CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Matheus dos Santos Teles Ribeiro

ANÁLISE DE DESEMPENHO ENERGÉTICO E DE REDE DE PROTOCOLOS DE COMUNICAÇÃO PARA INTERNET DAS COISAS EM UM CENÁRIO DE VAREJO INTELIGENTE

Timóteo

2019

Matheus dos Santos Teles Ribeiro

ANÁLISE DE DESEMPENHO ENERGÉTICO E DE REDE DE PROTOCOLOS DE COMUNICAÇÃO PARA INTERNET DAS COISAS EM UM CENÁRIO DE VAREJO INTELIGENTE

Monografia apresentada à Coordenação de Engenharia de Computação do Campus Timóteo do Centro Federal de Educação Tecnológica de Minas Gerais para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Me. Talles Quintão Pessoa

Timóteo

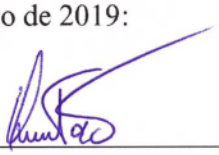
2019

Matheus dos Santos Teles Ribeiro

ANÁLISE DE DESEMPENHO ENERGÉTICO E DE REDE DE PROTOCOLOS DE COMUNICAÇÃO PARA INTERNET DAS COISAS EM UM CENÁRIO DE VAREJO INTELIGENTE

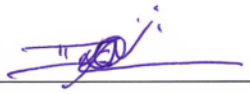
Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais, campus Timóteo, como requisito parcial para obtenção do título de Engenheiro de Computação.

Trabalho aprovado. Timóteo, 07 de agosto de 2019:



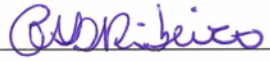
Prof. Me. Talles Quintão Pessoa

Orientador



Prof. Dr. Elder de Oliveira Rodrigues

Professor Convidado



Prof. Gustavo Henrique dos Santos Ribeiro

Professor Convidado

Timóteo

2019

Resumo

A criação do conceito de internet das coisas teve impacto direto na sociedade, possibilitando a integração de tecnologias de sensoriamento distribuído e transmissão de dados em processos corriqueiros. Porém, a adoção efetiva deste paradigma de estruturação de processos ainda não se deu por completo, devido a desafios encontrados ao se projetar e implantar sistemas de internet das coisas, destacando-se as restrições de fonte de energia e o tráfego de rede destes dispositivos. Dado que uma das causas dos problemas apontados se encontra no protocolo utilizado para comunicação entre os dispositivos, foi implantado neste trabalho um exemplo de sistema de internet das coisas, utilizando-se dois protocolos distintos : o HTTP, protocolo utilizado em larga escala atualmente para transferência de informações e acesso a recursos pela internet, e o MQTT, um protocolo leve e especializado para comunicação entre dispositivos com recursos limitados. Foi utilizado um microcontrolador Arduino Uno munido de um módulo de comunicação *wireless* Xbee Series 2 como dispositivo cliente, e um Raspberry Pi como dispositivo servidor. Foi estabelecida conexão entre os dispositivos comunicando-se através dos protocolos selecionados, comparando-se as respectivas performances nos quesitos consumo energético e tráfego e rede. Pôde-se observar que o MQTT obtém uma grande vantagem quando utilizado em aplicações da internet das coisas.

Palavras-chave: Internet das coisas, Xbee, HTTP, MQTT.

Abstract

The creation of the internet of things concept had a direct impact on society, which enabled the integration of distributed sensing and data transmission technologies into everyday processes. However, the effective adoption of this process structuring paradigm is not yet widespread, due to challenges encountered in designing and deploying IoT systems, especially the power source and network traffic constraints of these devices. Given that one of the causes of these problems lies in the communication protocol embedded in the devices, an example of an IoT system was developed, utilizing two distinct protocols: HTTP, a protocol currently used in large scale in information transfer and access to resources across the internet, and MQTT, a lightweight, specialized protocol for communication between resource-constrained devices. An Arduino Uno microcontroller with an Xbee Series 2 wireless communication module was used as a client device and a Raspberry Pi as a server device. A connection between the devices was established by communicating through the selected protocols, comparing their performances in energy consumption and network traffic. It was observed that MQTT has great advantages when used in IoT applications.

Keywords: Internet of things, Xbee, HTTP, MQTT.

Lista de ilustrações

Lista de abreviaturas e siglas

IoT	*Internet of Things*, em português, Internet das Coisas
TIC's	Tecnologias de Informação e Comunicação
M2M	*Machine to Machine*, em português, Máquina para Máquina
MQTT	*Message Queue Telemetry Transport*
HTTP	*HyperText Transfer Protocol*
RFID	*Radio-Frequency Identification*, em português, Identificação por Rádio-Frequência
WSN	*Wireless Sensor Network*, em português, Rede de sensores sem-fio
API	*Application Programming Interface*, em português, Interface de Programação de Aplicações
QoS	*Quality of Service*, em português, Qualidade de Serviço
TCP	*Transmission Control Protocol*, em português, Protocolo de Controle de Transmissão
XML	*Extended Markup Language*, formato de marcação de dados para compartilhamento
JSON	*Javascript Object Notation*, formato de marcação de dados para compartilhamento

Sumário

1 Introdução

Dados os avanços conquistados na última década no que se refere a criação de tecnologias distribuídas, o conceito de Cidade Inteligente (do inglês Smart City) e pesquisas sobre os serviços necessários para sua consolidação, vem ganhando popularidade entre academicistas e empresários de diversos ramos de negócio (PANTANO; TIMMERMANS, 2014).

A expressão Cidade Inteligente foi cunhada no começo dos anos 90 para designar um novo paradigma de estruturação das cidades, onde o ambiente urbano se tornaria "dependente de tecnologia, inovação e globalização, principalmente em uma perspectiva econômica" (RIZZON et al., 2017).

Posteriormente, o conceito de cidades inteligentes consolidou-se como uma cidade em que há coordenação e integração de Tecnologias de Informação e Comunicação (TIC's) nos diversos ambientes do espaço urbano, com o objetivo de promover a evolução contínua, a melhoria da qualidade de vida, e a eficiência dos serviços oferecidos (RIZZON et al., 2017).

Diretamente associados aos conceitos de cidade inteligente, "serviços inteligentes" vêm sendo desenvolvidos, com a proposta de melhorar a qualidade de processos do dia-a-dia das cidades tanto quanto a satisfação dos agentes envolvidos (usuários e provedores de serviços). Exemplos desses serviços inteligentes incluem: automação e segurança residencial e industrial, monitoramento de saúde, monitoramento de trânsito, controle de vendas e estoque, dentre diversos outros (PANTANO; TIMMERMANS, 2014).

Dentro do escopo de serviços inteligentes, emerge a ideia de "mercado inteligente", um novo paradigma de gerenciamento de comércios, que se baseia na tecnologia tanto para oferecer serviços personalizados, aumentando a qualidade de vida do consumidor, quanto para melhorar a rentabilidade do negócio através de coleta e análise de dados (PANTANO; TIMMERMANS, 2014).

1.1 Motivação

A constante evolução das TIC's nas últimas décadas foi causa de uma mudança radical no papel que a tecnologia ocupa no dia-a-dia da sociedade. Há poucas décadas, ter um computador pessoal em casa era raridade, e hoje vive-se num mundo onde é previsto que até 2021, 28 bilhões de dispositivos estarão conectados à internet (ERICSSON, 2015).

Naturalmente, com esta exposição contínua à tecnologia (principalmente entre as gerações que já nasceram imersas nas tecnologias mais recentes), são geradas maiores expectativas quanto à inserção de novas tecnologias nos diversos âmbitos da vida moderna. No que se refere ao varejo inteligente, encontram-se inovações tecnológicas como *checkout* próprio, localização de produtos em tempo real no espaço físico de uma loja, painéis informativos inteligentes, entre outras (PRIPORAS; STYLOS; FOTIADIS, 2017).

Porém, a inserção de tais tecnologias no ambiente de varejo não beneficia somente o consumidor final. Varejistas de todos os ramos tem a ganhar utilizando sistemas de IoT (Internet of Things, do inglês “Internet das coisas”) em seus estabelecimentos. Processos rotineiros como cálculo de estoque, controle de vendas e perdas, e balanço financeiro são grandemente facilitados, ao mesmo tempo em que se possibilita a implantação de serviços completamente novos, como mineração e análise de dados das vendas (LEE; LEE, 2015).

1.2 Problema

Ainda que o surgimento dos conceitos de “mercado inteligente” apresentem inovações extremamente relevantes para a economia atual, a modernização do varejo clássico tem encontrado resistência devido a desafios existentes nas soluções IoT atuais (PEREIRA; AGUIAR, 2014). Segundo Lee e Lee (2015) entre estes desafios destacam-se:

- Gerenciamento de dados

  As empresas que decidirem adotar o paradigma IoT em seus estabelecimentos necessitarão de centros de processamentos de dados suficientemente grandes para que consigam acomodar o volume de dados gerado pelas redes de sensores.

- Mineração de dados

  Para extrair informação relevante ao negócio a partir dos dados coletados pelos sensores, será necessária a utilização de ferramentas especializadas em interpretação e análise de dados (Big Data Analytics) operadas por um profissional da área.

- Manutenção da privacidade e segurança

  Provedores de serviços inteligentes e seus usuários estarão preocupados com a privacidade e a segurança dos dados pessoais gerados. Logo será de responsabilidade do provedor disponibilizar um ambiente seguro contra ameaças de rede externas (munido de autenticação de dados, criptografia ponta-a-ponta e proteção contra ataques de rede).

- Contenção de reações em cadeia (caos).

  Os sistemas devem ser tolerantes a falhas, ou seja, um erro fatal em um dispositivo não pode comprometer o funcionamento do sistema, tampouco a integridade dos dados gerados.

Outra grande preocupação sobre a viabilidade de dispositivos IoT reside na manutenção da vida útil da fonte de energia, que em sua grande maioria, consiste em baterias (recarregáveis ou não). É desejável que um dispositivo consuma o mínimo de energia possível em suas operações corriqueiras (sensoriamento, processamento e transmissão de dados), e entre em standby quando finalizá-las, a fim de estender o tempo de funcionamento do mesmo.

Blaauw et al. (2014) ainda menciona que devido à tendência dos sensores IoT de estarem localizados em locais em que a substituição da fonte de energia é uma tarefa custosa,

podemos dizer que o desafio universal da arquitetura IoT é desenvolver dispositivos com baixo consumo de bateria.

Kiani (2018) aponta um desafio no balanceamento de carga da rede. Uma rede desbalanceada (onde o tráfego está mal distribuído entre os nós da rede) pode levar a um consumo elevado de largura de banda e um aumento de tráfego em certos dispositivos, aumentando o gasto energético e consequentemente, reduzindo o tempo de vida da rede como um todo.

Segundo Gupta e Quamara (2018), existem diversas tecnologias e pesquisas em desenvolvimento com o intuito de solucionar (ou ao menos mitigar) tais desafios. Para os problemas de armazenamento, mineração de dados e manutenção da privacidade e segurança, podemos pontuar a utilização de aplicações Big Data, inteligências artificiais e redes neurais e a implementação de algoritmos de criptografia, respectivamente, como possíveis técnicas de se contornar as dificuldades apresentadas.

Porém, embora também existam tentativas de se solucionar o problema de vida útil da fonte de energia de dispositivos IoT, tal questão continua crítica para a progressão do desenvolvimento das tecnologias inteligentes como um todo, consequentemente atrasando a adoção total de tais tecnologias pelos grandes nomes do mercado e desencorajando empresários de negócios de pequeno e médio porte a investirem neste modelo de negócio. Logo, este trabalho terá como objeto de estudo os problemas de consumo de energia em dispositivos IoT.

1.3 Objetivos

Dado o problema apresentado, foi levantada a hipótese de que parte da ineficiência energética que acomete os dispositivos IoT advém do protocolo de comunicação escolhido para a comunicação dispositivo-servidor e dispositivo-dispositivo. Isso se justifica pelo fato de que protocolos que possuem uma necessidade alta de processamento para realizar uma comunicação efetiva, acabam por utilizar maior poder computacional e, consequentemente, energia (recursos escassos em dispositivos embarcados, como já citado).

Feitas estas considerações, e estudando-se os protocolos de comunicação mais difundidos atualmente, o protocolo MQTT foi eleito como possível solução para o problema apresentado, devido às suas características, descritas na Seção 2.3.

Assim, o presente trabalho tem como objetivo analisar o desempenho energético de protocolos de aplicação utilizados em IoT, e ainda, verificar a viabilidade do protocolo MQTT como protocolo de comunicação M2M (*Machine to Machine*, do inglês "Máquina para Máquina").

O trabalho tem como objetivos específicos:

- Construir um cenário de rede que se encaixe no conceito de mercado inteligente apresentado anteriormente.
- Aplicar diferentes protocolos de comunicação neste cenário, sendo um deles o MQTT.

- Realizar testes de eficiência energética e de tráfego de rede nos dispositivos e discutir o desempenho e o impacto da utilização de cada protocolo na vida útil dos dispositivos.
- Discutir a utilização do MQTT como protocolo de comunicação M2M em relação aos outros protocolos existentes.

Ao fim deste trabalho, espera-se confirmar que a aplicação do protocolo MQTT em um cenário de mercado inteligente contribuirá para a mitigação (ou solução) do desafio de consumo de bateria e tráfego de rede dos dispositivos IoT.

2 Fundamentação Teórica

2.1 Redes de Sensores sem Fio

Avanços recentes na tecnologia possibilitaram a criação de novos tipos de sensores, caracterizados por seu baixo custo, baixo consumo de energia, tamanho reduzido e habilidade de comunicação em curtas distâncias, sendo dotados de unidades de sensoriamento, processamento de dados e comunicação. Estes dispositivos possibilitaram o desenvolvimento de paradigmas de utilização colaborativa de uma malha composta de grande número destes sensores, o que deu origem ao conceito de Redes de Sensores sem Fio (WSNs, do inglês *Wireless Sensor Networks*).

Formalmente, uma rede de sensores sem fio é composta de um grande número de nós sensores que são dispostos densamente nas proximidades de uma região (ou fenômeno) que se deseje monitorar. Devido a esta densidade de nós sensores, a rede possibilita que a posição geográfica dos sensores tenha menor importância, facilitando a disposição dos mesmo em áreas inacessíveis ou de risco. Outra característica importante das redes de sensores sem fio é a natureza colaborativa dos nós sensores, que, munidos de unidades de processamento, realizam pequenos tratamentos de dados localmente para transmitir somente a informação necessária e já parcialmente processada.

Uma das principais limitações nos dispositivos sensores é manter o mínimo de consumo de energia o possível, pois sensores acabam por ter fontes de energia limitadas e, em alguns casos, insubstituíveis. Logo, enquanto redes de sensoriamento tradicionais tem como foco conseguir altos índices de QoS, protocolos para WSNs se preocupam primariamente em conservação de energia, prolongando a vida útil da rede ao invés de atingir grandes taxas de transferência.

WSNs podem consistir de dispositivos com sensores de diversos tipos, e assim, monitorar uma grande variedade de situações e fenômenos naturais como temperatura, umidade, iluminação, ruído, velocidade, direção, tamanho, pressão, entre muitos outros.

Segundo Akyildiz et al. (2002), nós sensores podem ser usados para sensoriamento contínuo, detecção e identificação de eventos, localização e controle de atuadores. Já Yick, Mukherjee e Ghosal (2008) classificam dois tipos principais de aplicações das WSNs: rastreamento e monitoramento. Akyildiz et al. (2002) e Yick, Mukherjee e Ghosal (2008) ainda define que as WSNs podem ser aplicadas em uma variedade de áreas, que incluem:

- Ambiental

  Aplicações ambientais incluem detecção de incêndios florestais e inundações, monitoramento de espécies animais em extinção, mapeamento do movimento migratório de certas espécies, agricultura de precisão, monitoramento do clima, entre outras.

- Logística

Aplicações para logística de pessoas e materiais incluem rastreamento e monitoramento de estoque, frotas de veículos e movimentação de pessoal, entre outras.

- Militar

  WSNs podem ser parte integral dos sistemas militares denominados C4ISRT (*Command, control, communication, computing, intelligence, surveillance, reconaissance and targeting*). Devido ao baixo custo e grande densidade de sensores numa WSN, a destruição de parte deles por ações hostis não afetaria profundamente o andamento de uma operação militar, tornando-os uma alternativa mais viável para áreas de confronto.

  Aplicações incluem monitoramento de forças aliadas, munição e equipamento, monitoramento do campo de batalha e terrenos críticos, reconhecimento de forças inimigas e terrenos hostis, detecção e reconhecimento de ataques biológicos, sistemas de mira, entre muitos outras.

- Residencial

  Aplicações residenciais incluem principalmente a automação residencial, segurança doméstica, monitoramento de consumo de energia e água, entre outras.

- Saúde

  Aplicações na área da saúde incluem monitoramento de pacientes, controle da administração de remédios em pacientes, monitoramento de sinais vitais, temperatura, monitoramento da saúde de bebês, entre outras.

2.2 O conceito de Internet das Coisas (IoT)

O conceito de IoT foi proposto em 1999 por Kevin Ashton, co-fundador do Auto-ID Center no Instituto de Tecnologia de Massachusetts (MIT). Na época, IoT foi descrita como uma interconexão do mundo físico e a Internet através do uso de sensores para observar e identificar o mundo real. (GUPTA; QUAMARA, 2018).

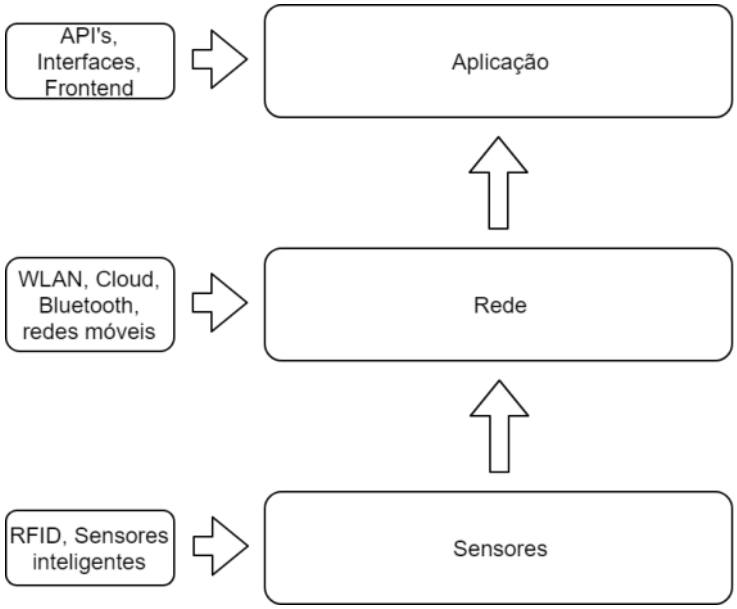
Porém, sendo uma tecnologia recente e em fase de pesquisa, uma definição padrão e aceita globalmente para IoT inexiste. Chen et al. (2014) agrupou definições de IoT dadas por diferentes organizações, dispostas no Quadro 1.

A arquitetura de IoT mais difundida na atualidade propõe um modelo de desenvolvimento em três camadas: sensoriamento, rede e aplicação (Figura 1). A camada de sensoriamento contém todos os sensores, leitores RFID e controladores responsáveis pela coleta de dados do mundo externo. A camada de rede define as tecnologias de rede utilizadas para transmitir os dados da camada de sensoriamento para a camada de aplicação. Por fim, na camada de aplicação, encontram-se as aplicações e API's IoT que utilizam dos dados das camadas inferiores para gerar informações (CHEN et al., 2014).

Quadro 1 – Definições de IoT segundo diferentes organizações (CHEN et al., 2014).

Organização	Definição
CCSA (China Communications Standards Association)	Uma rede, que pode coletar informações do mundo físico ou controlar os objetos do mundo físico através de vários dispositivos implantados com capacidade de percepção, computação, execução e comunicação, e suportar comunicações entre seres humanos e dispositivos (ou entre dispositivos), transmitindo, classificando e processando informações.
ITU-T (International Telecommunication Union - Telecommunication Standardization Sector)	Uma infraestrutura para a sociedade de informação, habilitando serviços avançados interconectando objetos (físicos e virtuais) baseados em tecnologias interoperáveis de informação e comunicação existentes em evolução.
EU FP7 (European Union Framework Programme 7) CASAGRAS (Coordination and Support Action for Global RFID-related Activities and Standardisation)	Uma infraestrutura de rede global, conectando objetos físicos e virtuais através da exploração de capacidades de captura e comunicação de dados.
IETF (Internet Engineering Task Force)	Uma rede mundial de objetos unicamente endereçáveis interconectados, baseados em um protocolo padronizado de comunicações.

Figura 1 – Representação da arquitetura de IoT e suas tecnologias associadas.



(Fonte: o Autor)

2.3 MQTT

O MQTT (Message Queuing Telemetry Transport) é um protocolo de transporte de mensagens entre cliente e servidor, fundado em 1999 por Arlen Nipper (Arcom, hoje Eurotech) e Andy Stanford-Clark (IBM), e se tornou um padrão OASIS em 2014.

O protocolo é baseado em um paradigma de publicar/inscrever, onde cada mensagem gerada é associada a um tópico, e clientes inscritos em um determinado tópico receberão todas as mensagens associadas a ele.

É caracterizado como leve, simples, de código aberto e de fácil implementação, tornando-o ideal para aplicações M2M (Machine to Machine, do inglês “máquina para máquina”) embutidas em dispositivos com recursos limitados, onde é necessária a aplicação de um protocolo com baixo impacto de memória e baixo consumo de largura de banda . O protocolo funciona sobre a arquitetura TCP/IP (ou qualquer outro protocolo que suporte conexões bidirecionais, ordenadas e sem perda)(OASIS, 2014). Características do MQTT incluem :

- Uso de padrão publicar/inscrever, o que possibilita distribuição de mensagens “de um para muitos” ;
- Um transporte de mensagens que é agnóstico ao conteúdo do pacote.
- Três formas de QoS (“Quality of Service”, do inglês “Qualidade do Serviço”)
  - QoS 0 - At most once (“No máximo uma vez”): Mensagens são enviadas sem a preocupação com a chegada das mesmas ao destino.
  - QoS 1 - At least once (“No mínimo uma vez”): O protocolo garante a chegada da mensagem ao seu destino final, porém pode ocorrer a transmissão de duplicatas.
  - QoS 2 - Exactly once (“Exatamente uma vez”): O protocolo garante a chegada da mensagem ao seu destino e garante a inexistência de duplicatas.
- Baixo overhead de transporte para reduzir o tráfego de rede.
- Mecanismo de notificação em caso de ocorrência de eventos anormais.(OASIS, 2014)

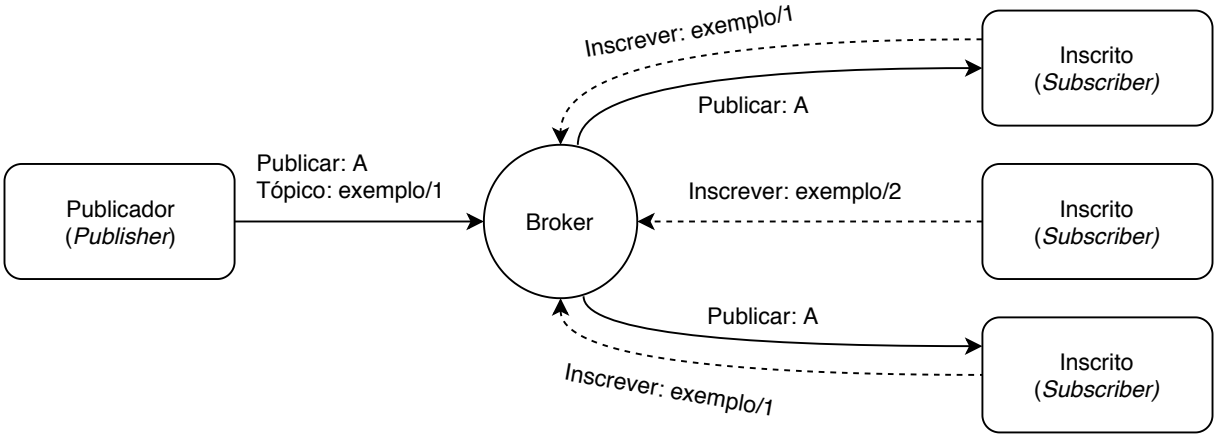
O funcionamento do protocolo se dá através de dois elementos fundamentais como demonstrado na Figura 2: Os clientes MQTT, e o Broker. Um cliente MQTT pode ser qualquer dispositivo IoT capaz de enviar *(publisher)* e/ou receber *(subscriber)* dados. Para isso é necessário que o dispositivo possua uma biblioteca cliente MQTT instalada e esteja em uma conexão com um broker MQTT.

O Broker MQTT é um dispositivo central que tem como objetivo gerenciar a comunicação entre os *publishers* e os *subscribers*, recebendo as mensagens de todos os *publishers*, e as direcionando para os *subscribers* inscritos nos tópicos de interesse. Brokers MQTT também são responsáveis por realizar autenticação, autorização dos clientes e encriptação das mensagens. Diversos brokers inplementam o protocolo MQTT, como Mosquitto, HiveMQ, Eclipse IOT, Solace, entre outros (XU; MAHENDRAN; RADHAKRISHNAN, 2016).

2.4 HTTP

O protocolo HTTP (Hypertext Transfer Protocol, do inglês *Protocolo de Transferência de Hipertexto*) é um protocolo da camada de aplicação utilizado em sistemas de informação

Figura 2 – Representação do funcionamento do MQTT.



(Fonte: o Autor)

distribuída, sendo usado como principal forma de comunicação na internet desde os anos 90 (FIELDING et al., 1999).

Sua primeira versão, o HTTP/0.9 permitia somente a transferência de dados puros através da Internet. Posteriormente, com a definição do protocolo HTTP/1.0, em 1996, as funcionalidades do HTTP foram extendidas, permitindo mensagens a conter vários tipos de dados em uma mesma requisição (MIME) e ainda, possuir metadados. Motivado por limitações da versão 1.0, em Junho de 1999 foi proposta uma melhoria ao protocolo HTTP denominada HTTP/1.1 (FIELDING et al., 1999).

O protocolo HTTP utiliza conexões TCP para transportar dados através da rede. Versões antigas do protocolo (HTTP/0.9 e HTTP/1.0) utilizavam uma conexão TCP por operação, o que se mostrava ineficiente, pois como as informações que trafegam na rede geralmente são pequenas, grande parte dos pacotes enviados eram pacotes de controle do próprio protocolo TCP utilizados para abertura e fechamento de conexões. Logo, era gerado *overhead* desnecessário na comunicação e ainda congestão na rede (NIELSEN et al., 1997).

A partir do HTTP/1.1 foi introduzida a técnica de *pipelining*, que consiste em manter conexões TCP persistentes entre operações consecutivas no mesmo servidor, o que permite o envio de várias requisições sem esperar por uma resposta, evitando envio desnecessário de pacotes o *overhead* causado por aberturas de conexão sucessivas, melhorando a performance da rede como um todo.

O protocolo HTTP se utiliza de mensagens padronizadas para o envio de mensagens aos servidores, denominadas requisições. Uma requisição HTTP deve conter obrigatoriamente o método HTTP a ser utilizado, o recurso a ser acessado, a versão do protocolo e o host de destino. Podem também ser adicionados à requisição diversos tipos de cabeçalhos, com o objetivo de definir os parâmetros de operação da transação HTTP. Fielding et al. (1999), na especificação do protocolo HTTP define os métodos utilizados para identificar a ação que será executada em um recurso (URI).

- GET

  O método GET é responsável por obter qualquer informação identificável por uma URI. Se, por exemplo, a URI requisitada endereça um processo de criação de dados, estes dados serão retornados juntamente com a resposta da requisição. A resposta de uma requisição GET pode ser armazenada em cache e reutilizada novamente, evitando envio desnecessário de requisições.

- HEAD

  O método HEAD é idêntico ao GET, exceto pelo fato que o servidor não retornará nenhum conteúdo no corpo da mensagem de resposta. Este método é utilizado somente para obter metadados sobre o recurso endereçado pela URI sem a necessidade de se transferir todo o conteúdo da requisição, o que o torna útil para validação de links e *web-crawling*.

- POST

  O método POST é utilizado para requisitar que o servidor destino aceite os dados enviados pelo cliente no corpo da requisição. A função real executada pelo método POST é determinada pelo servidor, e pode significar um envio de dados para uma aplicação ou uma operação no banco de dados do servidor.

- PUT

  O método PUT cria uma requisição para modificar os dados de um recurso identificado por uma URI. Este método se difere do POST, pois enquanto o POST envia os dados para um recurso que trata dados genéricos, o PUT envia dados assumindo que se saiba de antemão o recurso a ser alterado.

- DELETE

  O método DELETE tem a funcionalidade de indicar a deleção de um recurso no servidor. Porém, como na maioria dos casos é indesejável manter esta funcionalidade disponível para aplicações cliente, poucos serviços a utilizam de acordo com as especificações.

- TRACE

  O método TRACE possibilita a aplicação cliente visualizar como os dados enviados por ele são recebidos ao final do processo da requisição. O destino da requisição TRACE é o servidor final ou o primeiro servidor proxy encontrado.

- OPTIONS

  O método OPTIONS representa uma requisição que retorna todos os métodos que podem ser aplicados a um recurso no determinado servidor, sem necessitar de se iniciar um envio de dados ou uma obtenção de recursos.

- CONNECT

  O método CONNECT converte a conexão atual em um tunelamento TCP/IP transparente, normalmente utilizado para comunicação HTTPS.

- PATCH

  O método PATCH cria uma requisição com o objetivo de realizar modificações parciais a um recurso.

Servidores HTTP de propósito geral devem implementar ao menos os métodos GET e HEAD, tornando os outros opcionais pela definição. Alguns dos cabeçalhos de utilização mais comuns incluem:

- Accept-Charset

  Determina a codificação do conjunto de caracteres a ser utilizado na requisição.

- Cookie

  Envia ao servidor uma variável de sessão armazenada no browser do cliente.

- Content-Length

  Determina o tamanho do conteúdo enviado pela requisição em octetos (bytes de 8 bits).

- Content-Type

  Determina o tipo do conteúdo enviado pela requisição (texto, imagens, arquivos, respostas XML e JSON, entre outros).

Logo uma requisição HTTP válida tem o seguinte formato:

Método HTTP + URI da Requisição + Versão HTTP + CRLF
Host : <endereço> + CRLF
Cabeçalhos + CRLF
Dados + CRLF

Onde CRLF representa os caracteres de *carriage return* e *line feed*, utilizados para representar uma nova linha.

2.5 REST

REST, do inglês *Representational State Transfer* (Transferência de Estado Representacional), é uma arquitetura de software projetada para descrever sistemas de hipermídia distribuídos (como a Internet), primeiramente definido por Fielding e Taylor (2000) em sua dissertação de PhD. Devido a sua simplicidade e sua correlação natural com o protocolo HTTP, REST se tornou a arquitetura mais utilizada para compartilhamento de informações em aplicações na Internet (BATTLE; BENSON, 2008).

Tal arquitetura utiliza métodos básicos de interação remota e transferência de estados para sistemas de armazenamento de dados: Inserir, Buscar, Alterar e Deletar (geralmente referenciado como CRUD, do inglês *Create, Read, Update, Delete*). Assim, uma requisição REST possibilita um usuário realizar uma operação em algum recurso na web endereçado por

uma URL. Pode-se realizar um mapeamento informal das operações CRUD com os comandos providenciados pelo protocolo HTTP (descrito no Quadro 2).

Quadro 2 – Relação entre operações CRUD e o protocolo HTTP. (BATTLE; BENSON, 2008)

Operação CRUD	Comando HTTP	Dados de Entrada	Retorno da Requisição
Inserir (Create)	POST	Formulário Codificado HTTP	Status 201 CREATED
Buscar (Read)	GET	-	Determinado pela requisição
Alterar (Update)	PUT	Formulário Codificado HTTP	Status 200 OK
Deletar (Delete)	DELETE	-	Status 200 OK

Nesta filosofia, a unidade básica de informação é denominada *recurso*. Para Fielding e Taylor (2000), recurso se define como qualquer informação que pode ser nomeada, por exemplo, um documento, uma imagem, a temperatura em uma determinada região, um objeto não-virtual, uma coleção de outros recursos, entre outros. Battle e Benson (2008) definem recurso como sendo cada componente de uma aplicação que será visível para um usuário.

Ainda, recursos contidos em uma aplicação REST devem ser identificados por uma URL (Localização Universal de Recursos, do inglês *Universal Resource Locator*) ou URI (Identificador Universal de Recursos, do inglês *Universal Resource Identifier*), que geralmente tem o seguinte formato:

Protocolo://Hospedeiro/Aplicação/TipoDeRecurso/IDRecurso
http://exemplo.org/api/posts/bWV1cGF1ZGVhc2E=

A arquitetura REST especifica uma série de restrições que, quando aplicadas enfatizam escalabilidade das interações entre os componentes, generalização de interfaces e publicação independente de componentes (FIELDING; TAYLOR, 2000).

Fielding e Taylor (2000) definiram seis características necessárias para se descrever um serviço como REST.

1. Cliente-Servidor

   Deve haver separação entre a interface de usuário e a aplicação que armazena os dados, permitindo que cada um dos elementos possa se desenvolver independentemente sem afetar o funcionamento do outro, garantindo assim, portabilidade da interface de usuário e escalabilidade dos servidores.

2. Sem-Estado (Stateless)

   Cada requisição realizada pelo cliente deve ser sem-estado por natureza, ou seja, deve conter toda a informação necessária para ser processada, sem se beneficiar de alguma informação armazenada no servidor. Assim, a responsabilidade de armazenar informações relevantes ao contexto do usuário fica na aplicação cliente.

3. Cacheável

   Para otimizar os recursos da rede, é possibilitado que qualquer resposta a uma requisição ao servidor seja implícita ou explicitamente caracterizada como cacheável, dando à aplicação cliente o direito de reutilizar tal resposta mais tarde em requisições semelhantes.

4. Interface Uniforme

   Aplicando-se generalização à interface dos componentes acarreta em uma simplificação da arquitetura do sistema. Assim, as implementações das interfaces são desacopladas dos serviços que elas oferecem, o que possibilita desenvolvimento independente das mesmas.

5. Sistema em camadas

   Um sistema em camadas permite a hierarquização dos recursos do sistema, fazendo com que cada componente não possa acessar informações além da camada onde está inserido, promovendo assim, independência dos componentes.

6. Código por demanda

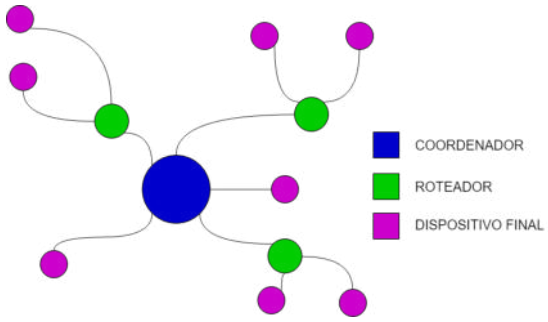
   A arquitetura REST permite estender a funcionalidade das aplicações clientes através do download e execução de scripts diretamente, o que reduz a quantidade de código a ser pré-implementado no cliente.

2.6 ZigBee

Zigbee é uma especificação baseada no padrão técnico IEEE 802.15.4, que nomeia um grupo de protocolos de comunicação de alto nível para criar Redes de área pessoal (PAN's) com dispositivos munidos de rádios digitais. Logo, tecnologias no padrão Zigbee tem como objetivo atender a aplicações embarcadas que tenham como prioridade o baixo consumo de energia, tolerando baixas taxas de transferência, tais como automação residencial, sensoriamento distribuído, monitoramento hospitalar, controle e automação industrial e segurança (Wheeler, 2007).

A camada de rede de dispositivos Zigbee oferece suporte a topologias de rede em estrela, árvore e também redes em malha. Para isso, o padrão Zigbee se vale de três tipos primordiais de dispositivos: *End-device* (ou, dispositivo final), Roteador e Coordenador (ALLIANCE, 2005).

Figura 3 – Dispositivos Zigbee em uma rede de malha



(Fonte: o Autor)

O coordenador é o nó central de uma rede, que armazena todas as informações da mesma. É necessária a presença de um (e somente um) dispositivo coordenador para a criação e manutenção de uma rede Zigbee. O dispositivo final (*end device*) é um dispositivo de

funcionalidade limitada que tem o objetivo de se comunicar com seu nó pai na rede (coordenador ou roteador). Isso permite que o dispositivo permaneça inativo (em *sleep*) na maior parte de sua operação, garantindo um maior tempo de vida útil de sua fonte de energia. Finalmente, o dispositivo roteador pode operar tanto como um dispositivo final, transmitindo dados ao seu nó pai, quanto transmitindo dados de outros *end devices* pela rede, possibilitando a criação de uma topologia em árvore e/ou malha. (ALLIANCE, 2005). A disposição de dispositivos Zigbee em uma rede pode ser visualizada conforme a Figura 3.

3 Trabalhos Relacionados

O objetivo deste capítulo é apresentar os trabalhos relacionados que inspiraram o desenvolvimento desta pesquisa. Os trabalhos selecionados podem ser divididos em três temas principais: Internet das Coisas, Protocolos de Comunicação e Varejo Inteligente.

3.1 Internet das Coisas

O surgimento do conceito de IoT proporcionou o desenvolvimento de diversos trabalhos acadêmicos com os mais diversos objetivos. Pesquisas como as de Al-Fuqaha et al. (2015), Santos et al. (2016) e Gupta e Quamara (2018) se preocuparam em conceituar IoT e mapear as tecnologias que possibilitam o seu desenvolvimento. Gupta e Quamara (2018) ainda apresentou a arquitetura atual da Internet das Coisas e os principais desafios que este paradigma encontra no presente. Já Al-Fuqaha et al. (2015) e Santos et al. (2016) apresentaram os principais protocolos de comunicação e ambientes de desenvolvimento existentes.

Outras frentes de pesquisa buscaram otimizar a utilização de recursos por dispositivos IoT. Afzal et al. (2017) mapeou as principais características de hardware e sistema operacional utilizadas como métrica de performance, e realizou um estudo sobre quais combinações de sistema operacional e hardware são mais adequados para os diversos tipos de problemas que podem ser solucionados por IoT. Outro trabalho, desenvolvido por Blaauw et al. (2014), buscou expor limitações de hardware existentes em dispositivos IoT e por fim, apresentou uma possível solução de arquitetura de dispositivos de sensoriamento.

Finalmente, distanciando-se da conceituação e mapeamento de requisitos e tecnologias, trabalhos como os de Chen et al. (2014) e Lee, Bae e Kim (2017) buscaram oferecer um prospecto das oportunidades e tendências de mercado de IoT. Chen et al. (2014) descreve as capacidades e oportunidades da Internet das coisas e apresenta os campos de aplicação mais proeminentes na China (país estimado a possuir 9 bilhões de dispositivos conectados), e os desafios associados ao crescimento das tecnologias IoT. Já Lee, Bae e Kim (2017) realizam um prospecto sobre as tendências de mercado em IoT e os programas de pesquisa e desenvolvimento sendo realizados nos EUA, União Européia e na Ásia.

3.2 Protocolos de Comunicação

Foram produzidos também trabalhos com foco na especificação e avaliação de protocolos de comunicação entre dispositivos IoT existentes. Pereira e Aguiar (2014) apresentaram dois protocolos de comunicação M2M e realizaram um estudo comparativo sobre suas características e funcionalidades. Em seu trabalho, também sugeriram a utilização de *smartphones* como dispositivos IoT de coleta e transmissão de dados.

Já Gündoğan et al. (2018) realizou extensas análises experimentais entre os protocolos

de comunicação mais proeminentes em IoT. Foram testados consumo de RAM e CPU, perda de pacotes, tempo de chegada das mensagens, *overhead* e consumo de energia. Gündoğan et al. (2018) ainda conclui que protocolos de comunicação mais simples tendem a obter maior eficiência em ambientes de rede relaxados (baixos intervalos de transferência) e com baixas taxas de erro.

3.3 Varejo Inteligente

Buscaram-se também desenvolver pesquisas sobre a aplicação dos conceitos de IoT em cenários de varejo, com o objetivo de consolidar a ideia de varejo inteligente e medir o possível impacto na mudança de dinâmicas operacionais causadas pela implantação das novas tecnologias.

Priporas, Stylos e Fotiadis (2017) realizaram um estudo com o objetivo de compreender as expectativas da geração Z (jovens adultos nascidos após o ano de 1995) quanto à inserção de tecnologias inteligentes no ambiente de varejo, concluindo que os integrantes dessa geração já possuem projeções formadas sobre a integração dos serviços de varejo com a tecnologia de ponta. Isso implica na necessidade de se investir em aplicações que visem cumprir as demandas dessa geração por melhorias na velocidade de transações, provisão de informações e conveniência (como apontado pelos autores).

Discutindo os impactos da implantação de tecnologias inteligentes, surgiram trabalhos como os de Pantano e Timmermans (2014) e Pantano et al. (2018), que verificaram os requisitos necessários e os desafios emergentes para que um estabelecimento conquiste vantagem econômica ao incorporar estas tecnologias em suas atividades. Enquanto o primeiro trabalho oferece uma visão geral sobre as aplicações de tecnologias inteligentes em varejo, o segundo trabalho tem como objeto de estudo o mercado de produtos de luxo.

Ainda nesta vertente, Foroudi et al. (2018) realizou um extenso estudo com o objetivo de investigar os efeitos das tecnologias inteligentes nas dinâmicas e experiências do consumidor. Em seu trabalho, é concluído que neste mundo de extensa inovação digital, o poder de decisão de compra se encontra cada vez mais nas mãos do consumidor final, fazendo-se necessário que os varejistas se conscientizem desta mudança de paradigma e ofereçam uma melhor experiência para o usuário, garantindo a fidelidade dos clientes existentes e possibilitando a conquista de novos.

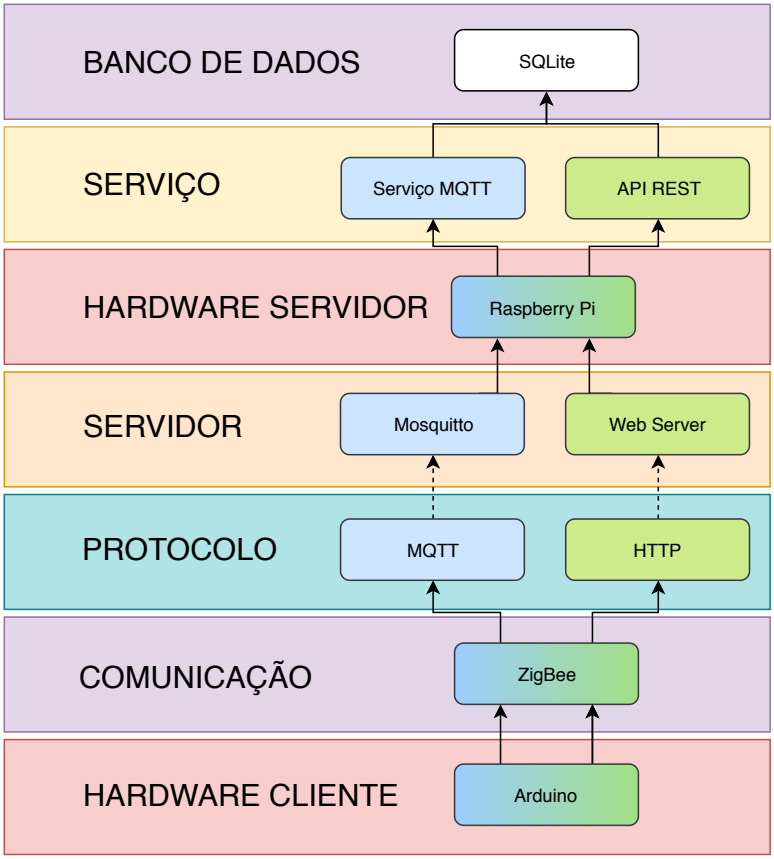
Por fim, alguns trabalhos como os de Li et al. (2016) e Kumar et al. (2017) ofereceram soluções mais concretas para o ambiente de varejo inteligente. Kumar et al. (2017) propôs um modelo de carrinho de compras baseado nas tecnologias Arduino e Xbee, possibilitando o reconhecimento de produtos via RFID (Identificação via Rádio-Frequência) e geração de informações sobre a compra. Li et al. (2016) especificou toda a estrutura necessária para a implementação de uma solução de varejo inteligente, definindo a arquitetura de um carrinho de compras inteligente, das tags RFID utilizadas e dos algoritmos que representam o fluxo de dados no sistema.

4 Desenvolvimento

4.1 Materiais

Para a execução da proposta do trabalho, foram selecionados os materiais evidenciados pelo diagrama da Figura 4.

Figura 4 – Materiais utilizados e estruturação da rede.



(Fonte: o Autor)

- Arduino

Devido a seu baixo custo, facilidade de interfaceamento com sensores e dispositivos de entrada e saída de dados e quantidade de documentação e trabalhos correlatos realizados com a plataforma, utilizou-se o Arduino Uno como plataforma de hardware para as aplicações cliente.

- ZigBee

Foram utilizados dois dispositivos XBee Series 2 como plataforma de comunicação entre o cliente (Arduino) e o servidor (Raspberry Pi), configurados como Endpoint e Coordenador, respectivamente.

- Raspberry Pi

O microcomputador Raspberry Pi foi utilizado para hospedar o servidor Web e o broker MQTT, devido a seu alto poder computacional, suporte a um sistema operacional baseado em Linux (Raspbian) e grande quantidade de material disponível e trabalhos realizados com a plataforma disponíveis para aprendizado.

- Mosquitto

Para coordenar a comunicação entre cliente-servidor, foi utilizado o broker MQTT Mosquitto, um projeto de código aberto da Eclipse Foundation que providencia implementações cliente-servidor que cumprem os padrões do protocolo MQTT (LIGHT, 2017). A escolha desta ferramenta foi motivada por sua documentação completa, disponibilidade de um servidor para testes e sua presença em trabalhos acadêmicos correlatos.

- Serviço MQTT

Foi implementada uma aplicação que recebe as mensagens dos clientes e realiza operações no banco de dados. Sua criação será discutida na seção 4.2.

- API REST

Foi implementada uma API REST que fornece e modifica recursos do banco de dados. Sua criação será discutida na seção 4.2.

- Banco de Dados

Foi criado um banco de dados para as aplicações servidoras utilizando-se o mecanismo de banco de dados SQLite, uma biblioteca de código aberto que oferece um serviço de banco de dados SQL transacional auto-contido, sem a necessidade de um servidor dedicado de banco de dados, nem de configurações prévias. Ainda, SQLite é uma biblioteca leve, com interfaces para uma multitude de linguagens de programação (C, C++, BASIC, C#, Java, Python, etc) e Sistemas Operacionais embarcados (Android, iOS) (BHOSALE; PATIL; PATIL, 2015).

4.2 Metodologia

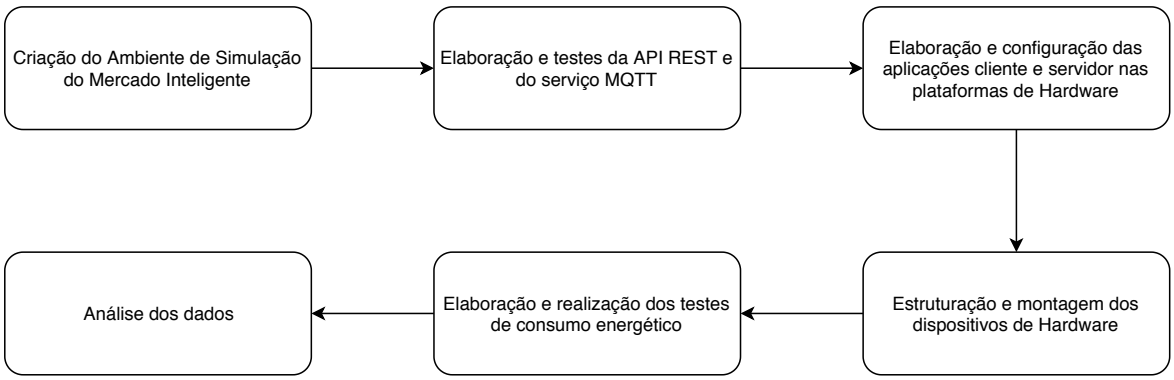
Para a execução do trabalho, e consequentemente, o cumprimento da proposta de se verificar a eficiência energética do protocolo MQTT em relação ao HTTP em uma aplicação IoT baseada num cenário de varejo inteligente, foi elaborado um plano de ação (Figura 5) que contemplasse todas as etapas necessárias.

Segue uma breve descrição sobre cada etapa do processo.

- Criação do Ambiente de Simulação do Mercado Inteligente

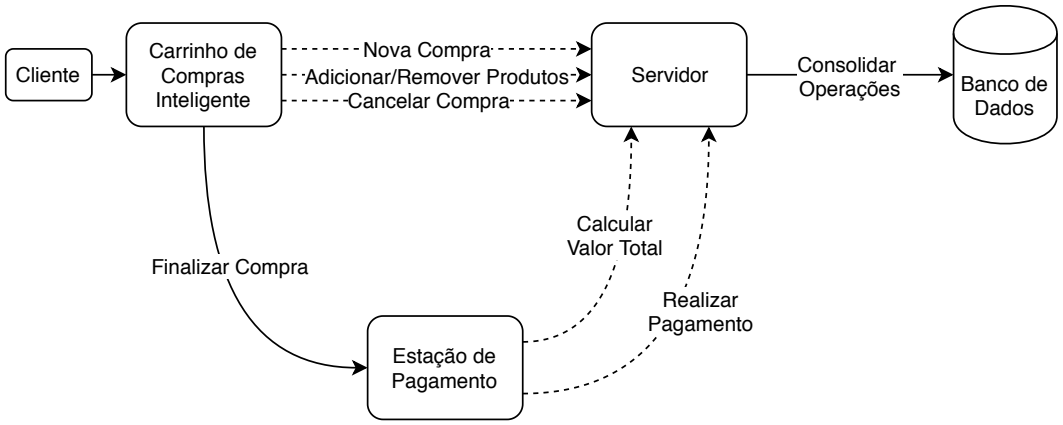
Atendo-se aos conceitos de varejo inteligente, foi determinado um modelo de funcionamento (Figura 6) onde o cliente, munido de um carrinho de compras capaz de realizar identificação de produtos através de radiofrequência, realiza ações corriqueiras de uma compra (adicionar e remover produtos) enquanto o carrinho envia mensagens correspondentes a estas ações ao servidor, que por sua vez, armazena os dados da compra no banco de dados.

Figura 5 – Plano de ação para execução da proposta.



(Fonte: o Autor)

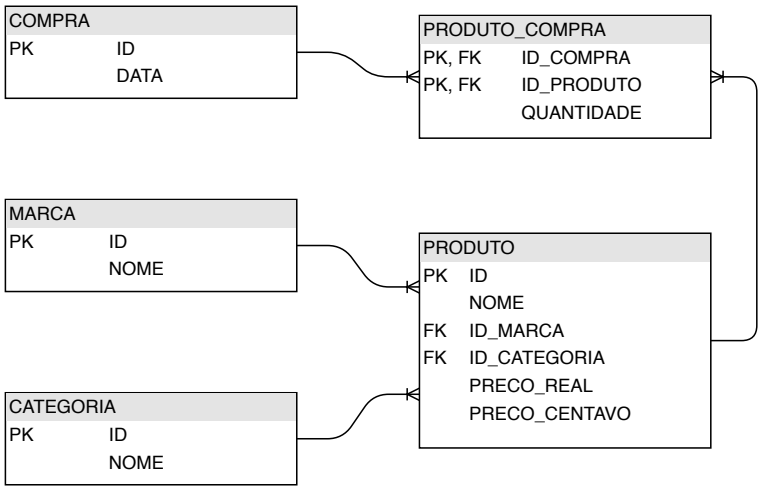
Figura 6 – Representação simplificada do funcionamento de um varejo inteligente



(Fonte: o Autor, adaptado de Li et al. (2016))

Feitas estas considerações, foi elaborado um banco de dados que simulasse de maneira simplificada a realidade deste mercado, conforme demonstrado na Figura 7.

Figura 7 – Modelo de Entidade Relacional do Banco de Dados criado.



(Fonte: o Autor)

- Elaboração da API REST e do serviço MQTT

Foram elaborados os serviços que manipulariam as informações do banco de dados, obedecendo as características dos protocolos HTTP e MQTT, respectivamente. Ambas tem a função de receber uma instrução (requisição GET, POST, PUT, etc. no HTTP e um código específico no MQTT) e uma string JSON contendo os dados da operação a ser realizada, e realizar as alterações necessárias no banco de dados, retornando uma mensagem ao cliente.

A API REST utilizada pelo protocolo HTTP foi estruturada conforme demonstrado no Quadro 3. Seu código está disponível no Anexo D.

Quadro 3 – Estrutura da API REST para o mercado inteligente

Requisição	URI	Descrição do recurso
GET	http://localhost:5000/smartmarket/api/compras	Lista das compras
GET	http://localhost:5000/smartmarket/api/compras/nova	Cria uma compra e retorna sua URI
GET	http://localhost:5000/smartmarket/api/compras<id>	Descrição de uma compra
PUT	http://localhost:5000/smartmarket/api/compras<id>/adicionar	Adicionar um produto a uma compra
PUT	http://localhost:5000/smartmarket/api/compras<id>/remover	Remover um produto de uma compra

O serviço MQTT foi estruturado conforme demonstrado no Quadro 4:

Quadro 4 – Estrutura do serviço MQTT para o mercado inteligente

Código	Dados	Tópico	Descrição
NEW	-	smartmarket/compra	Cria uma nova compra e retorna seu código
ADD	JSON com código do produto	smartmarket/compra	Adiciona um produto a uma compra
RMV	JSON com código do produto	smartmarket/compra	Remove um produto de uma compra
PAY	JSON com código da compra	smartmarket/pagamento	Realiza o pagamento e finaliza uma compra
CNL	JSON com código da compra	smartmarket/pagamento	Cancela a compra atual

As aplicações foram desenvolvidas na linguagem de programação Python, utilizando-se o microframework *Flask* para criação da API REST e a biblioteca *paho-mqtt* para criação do serviço MQTT. Estas ainda possuem a funcionalidade de receber a medição do consumo de bateria do hardware ao longo do tempo, gerando logs periódicos e os armazenando.

A linguagem Python foi escolhida por ser uma linguagem de alto nível que oferece praticidade e agilidade no desenvolvimento de aplicações, e possibilita rápida prototipação e testes.

No log serão registradas as seguintes informações:

1. Tempo de simulação
2. nº da requisição
3. Medida instantânea da tensão de alimentação e do Arduino

De posse destas informações, será possível calcular e comparar a eficiência dos protocolos nos quesitos

- Vida útil
- Consumo de energia x tempo
- Throughput

- Elaboração e configuração das aplicações cliente nas plataformas de Hardware

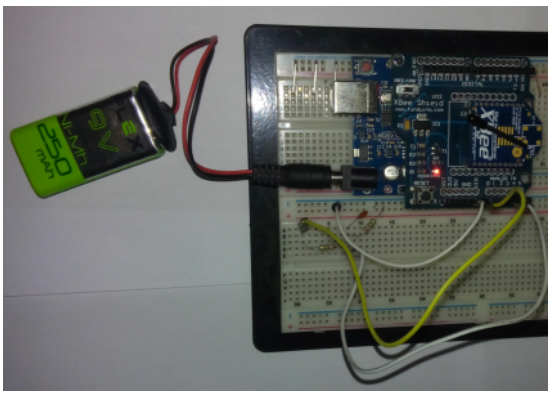
Foram elaboradas as aplicações que seriam responsáveis por se conectar aos seus respectivos servidores, gerar requisições e receber as respostas esperadas. Ambas as aplicações foram embarcadas na plataforma Arduino Uno. Foi utilizadas as bibliotecas PubSubClient para implementação do cliente MQTT (segundo Anexo B) e HTTPCLient para implementação do cliente HTTP (segundo Anexo A).

Foram também configuradas as aplicações servidoras no microcomputador Raspberry Pi. O broker MQTT foi configurado com nível de QoS 2 (2.3), devido às características do ambiente proposto, onde deve-se garantir que todas as informações foram recebidas pelo servidor exatamente uma vez, a fim de evitar divergências.

- Estruturação e montagem dos dispositivos de Hardware

Foram montados os dispositivos de hardware segundo as figuras 8 e 9:

Figura 8 – Montagem do Arduino



(Fonte: o Autor)

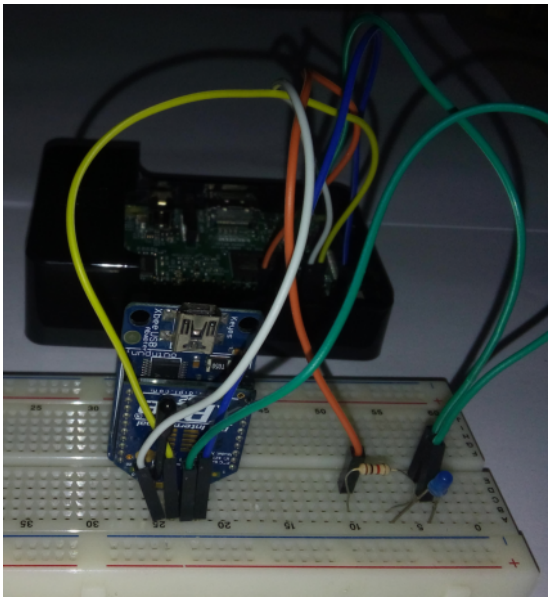
O circuito utilizado para medição de consumo de bateria no Arduino foi montado conforme demonstrado na Figura 10. Foi utilizado um divisor de tensão no intuito de respeitar a limitação das portas de leitura analógica do Arduino. Como os resistores do divisor de tensão são iguais, o leitor analógico fará uma medição de metade da tensão de alimentação do sistema. Logo, para obter a tensão real do sistema em um determinado momento, deve-se multiplicar o valor encontrado na leitura analógica por 2. A construção do circuito medidor de tensão da bateria foi baseada no trabalho de Steichen, Sutton e Thiele (2013).

- Testes de consumo energético e de rede - HTTP x MQTT

Foram realizados os testes de consumo energético de acordo com o procedimento da Figura 11. O dispositivo cliente (Arduino UNO) é responsável por gerar requisições ao dispositivo servidor (Raspberry Pi). Para o protocolo MQTT, são gerados *publishes* num tópico gerenciado pelo broker Mosquitto, e para o protocolo HTTP, são geradas requisições HTTP em formato *raw* segundo a especificação do HTTP/1.1.

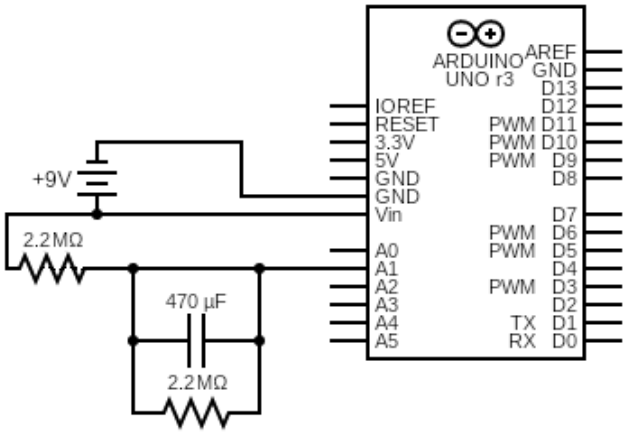
Logo, uma requisição GET tem o formato:

Figura 9 – Montagem do Raspberry Pi



(Fonte: o Autor)

Figura 10 – Circuito para medição de bateria do Arduino Uno



(Fonte: o Autor)

GET /smartmarket/api/<recurso> HTTP/1.1
Host: <endereço>

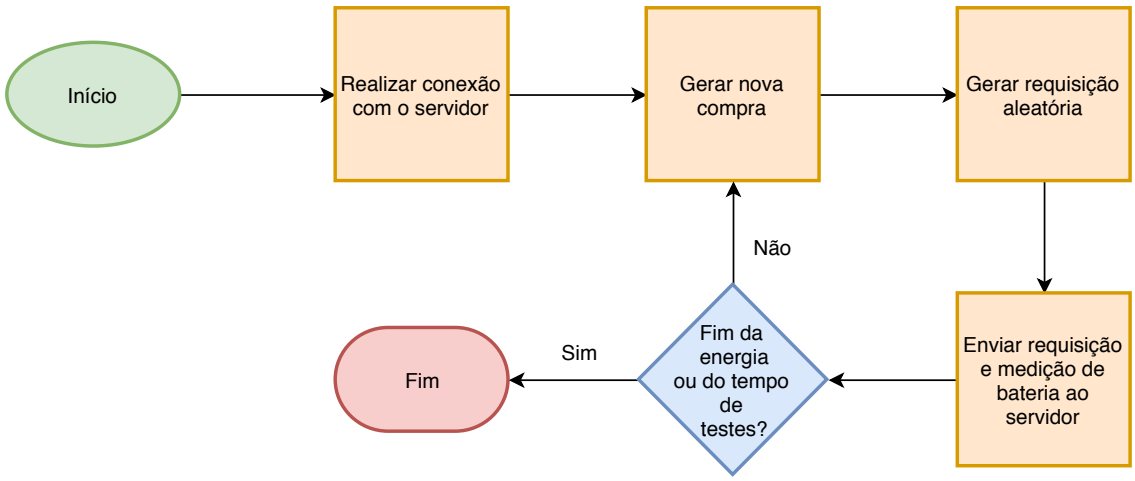
E requisições POST/PUT tem o formato:

POST/PUT /smartmarket/api/<recurso> HTTP/1.1
Host: <endereço>
Content-Type: application/json
<dados>

Como a comunicação é realizada de maneira *wireless* pela porta serial dos dispositivos (/dev/ttyS0 no Raspberry Pi), foi necessário um roteamento dos bytes recebidos na porta serial do servidor para a porta de rede na qual se encontra a aplicação . Este roteamento foi realizado

através de um script Python (Anexo C) que realiza leitura da porta /dev/ttyS0 e envia, via *sockets*, um pacote TCP contendo os dados da requisição para a porta de rede determinada.

Figura 11 – Fluxograma dos testes de consumo de energia



(Fonte: o Autor)

Para cada protocolo, foram realizados testes com baterias recarregáveis 9V Ni-Mh 250mAh, com o módulo Xbee enviando requisições contendo uma *string* JSON que contém o estado da bateria a cada segundo, através de comunicação serial com *baud rate* de 9600. Os testes visaram obter as informações sobre o tempo de execução e o consumo de bateria ao longo do tempo.

Para realização dos testes de tráfego de rede, foram executadas sequências de requisições predefinidas, e utilizou-se a ferramenta *bmon* no Raspberry Pi para medir a quantidade de bytes e pacotes enviados pela rede durante o tempo de execução.

- Análise dos dados

  Após coleta dos dados, foi realizada análise e comparação dos resultados obtidos.

5 Resultados e Discussões

Os dados das tabelas 1 e 2 se referem aos testes realizados em um Arduino Uno ligado a um dispositivo Xbee Series 2 tendo como fonte de energia uma bateria de 9V Ni-Mh 250mAh totalmente carregada, utilizando os protocolos HTTP e MQTT, respectivamente. A vida útil da bateria foi medida calculando a diferença de tempo entre a primeira e a última requisição recebida pelo servidor (quando ocorre o fim da carga da bateria).

Tabela 1 – Resultados dos testes do protocolo HTTP

HTTP					
nº Teste	Vida Útil (minutos)	Quantidade de requisições	Requisições/ segundo	Consumo médio em mAh	Consumo médio em W/h
Teste 1	204	12.132	0,9912	73,52	0,49
Teste 2	216	11.884	0,917	69,44	0,46
Teste 3	211	12.057	0,9524	71,09	0,47
Teste 4	209	11.298	0,901	71,77	0,48
Teste 5	215	11.537	0,8944	69,76	0,47
Média	211	11.782	0,9312	71,12	0,47

Tabela 2 – Resultados dos testes do protocolo MQTT

MQTT					
nº Teste	Vida Útil (minutos)	Quantidade de publishes	Publishes/ segundo	Consumo médio em mAh	Consumo médio em W/h
Teste 1	269	15.977	0,9899	55,76	0,37
Teste 2	278	15.296	0,917	53,95	0,36
Teste 3	276	15.969	0,9643	54,34	0,36
Teste 4	262	15.377	0,9782	57,25	0,38
Teste 5	271	16.058	0,9876	55,35	0,37
Média	271,2	15.735	0,9674	55,33	0,37

(Fonte: o Autor)

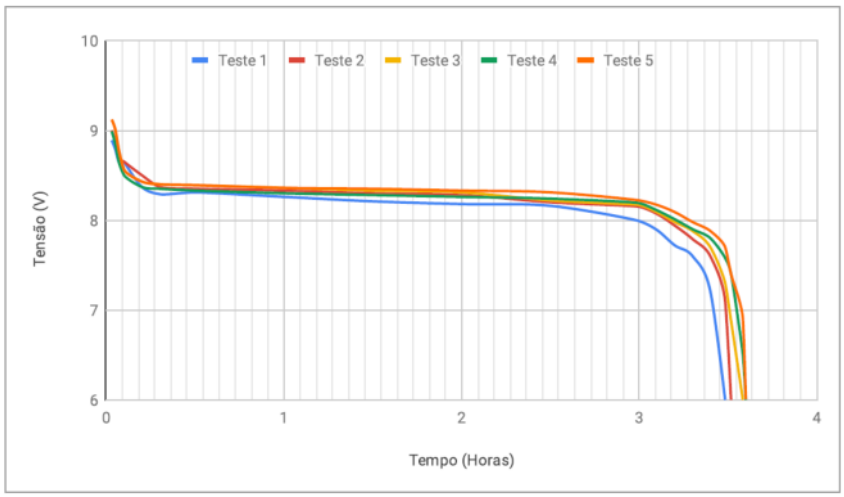
Os gráficos das figuras 12 e 13 demonstram as curvas de descarregamento da bateria para cada teste. Pôde-se observar que, mesmo que com tempos de vida útil distintos (aproximadamente 3.5h para o HTTP e 4.5h para o MQTT), houve um padrão de descarregamento da bateria, que ofereceu tensão máxima por um breve período de tempo, transicionou para um pequeno decrescimento linear da tensão (que acomete a maioria do seu período de vida útil) e por fim, sofreu queda súbita quando se aproximou do fim de sua carga. Este padrão é característico de baterias compostas de Ni-Mh, logo, a utilização de outros tipos de fonte energética apresentarão outros padrões de descarregamento.

O consumo energético médio do sistema foi calculado tomando-se a tensão média do sistema e a corrente média medidas durante a execução dos testes.

Logo,

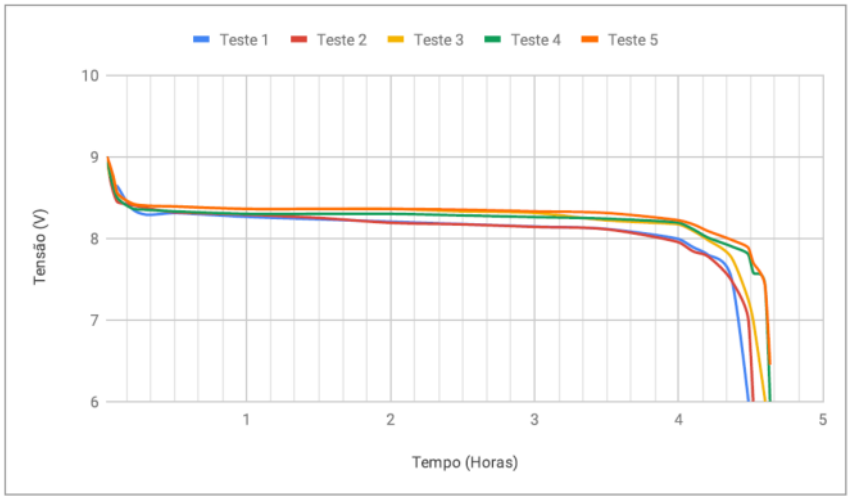
$$P_{med} = V_{med} \cdot I_{med}$$

Figura 12 – Curva de descarregamento da bateria - HTTP



(Fonte: o Autor)

Figura 13 – Curva de descarregamento da bateria - MQTT



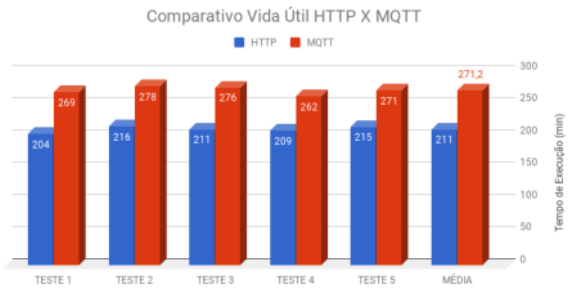
(Fonte: o Autor)

Comparando-se os resultados obtidos, pode-se observar diminuição de consumo energético (e consequentemente, aumento de vida útil) pelo sistema quando o MQTT é utilizado. Os gráficos das Figuras 14a e 14b ilustram este fenômeno. Ao se enviar os pacotes via comunicação serial sem fio pelo Xbee percebeu-se que a maior utilização de corrente no sistema era proveniente das portas RX (recepção de dados seriais) e TX (transmissão de dados seriais), justamente as responsáveis pela comunicação entre os módulos. Logo, concluiu-se que diminuição do consumo energético está relacionada com o protocolo de comunicação utilizado para comunicação de dispositivos IoT.

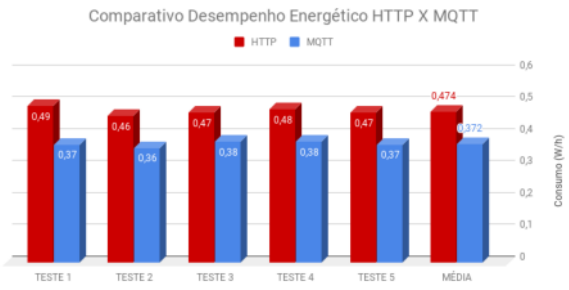
Os dados das tabelas 3 e 4 se referem aos testes realizados para determinar o tráfego de rede para cada protocolo. Foram definidas três quantidades de requisições/*publishes* a serem enviados ao servidor (1000, 5000 e 10000) de maneira contínua e ininterrupta. Utilizou-se a ferramenta *bmon* no Raspberry Pi para medir a quantidade de pacotes e de bytes, tal

Figura 14 – Comparativo do consumo energético HTTP X MQTT

(a) Vida Útil Observada



(b) Consumo Energético Observado



(Fonte: o Autor)

como o tempo necessário para realização de cada teste. Os resultados dos testes de tráfego de rede podem ser visualizados graficamente nas Figuras 15a (em escala logarítmica) e 15b.

Tabela 3 – Resultados dos testes de tráfego de rede do protocolo HTTP

HTTP				
nº Teste	**Tempo (segundos)**	**Bytes enviados (MB)**	**Pacotes enviados (x1000)**	**Quantidade de requisições**
Teste 1	69	1,07	14,01	1000
Teste 2	340	5,37	70,06	5000
Teste 3	682	12.057	140,13	10000

(Fonte: o Autor)

Tabela 4 – Resultados dos testes de tráfego de rede do protocolo MQTT

MQTT				
nº Teste	**Tempo (segundos)**	**Bytes enviados (MB)**	**Pacotes enviados (x1000)**	**Quantidade de requisições**
Teste 1	1,7	0,2	3,89	1000
Teste 2	8,37	1,56	18,31	5000
Teste 3	17,49	2,73	40,36	10000

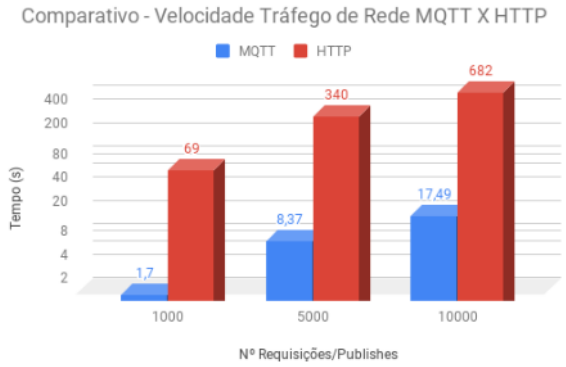
(Fonte: o Autor)

Realizando-se uma análise comparativa, pôde-se observar vantagens na utilização do protocolo MQTT sobre o HTTP. No quesito de consumo energético, foi observado um ganho de aproximadamente 22% de vida útil da bateria utilizando-se o MQTT em relação ao HTTP. Já no quesito tráfego de rede, foi observado que o MQTT realiza transmissão de mensagens aproximadamente 44 vezes mais rápido que o HTTP, enviando uma média de 4,4 vezes menos bytes e 3,3 vezes menos pacotes TCP na rede.

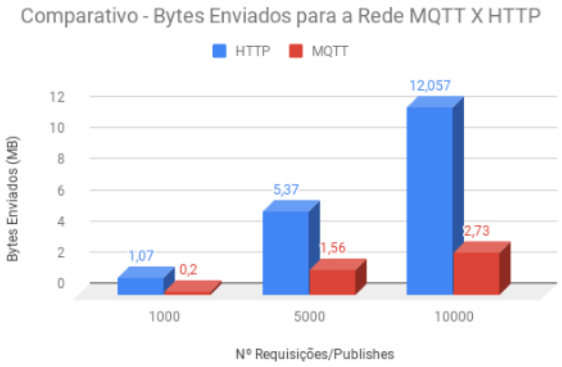
Parte desta vantagem se deve ao fato de o MQTT ser um protocolo projetado para ter cabeçalhos de mensagem extremamente pequenos, contendo, em sua maioria, dados binários. No HTTP, todos os parâmetros que moldam o comportamento de uma requisição são

Figura 15 – Comparativo entre testes de desempenho de rede HTTP X MQTT

(a) Gráfico comparativo Requisições X Tempo



(b) Gráfico comparativo Requisições X Bytes



(Fonte: o Autor)

formados por strings de bytes. Assim, requisições que contenham pequenas quantidades de dados, devem ser envelopadas em *headers* verbosos, que acabam por superar o tamanho dos dados em várias ordens de magnitude. Já no MQTT, todos os parâmetros de controle que encapsulam os dados são binários, o que faz com que os *headers* MQTT ocupem uma menor parcela nos pacotes enviados, gerando menos *overhead*.

Além disso, após estabelecida uma conexão MQTT, esta é reutilizada continuamente até que o cliente se desconecte, o que diminui a necessidade de se enviar informações redundantes ao servidor. Já para o HTTP, cada requisição acontece em uma conexão TCP isolada, fazendo com que seja necessário reenvio dos *headers* para uma nova requisição. Pode-se observar que estas características são causa direta da redução de consumo energético, já que são necessárias menos operações para se encapsular os dados nos moldes do protocolo, e consequentemente, o dispositivo se encarregará de enviar menos dados através da rede, processo que demanda grande gasto energético.

Isso é mais agravado pelas características da Internet das Coisas, onde se deseja que os dispositivos sensores enviem um fluxo constante de dados ao longo do tempo, e um dispositivo que se comunique por um protocolo que exija muito processamento para transmitir os dados no formato desejado, sofrerá de uma redução de desempenho significativa, o que pôde ser constatado nas medições realizadas.

Logo, comparadas as características do HTTP e do MQTT e o desempenho de ambos os protocolos em um cenário hipotético de Internet das Coisas, pode-se concluir que de fato, o protocolo MQTT oferece ganho de desempenho e redução de tráfego em relação ao HTTP, sendo indicado a sua utilização em aplicações semelhantes.

6 Considerações Finais

6.1 Conclusão

O presente trabalho teve como objetivo analisar a eficiência energética de dois protocolos da camada de aplicação (HTTP e MQTT) quando utilizados como protocolo de comunicação entre cliente-servidor em um dispositivo IoT. Primeiramente foi construído um modelo de solução IoT para um mercado inteligente, constituído de um banco de dados, uma API REST e um serviço MQTT embarcados em um Raspberry Pi, e ainda duas aplicações cliente (uma HTTP e uma MQTT) embutidas em um Arduino Uno utilizando-se de comunicação Ethernet.

Posteriormente, foram medidos os respectivos consumos de energia no dispositivo cliente para cada configuração da rede. Foi também medida a quantidade de bytes e pacotes que trafegaram na rede, de modo a determinar qual protocolo provoca menos *overhead*.

Após análise dos resultados obtidos, pôde-se concluir que no cenário apresentado, o protocolo MQTT se mostra superior ao HTTP no quesito de consumo de bateria, obtendo um ganho de aproximadamente 22% em tempo de vida útil da aplicação, o que beneficia o paradigma da Internet das Coisas, onde a fonte de energia é principal restrição e sua preservação é crítica. Pôde-se observar também que o MQTT possibilita a transmissão da mesma quantidade de informação utilizando uma menor quantidade de pacotes, apresentando uma redução de aproximadamente três vezes no tráfego de rede analisado.

6.2 Trabalhos Futuros

Devido à característica interdisciplinar do paradigma da Internet das Coisas, é possível observar diversos pontos de melhoria e pesquisa futura sobre o presente trabalho. Desta forma, o texto descrito abaixo procura enfatizar possíveis melhoras para trabalhos futuros.

Partindo do quesito do tratamento dos dados gerados pela IoT, encoraja-se buscar soluções que se distanciem do modelo de banco de dados relacional, preferindo soluções Big Data, que prometem melhorar o desempenho de aplicações que operem sobre um volume de dados maior. Logo, pode-se realizar uma análise da eficiência de um banco de dados não-relacional em aplicações IoT. Também é possível realizar pesquisa na área de análise de dados, aplicando-se algoritmos inteligentes (inteligências artificiais, redes neurais) para extrair informações e identificar padrões nos dados gerados.

Em relação ao varejo inteligente, pode-se expandir o presente trabalho nos quesitos identificação (identificar produtos via RFID), apresentação (utilizar um display para exibir os dados de uma compra, ou ainda, encapsular as funcionalidades do modelo em um *app* que se comunica com a API) e autenticação e segurança (implementar autenticação via OAuth ou Tokens).

Na vertente de protocolos de rede, pode-se realizar esta análise de consumo de bate-

ria de dispositivos IoT com outros protocolos, como o CoAP, SOAP, entre outros. Outra análise possível é medir o desempenho de rede dos protocolos testados a fim de verificar qual protocolo é mais indicado para operar em uma rede de sensores sem fio com limitações na largura de banda disponível.

Sobre dispositivos de rede, há a possibilidade de se analisar outras plataformas de comunicação, como a ESP8622 (comunicação via WiFi), GSM900 (comunicação via redes celulares), módulos Bluetooth, LoRaWAN entre outros. Também há a possibilidade de se analisar o desempenho dos dispositivos sob outras configurações e aumentar o número de dispositivos clientes.

Referências

AFZAL, B. et al. Enabling iot platforms for social iot applications: Vision, feature mapping, and challenges. *Future Generation Computer Systems*, Elsevier, 2017. Citado na página 22.

AKYILDIZ, I. et al. Wireless sensor networks: a survey. *Computer Networks*, v. 38, n. 4, p. 393 – 422, 2002. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128601003024>. Citado na página 12.

AL-FUQAHA, A. et al. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, IEEE, v. 17, n. 4, p. 2347–2376, 2015. Citado na página 22.

ALLIANCE, Z. *Zigbee specification version 1.0*. [S.l.]: April, 2005. Citado nas páginas 20 e 21.

BATTLE, R.; BENSON, E. Bridging the semantic web and web 2.0 with representational state transfer (rest). *Journal of Web Semantics*, v. 6, n. 1, p. 61 – 69, 2008. ISSN 1570-8268. Semantic Web and Web 2.0. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1570826807000510>. Citado nas páginas 18 e 19.

BHOSALE, S.; PATIL, T.; PATIL, P. Sqlite: Light database system. *International Journal of Computer Science and Mobile Computing*, p. 882–885, 2015. Citado na página 25.

BLAAUW, D. et al. Iot design space challenges: Circuits and systems. In: IEEE. *2014 Symposium on VLSI Technology (VLSI-Technology): Digest of Technical Papers*. [S.l.], 2014. p. 1–2. Citado nas páginas 9 e 22.

CHEN, S. et al. A vision of iot: Applications, challenges, and opportunities with china perspective. *IEEE Internet of Things journal*, IEEE, v. 1, n. 4, p. 349–359, 2014. Citado nas páginas 13, 14 e 22.

ERICSSON, A. Ericsson mobility report: On the pulse of the networked society. *Ericsson, Sweden, Tech. Rep. EAB-14*, v. 61078, 2015. Citado na página 8.

FIELDING, R. et al. *Hypertext transfer protocol–HTTP/1.1*. [S.l.]: RFC 2616, june, 1999. Citado na página 16.

FIELDING, R. T.; TAYLOR, R. N. *Architectural styles and the design of network-based software architectures*. [S.l.]: University of California, Irvine Doctoral dissertation, 2000. v. 7. Citado nas páginas 18 e 19.

FOROUDI, P. et al. Investigating the effects of smart technology on customer dynamics and customer experience. *Computers in Human Behavior*, Elsevier, v. 80, p. 271–282, 2018. Citado na página 23.

GÜNDOĞAN, C. et al. Ndn, coap, and mqtt: A comparative measurement study in the iot. *arXiv preprint arXiv:1806.01444*, 2018. Citado nas páginas 22 e 23.

GUPTA, B.; QUAMARA, M. An overview of internet of things (iot): Architectural aspects, challenges, and protocols. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, p. e4946, 2018. Citado nas páginas 10, 13 e 22.

KIANI, F. A survey on management frameworks and open challenges in iot. *Wireless Communications and Mobile Computing*, Hindawi, v. 2018, 2018. Citado na página 10.

KUMAR, A. et al. Smart shopping cart. In: IEEE. *Microelectronic Devices, Circuits and Systems (ICMDCS), 2017 International conference on*. [S.l.], 2017. p. 1–4. Citado na página 23.

LEE, I.; LEE, K. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, Elsevier, v. 58, n. 4, p. 431–440, 2015. Citado na página 9.

LEE, S. K.; BAE, M.; KIM, H. Future of iot networks: A survey. *Applied Sciences*, Multidisciplinary Digital Publishing Institute, v. 7, n. 10, p. 1072, 2017. Citado na página 22.

LI, R. et al. Iot applications on secure smart shopping. In: IEEE. *Identification, Information and Knowledge in the Internet of Things (IIKI), 2016 International Conference on*. [S.l.], 2016. p. 238–243. Citado nas páginas 23 e 26.

LIGHT, R. A. Mosquitto: server and client implementation of the mqtt protocol. *The Journal of Open Source Software*, The Open Journal, v. 2, n. 13, p. 265, 5 2017. ISSN 2475-9066. Disponível em: <http://dx.doi.org/10.21105/joss.00265>. Citado na página 25.

NIELSEN, H. F. et al. Network performance effects of http/1.1, css1, and png. In: ACM. *ACM SIGCOMM Computer Communication Review*. [S.l.], 1997. v. 27, n. 4, p. 155–166. Citado na página 16.

OASIS. *MQTT Version 3.1.1. Edited by Andrew Banks and Rahul Gupta.* [S.l.]: October, 2014. Citado na página 15.

PANTANO, E. et al. To what extent luxury retailing can be smart? *Journal of Retailing and Consumer Services*, Elsevier, v. 43, p. 94–100, 2018. Citado na página 23.

PANTANO, E.; TIMMERMANS, H. What is smart for retailing? *Procedia Environmental Sciences*, Elsevier, v. 22, p. 101–107, 2014. Citado nas páginas 8 e 23.

PEREIRA, C.; AGUIAR, A. Towards efficient mobile m2m communications: Survey and open challenges. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 14, n. 10, p. 19582–19608, 2014. Citado nas páginas 9 e 22.

PRIPORAS, C.-V.; STYLOS, N.; FOTIADIS, A. K. Generation z consumers' expectations of interactions in smart retailing: A future agenda. *Computers in Human Behavior*, Elsevier, v. 77, p. 374–381, 2017. Citado nas páginas 8 e 23.

RIZZON, F. et al. Smart city: um conceito em construção. *Revista Metropolitana de Sustentabilidade (ISSN 2318-3233)*, v. 7, n. 3, p. 123–142, 2017. Citado na página 8.

SANTOS, B. P. et al. Internet das coisas: da teoriaa prática. *Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos*, 2016. Citado na página 22.

STEICHEN, M.; SUTTON, F.; THIELE, P. D. L. Nimh battery state-of-charge approximation. 2013. Citado na página 28.

Wheeler, A. Commercial applications of wireless sensor networks using zigbee. *IEEE Communications Magazine*, v. 45, n. 4, p. 70–77, April 2007. ISSN 0163-6804. Citado na página 20.

XU, Y.; MAHENDRAN, V.; RADHAKRISHNAN, S. Towards sdn-based fog computing: Mqtt broker virtualization for effective and reliable delivery. In: IEEE. *Communication Systems and Networks (COMSNETS), 2016 8th International Conference on*. [S.l.], 2016. p. 1–6. Citado na página 15.

YICK, J.; MUKHERJEE, B.; GHOSAL, D. Wireless sensor network survey. *Computer Networks*, v. 52, n. 12, p. 2292 – 2330, 2008. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128608001254>. Citado na página 12.

A Código-fonte cliente HTTP Arduino

```
#include <SoftwareSerial.h>
#define rxPin 2
#define txPin 3

char buf_measurements[30];
float measurements[4];
String data_measured;
int analogPin = 3;
String payload;
SoftwareSerial xbee =  SoftwareSerial(rxPin, txPin);

void setup() {
    pinMode(rxPin, INPUT);
    pinMode(txPin, OUTPUT);
    xbee.begin(9600);
}

void loop() {
    for (int i = 0; i < 4; ++i) {
        analogRead(analogPin);
        measurements[i] = analogRead(analogPin);
    }

    data_measured = "";
    for (int i = 0; i< 4; i++){
        data_measured += (measurements[i]* (5.0 / 1024.0) * 2);
        data_measured += " ";
    }
    payload = "POST /smartmarket/api/compras/1 HTTP/1.1\r\n";
    payload +="Host: http://10.0.0.127\r\n";
    payload +="User-Agent: arduino-ethernet\r\n";
    payload +="Content-Type:      application/json;\r\n";
    payload +="Content-Length: ";
    payload +="\r\n";
    payload += "{\"ID\":";
    payload += "\"1\"";
    payload += ",\"QUANTIDADE\":";
    payload += "\"1\"";
    payload += ",\"LOG\":\"";
    payload += data_measured;
    payload += "\"}";
    payload += "\r\n";

    xbee.write(payload.c_str());
    delay(1000);
}
```

B Código-fonte cliente MQTT Arduino

```
#include <Arduino.h>
#include <SoftwareSerial.h>

// Enable MqttClient logs
#define MQTT_LOG_ENABLED 1
// Include library
#include <MqttClient.h>

#define LOG_PRINTFLN(fmt, ...)  printfln_P(PSTR(fmt), ##__VA_ARGS__)
#define LOG_SIZE_MAX 128
void printfln_P(const char *fmt, ...) {
    char buf[LOG_SIZE_MAX];
    va_list ap;
    va_start(ap, fmt);
    vsnprintf_P(buf, LOG_SIZE_MAX, fmt, ap);
    va_end(ap);
    Serial.println(buf);
}

#define HW_UART_SPEED                                   57600L
#define MQTT_ID                                         "TEST-ID"
const char* MQTT_TOPIC_SUB = "test/" MQTT_ID "/sub";
const char* MQTT_TOPIC_PUB = "test/" MQTT_ID "/pub";
MqttClient *mqtt = NULL;

// ============== Object to supply system functions ================================
class System: public MqttClient::System {
public:
    unsigned long millis() const {
        return ::millis();
    }
};

// ============== Object to implement network connectivity ======================
// Current example assumes the network TCP stack is connected using serial
// interface to pins 10(RX) and 11(TX). The SoftwareSerial library is used
// for actual communication.
#define SW_UART_PIN_RX                                  2
#define SW_UART_PIN_TX                                  3
#define SW_UART_SPEED                                   9600L
class Network {
public:
    Network() {
        mNet = new SoftwareSerial(SW_UART_PIN_RX, SW_UART_PIN_TX);
        mNet->begin(SW_UART_SPEED);
    }

    int connect(const char* hostname, int port) {
```

```
49          // TCP connection is already established otherwise do it here
50          return 0;
51      }
52
53      int read(unsigned char* buffer, int len, unsigned long timeoutMs) {
54          mNet->setTimeout(timeoutMs);
55          return mNet->readBytes((char*) buffer, len);
56      }
57
58      int write(unsigned char* buffer, int len, unsigned long timeoutMs) {
59          mNet->setTimeout(timeoutMs);
60          for (int i = 0; i < len; ++i) {
61              mNet->write(buffer[i]);
62          }
63          mNet->flush();
64          return len;
65      }
66
67      int disconnect() {
68          // Implement TCP network disconnect here
69          return 0;
70      }
71
72  private:
73      SoftwareSerial                                          *mNet;
74  } *network = NULL;
75
76  // ============== Setup all objects ==========================================
77  void setup() {
78      // Setup hardware serial for logging
79      Serial.begin(HW_UART_SPEED);
80      while (!Serial);
81      // Setup network
82      network = new Network;
83      // Setup MqttClient
84      MqttClient::System *mqttSystem = new System;
85      MqttClient::Logger *mqttLogger = new MqttClient::LoggerImpl<HardwareSerial>(
            Serial);
86      MqttClient::Network * mqttNetwork = new MqttClient::NetworkImpl<Network>(*
            network, *mqttSystem);
87      //// Make 128 bytes send buffer
88      MqttClient::Buffer *mqttSendBuffer = new MqttClient::ArrayBuffer<128>();
89      //// Make 128 bytes receive buffer
90      MqttClient::Buffer *mqttRecvBuffer = new MqttClient::ArrayBuffer<128>();
91      //// Allow up to 2 subscriptions simultaneously
92      MqttClient::MessageHandlers *mqttMessageHandlers = new MqttClient::
            MessageHandlersImpl<2>();
93      //// Configure client options
94      MqttClient::Options mqttOptions;
95      ////// Set command timeout to 10 seconds
96      mqttOptions.commandTimeoutMs = 10000;
97      //// Make client object
```

```
98      mqtt = new MqttClient(
99          mqttOptions, *mqttLogger, *mqttSystem, *mqttNetwork, *mqttSendBuffer,
100         *mqttRecvBuffer, *mqttMessageHandlers
101     );
102 }
103
104 // ============== Subscription callback ========================================
105 void processMessage(MqttClient::MessageData& md) {
106     const MqttClient::Message& msg = md.message;
107     char payload[msg.payloadLen + 1];
108     memcpy(payload, msg.payload, msg.payloadLen);
109     payload[msg.payloadLen] = '\0';
110     LOG_PRINTFLN(
111         "Message arrived: qos %d, retained %d, dup %d, packetid %d, payload:[%s]",
112         msg.qos, msg.retained, msg.dup, msg.id, payload
113     );
114 }
115
116 // ============== Main loop ====================================================
117 void loop() {
118     // Check connection status
119     if (!mqtt->isConnected()) {
120         // Re-establish TCP connection with MQTT broker
121         network->disconnect();
122         network->connect("mymqttserver.com", 1883);
123         // Start new MQTT connection
124         LOG_PRINTFLN("Connecting");
125         MqttClient::ConnectResult connectResult;
126         // Connect
127         {
128             MQTTPacket_connectData options = MQTTPacket_connectData_initializer;
129             options.MQTTVersion = 4;
130             options.clientID.cstring = (char*)MQTT_ID;
131             options.cleansession = true;
132             options.keepAliveInterval = 15; // 15 seconds
133             MqttClient::Error::type rc = mqtt->connect(options, connectResult);
134             if (rc != MqttClient::Error::SUCCESS) {
135                 LOG_PRINTFLN("Connection error: %i", rc);
136                 return;
137             }
138         }
139         // Subscribe
140         {
141             MqttClient::Error::type rc = mqtt->subscribe(
142                 MQTT_TOPIC_SUB, MqttClient::QOS0, processMessage
143             );
144             if (rc != MqttClient::Error::SUCCESS) {
145                 LOG_PRINTFLN("Subscribe error: %i", rc);
146                 LOG_PRINTFLN("Drop connection");
147                 mqtt->disconnect();
148                 return;
149             }
```

```
150         }
151     } else {
152         // Publish
153         {
154             const char* buf = "Hello";
155             MqttClient::Message message;
156             message.qos = MqttClient::QOS0;
157             message.retained = false;
158             message.dup = false;
159             message.payload = (void*) buf;
160             message.payloadLen = strlen(buf);
161             mqtt->publish(MQTT_TOPIC_PUB, message);
162         }
163         // Idle for 1 seconds
164         mqtt->yield(1000L);
165     }
166 }
```

C Código-fonte roteador serial

```
import time
import serial
import socket

addr = ('10.0.0.127', 80)

ser = serial.Serial(
        port = '/dev/ttyS0',
        baudrate = 9600,
        parity = serial.PARITY_NONE,
        stopbits = serial.STOPBITS_ONE,
        bytesize = serial.EIGHTBITS,
        timeout = None
)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(addr)

while 1:
    x = ""
    while ser.available():
        x += ser.read(1)
    print(x)
    try:
        s.send(x.encode('utf-8'))
    except Exception as e:
        print(e)
s.close()
```

D Código fonte da API REST

```
1
2     from flask import (Flask, jsonify, make_response, abort, url_for, request)
3     from smartmarket_data import data
4
5     app = Flask(__name__)
6     logfile = 'smartmarket_log.txt'
7     db = data('smartmarket.db')
8
9     @app.errorhandler(404)
10    def not_found(error):
11        return make_response(jsonify({'error': 'Not found'}), 404)
12
13    @app.route('/')
14    def index():
15        return "<html><body><p><h1>API</h1></p></body></html>"
16
17    @app.route('/smartmarket/api/compras/nova', methods=['GET'])
18    def start_compra():
19        _id = db.start_compra()
20        return jsonify({"URI":url_for('get_compra', id_compra=_id["ID"], _external=
              True)})
21
22    @app.route('/smartmarket/api/compras/<int:id_compra>', methods=['GET'])
23    def get_compra(id_compra):
24        result = db.get_compra(id_compra)
25        if not result:
26            abort(404)
27        else:
28            result["URI"] = url_for('get_compra', id_compra=result["ID"], _external=
                  True)
29            result.pop("ID")
30            for produto in result["PRODUTOS"]:
31                produto["URI"] = url_for('get_produto', id_produto=produto["ID"],
                      _external=True)
32                produto.pop("ID")
33            return jsonify(result)
34
35    @app.route('/smartmarket/api/compras/<int:id_compra>', methods=['POST'])
36    def api_call(id_compra):
37        r = db.find_compra(id_compra)
38        if not r:
39            abort(404)
40        if not request.json:
41            abort(400)
42        if not "ID" in request.json:
43            abort(400)
44        else:
45            id_produto = request.json.get("ID")
```

```
46              quantidade = request.json.get("QUANTIDADE")
47              if not db.find_produto(id_produto):
48                  abort(404)
49              db.add_produto(id_compra, id_produto, quantidade)
50              with open(logfile, 'w', encoding = 'utf8') as log:
51                  log.write(request.json.get("LOG"))
52              return request.json
53
54
55      @app.route('/smartmarket/api/compras/<int:id_compra>/adicionar', methods=['PUT'
            ])
56      def add_produto(id_compra):
57          r = db.find_compra(id_compra)
58          if not r:
59              abort(404)
60          if not request.json:
61              abort(400)
62          if not "ID" in request.json:
63              abort(400)
64          else:
65              id_produto = request.json.get("ID")
66              quantidade = request.json.get("QUANTIDADE")
67              if not db.find_produto(id_produto):
68                  abort(404)
69              db.add_produto(id_compra, id_produto, quantidade)
70              return request.json
71
72      if __name__ == "__main__":
73          app.run(host = '10.0.0.147', port=80, debug=True)
```