conference on Image Information Processing (ICIIP)*. [S.l.: s.n.], 2017. p. 1–6. ISSN null.

[129] PETERSEN, H.; BACCELLI, E.; WäHLISCH, M. Interoperable services on constrained devices in the internet of things. 06 2014.

[130] KONG, J. H.; ANG, L.-M.; SENG, K. P. A comprehensive survey of modern symmetric cryptographic solutions for resource constrained environments. 06 2015.

[131] BELLAVISTA, P.; ZANNI, A. Towards better scalability for iot-cloud interactions via combined exploitation of mqtt and coap. In: *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*. [S.l.: s.n.], 2016.

[132] FORTINO, G. et al. Integration of agent-based and cloud computing for the smart objects-oriented iot. In: *Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. [S.l.: s.n.], 2014. p. 493–498.

[133] SCHAAF, H. van der; HERZOG, R. Mapping the ogc sensorthings api onto the openiot middleware. In: _____. *Interoperability and Open-Source Solutions for the Internet of Things: International Workshop, FP7 OpenIoT Project, Held in Conjunction with SoftCOM 2014, Split, Croatia, September 18, 2014, Invited Papers*. Cham: Springer International Publishing, 2015. p. 62–70. ISBN 978-3-319-16546-2. Disponível em: <http://dx.doi.org/10.1007/9783319165462_6>.

[134] ENNESSER, F.; SHAH, Y. *Security solutions and services for the IoT*. 2016. `www.onem2m.org/images/files/oneM2M_Security_Briefing_A4.pdf`. (IoThink Thought Leader Series) Online. February/2017.

[135] GAJJAR, S. et al. Comparative analysis of wireless sensor network motes. In: *2014 International Conference on Signal Processing and Integrated Networks (SPIN)*. [S.l.: s.n.], 2014. p. 426–431.

[136] LUZURIAGA, J. E. et al. A comparative evaluation of amqp and mqtt protocols over unstable and mobile networks. In: *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*. [S.l.: s.n.], 2015. p. 931–936. ISSN 2331-9852.

[137] PERERA, C. et al. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys Tutorials*, v. 16, n. 1, p. 414–454, First 2014. ISSN 1553-877X.

[138] OASIS. *MQTT For Sensor Networks (MQTT-SN)*. 2013. OASIS Version 1.2. Web page. `http://www.mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf`.

[139] Internet Engineering Task Force (IETF). *Extensible Messaging and Presence Protocol (XMPP)*. 2011. Web page. `https://xmpp.org/`. Internet Engineering Task Force (IETF) Standard (RFC 6120, RFC 6121, and RFC 7622).

[140] SICARI, S. et al. Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, v. 76, p. 146 – 164, 2015. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128614003971>.

[141] ROMAN, R.; NAJERA, P.; LOPEZ, J. Securing the internet of things. *Computer*, v. 44, n. 9, p. 51–58, Sept 2011. ISSN 0018-9162.

[142] ROMAN, R.; ZHOU, J.; LOPEZ, J. On the features and challenges of security and privacy in distributed internet of things. *Comput. Netw.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 57, n. 10, p. 2266–2279, jul. 2013. ISSN 1389-1286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2012.12.018>.

[143] CISCO. *Seize New Product and Revenue Opportunities with the Internet of Things*. [S.l.].

[144] DESAI, P.; SHETH, A.; ANANTHARAM, P. Semantic gateway as a service architecture for iot interoperability. *2015 IEEE International Conference on Mobile Services (MS)*, IEEE Computer Society, Los Alamitos, CA, USA, v. 00, p. 313–319, 2015.

[145] GANZHA, M. et al. Semantic technologies for the iot - an inter-iot perspective. In: *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*. [S.l.: s.n.], 2016. p. 271–276.

[146] KALRA, S.; SOOD, S. K. Secure authentication scheme for iot and cloud servers. p. 83–92, 2016.

[147] SECURITY, U. D. of H. *STRATEGIC PRINCIPLES FOR SECURING THE INTERNET OF THINGS (IoT)*. November 2016. `https://www.dhs.gov/sites/default/files/publications/Strategic_Principles_for_Securing_the_Internet_of_Things-2016-1115-FINAL_v2-dg11.pdf`.

[148] BARRETO, F. M. et al. Coap-ctx: A context-aware coap extension for smart objects discovery in internet of things. In: *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*. [S.l.: s.n.], 2017. v. 1, p. 575–584. ISSN 0730-3157.

[149] OLIVEIRA, L. B. et al. The computer for the 21st century: Security privacy challenges after 25 years. In: *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. [S.l.: s.n.], 2017. p. 1–10.

[150] XIONG, X.; FU, J. Active status certificate publish and subscribe based on amqp. In: *2011 International Conference on Computational and Information Sciences*. [S.l.: s.n.], 2011. p. 725–728.

[151] YASSEIN, M. B.; SHATNAWI, M. Q.; AL-ZOUBI, D. Application layer protocols for the internet of things: A survey. In: *2016 International Conference on Engineering MIS (ICEMIS)*. [S.l.: s.n.], 2016. p. 1–4.

[152] KAYAL, P.; PERROS, H. A comparison of iot application layer protocols through a smart parking implementation. In: *Innovations in Clouds, Internet and Networks (ICIN)*. [S.l.: s.n.], 2017. p. 1–4.

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

[154] MANDYAM, G. D.; EHSAN, N. Impact of html5 persistent connectivity to power consumption for the mobile web. In: *2015 7th International Conference on Communication Systems and Networks (COMSNETS)*. [S.l.: s.n.], 2015. p. 1–2. ISSN 2155-2487.

[155] EUROTECH Web Site. *Multi-service IoT Edge Gateways*. 2018. Web page. Https://www.eurotech.com/en/products/iot.

[156] Eclipse Gateways. *Eclipse Gateways*. 2018. Web page. Https://iot.eclipse.org/gateways/.

[157] CISCO. *Cisco 900 Series Industrial Gateway*. 2018. Web page. Https://www.cisco.com/c/en/us/products/routers/900-series-industrial-routers/index.html.

[158] IEEE Cloud Definition. *Cloud Computing Definition, Reference Architecture, and General Use Cases*. 2018. Web page. `https://cloudcomputing.ieee.org/images/files/education/studygroup/Cloud_Computing_Definition_Reference_Architecture_and_General_Use_Cases.pdf`.

[159] Eclipse Foundation. *SmartHome*. 2018. Web page. Https://www.eclipse.org/smarthome/.

[160] The Apache Software Foundation. *Apache Server*. 2017. Web page. Apache Server, `https://httpd.apache.org/`.

[161] https://meioambientedigital.pmnf.rj.gov.br/. *Fiware Nova Friburgo*. 2017. Web page. Fiware Nova Friburgo, `https://meioambientedigital.pmnf.rj.gov.br/`.

[162] OLIVEIRA, G. et al. Computação urbana: Tecnologias e aplicações para cidades inteligentes. In: *SBRC 2016. Minicurso 1 (MC-2). Salvador, BA.* [S.l.: s.n.], 2016. p. 256–315.

[163] COUTINHO, A. A. R.; CARNEIRO, E.; GREVE, F. Computação em névoa: Conceitos aplicações e desafios. In: *SBRC 2016. Minicurso 6 (MC-6). Salvador, BA.* [S.l.: s.n.], 2016. p. 1–50.

[164] PIRES, P. F. et al. Plataformas para a internet das coisas. In: *SBRC 2015. Minicurso 3 (MC-3). Vitória, ES.* [S.l.: s.n.], 2015. p. 110–169.

[165] INTERSCITY.ORG. *Escola São Paulo de Ciência Avançada em Cidades Inteligentes*. 2017. IME-USP. INCT da Internet do Futuro para Cidades Inteligentes. `http://interscity.org/advanced-school/`.

[166] ZHENG, G. et al. A link quality inference model for ieee 802.15.4 low-rate wpans. In: *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011.* [S.l.: s.n.], 2011. p. 1–6. ISSN 1930-529X.

[167] TABARI, B. M. et al. Low latency live video streaming on android devices using web-socket. In: *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT).* [S.l.: s.n.], 2017. p. 1–6.

[168] WORKING GROUP RFID - ETP EPOSS. Internet of things in 2020. *European Technology Platform on Smart Systems (EPoSS)*, September 2008. `http://www.smart-systems-integration.org/public/documents/publications/Internet-of-Things_in_2020_EC-EPoSS_Workshop_Report_2008_v3.pdf`.

[169] Raspberry Pi Foundation. *Raspberry Pi 3 Model B*. 2017. Web page. `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`.

[170] PARTICLE. *Security Checklist for the Internet of Things. An essential guide to securing connected products*. 2017. Web page. `https://www.particle.io/white-papers/securing-internet-of-things-products`.

[171] PARTICLE. *Particle Photon (Wi-Fi)*. 2017. Web page. `https://www.particle.io/products/hardware/photon-wifi-dev-kit`.

[172] RICCIARDI, F. *KERBEROS PROTOCOL TUTORIAL*. 2007. `http://www.kerberos.org/software/tutorial.html`. MIT Kerberos & Internet trust (MIT-KIT) Consortium.

[173] HARDT, D. *The OAuth 2.0 Authorization Framework*. 10 2012. Request for Comments: 6749, Internet Engineering Task Force (IETF). Category: Standards Track. ISSN: 2070-1721. `https://tools.ietf.org/html/rfc6749`.

[174] http://couchdb.apache.org/. *noSql database*. 2017. Web page. NoSql database, `http://couchdb.apache.org/`.

[175] ITU-T. *ITU-T*. 2018. Web page. `https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx`.

[176] ITU-T. *Reference architecture for IoT device capability exposure*. 4 2017. Global Information InfraStructure, Internet Protocol Aspects, Next-generation Networks, Internet of Things and Smart Cities. https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Y.4115-201704-I!!PDF-E&type=items.

[177] ITU-T. *Common requirements and capabilities of device management in the Internet of Things*. 3 2016. Global Information InfraStructure, Internet Protocol Aspects, Next-generation Networks, Internet of Things and Smart Cities. https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Y.4702-201603-I!!PDF-E&type=items.

[178] ISO/IEC. *International Organization for Standardization*. 2014. Web page. https://www.iso.org/standard/64955.html.

[179] Tigase, Inc. *Tigase JaXMPP Client Library*. 2017. Web page. https://tigase.tech/projects/jaxmpp2.

[180] Many. *Arduino*. 2017. Web page. https://www.arduino.cc/.

[181] https://github.com/1248/microcoap. *Biblioteca CoAP para Arduino*. 2017. Web page. https://github.com/1248/microcoap.

[182] The Eclipse Foundation. *Californium*. 2017. Web page. https://www.eclipse.org/californium/.

[183] Sociedade Brasileira de Computação. *37º Jornada de Atualização em Informática (JAI)*. 2018. Web page. http://natal.uern.br/eventos/csbc2018/?page_id=331.

[184] Sociedade Brasileira de Computação. *Anais - 37º Jornada de Atualização em Informática (JAI)*. 2018. Web page. http://natal.uern.br/eventos/csbc2018/wp-content/uploads/2018/08/Anais-Final-JAI-2018.pdf.

[185] GREENWADE, G. D. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, v. 14, n. 3, p. 342–351, 1993.

[186] SATYANARAYANAN, M.; BAHL, P.; CACERES, R. The case for vm-based cloudlets in mobile computing. In: . [S.l.: s.n.], 2009.

[187] IEEE Cloud Definition. *Cloud Computing Definition, Reference Architecture, and General Use Cases*. 2018. Web page. https://cloudcomputing.ieee.org/images/files/education/studygroup/Cloud_Computing_Definition_Reference_Architecture_and_General_Use_Cases.pdf.

[188] TUPINAMBá, A.; SZTAJNBERG, A. Distributedcl: A framework for transparent distributed gpu processing using the opencl api. In: *2012 13th Symposium on Computer Systems*. [S.l.: s.n.], 2012. p. 187–193.

[189] thingTronics Innovations. *ESP-CoAP server/client library for Arduino*. 2018. Web page. https://github.com/automote/ESP-CoAP.

[190] O'LEARY, N. *Arduino Client for MQTT*. 2017. Web page. https://pubsubclient.knolleary.net/.

[191] Eclipse Paho. *Eclipse Paho JavaScript Client*. 2018. Web page. https://www.eclipse.org/paho/clients/js/.

[192] NODEMCU. *NodeMcu. Connect Things EASY*. 2014. Web page. http://www.nodemcu.com.

[193] Node.js Foundation. *Node.js*. 2018. Web page. https://nodejs.org/en/.

[194] ORACLE. *Oracle GlassFish Server*. 2017. Web page. http://www.oracle.com/technetwork/middleware/glassfish/overview/index.html.

[195] Eclipse Foundation. *Eclipse Tyrus*. 2018. Web page. https://projects.eclipse.org/projects/ee4j.tyrus.

[196] https://github.com/Links2004. *WebSocket Server and Client for Arduino*. 2018. Web page. https://github.com/Links2004/arduinoWebSockets.

[197] Arduino team. *Arduino core for ESP8266 WiFi chip*. 2018. Web page. https://github.com/esp8266/Arduino.

[198] IETF. *The JavaScript Object Notation (JSON) Data Interchange Format*. 2017. RFC 8259. IETF. Web page. https://tools.ietf.org/html/rfc8259.

[199] Nghia TH. *IoT MQTT Dashboard*. 2018. Google Play. https://play.google.com/store/apps/details?id=com.thn.iotmqttdashboard&hl=pt_BR.

[200] POSTSCAPES. *Postscapes - IoT Protocols*. 2019. Web Site. Web page. https://www.postscapes.com/internet-of-things-protocols/.