UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Bauru

ISRAEL VIEIRA FERREIRA

Uma Arquitetura para a Aplicação da Internet das Coisas Industrial

Bauru

2018

ISRAEL VIEIRA FERREIRA

Uma Arquitetura para a Aplicação da Internet das Coisas Industrial

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Faculdade de Engenharia Elétrica da Universidade Estadual Paulista "Júlio de Mesquita Filho" para obtenção do título de Mestre em Engenharia Elétrica.

Área de Concentração: Automação

Linha de Pesquisa: Mecatrônica

Orientador: Prof. Dr. Eduardo Paciência Godoy

Bauru – SP

2018

Ferreira, Israel Vieira.
Uma arquitetura para aplicação da internet das coisas industrial / Israel Vieira Ferreira, 2018
93 f. : il.

Orientador: Eduardo Paciencia Godoy

Dissertação (Mestrado)-Universidade Estadual Paulista. Faculdade de Engenharia, Bauru, 2018

1. Indústria 4.0. 2. Internet das Coisas. 3. Sistemas Embarcados. 4. OpenPLC. I. Universidade Estadual Paulista. Faculdade de Engenharia. II. Uma arquitetura para aplicação da internet das coisas industrial.

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Bauru

ATA DA DEFESA PÚBLICA DA DISSERTAÇÃO DE MESTRADO DE ISRAEL VIEIRA FERREIRA, DISCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA, DA FACULDADE DE ENGENHARIA - CÂMPUS DE BAURU.

Aos 04 dias do mês de setembro do ano de 2018, às 14:00 horas, no(a) Instituto de Ciência e Tecnologia de Sorocaba-UNESP, reuniu-se a Comissão Examinadora da Defesa Pública, composta pelos seguintes membros: Prof. Dr. EDUARDO PACIÊNCIA GODOY - Orientador(a) do(a) Departamento de Engenharia de Controle e Automação / Instituto de Ciência e Tecnologia/UNESP/Sorocaba, Prof. Dr. WESLEY ANGELINO DE SOUZA do(a) Departamento de Computação / Universidade Federal de São Carlos - Campus de Sorocaba / SP, Prof. Dr. IVANDO SEVERINO DINIZ do(a) Engenharia de Controle e Automação / Instituto de Ciência e Tecnologia/UNESP/Sorocaba, sob a presidência do primeiro, a fim de proceder a arguição pública da DISSERTAÇÃO DE MESTRADO de ISRAEL VIEIRA FERREIRA, intitulada **UM MODELO PARA A APLICAÇÃO DA INTERNET DAS COISAS INDUSTRIAL**. Após a exposição, o discente foi arguido oralmente pelos membros da Comissão Examinadora, tendo recebido o conceito final: APROVADO. Nada mais havendo, foi lavrada a presente ata, que após lida e aprovada, foi assinada pelos membros da Comissão Examinadora.

Prof. Dr. EDUARDO PACIÊNCIA GODOY

Prof. Dr. WESLEY ANGELINO DE SOUZA

Prof. Dr. IVANDO SEVERINO DINIZ

AGRADECIMENTOS

Ao professor Eduardo Paciência Godoy pela orientação, apoio e confiança em todos os momentos.

Aos meus pais pelo amor e incentivo incondicional.

À Fernanda Mildemberger pelo companheirismo e compreensão nos momentos mais difíceis.

E aos colegas de pós-graduação que tanto me ajudaram nesse percurso.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

RESUMO

A busca por eficiência e maior produtividade tem levado inovação para a indústria. A Indústria 4.0 é um novo conceito que representa uma evolução dos sistemas produtivos atuais a partir da convergência entre novas tecnologias de automação industrial e tecnologia da informação. Nesse contexto, a Internet das Coisas Industrial (IIoT) se destaca pela evolução na comunicação entre sistemas e equipamentos, disponibilizando para os usuários diversas informações úteis para o gerenciamento e aperfeiçoamento dos sistemas de produção. Este trabalho apresenta o desenvolvimento de um modelo para aplicação da IIoT baseado em soluções de código aberto (*open source*), que habilita a programação, comunicação em rede e supervisão dos equipamentos, permitindo monitorar máquinas e processos e controlar uma planta industrial. A arquitetura do modelo de aplicação é composta por três componentes: o servidor da aplicação, o módulo gateway e os módulos remotos. Os protocolos de comunicação desenvolvidos para a solução são: Modbus TCP/IP, MQTT e CoAP para conexão entre os módulos remotos e o gateway e o HTTP utilizando o padrão REST no formato JSON para a comunicação da aplicação externa com o gateway. Para viabilizar a programação da lógica de controle das máquinas e plantas foi utilizado o projeto OpenPLC. O modelo, adicionalmente, permite armazenar e disponibilizar em tempo real, de forma padronizada e interoperável, as informações do processo ou sistema, para qualquer tipo de plataforma conectada à rede e/ou Internet. Testes de operação realizados validaram o desenvolvimento dos módulos remotos e do gateway. Resultados obtidos com a implementação do modelo para o controle e monitoramento de um processo de manufatura demonstraram a eficácia e o potencial de uso da solução para aplicações de IIoT e Industria 4.0.

Palavras-chave: *Industria 4.0, Internet das Coisas, Sistemas Embarcados, OpenPLC.*

ABSTRACT

The focus on efficiency and higher productivity has brought innovation to the industry. The Industry 4.0 is a new concept representing the evolution of the current production systems through the convergence of new technologies of industrial automation and information technology. In this context, the Industrial Internet of Things (IIoT) stands out for the evolution on the communication among equipment and systems, offering to users a variety of useful information for the management of improvement of production systems. This dissertation presents the development of an application model for IIoT based on open source solutions, which enables the programming, network communication and supervision of equipment, and allows monitoring machines and processes and controlling an industrial plant. The architecture of the application model consists of three components: the application server, the gateway module, and the remote modules. The communication protocols developed for the solution are: Modbus TCP/IP, MQTT and CoAP for connection between the remote modules and the gateway; and HTTP using the REST standard with JSON format for communication between the external application with the gateway. The OpenPLC project was used to enable the programming of equipment and plant control logic. Besides, this model stores information of the process or system and makes it available online and accessible in real-time in a standardized and interoperable way to any point connected to the network and/or the Internet. Operation tests validated the development of the gateway and remote modules. Results obtained with the implementation of the model for the control and monitoring of a manufacturing process have demonstrated the efficacy and the potential use of the solution to IIoT and Industry 4.0 applications.

Keywords: *Industry 4.0, Internet of Things, Embedded Devices, OpenPLC.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ACK	*Acknowledgement*
AD	Analógico/Digital
API	*Application Programming Interface*
ARM	*Advanced* RISC *Machine*
ASCII	*American Standard Code for Information Interchange*
CLP	Controlador Lógico Programável
CoAP	*Constrained Application Protocol*
CON	*Confirmable*
CoRE	*Constrained RESTful Environments*
CPS	*Cyber-Physical Systems*
CR	*Carriage Return*
CRC	*Cyclic Redundancy Check*
CSI	*Camera Serial Interface*
CSMA/CD	*Carrier Sense Multiple Access with Collision Detection*
DCS	*Distributed Control Systems*
DDS	*Data Distribution Service*
DIO	*Dual I/O*
DIRT	*Data-Intensive Real-Time application*
DPWS	*Devices Profile for Web Services*
DRBD	*Distributed Replicated Block Device*
DSI	*Display Serial Interface*
ECMA	*European Computer Manufacturers Association*
ERP	*Enterprise Resource Planning*, Sistema de Gestão Empresarial
ETSI	*European Telecommunications Standards Institute*
FBD	*Function Block Diagram*, Diagrama de Blocos Funcionais
GIF	*Graphics Interchange Format*
GPS	*Global Positioning System*, Sistema de Posicionamento Global
HDMI	*High-Definition Multimedia Interface*, Interface Multimídia de Alta Resolução
HMI	*Human Machine Interface*, Interface Homem-Máquina
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
I/O	*Input/Output*, Entrada/Saída
IBM	*International Business Machines*
IDE	*Integrated Development Environment*, Ambiente de Desenvolvimento Integrado
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IETF	*Internet Engineering Task Force*
IIoT	*Industrial Internet of Things*
IL	*Instruction List*, Lista de Instruções
IoT	*Internet of Things*
IP	*Internet Protocol*
IPv4	*Internet Protocol* versão 4
IPV6	*Internet Protocol* versão 6

JPEG *Joint Photographic Experts Group*
JSON *JavaScript Object Notation*
LD *Ladder*
LF *Line Feed*
LRC *Longitudinal Redundancy Check*
M2M *Machine to Machine*
MAC *Medium Access Control*, Controle de Acesso ao Meio
MES *Manufacturing Execution System*
MIME *Multipurpose Internet Mail Extensions*
MID *Message Identifier*
MQTT *Message Queue Telemetry Transport*
NAT *Network Address Translation*
NON *Non-confirmable*
OLE *Object Linking and Embedding*
OPC *OLE for Process Control* ou *Open Platform Communications*
OPC DA *OPC Data Access*
OPC UA *Open Platform Communications Unified Architecture*
OSI *Open System Interconnection*
OTA *Over The Air programming*
PAC *Programmable Automation Controller*
PC *Personal Computer*
QIO *Quad I/O*
QoS *Quality of Service*, Qualidade de Serviço
RAM *Random Access Memory*, Memória de Acesso Aleatório
REST *Representational State Transfer*
RISC *Reduced Instruction Set Computer*
RFC *Request for Comments*
RFID *Radio-Frequency Identification*, Identificação por radiofrequência
RST *Reset*
RTU *Remote Terminal Unit*
SBC *Single Board Computer*, Computador de Placa Única
SCADA *Supervisory Control & Data Acquisition Systems*
SD *Secure Digital Card*
SDCD Sistema Digital de Controle Distribuído
SFC Sequenciamento Gráfico de Funções
SOA *Service Oriented Architecture*, Arquitetura Orientada a Serviços
SOAP *Simple Object Access Protocol*
SSID *Service Set Identifier*
STL *Structured Text Language*, Texto Estruturado
TA Tecnologias de Automação
TCP *Transmission Control Protocol*
TI Tecnologia da Informação
TLV *Type-Length-Value*
UDP *User Datagram Protocol*

URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locater*
USB	*Universal Serial Bus*
WSDL	*Web Services Description Language*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

1.1 Justificativa

A crescente necessidade de se produzir mais e melhor tem levado o mundo industrial a passar por várias revoluções. A primeira revolução industrial aconteceu no século 18 com o advento da máquina a vapor, revolucionando a produção têxtil. O desenvolvimento da energia elétrica e da produção em massa levaram à segunda revolução industrial no final do século 19. Em 1969, a crescente utilização da tecnologia da informação e eletrônicos na indústria gerou a terceira revolução industrial (A VOZ DA INDUSTRIA, 2016). Dessa forma, a indústria tem evoluído de forma a incorporar novas tecnologias e consequentemente obter maior produtividade (SAUTER et al., 2011).

A Indústria 4.0 (I4.0), também chamada de Quarta Revolução Industrial, é um novo conceito que representa uma evolução dos sistemas produtivos atuais a partir da convergência entre novas tecnologias de automação industrial (TA) e Tecnologia da Informação (TI) (BERGER, 2014). A I4.0 se destaca pela evolução na comunicação entre sistemas e equipamentos, disponibilizando para os usuários diversas informações úteis para o gerenciamento e aperfeiçoamento dos sistemas de produção. É possível monitorar e controlar todas as ferramentas da produção, melhorando a produtividade e qualidade, através de um melhor ajuste das máquinas (MANYIKA, 2015).

A maioria das tecnologias necessárias à implementação da I4.0 já existem atualmente. Entre elas pode-se citar o protocolo IPV6 para ampliação dos pontos de conexão IP a todos os equipamentos, a ampliação do uso de redes sem fio para flexibilização da comunicação e aquisição de dados, o uso de virtualização através da disponibilização de sistemas e serviços a partir de softwares; a Internet das Coisas (IoT – *Internet of Things*) e Computação em Nuvem (Cloud Computing) visando a conexão de todos os equipamentos em redes e o compartilhamento das informações na nuvem (ZUEHLKE, 2010; STANKOVIC, 2014, HEGAZY & HEFEEDA, 2015). O grande desafio, portanto, é promover a integração entre essas tecnologias, visando a obtenção de uma nova realidade produtiva, onde tudo estará conectado para que as melhores decisões de produção, custo e segurança sejam tomadas, tudo sob demanda e em tempo real.

Uma das tecnologias base para este conceito é a Internet das Coisas (IoT) e a comunicação Máquina para Máquina (M2M – *Machine to Machine*) (STANKOVIC, 2014). A IoT contempla a conexão lógica de todos os dispositivos e meios relacionados ao ambiente produtivo em questão como os sensores, controladores, computadores, células de produção, sistema de planejamento produtivo, sendo todas as informações compartilhadas através de bancos de

dados. A M2M representa a interconexão entre células de produção, onde tais sistemas podem trocar informações entre si, de forma autônoma, e tomar decisões de produção e segurança relacionadas ao processo através de um modelo de inteligência gerenciado pela IoT.

A IoT é um paradigma que preconiza um mundo de objetos físicos embarcados com sensores e atuadores, conectados por redes de comunicação, principalmente sem fio, e que se comunicam usando a Internet, moldando uma rede de objetos inteligentes capazes de realizar variados processamentos, capturar variáveis e reagir a estímulos externos. O termo IoT foi inventado por Kevin Asthon em 1999 no contexto de *Supply Chain* da indústria. Porém a definição se expandiu para outras aplicações, como na área de saúde, transporte, comunicação (SUNDMAEKER et al., 2010). Atualmente, as pesquisas em torno da IoT têm atraído interesse e focado em diferentes aplicações (STANKOVIC, 2014). Cada foco de aplicação da IoT possui características e requisitos específicos. Quando aplicada à área industrial, a IoT industrial (IIoT – *Industrial Internet of Things*) busca simplificar e criar arquiteturas de sistemas que são mais acessíveis, responsivas e efetivas, com comunicações eficientes e interação entre a produção e sensores, atuadores, entre outros, para aprimorar o desempenho e flexibilidade da indústria (LYDON, 2014).

A IoT apoia-se em três elementos: o Hardware, o Middleware e o Interfaceamento (GUBBI et al., 2013). O hardware é parte física do sistema, podendo ser composto por dispositivos programáveis (CLP e microprocessadores), sensores, atuadores e dispositivos de comunicação. O Interfaceamento ou interface de apresentação retorna ao usuário final as informações obtidas pelo hardware em conjunto com o software, além de possibilitar ao usuário o controle total ou parcial do sistema. O middleware é o mediador entre os outros dois elementos, por exemplo criando o elo de ligação entre softwares, hardwares e aplicações de diferentes protocolos de comunicação. Além disso, o middleware na IoT também disponibiliza uma interface de armazenagem por demanda e ferramentas computacionais, para viabilizar o acesso e análise dos dados da aplicação.

A manufatura é uma indústria bem estabelecida, com máquinas e linhas de produção que funcionam com um alto grau de automação. Sistemas SCADA (*Supervisory Control & Data Acquisition Systems*) e sistemas de controle distribuído (*Distributed Control Systems* - DCS) são padrões na indústria. A IIoT é complementar a esses sistemas, utilizando as informações provenientes dos sistemas SCADA e DCS como fontes de dados. Sistemas SCADA tem foco em monitoramento e controle, enquanto o foco da IIoT é analisar os dados para aumentar a produtividade e eficiência (KULKARNI, 2016).

Sistemas SCADA são ideais para o monitoramento da fábrica. A IIoT se aplica em um nível acima, como por exemplo informando qual é a eficiência operacional da planta como um

todo, ou das máquinas individualmente e dizendo como aumentar essa eficiência através da identificação e eliminação dos pontos fracos. Existe também a aplicação da IIoT na manutenção preventiva, possibilitando prever quando falhas irão ocorrer nas máquinas baseando-se em dados de telemetria (KULKARNI, 2016).

A arquitetura dos sistemas SCADA mudou conforme a computação evoluiu, seguindo as tendências da tecnologia. Os grandes avanços nas áreas da computação e comunicação levaram à inovação nos sistemas SCADA, de sistemas centralizados para sistemas distribuídos pela rede, sendo a nova geração destes sistemas baseada na grande disponibilidade da Internet atualmente. Um sistema SCADA Web é capaz de prover as mesmas funções críticas de um sistema SCADA tradicional. Entretanto, essas funções estão implementadas através da mesma tecnologia característica de um Web site, podendo rodar diretamente no navegador do usuário (LI, SERIZAWA & KIUCHI, 2002).

A partir da integração entre as tecnologias citadas, como a comunicação entre dispositivos utilizando os conceitos de M2M e IoT, o consumo e processamento dessas informações trocadas entre os dispositivos e a exibição dessas informações de modo gráfico e organizado, como ocorre em um sistema SCADA, é proposto neste trabalho a criação de uma arquitetura para a aplicação da IIoT.

1.2 Objetivos

Esta proposta objetivou criar uma arquitetura, através da integração de hardware e software, para aplicação da IoT na indústria. Esta arquitetura para a IIoT é baseada em ferramentas open source e fornece as funcionalidades de programação, comunicação em rede e supervisão dos equipamentos de automação e adicionalmente armazenamento e disponibilização online de forma padronizada e interoperável, das informações do processo ou sistema em questão, para qualquer tipo de plataforma, como smartphones, tablets ou computadores, tornando os dados de monitoramento acessíveis em tempo real, para qualquer ponto conectado à rede e/ou Internet.

1.3 Estrutura e Conteúdo

A estrutura deste trabalho ficou dividida em 8 capítulos, considerando esta introdução e desconsiderando as referências bibliográficas e os anexos.

O capítulo 2 trata da revisão bibliográfica, buscando as mais relevantes pesquisas que já foram publicadas acerca do tema, as definições e pesquisas sobre a Indústria 4.0 e as tecnologias de automação e informática envolvidas nesse contexto.

O capítulo 3 traz uma revisão dos conceitos importantes utilizados no trabalho, como os protocolos de comunicação utilizados na Ethernet Industrial, os protocolos da IoT e as tecnologias utilizadas na Internet, que envolvem protocolos de comunicação, tecnologia de Web Service e formatos para transporte da informação.

O capítulo 4 apresenta o modelo de aplicação da IIoT proposto no trabalho, toda a arquitetura contendo os módulos e suas interações e a descrição dos hardwares e softwares utilizados neste trabalho.

O capítulo 5 discorre sobre a descrição detalhada dos módulos remotos e gateway, como é o funcionamento, como é feita a configuração dos módulos remotos, como foi implementado o gateway, o formato dos dados e funcionamento da API REST, utilizada para comunicação com o módulo gateway.

O capítulo 6 apresenta os resultados dos testes preliminares dos softwares desenvolvidos para os módulos remotos e para o gateway, além da validação do hardware, teste da comunicação dos módulos remotos com o gateway e da API REST.

O capítulo 7, por sua vez, apresenta as considerações finais e as conclusões obtidas neste trabalho.

O capítulo 8, propõe alguns temas para trabalhos futuros, baseados nos resultados obtidos neste trabalho.

Por fim, o capítulo 9 apresenta as referências bibliográficas utilizadas neste trabalho.

2 REVISÃO BIBLIOGRÁFICA

2.1 Indústria 4.0

Sistemas embarcados com microprocessadores cada vez mais potentes estão mais conectados por redes sem fio e com a Internet. Isso possibilita a convergência do mundo real com o virtual, originando o conceito de Sistemas Ciber-físicos (CPS - *Cyber-Physical Systems*). No contexto da indústria, o termo CPS descreve uma composição de sistemas mecatrônicos, sistemas de comunicação e tecnologias de informação, formando uma rede integrada de tecnologias cibernéticas e elementos físicos, interagindo entre si para controlar sistemas e processos de forma distribuída e com grande capacidade de tomada decisões, seja de forma autônoma ou de forma colaborativa (RAJKUMAR et al., 2010).

Com a introdução do IPv6, em 2012, atualmente existem endereços IP suficientes para possibilitar a conexão direta de objetos inteligentes à Internet. Isso viabiliza a IoT, possibilitando conectar recursos, informação, objetos e pessoas entre si. Os efeitos desse fenômeno já podem ser observados na indústria. Na manufatura, a influência deste avanço na comunicação entre dispositivos, atrelado a outros, norteou o que ser descrito como a quarta revolução industrial, ou Industria 4.0 (KAGERMANN, WAHLSTER & HELBIG, 2013).

A Alemanha foi a primeira a propor a integração da IoT com a indústria e apresentou o conceito de Indústria 4.0 em 2011, durante a feira de Hannover. A indústria 4.0 foi uma estratégia do governo alemão visando desenvolver tecnologia de ponta (A VOZ DA INDUSTRIA, 2016). Kagermann, Wahlster & Helbig (2013) descrevem sobre o uso de estratégias para a implementação da Industria 4.0, visando manter a Alemanha como uma das principais produtoras de máquinas e tecnologia, além de possibilitar que suas indústrias continuem figurando entre as mais avançadas do mundo.

Segundo Kagermann Wahlster & Helbig (2013), o potencial da Indústria 4.0 fica evidente devido à algumas características e possibilidades, como:

- Atender aos requisitos individuais do cliente: A linha de produção pode se modificar para atender critérios individuais, específicos do cliente. Mudanças de última hora e volumes de produção muito baixos (tamanho de lote de 1), também são possíveis;
- Flexibilidade: É possível configurar dinamicamente diferentes aspectos dos processos de negócios, tais como qualidade, tempo, risco e preço. Isso possibilita a otimização do uso de materiais, os processos de engenharia podem ser tornados mais ágeis, processos de fabricação podem ser alterados, a escassez temporária de algum material (devido a problemas de

fornecimento, por exemplo) pode ser compensada e grandes aumentos no tamanho dos lotes produzidos podem ser alcançados em um curto espaço de tempo;

- Tomada de decisão otimizada: A I4.0 permite a verificação antecipada das decisões de projeto, respostas mais flexíveis à interrupção e otimização global em todos os setores de uma empresa;

- Produtividade e eficiência dos recursos: A mesma motivação que resultou nas revoluções industriais anteriores impulsiona a I4.0, a obtenção da maior produção possível a partir de um determinado volume de recursos (produtividade dos recursos) e utilização da menor quantidade possível de recursos para produzir uma determinada quantidade de produtos (eficiência de recursos). Os processos de fabricação podem ser otimizados para cada caso individual, até mesmo sem parar a produção, sendo continuamente otimizados em termos de consumo de recursos e energia ou de redução de suas emissões.

No Brasil, o conceito da Industria 4.0 ficou conhecido como Manufatura Avançada, e os estudos das tecnologias envolvidas começaram a acontecer por volta de 2013. Nos Estados Unidos, o conceito recebeu o nome de *Smart Industry* (Indústria Inteligente) (A VOZ DA INDUSTRIA, 2016). Segundo A Voz da Indústria (2016), estima-se que no Brasil, contabilizando-se aplicações da IoT nos diversos setores da economia (saúde, energia, mobilidade urbana, agricultura e manufatura), haja um impacto de US$ 39 bilhões até 2030, podendo chegar a US$ 210 bilhões se forem criadas condições favoráveis para acelerar a absorção das tecnologias relacionadas à IoT.

Nos Estados Unidos, segundo estudo da consultoria americana *McKinsey*, as aplicações de IoT na indústria tem o potencial de criar um valor de U$ 1,2 trilhões à U$ 3,7 trilhões por ano em 2025, sendo que se estima que serão de U$ 633 bilhões à U$ 1,2 trilhões apenas em otimização de operações. Isso inclui a utilização de sensores, ao invés do julgamento humano para ajustes nas máquinas. Os dados gerados também podem ser utilizados para fins de manutenção preventiva (MANYIKA, 2015).

Antes mesmo do conceito de I4.0 surgir, já se estudavam novas tecnologias para uso industrial, como as redes sem fio. A utilização delas na indústria cria novas possibilidades de *layout* no chão de fábrica, possibilitando a modificação da posição dos equipamentos, sem se preocupar com a posição dos cabos (WOLLSCHLAEGER, SAUTER, JASPERNEITE, 2017). Nesse sentido, em junho de 2005 foi criada uma organização sem fins lucrativos chamada ‘‘*Technology Initiative SmartFactory*’’. Os membros dessa organização são empresas de vários setores da economia, além de algumas universidades e centros de pesquisa. O objetivo dessa

organização é a criação, implementação e teste de tecnologias inovadoras em plantas industriais (ZUEHLKE, 2010).

A utilização de comunicação sem fio, em conjunto com uma abordagem modular de construção da linha de produção, permite que a planta opere de acordo com o princípio "*plug'n work*" de funcionamento. Segundo esta abordagem, cada módulo tem um certo grau de automação e sabe a sua função na cadeia de processos. Como não há conexões de rede físicas, é simples trocar ou adicionar novos módulos, para modificação ou extensão da funcionalidade dos processos produtivos. Os componentes modulares reconhecem sua função e se posicionam na cadeia de processos, se integrando automaticamente nos sistemas de controle da planta. Os módulos se configuram automaticamente de acordo com os módulos vizinhos, identificando seus papéis na linha de produção (ZUEHLKE, 2010).

2.2 Tecnologias de Automação e Informática

Atualmente com a IIoT e a Indústria 4.0, se torna imprescindível a convergência entre as tecnologias de informação e automação. Existe uma tendência nos últimos anos de uma maior aplicação de tecnologias de TI como sistemas Web, Computação em Nuvem, Serviços Web e arquiteturas orientadas a serviços (*SOA – Service Oriented Architecture*) em sistemas de automação e controle industrial. Também pode ser verificado o surgimento de novas tecnologias como *Devices Profile for Web Services* (DPWS), *Open Platform Communications Unified Architecture* (OPC UA), *Data Distribution Service* (DDS) e o *AutomationML* (*Automation Markup Language*) para suprir as demandas desse novo formato da indústria (JAMMES et al., 2014).

No início a Ethernet TCP/IP não era aceita na indústria, devido à sua característica não determinística. Não haviam garantias de que as mensagens seriam entregues, pois o mecanismo de controle de acesso ao meio de transmissão (CSMA/CD - *Carrier Sense Multiple Access with Collision Detection*) empregado por essa tecnologia tratava as colisões detectando-as e reenviando as mensagens após um período de tempo aleatório. Múltiplas colisões sucessivas poderiam acontecer, tornando obsoleto o dado transmitido na mensagem até chegar ao seu destino. Uma das soluções encontradas foi a utilização de um switch industrial, que por possuir diversas portas independentes com buffer, é capaz de guardar a mensagem de uma das portas para reenviar posteriormente, além de permitir a programação de prioridades e tempos de esperas das mensagens. Isso amenizou o problema, norteando a aceitação da rede Ethernet TCP/IP no chão de fábrica (LUGLI; SANTOS; FRANCO, 2008, 2009).

A terminologia Ethernet Industrial compreende, portanto, uma vertente de melhorias e novos desenvolvimentos para cumprir os requisitos de comunicação industrial e viabilizar o

uso da Ethernet TCP/IP na indústria (DECOTIGNIE, 2009). Nesse sentido, diversas propostas podem ser encontradas atualmente como o Modbus TCP/IP, Profinet e EtherNet/IP, entre outros. Lugli, Santos & Franco (2008, 2009) descrevem sobre as tecnologias inovadoras do mercado de redes industriais, além do funcionamento e características de cada um dos principais protocolos utilizados na Ethernet Industrial.

Soluções baseadas em Ethernet Industrial permitem ao usuário a integração dos equipamentos de campo a serviços de TI fornecidos por servidores (DECOTIGNIE, 2005). O uso de protocolos de rede suportados a nível industrial expandiu-se rapidamente, possibilitando a comunicação de dados dos sistemas de automação provenientes de chão de fábrica e de dados armazenados em bancos de dados de servidores. Dessa forma, tornou-se possível disponibilizar informações provenientes desses sistemas de automação através de protocolos com suporte ao Ethernet TCP/IP em aplicações Web.

Essa interconexão de dados e infraestruturas de redes de TI e TA em alto nível e de forma segura e confiável representa uma tendência para o desenvolvimento de soluções inovadoras e integradoras de automação e controle na área industrial com a Indústria 4.0 (BERGER, 2014). Algumas dessas soluções inovadoras envolvem a aplicação dos padrões utilizados por Web Services em camadas cada vez mais baixas da hierarquia industrial, chegando até mesmo ao nível de dispositivos, com padrões como o OPC UA e o DPWS.

OPC UA é um padrão para comunicação industrial do tipo cliente-servidor. É o sucessor do OPC Classic ou OPC DA (*Data Access*), sendo utilizado para a comunicação flexível em aplicações industriais que não tem requisitos críticos de tempo real. O OPC Classic foi criado em 1996, numa iniciativa de usar a tecnologia da Microsoft chamada OLE (*Object Linking and Embedding*) para aplicações em controle de processo (*OPC – OLE for Process Control*). Com a adoção do padrão em outras áreas, além do controle de processos, e a flexibilização para uso em outros sistemas operacionais (Linux, Android, etc), a OPC Foundation mudou seu nome para *Open Platform Communications* em 2011 (OPC, 2017).

O OPC UA tem sido visto como um padrão muito importante para as aplicações da IIoT e Industria 4.0 (HOPPE, 2017). O servidor fornece acesso, através de uma estrutura de dados similar ao conceito de objeto em programação orientada a objetos, a dados e funções com os quais os clientes podem interagir através de um conjunto de serviços padronizados. Estruturas de dados do tipo pedido-resposta (*request-response*) são definidos para cada serviço. Essas estruturas são construídas a partir de tipos básicos, como inteiros, e codificadas em um formato binário personalizado ou em XML (GRÜNER, PFROMMER & PALM, 2016).

Cândido et al. (2010) avaliaram os padrões OPC UA e DPWS quanto à sua aplicabilidade em uma arquitetura orientada a serviços no nível de dispositivos. DPWS foi apresentado como

uma proposta para uso de protocolos relacionados à tecnologia de Web Services na camada de dispositivos de uma rede industrial. Define dois elementos fundamentais: o dispositivo e seus serviços. Além dos serviços desenvolvidos pelo usuário, também são oferecidos serviços relacionados à infraestrutura, como descoberta de dispositivos utilizando UDP, troca de metadados entre dispositivos utilizando mecanismos como o WSDL (*Web Services Description Language*), oferecido por Web Services SOAP e serviços relacionados a eventos. A aplicação de Web Services a nível de dispositivos tem diversos benefícios, como a unificação dos protocolos utilizados no nível de TI e chão de fábrica, possibilitando a comunicação direta entre o nível de dispositivos e o nível de MES ou ERP da empresa.

DPWS foi desenvolvido para operar a partir do nível mais baixo, de dispositivos mais simples, enquanto o OPC UA é mais focado no nível de gateway, que se apoia em protocolos de comunicação proprietários para acessar os dispositivos do nível mais baixo. Embora existam algumas implementações do OPC UA em dispositivos mais simples, é um padrão que continua focado em comunicação entre controladores e sistemas SCADA e HMI. Cândido et al. (2010) concluem que nenhuma dessas duas especificações preenchem todos os requisitos de uma arquitetura orientada a serviços a nível de dispositivos, sendo proposta uma combinação dos dois padrões para a obtenção de uma solução mais completa. Isso também é proposto por Izaguirre, Lobov & Lastra (2011), que aponta como benefícios dessa abordagem sinérgica a possibilidade de customizar operações e eventos de cada dispositivo, pois o DPWS permite que isso seja feito via WSDL personalizado pelo usuário, que pode ser descoberto e invocado utilizando-se o paradigma ponto a ponto.

Enquanto a maioria das tecnologias de Web Services utilizada na indústria é baseada em SOAP/XML, existe uma tendência crescente visando reduzir a complexidade dos Web Services, utilizando uma alternativa ao SOAP que é o *Representational State Transfer* (REST). REST é um estilo de arquitetura muito usada em aplicações descentralizadas orientadas a recursos. Ao invés de se definir interfaces personalizadas para cada aplicativo, REST propõe o uso de um conjunto fixo de interfaces de serviço para transferir representações de recursos heterogêneas. Um recurso pode ser uma entidade física ou virtual (GRÜNER, PFROMMER& PALM, 2016).

Richardson & Ruby (2007) descrevem quatro propriedades de Web Services RESTful como parte de uma arquitetura orientada a recursos.

- *Addressability*: É atribuído um endereço a cada recurso, atuando como seu identificador e que contém informações suficientes para estabelecer um canal de comunicação;
- *Statelessness*: As solicitações e respostas devem ser autossuficientes e não dependem de o receptor armazenar informações para cada conexão. Isso permite a comunicação

assíncrona, onde várias solicitações são enviadas em paralelo, sem esperar por uma resposta. Além disso, os requisitos na memória principal para servidores que precisam para lidar com muitas conexões são reduzidos. Este requisito pode ser enfraquecido para permitir um meio de transporte com suporte para pedidos de pacotes, retransmissão, bem como autenticação e comunicação criptografada;

- *Connectedness*: Descreve como os recursos se referem uns aos outros em suas representações, através de seus endereços de identificação. Os usuários podem então navegar nesses links e descobrir sobre o contexto e as relações de cada recurso;
- *Uniform Interface*: A interface para interagir com representações de recursos é sempre a mesma, independente do recurso. Em HTTP, o protocolo usado para acessar sites, os métodos disponíveis (GET, POST, PUT, DELETE e outros) são os mesmas para todos os sites e tipos de mídia transferidos. Em abordagens orientadas a serviços, muitos serviços podem estar disponíveis em um local, mas cada serviço tem argumentos diferentes e retornam valores diferentes, que os usuários precisam considerar em suas aplicações.

Grüner, Pfrommer & Palm (2016) propõem uma extensão RESTful ao OPC UA. Os testes de desempenho em hardware industrial mostraram um aumento na eficiência da comunicação, com taxas de transferência de dados até oito vezes maior para interações de curta duração. Isso ocorre devido a uma redução de sobrecarga de comunicação, sendo especialmente relevante para o uso e adoção do OPC UA como solução emergente para a Internet das Coisas Industrial (WOLLSCH LAEGER, SAUTER & JASPERNEITE, 2017).

Ungurean, Gaitan & Gaitan (2016) propõem e implementam uma arquitetura de quatro níveis para a IIoT, utilizando DDS como middleware. O objetivo é integrar a maior quantidade possível de protocolos a nível de dispositivos ao sistema. A grande vantagem da utilização de DDS como middleware é a possibilidade de configuração dos parâmetros de QoS de modo a alcançar a performance de tempo real exigida por uma aplicação industrial, uma vantagem sobre especificações OPC (OPC DA, OPC .NET, OPC UA).

O CoAP é um protocolo de comunicação criado por um grupo de trabalho da *Internet Engineering Task Force* (IETF) chamado *Constrained RESTful Environments* (CoRE). A IETF é uma comunidade internacional aberta de profissionais, pesquisadores e empresas preocupados com a evolução da arquitetura da Internet e seu funcionamento. Todo o trabalho técnico da IETF é feito por grupos de trabalho (*working groups*), que são organizados por assuntos de diversas áreas, como roteamento, transporte, segurança, dentre outros. O objetivo do CoRE era desenvolver um framework para aplicações simples, como monitoramento de sensores, em redes com largura de banda limitada.

O CoAP é parte deste framework, sendo que seu primeiro esboço foi publicado no IETF em junho de 2010. Em junho de 2014 tornou-se um *Request for Comments* (RFC), que é o primeiro passo para se tornar um protocolo padrão (MARTINS & ZEM, 2015). O CoAP é baseado nos mesmos princípios do modelo REST. Servidores disponibilizam recursos em um URL e os clientes acessam esses recursos usando os mesmos métodos do HTTP (GET, PUT, POST e DELETE). Como o HTTP e o CoAP compartilham o modelo REST, eles podem ser facilmente conectados. Um cliente da Web pode nem sequer perceber que ele acessou um recurso de um sensor, pois o resultado é o mesmo de uma requisição HTTP comum (BORMANN, 2014).

Lerche, Hartke & Kovatsch (2012) apresentam uma visão geral das implementações atuais do CoAP e discutem os resultados do primeiro encontro voltado para a interoperabilidade entre soluções de diferentes fabricantes, organizado pelo *European Telecommunications Standards Institute* (ETSI), em março de 2012. Dentre as implementações do CoAP, haviam implementações em servidores, que são dispositivos ricos em recursos, utilizando Java ou C++. Alguns fabricantes apresentaram dispositivos que permitem a interoperabilidade entre o CoAP e o HTTP. Apesar de ser um protocolo relativamente novo, diversas implementações diferentes foram observadas, indo desde dispositivos simples como sensores e atuadores até implementações em smartphones e servidores.

MQTT é um protocolo de comunicação criado pela *IBM* e pela *Eurotech* em 1999, que apresenta como principal característica o fato de ser leve e aplicável em redes com alta latência e largura de banda limitada (MARTINS; ZEM, 2015). O MQTT é baseado na arquitetura *publish/subscribe*, em que existem dispositivos que publicam informações na rede, organizando essas informações em categorias ou tópicos, e dispositivos que declaram interesse em determinadas categorias de informação, recebendo assim apenas as informações relevantes para si. Um dispositivo central funcionando como servidor, chamado *broker*, é responsável pela comunicação entre produtores e consumidores de dados. Os dispositivos se comunicam com o broker usando TCP (MARTINS; ZEM, 2015).

Grehs (2016) utilizou o MQTT para a comunicação no seu sistema de irrigação doméstico. Um ponto interessante é que neste trabalho o broker MQTT está na nuvem da *Amazon*, o que possibilitou o levantamento de questões como a utilização de *Network Address Translation* (NAT) pelas operadoras de Internet e como isso impacta a conexão para um sistema baseado na IoT. NAT é uma tecnologia utilizada para traduzir IPv4s entre duas redes. Recentemente passou a ser utilizado pelos provedores de Internet para virtualização do IPv4, possibilitando que um único IPv4 público seja dividido por vários IPv4 privados. Até que o IPv6 seja completamente adotado, muitos usuários ainda utilizarão IPv4 privados.

Isso é um problema caso o servidor esteja utilizando um IPv4 compartilhado, sendo necessário o uso de proxy para o acesso. Grehs (2016) afirma que haveriam problemas se fosse utilizado o protocolo CoAP para a comunicação em seu sistema de irrigação doméstico, pois a maioria dos usuários residenciais estão utilizando a tecnologia NAT. Entretanto, como foi utilizado um servidor localizado na nuvem, através de um broker MQTT, não houveram problemas, pois os clientes utilizaram conexões TCP, mantidas através de *keep-alives*1, e a troca de informações pertinentes ao protocolo foram realizadas com sucesso.

Das tecnologias e conceitos apresentados, foram escolhidos para a comunicação entre máquinas e dispositivos do chão de fábrica os protocolos CoAP, MQTT e Modbus (Ethernet Industrial). Um gateway industrial obtém dados destas máquinas e dispositivos utilizando os protocolos citados e disponibiliza estes dados através do protocolo HTTP seguindo o modelo REST. Estes conceitos são apresentados no próximo capítulo, como conceitos fundamentais para a arquitetura proposta.

[1] *Keep-Alive* é um mecanismo do protocolo TCP para verificar se uma conexão ainda está ativa. Um pacote TCP sem dados, com uma *flag* indicando que deve ser retornada uma mensagem de *Acknowledgement* é enviado periodicamente. A ocorrência ou não de uma resposta indica se a conexão ainda está ativa.

3 REVISÃO CONCEITUAL

3.1 Ethernet Industrial e Modbus TCP/IP

A Ethernet tem se tornado cada vez mais presente na indústria, com muitos protocolos de comunicação industrial sendo adaptados para soluções baseadas em Ethernet. As comunicações Ethernet com TCP/IP normalmente não são determinísticas e o tempo de reação é geralmente de cerca de 100 ms. Os protocolos de Ethernet Industrial usam uma camada de Controle de Acesso ao Meio (MAC) modificada para obter respostas de latência muito baixas. Outra característica interessante da Ethernet é permitir uma topologia de rede flexível e um número flexível de nós no sistema (LIN & PEARSON, 2013).

O Modbus é um protocolo aberto que define uma estrutura de mensagens, usadas para transferir dados discretos e analógicos entre dispositivos microprocessados com detecção e informação de erros de transmissão. O protocolo Modbus é baseado no modelo de comunicação mestre-escravo, onde apenas o único dispositivo mestre pode inicializar a comunicação¸ e os demais dispositivos escravos, respondem enviando os dados solicitados pelo mestre, ou realizam alguma ação solicitada.

O dispositivo mestre pode endereçar e se comunicar com cada dispositivo escravo da rede individualmente ou acessar a todos da rede através de mensagens em transmissão (*broadcast*). Quando o mestre envia uma mensagem (*query*) endereçada a um escravo, apenas o dispositivo endereçado retorna uma resposta (*response*) e nunca são gerados responses quando uma query for do tipo *broadcast*, pois nesse caso vários dispositivos estão recebendo essa informação ao mesmo tempo.

O formato das mensagens (*query*) definidas pelo protocolo Modbus é estabelecido da seguinte forma:

- *Address*: Tamanho de 1 byte. Número que define o endereço do dispositivo (escravo);
- *Function Code*: Tamanho de 1 byte. Define a ação a ser executada pelo escravo (ler dado, aceitar dado, reportar estado, reporte de erros);
- *Data*: Tamanho 0 a 252 bytes. Contém informações que o Dispositivo/Servidor deve usar para executar a ação definida pela função requisitada pelo Mestre/Cliente (endereços de memória, quantidade de itens transmitidos, tamanho do campo Data);
- *Error check*: Tamanho de 2 bytes representando a checagem de erro na comunicação (enviado por um e checado pelo outro). Em caso de erro, é solicitada a retransmissão da mensagem.

Já o formato das respostas (response) seguem o mesmo modelo de uma mensagem (query), porém, são ajustadas obedecendo o formato da função requerida:

- confirmação da função;
- parâmetros pertinentes as funções;
- campo de *checksum*.

Se o dispositivo não estiver apto para atender à função requisitada, ou se houver algum erro de comunicação, ele responde com uma mensagem específica (exception).

O protocolo Modbus pode ser configurado para trabalhar com um dos dois modos de transmissão disponíveis: ASCII (American Standard Code for Information Interchange) ou RTU (Remote Terminal Unit). O modo de transmissão é o que define como os dados serão empacotados na mensagem. Em uma rede industrial utilizando o protocolo Modbus, qualquer um dos modos pode ser usado, mas todos os dispositivos da rede devem ser configurados com o mesmo modo de transmissão.

Para cada um dos modos de transmissão, existe um método para identificar o início e o fim de uma mensagem. Isso é importante para que todos os dispositivos sejam capazes de identificar os campos constituintes do frame Modbus. Deste modo, o dispositivo pode identificar o endereço do destinatário de determinada mensagem e saber se deve ou não atender àquela requisição.

No modo de transmissão RTU o início e o fim de uma mensagem são identificados por períodos de silêncio (silent), quando não pode haver nenhuma transmissão de dados. Em contrapartida, no modo ASCII o início de uma mensagem é identificado pelo caractere dois pontos (:), enquanto o final da mensagem é identificado pelo conjunto de caracteres “Retorno de Carro” (Carriage Return - CR) e “Avanço de Linha” (Line Feed - LF). Devido a isto, a principal vantagem do modo ASCII é permitir intervalos grandes entre o envio de dados de uma mesma mensagem (ALFA INSTRUMENTOS, 2000).

No modo ASCII todos os dados são enviados conforme o padrão ASCII. Isso significa que são permitidos apenas valores 30H a 39H e 41H a 46H, que correspondem aos caracteres 0 a 9 e A a F, em hexadecimal. A quantidade de bits por cada palavra de dados do frame sempre será igual a 10, independente da configuração escolhida. Estas são as possíveis configurações:

- 1 Start bit, 7 data bits, sem paridade e 2 stop bits;
- 1 Start bit, 7 data bits, paridade PAR e 1 stop bits;
- 1 Start bit, 7 data bits, paridade IMPAR e 1 stop bits.

No campo de checksum é utilizado o método LRC (*Longitudinal Redundancy Check*).

No modo RTU, para cada palavra de dados da mensagem é enviado apenas um caractere no padrão hexadecimal. A palavra de dados sempre será igual a 11, independente da configuração dos parâmetros.

- 1 Start bit, 8 data bits, sem paridade e 2 stop bits;
- 1 Start bit, 8 data bits, paridade PAR e 1 stop bits;
- 1 Start bit, 8 data bits, paridade IMPAR e 1 stop bits;

O campo checksum do frame é gerado pelo método CRC (*Cyclic Redundancy Check*). A principal vantagem deste modo RTU em relação ao ASCII é a maior densidade de caracteres que é enviada numa mesma mensagem, aumentando o desempenho da comunicação.

Embora seja utilizado normalmente sobre conexões seriais padrão RS-232, ele também pode ser usado como um protocolo da camada de aplicação de redes industriais como a Ethernet TCP/IP. O Modbus talvez seja o protocolo de mais ampla utilização no campo das redes industriais, sendo utilizado por diversos controladores e ferramentas para desenvolvimento de sistemas supervisórios. Isto se deve a sua grande simplicidade e facilidade de implementação.

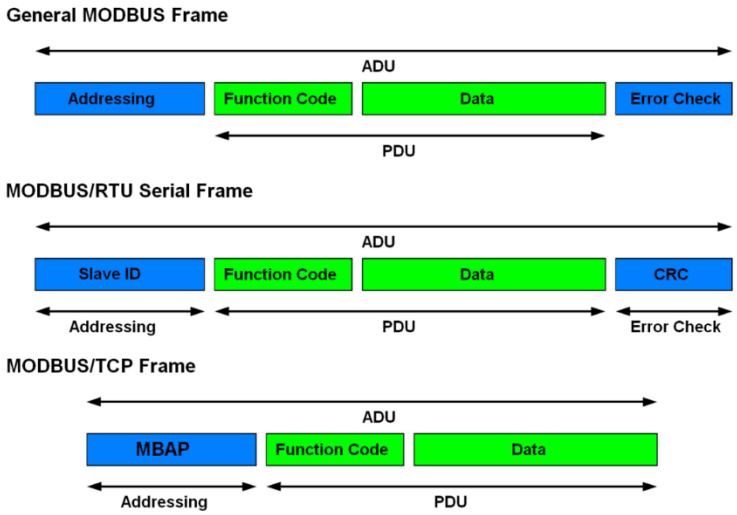
Modbus TCP/IP (também Modbus-TCP) é simplesmente o protocolo Modbus RTU com uma interface TCP e endereçamento IP que funciona com uma rede Ethernet. Na estrutura de camadas de redes de comunicação do modelo OSI, Modbus é o protocolo de Aplicação que define as regras para organizar e interpretar a dados independentes do meio de transmissão de dados. TCP/IP refere-se ao Protocolo de Internet (IP - Internet Protocol) da Camada de Rede e ao protocolo de controle de transmissão (TCP - Transmission Control Protocol) da Camada de Transporte para comunicação Modbus (CALDIÉRI, 2016).

Simplificando, TCP/IP permite que blocos de dados binários possam ser trocados entre computadores. Ele também é um padrão mundial que serve como base para a World Wide Web. A função primária de TCP é de assegurar que todos os pacotes de dados serão recebidos corretamente, enquanto o IP que garante que as mensagens sejam dirigidas corretamente e encaminhadas. Note-se que a combinação de TCP/IP é apenas um protocolo de transporte, e não define o que os dados significam ou como os dados devem ser interpretados (este é o trabalho da camada de aplicação, Modbus, neste caso) (CALDIÉRI, 2016). Em resumo, mensagem TCP/IP Modbus é simplesmente uma comunicação Modbus encapsulado em um invólucro de Ethernet TCP/IP.

Na prática, Modbus TCP incorpora um frame de dados Modbus padrão em um frame TCP, sem a soma de verificação Modbus. Os comandos Modbus e dados do utilizador são encapsulados no interior do recipiente de dados de uma mensagem TCP/IP sem modificar a

estrutura básica da mensagem Modbus. As diferenças estão na interpretação do endereço (IP) e na verificação de erro (CRC-32) conforme mostrado na Figura 1.

Figura 1 - Frame Modbus.



Fonte: Modbus (2017).

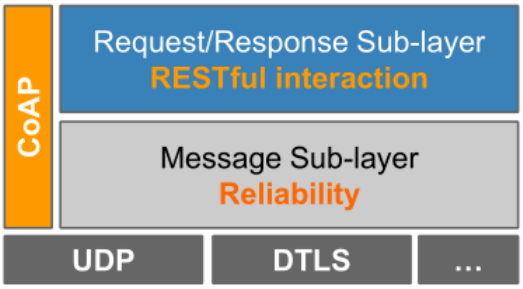
3.2 Protocolos da IoT

3.2.1 CoAP

O *Constrained Application Protocol* (CoAP) é um protocolo de comunicação desenvolvido para dispositivos com baixo consumo energético, processamento e em redes limitadas. Com frequência esses dispositivos têm microcontroladores de 8 bits, com quantidades reduzidas de memória e estão se comunicando através de redes com alta taxa de perda de pacotes e baixa velocidade. É um protocolo pensado para aplicações M2M (SHELBY; HARTKE; BORMANN, 2014).

O objetivo do CoAP não é apenas comprimir o HTTP, mas sim ter um subconjunto de características REST em comum com o HTTP, mas otimizadas para aplicações M2M (SHELBY; HARTKE; BORMANN, 2014). CoAP é baseado na troca de mensagens sobre UDP/IP, que oferece um melhor desempenho do que o TCP/IP para comunicação sem fio de baixo consumo de energia e em redes com altas taxas de perda de pacotes (HUI & CULLER, 2008). Em alguns casos, existe a necessidade de mais garantias de entrega das mensagens do que o UDP pode fornecer, sendo implementada para isso uma fina camada de controle, além da camada de requisição-resposta, conforme a Figura 2. O CoAP pode ser dividido em duas subcamadas: A subcamada de requisição/resposta, responsável pelas interações RESTful e a subcamada de mensagem, que gerencia o reenvio de mensagens.

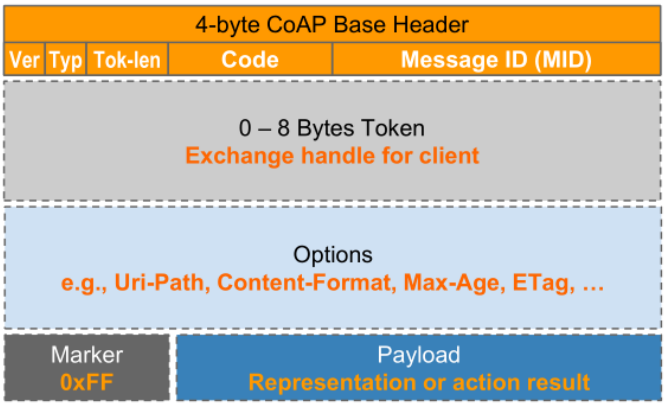
Figura 2 – Divisão em Camadas do Protocolo CoAP.



Fonte: Kovatsch (2015).

As mensagens são codificadas em um formato binário com um cabeçalho de 4 bytes, seguido por um token de largura variável (de 0 a 8 bytes). Uma série de opções do CoAP no formato *Type-Length-Value* (TLV) pode vir ou não depois do token, que são seguidas também opcionalmente por um payload, ocupando o resto do datagrama. A Figura 3 mostra o formato da mensagem CoAP (MARTINS; ZEM, 2015).

Figura 3 - Frame CoAP.

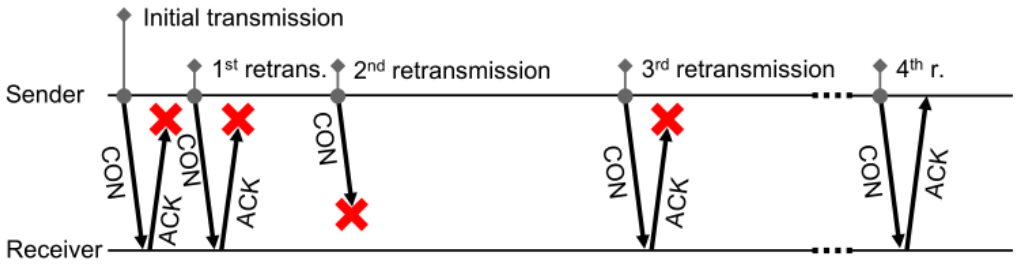


Fonte: Kovatsch (2015).

A subcamada de mensagem usa um identificador chamado *Message Identifier* (MID) para detectar mensagens duplicadas e quatro tipos de mensagem para aumentar a confiabilidade de entrega. O MID é compacto, tem 16 bits e permite a troca de cerca de 250 mensagens por segundo entre dois dispositivos. A confiabilidade de entrega da mensagem pode ser aumentada, marcando a mensagem como *Confirmable* (CON). A mensagem *Confirmable* é retransmitida em determinados intervalos de tempo, até que o destinatário envie uma mensagem de *Acknowledgement* (ACK) com o mesmo MID de mensagem (Figura 4). Ao receber uma mensagem CON, o receptor precisa responder com uma confirmação de recebimento (ACK). Caso uma das mensagens (CON ou ACK) seja perdida, a mensagem CON é reenviada até quatro vezes, com um intervalo inicial aleatório de 2 a 3 segundos entre cada mensagem, que é dobrado após cada retransmissão. Se o destinatário não for capaz de processar a mensagem, ele

responde com uma mensagem *Reset* (RST) ao invés do *Acknowledgement* (ACK). Uma mensagem que não precise de tanta confiabilidade na transmissão pode ser enviada como uma mensagem *Non-confirmable* (NON). Nesses casos, não há *Acknowledgement* por parte do destinatário, mas a mensagem ainda tem um MID para detecção de duplicatas. Se o receptor não for capaz de processar a mensagem, ele pode responder com uma mensagem Reset (RST) (SHELBY, HARTKE & BORMANN, 2014; KOVATSCH, 2015).

Figura 4 – Procedimento de Comunicação via Mensagem com Confiabilidade de Entrega (CON) no CoAP.



Fonte: Kovatsch (2015).

O token é gerado pelo cliente e espelhado pelo servidor, sendo necessário que seja único para cada requisição aberta ou, pelo menos, para cada destinatário, se o endereço do servidor também for utilizado para identificação das mensagens. Ele também pode ter tamanho de zero bytes, sendo chamado de empty token nesse caso, útil para minimizar o tamanho das mensagens. Convém notar que o que identifica uma resposta a uma requisição é a utilização do mesmo token e não o MID. Em alguns casos onde a resposta leva algum tempo, pode ser enviada uma mensagem ACK vazia, com o mesmo MID da requisição, enquanto a resposta é enviada posteriormente como uma mensagem CON com o mesmo token da requisição e um MID diferente. Mensagens NON sempre geram uma mensagem com MID diferente como resposta, precisando ser feita a correlação entre as mensagens através do token. Quando o cliente recebe uma resposta a uma mensagem CON que ainda não recebeu um ACK, ele interrompe as retransmissões, pois a resposta funciona como um *Acknowledgement* implícito. Uma mensagem CON vazia pode ser enviada para desencadear uma mensagem RST, comportamento que pode ser utilizado de modo similar ao PING, para testar a conectividade em uma rede (KOVATSCH, 2015).

Com funcionamento similar aos *headers* no HTTP, as requisições e respostas podem ter opções para especificar informações adicionais. Cada opção tem um número que a identifica. Uma mensagem pode conter várias opções, ordenadas em ordem crescente de acordo com seu identificador numérico. O identificador numérico não é utilizado diretamente, sendo utilizado um valor denominado delta, resultante de uma subtração entre o valor da opção que se deseja referenciar e da opção anterior. Caso não haja uma opção anterior, é considerado o próprio

identificador numérico da opção como o delta. Além do delta, cada referência a uma opção presente na mensagem deve especificar o tamanho do valor da opção e seu valor propriamente dito. O valor da opção pode ser uma sequência de bytes, que pode ter comprimento zero (campo valor vazio), um número inteiro não negativo ou uma string.

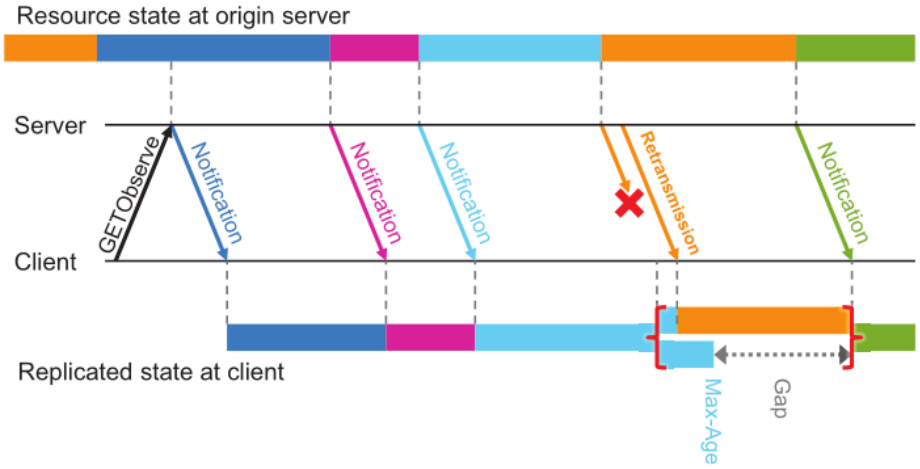
A subcamada de requisição/resposta implementa o modelo REST, o que faz com que o CoAP tenha muitas similaridades com o HTTP. Porém, como é voltado para dispositivos mais simples e para comunicação M2M, o CoAP tem algumas diferenças em relação ao HTTP, como o campo de código (Figura 3), usado no CoAP para identificar os métodos, além do status das operações como acontece no HTTP. No CoAP são utilizados quatro métodos para interação com os recursos: GET, PUT, POST e DELETE. Eles apresentam mesma semântica e funcionamento dos métodos análogos do HTTP. Os códigos de status também apresentam algumas diferenças, como um ponto separando a classe do código do restante, além de alguns códigos mais explícitos, como o "2.02 *Deleted*" e o "2.04 *Changed*", que equivalem ao código "204 *No Content*" do HTTP. As mensagens de erro podem conter um payload com uma mensagem curta para que desenvolvedores entendam o motivo de uma falha (KOVATSCH, 2015).

Existem extensões que adicionam funcionalidades ao protocolo CoAP. Uma delas é a funcionalidade de observar recursos. A implementação dessa funcionalidade é feita sobre uma mensagem com método GET, onde o cliente adiciona a opção "*observe*" igual a zero. Isso indica ao servidor que deve incluir o cliente na lista de "*observers*". Caso o servidor não consiga incluir o cliente na lista, ou não suporte essa funcionalidade, a resposta torna-se uma resposta padrão a uma requisição GET e não contém a opção "*observe*".

Ao incluir o cliente na lista, o servidor passa a enviar respostas ao cliente conforme o recurso observado se altera, fazendo o possível para manter a sincronia entre a representação do recurso no servidor e no cliente. Essas respostas são chamadas de notificações e são relacionadas com a requisição original através do token. Isso torna o modelo requisição/resposta em um modelo requisição/múltiplas-respostas. Convém notar que, diferentemente de um modelo *publish/subscribe*, o servidor pode não enviar notificações todas as vezes em que houver alterações no recurso observado. Isso pode ocorrer por congestionamento na rede ou recursos se alterando muito rapidamente. As notificações têm uma opção "*Max-Age*", utilizada pelo servidor para indicar até quando o cliente pode considerar aquela notificação uma representação válida do recurso observado. Elas podem ficar armazenadas em cache para serem utilizadas caso alguma notificação se perca. Caso a notificação se torne obsoleta, o cliente deve assumir que não está mais na lista de observadores e reenviar a requisição GET original. Isso pode gerar intervalos de tempo em que o cliente não tem uma representação válida daquele

recurso (Figura 5) até receber uma resposta. Se o cliente deseja parar de observar um determinado recurso, basta enviar uma requisição GET com a opção "*observe*" igual a um. Nas notificações, a opção "observe" apresenta um valor crescente, tornando possível ordená-las. Esse valor pode se iniciar em qualquer número e aumentar em passos arbitrários.

Figura 5 - Funcionamento da função Observe no CoAP.

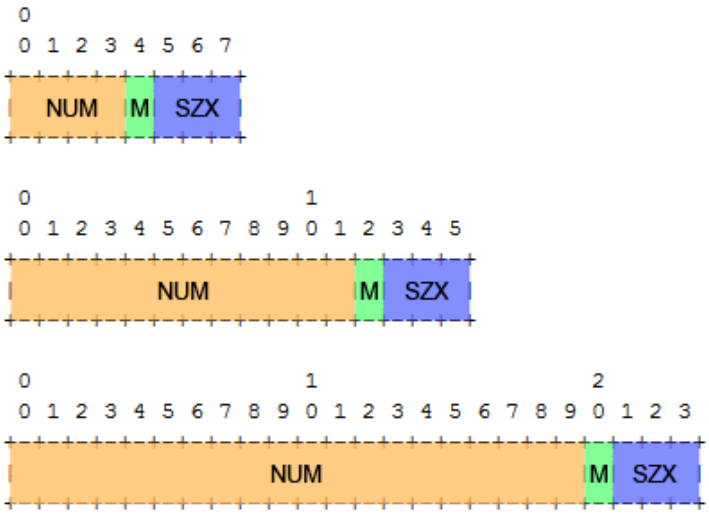


Fonte: Kovatsch (2015).

Outra extensão ao CoAP é a transferência *block-wise*, que é utilizada para dividir uma mensagem grande em várias mensagens menores. Isso é necessário, pois o CoAP é baseado em UDP/IP, que oferece apenas fragmentação de mensagens, o que pode ser problemático em um ambiente com restrições e redes com altas taxas de perdas de pacotes. A divisão das mensagens é feita através das opções "*Block1*" e "*Size1*", referentes à representação do recurso transferido na requisição e "*Block2*" e "*Size2*", referentes à representação do recurso transferido na resposta. O valor da opção *Block* é de tamanho variável, podendo ter entre 0 e 3 bytes. Conforme mostrado na Figura 6, ela deve conter o tamanho do bloco (SZX), quantos blocos virão a seguir (M) e o número relativo daquele bloco (NUM), em relação ao total de blocos. A opção *Size* indica o tamanho total estimado da representação do recurso que será transferida (BORMANN & SHELBY, 2016).

A descoberta de recursos contidos em um dispositivo também é feita por uma extensão do protocolo CoAP. O *Constrained RESTful Environments* (CoRE) *Link Format* padroniza essa descoberta de recursos voltada para a comunicação M2M. A URI "/.well-known/core" é definida como o ponto de entrada padrão para requisitar uma lista de links de recursos hospedados pelo servidor, possibilitando assim a descoberta de recursos. Esse mecanismo pode ser utilizado em outros protocolos web, como o HTTP (SHELBY, 2012).

Figura 6 - Formato da opção Block.



Fonte: Bormann & Shelby (2016).

A descoberta pode ser feita tanto por *unicast*, quanto por *multicast*, através de uma requisição com o método GET para a URI "/.well-known/core" relativa ao IP do servidor ou ao endereço de *multicast*, sendo encontrado no *payload* da resposta o *CoRE Link Format*.

O *CoRE Link Format* é composto por links entre parênteses angulares e atributos que são conectados por ponto e vírgula, sendo utilizada a vírgula para separação entre os links.

A Figura 7 mostra um exemplo de utilização do *CoRE Link Format*, sendo possível ver os diferentes atributos utilizados. O atributo "*title*" contém um nome amigável para o recurso que pode ser lido por um usuário humano, enquanto "rt" indica uma etiqueta para classificar o recurso que deve ser previamente conhecida. Caso contrário, o cliente deve ser capaz de acessar o link indicado pelo atributo "if". O tamanho máximo da representação de um recurso pode ser indicado pelo atributo "sz", enquanto uma lista com os formatos possíveis dessa representação é indicada por "ct". Os atributos "anchor" e "rel" são utilizados para descrever uma relação entre dois recursos (KOVATSCH, 2015).

Figura 7 - Exemplo de utilização do CoRE Link Format. A quebra de linha após as vírgulas foi utilizada apenas para melhorar a legibilidade, não sendo utilizada na prática.

```
</sensors/temp>;rt="ucum:cel",
</large>;sz=1280;title="Large resource",
</multi-format>;ct="0 41";title="text/plain and application/xml",
</t>;anchor="/sensors/temp";rel="alternate";title="Shortcut to sensor",
</semantic>;rt="restdesc";if="http://api.example.com/usage/"
```

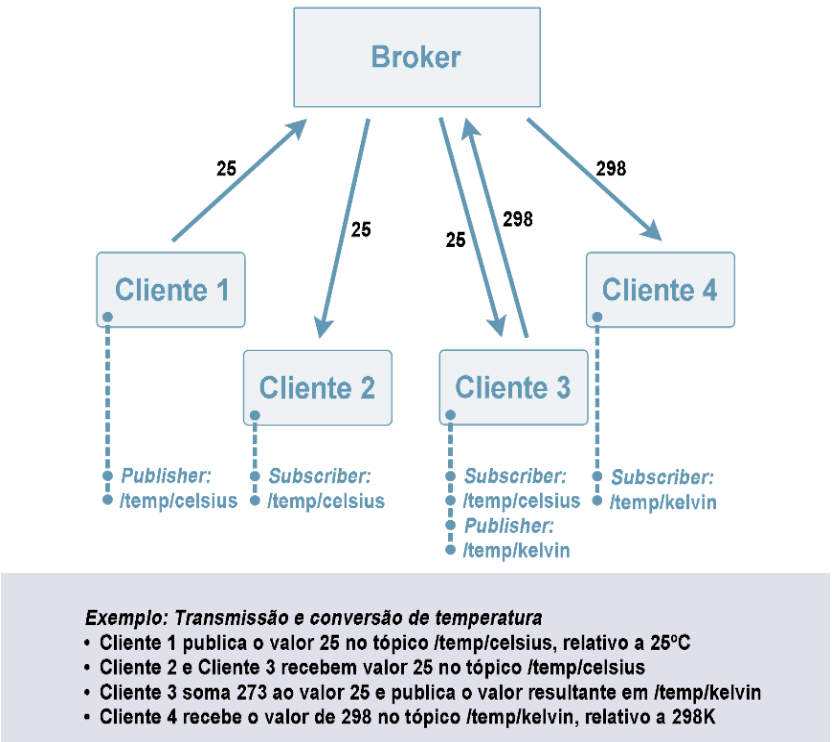
Fonte: Kovatsch (2015).

3.2.2 MQTT

O *Message Queue Telemetry Transport* (MQTT) é um protocolo de comunicação criado pela IBM e pela Eurotech no final dos anos 90, o qual está novamente ganhando visibilidade. Por ser leve e aplicável em redes com alta latência e largura de banda limitada, o MQTT tem sido visto como um dos protocolos com potencial para ser aplicado na IoT (JAFFEY, 2014). O MQTT é baseado na arquitetura *publish/subscribe* e organizando os dados em uma estrutura hierárquica de tópicos, separados por barras (/) semelhantemente a diretórios em um sistema operacional, conecta dispositivos que publicam informações na rede com dispositivos que declaram interesse em determinadas categorias de informação.

Os dispositivos consumidores de conteúdo recebem apenas as informações relevantes para si. Um dispositivo central funcionando como servidor, chamado *broker*, é responsável por gerenciar a comunicação entre produtores e consumidores de dados. Os produtores e consumidores de dados não se comunicam diretamente, sendo toda a comunicação centralizada pelo *broker*, que recebe os dados publicados pelos produtores em tópicos por eles determinados e encaminha aos consumidores, que declararam interesse naquele tópico específico em que os dados foram publicados. Essa comunicação com o *broker* é feita utilizando o protocolo TCP/IP e um exemplo detalhado do procedimento de comunicação é mostrado na Figura 8.

Figura 8 - Exemplo de comunicação utilizando a arquitetura publish/subscribe e a estrutura de tópicos e sub-tópicos.

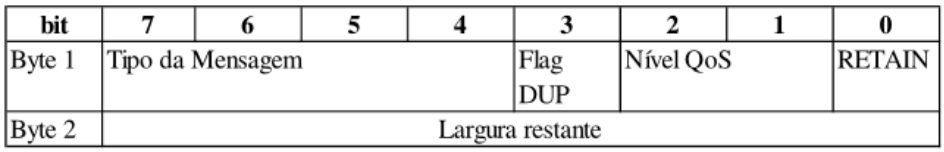


Fonte: Autoria própria.

Um exemplo para ilustrar essa estrutura foi dado por Jamie et al. (2005), em que se tem dois laboratórios com dois sensores de temperatura em cada. Os sensores publicam suas medições em tópicos específicos para cada sensor, porém esses tópicos são organizados dentro de categorias mais gerais, como o nome do laboratório em que estão localizados. Deste modo, é possível obter dados de diversos sensores de uma só vez, que podem estar distribuídos em um ambiente, não importando o tipo do sensor, apenas o tipo do dado que ele disponibiliza (como temperatura ou umidade, por exemplo). Isso favorece a escalabilidade, pois novos sensores poderiam ser adicionados e o dispositivo que faz uso daquela informação não precisaria ser reprogramado. Outro benefício é a independência em relação ao sensor físico, pois um sensor danificado pode ser substituído por outro, que publicaria o mesmo tipo de informação no mesmo tópico, fazendo com que essa substituição não seja percebida pelo sistema.

A mensagem (ou frame) MQTT é composta por um cabeçalho fixo de dois bytes. No primeiro byte existe um campo que identifica o tipo da mensagem. É também no primeiro byte em que estão as flags, que são bits reservados para identificar alguns tipos de mensagens especiais (DUP, Nível QoS e RETAIN). Isso pode ser visto na Figura 9. Os tipos de mensagem podem ser vistos na Tabela 1.

Figura 9 - Frame MQTT.



Fonte: Martins & Zem (2015).

Tabela 1 - Tipos de mensagens MQTT.

Nome	Valor	Direção da comunicação	Descrição
Reserved	0	-	Reservado
CONNECT	1	Cliente para o Servidor	Requisição do cliente para conexão com o servidor.
CONNACK	2	Servidor para o Cliente	Confirmação de recebimento da requisição de conexão.
PUBLISH	3	Cliente para o Servidor ou Servidor para o Cliente	Requisição para publicar
PUBACK	4	Cliente para o Servidor ou Servidor para o Cliente	Confirmação de recebimento da requisição para publicar
PUBREC	5	Cliente para o Servidor ou Servidor para o Cliente	Publish received (sistema de entrega assegurada de mensagem – parte 1)
PUBREL	6	Cliente para o Servidor ou Servidor para o Cliente	Publish release (sistema de entrega assegurada de mensagem – parte 2)
PUBCOMP	7	Cliente para o Servidor ou Servidor para o Cliente	Publish complete (sistema de entrega assegurada de mensagem – parte 3)
SUBSCRIBE	8	Cliente para o Servidor	Requisição do cliente para se inscrever em um tópico
SUBACK	9	Servidor para o Cliente	Confirmação de recebimento da requisição do cliente para se inscrever em um tópico

UNSUBSCRIBE	10	Cliente para o Servidor	Requisição do cliente para se desinscrever de um tópico
UNSUBACK	11	Servidor para o Cliente	Confirmação do recebimento da requisição do cliente para se desinscrever de um tópico
PINGREQ	12	Cliente para o Servidor	Requisição de PING
PINGRESP	13	Servidor para o Cliente	Resposta do PING
DISCONNECT	14	Cliente para o Servidor	Mensagem de desconexão do cliente
Reserved	15	-	

Fonte: Oasis (2015).

As flags *DUP*, Nível QoS e *RETAIN* tem um papel importante quando a mensagem é do tipo *PUBLISH*. A flag DUP é setada quando aquela mensagem é uma duplicata, ou seja, está sendo reenviada por algum motivo. O nível QoS determina o nível de confiabilidade de entrega da mensagem. Pode ser QoS 0, QoS 1 ou QoS 2. Quando uma mensagem é configurada como QoS 0, a mensagem é entregue de acordo com as capacidades da rede, não havendo garantias de entrega. Neste caso a mensagem pode ser entregue uma vez, ou não ser entregue. No caso do QoS 1, a mensagem é entregue ao menos uma vez, podendo ser entregue mais de uma vez. Isso é assegurado por uma resposta *PUBACK*, enviada pelo broker ao receber a mensagem *PUBLISH*. O nível QoS 2 garante que a mensagem será entregue apenas uma vez (OASIS, 2015).

Isso é feito por um sistema de entrega assegurada de mensagens, onde o broker recebe a mensagem do tipo *PUBLISH*, salva na memória o identificador daquela mensagem e responde com uma mensagem *PUBREC*, contendo o identificador. O cliente, ao receber a mensagem *PUBREC*, descarta a mensagem original e guarda apenas o identificador daquela mensagem. O cliente então envia uma mensagem do tipo *PUBREL* para o broker, contendo o identificador da mensagem original. Ao receber a mensagem *PUBREL*, o broker descarta os dados armazenados relativos àquela mensagem e envia uma mensagem *PUBCOMP*. O cliente, ao receber a mensagem *PUBCOMP* com o identificador correto, descarta os dados relativos àquela mensagem. Deste modo a entrega da mensagem original é assegurada (OASIS, 2015).

A flag *RETAIN* é utilizada para manter a última mensagem com esta flag setada na memória do broker, no tópico para qual foi enviada. Essa mensagem retida na memória do broker pode ser interpretada como o "último valor confiável" daquele dado e é enviada logo que um dispositivo se subscreve naquele tópico (JAMIE et al., 2005). Para retirar essa mensagem da memória do broker, basta enviar uma mensagem em branco com a *retained flag* setada (HIVEMQ, 2016).

O *broker* é o responsável por gerenciar as conexões, porém os demais dispositivos podem definir o que ocorre caso ocorra alguma desconexão inesperada. Isso é feito definindo uma mensagem que será enviada em um determinado tópico, caso haja uma desconexão inesperada.

Essa mensagem é conhecida como "*Last Will and Testament*". É um mecanismo para notificar que um dispositivo se desconectou da rede de forma anormal (MARTINS & ZEM, 2015).

O cliente que se conecta utilizando o protocolo MQTT, fica responsável por enviar um ping (mensagem para testar a conectividade entre dispositivos em uma rede) ao broker em um intervalo determinado de tempo, para certificar para o broker que ele ainda está conectado. Em contrapartida, o broker deve responder ao ping enviado pelo cliente, mostrando ao cliente que a conexão ainda está ativa. Caso o broker não receba o ping, a conexão é encerrada e o cliente tem que enviar uma nova requisição de conexão. Caso o cliente não receba a resposta do servidor, ele tenta se conectar novamente. Esse mecanismo é chamado *Keep Alive*.

3.3 Protocolos da Internet

3.3.1 HTTP

O HTTP (*Hypertext Transfer Protocol*, ou Protocolo de Transferência de Hipertexto) é um protocolo de comunicação que implementa o modelo cliente-servidor, da camada de aplicação, segundo o Modelo OSI (*Open System Interconnection*). É um dos protocolos que possibilitam a comunicação de dados da *World Wide Web*, sendo utilizado desde o início da década de 90 (FIELDING, 1999). Um sistema de comunicação em rede possui diversos protocolos que trabalham em conjunto para o fornecimento de serviços. O protocolo HTTP necessita que o protocolo TCP/IP forneça a conectividade necessária entre clientes e servidores, para que seja possível transferir seus dados pela Web.

Assim como a maioria dos protocolos de rede, o HTTP utiliza o paradigma de requisição e resposta. É um protocolo baseado em documentos, no qual o cliente envolve um documento em um envelope e o envia para o servidor. A resposta do servidor segue a mesma ideia, sendo enviado ao cliente um documento envolto em um envelope padronizado. O HTTP tem padrões bem definidos para como esse envelope deve ser, mas não se preocupa com seu conteúdo (RICHARDSON & RUBY, 2007).

A requisição enviada pelo cliente ao servidor deve conter um método, um URI (*Uniform Resource Identifier*, em português identificador uniforme de recurso), a versão do protocolo, seguido de uma mensagem MIME (padrão utilizado para codificar dados em formato de textos ASCII para serem transmitidos pela Internet) contendo os modificadores da requisição, informações sobre o cliente e, possivelmente, o conteúdo no corpo da mensagem. A estrutura dessa mensagem de requisição do cliente é composta por uma linha inicial (*Request-Line*); linhas de cabeçalhos (*Request-header*); uma linha em branco obrigatória e um corpo de mensagem opcional. A linha inicial de uma requisição é composta por três partes separadas por

espaços (caractere SP da tabela ASCII): o método, a identificação do URI e a versão do HTTP (FIELDING, 1999).

O método determina a ação que o servidor deve executar com o URI fornecido no momento da requisição de um recurso. O protocolo HTTP originalmente define oito métodos em sua versão 1.1 (GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS e CONNECT) (BASTOS, 2001; FIELDING & RESCHKE, 2014). Posteriormente foi definido o método PATCH, similar ao método PUT que é utilizado para atualizar os dados de um recurso, porém responsável pela atualização parcial deste recurso (DUSSEAULT & SNELL, 2010). No uso cotidiano da Web, os métodos mais utilizados são GET e POST. O método GET solicita ao servidor um recurso, que pode ser uma página HTML, uma imagem, um documento, XML, JSON, entre outros. O método POST é utilizado quando deseja-se enviar dados para o servidor, podendo estes dados serem provenientes de um formulário, de arquivos sendo enviados ao servidor, dentre outros.

O URI identifica o recurso sobre o qual será aplicada a requisição. No protocolo HTTP, o tipo de URI utilizado é chamado de URL (*Uniform Resource Locater*), composto pela identificação do protocolo, pelo endereço do servidor e pela localização do documento requisitado (BASTOS, 2001). A sintaxe geral de uma URL é composta pelo protocolo, servidor, porta, caminho e recurso. Se a porta não for especificada é utilizada a porta padrão, além disso caminho e recursos podem ser omitidos. As URLs podem conter dados ainda após o nome do recurso no formato “x-www-form-urlencoded”, que segue os seguintes critérios: não pode haver nenhum espaço em branco nos dados; dados são agrupados em pares nome=valor; pares são separados por &; espaços em branco são codificados com +; caracteres de 8 bits são codificados com %HH, onde HH é o código hexadecimal do caractere.

O cabeçalho da mensagem (*header*) permite ao cliente transmitir informações adicionais ao servidor relativas à requisição ou ao próprio cliente. O cabeçalho age como um modificador da requisição, funcionando de modo equivalente aos parâmetros passados a um método em uma linguagem de programação. Também é utilizado na resposta do servidor ao cliente, sendo sua função nesse caso transmitir informações sobre o servidor ou sobre o recurso identificado pelo URI da requisição (FIELDING, 1999).

O corpo da mensagem (*body*) é utilizado em alguns métodos para transmitir informação na requisição ou na resposta, sendo enviado abaixo das linhas de cabeçalho. No caso da resposta, o corpo da mensagem pode conter o recurso que foi requisitado pelo cliente ou uma mensagem de erro. Em uma mensagem de requisição, o corpo da mensagem pode conter dados estruturados em um formato padronizado, podendo ser até mesmo um arquivo que será enviado para o servidor. Neste caso, poderão ser incluídos cabeçalhos que descrevem as características

desse formato ou arquivo contido no corpo da mensagem, como por exemplo, o *Content-Type* que informa o tipo MIME dos dados e o *Content-Length* que informa a quantidade de bytes que o corpo da mensagem contém. A Tabela 2 apresenta alguns tipos MIME que podem ser incluídos do cabeçalho.

Tabela 2 – Lista de exemplos de MIME.

Exemplo	**Descrição**
text/plain	Arquivo no formato texto (ASCII)
text/html	Arquivo no formato HTML (padrão para páginas na WEB)
image/gif	Imagem com formato GIF
image/jpeg	Imagem com formato JPEG
application/zip	Arquivo compactado
text/json	Texto no formato JSON
text/xml	Texto no formato XML

Fonte: Autoria própria.

Ao receber e processar a requisição, o servidor responde com uma linha de status (*status line*) contendo a versão do protocolo, o código numérico que indica o status da operação, informando se houve erro ou sucesso, seguido por uma mensagem do tipo MIME, contendo informações do servidor, meta informações da entidade (entidade é a informação transferida como payload da mensagem) e possível conteúdo no corpo da mensagem. Após o envio da resposta pelo servidor, encerra-se a conexão estabelecida.

A linha de status contém um código numérico (código de status) e um texto explicativo, indicando se a requisição foi bem-sucedida ou se ocorreu algum erro. O código de status é formado por três dígitos e o primeiro dígito representa a classe que pertence classificada em cinco tipos (FIELDING, 1999):

- 1xx: Informação – utilizada para enviar informações para o cliente de que sua requisição foi recebida e está sendo processada;
- 2xx: Sucesso – indica que a requisição do cliente foi bem-sucedida;
- 3xx: Redirecionamento – informa a ação adicional que deve ser tomada para completar a requisição;
- 4xx: Erro no cliente – avisa que o cliente fez uma requisição que não pode ser atendida;
- 5xx: Erro no servidor – ocorreu um erro no servidor ao cumprir uma requisição válida.

Apenas alguns códigos em cada classe são descritos na RFC 2616 (documento que descreve os padrões do protocolo HTTP), porém existe a possibilidade de cada servidor definir seus próprios códigos.

3.3.2 Web Services RESTful

Web Service ou Serviço Web é um software modular com funções bem definidas que podem ser acessadas através de uma rede privada ou pública. É capaz de funcionar como um recurso para outras aplicações e sistemas, que interagem com os Web Services com ou sem a intervenção humana (PAPAZOGLOU, 2008).

Usa-se uma interface padronizada para se comunicar com os Web Services e formatos de arquivo como o XML e o JSON para transportar a informação. O REST, abreviação para *Representational State Transfer* (Transferência do Estado Representativo) pode ser usado para realizar a comunicação com um Web Service. O termo foi introduzido em 2000 por Roy Thomas Fielding, um dos autores do protocolo HTTP, amplamente utilizado para a troca de informações na Internet. O REST é uma filosofia que busca padronizar a comunicação entre aplicações e serviços em uma rede, apoiando-se em conceitos e tecnologias base para a comunicação da Internet como o HTTP e URIs.

Os principais princípios do REST são (FIELDING, 2000):

- As operações são baseadas em métodos HTTP padrões (GET, PUT, POST e DELETE);
- Os recursos são identificáveis por URIs;
- Comunicação sem estado (os serviços não guardam dados de sessão no servidor);
- Os dados são transferidos diretamente utilizando o protocolo HTTP, podendo ser utilizados arquivos como XML ou JSON para carregar a informação.

O termo RESTful é normalmente utilizado para se referir a implementação de Web Services seguindo esses princípios.

Nos últimos anos observou-se o crescimento dos Web Services que utilizam os princípios da arquitetura REST, como uma alternativa popular ao uso de tecnologias baseadas em SOAP (*Simple Object Access Protocol*, em português Protocolo Simples de Acesso a Objetos). A crescente popularidade desse tipo de Web Service deve-se ao aumento de performance do sistema, por aliviar o cliente de processar SOAP (CASTELEYN et al., 2009).

3.3.3 JSON

O *JavaScript Object Notation* (JSON) é um formato utilizado para troca de dados. As vantagens do JSON em relação ao XML são sua legibilidade para humanos e facilidade para máquinas o processarem e criarem. É baseado no padrão ECMA-262 3ª edição, de dezembro de 1999, derivado da linguagem de programação JavaScript. Apesar disso, o JSON é um formato de texto independente de linguagem de programação, podendo ser utilizado com

qualquer linguagem, apesar de suas convenções serem familiares aos programadores de linguagens baseadas em C, como o C, C++, C#, Java, JavaScript, Perl, Python, dentre outras (JSON, 2017).

O JSON é composto por basicamente duas estruturas:

- Uma coleção de pares compostos por atributos e valores, que dependendo da linguagem utilizada podem ser traduzidos como um objeto ou um array associativo, por exemplo;
- Uma lista ordenada de valores, podendo ser traduzida como um array ou um vetor.

Em JSON um objeto é composto por uma série de pares de atributos e valores não ordenados, contidos dentro de chaves. Em cada par o atributo e o valor são separados por dois pontos e cada par é separado entre si por vírgula. O valor pode ser uma string contida entre aspas duplas, um número, um valor booleano (true, false ou null), um objeto ou um array. Um array, é um conjunto ordenado de valores contidos em colchetes e separados por vírgula. Essas estruturas podem ser agrupadas de forma hierárquica.

Essas estruturas são suportadas por praticamente todas as linguagens de programação modernas, tornando o JSON uma escolha lógica para troca de informações entre sistemas.

Com protocolos de comunicação que atendem desde dispositivos mais simples e redes limitadas, como o MQTT e o CoAP, até protocolos como o HTTP e o modelo REST, utilizados para a criação de *Web Services*, uma arquitetura para a aplicação da IIoT é proposta no próximo capítulo.

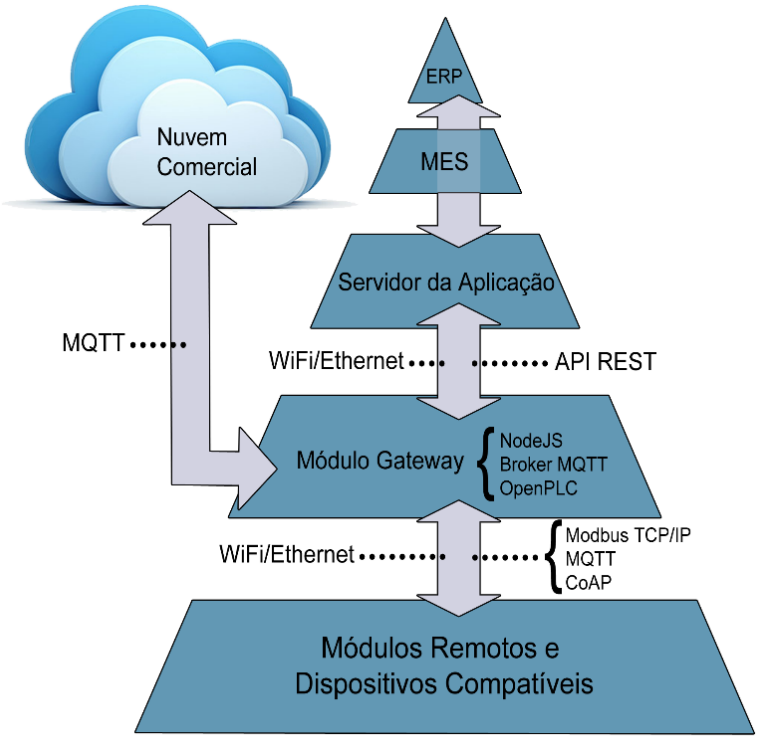
4 MATERIAIS E MÉTODOS

4.1 Proposta do Trabalho

Este projeto propõe a criação de um modelo para aplicação da IIoT. Este modelo para a IIoT é baseado em ferramentas open source e deve ser capaz de gerenciar a comunicação entre equipamentos e sistemas, monitorar máquinas e processos e controlar uma planta industrial. Adicionalmente, este modelo deve ser capaz de armazenar e disponibilizar online de forma padronizada e interoperável, essas informações do processo ou sistema em questão, para qualquer tipo de plataforma, como smartphones, tablets ou computadores, tornando os dados de monitoramento acessíveis em tempo real, para qualquer ponto conectado à rede e/ou Internet.

Deste modo, pretende-se integrar os dispositivos de chão de fábrica com os níveis superiores da chamada "pirâmide da automação", que é um modelo de rede organizado em diversos níveis hierárquicos, desenvolvido para lidar com a crescente complexidade dos dados em um ambiente de comunicação integrado tanto horizontalmente, quanto verticalmente (SAUTER, 2010). Essa estrutura proposta pode ser vista na Figura 10.

Figura 10 - Arquitetura proposta para aplicação da IIoT e sua integração com os níveis hierárquicos de uma indústria.

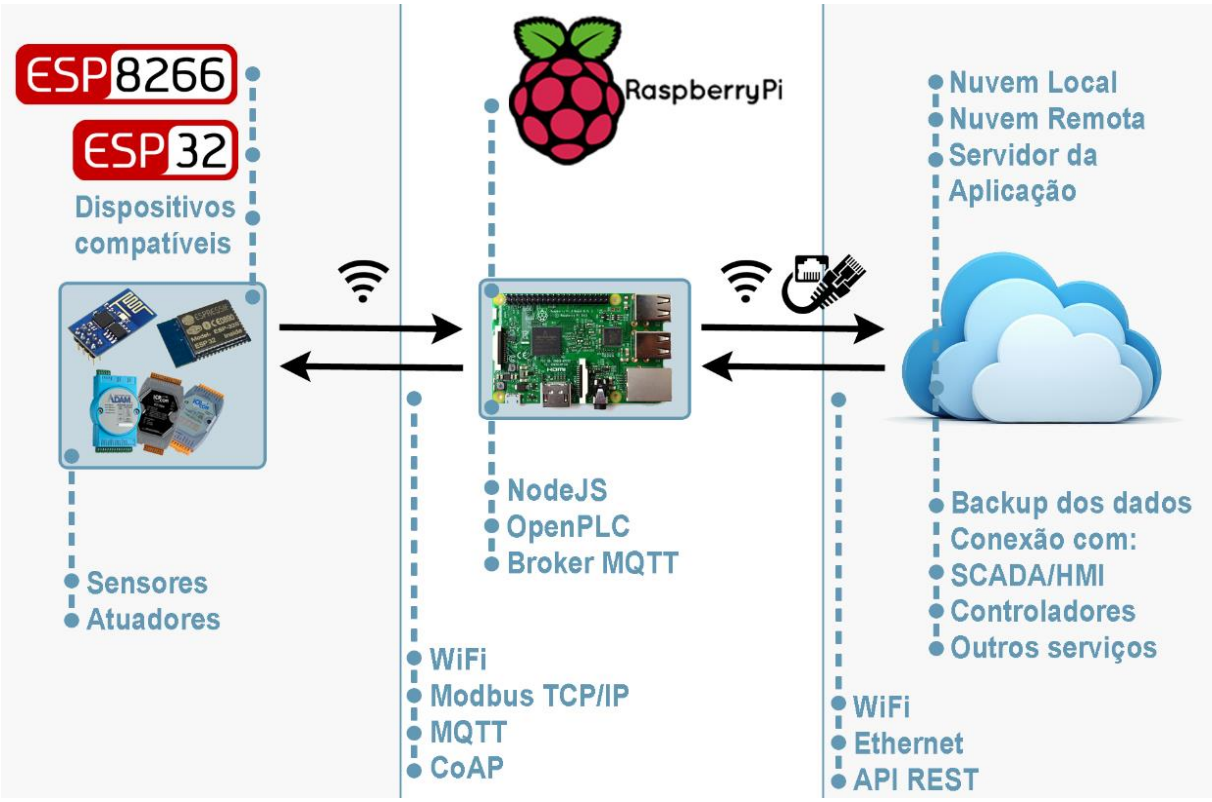


Fonte: Autoria própria.

A implementação desse modelo viabiliza a aplicação dos conceitos de IoT na indústria, como a utilização de redes sem fio, protocolos de comunicação específicos da IoT,

comunicação e acesso remoto, além de um meio diversificado de acesso e visualização dos dados da planta. A Figura 11 mostra mais detalhes da integração entre os módulos propostos neste modelo de aplicação da IIoT.

Figura 11 - Arquitetura proposta para o Modelo de Aplicação da IIoT.



Fonte: Autoria própria.

Os dispositivos que compõe a arquitetura do modelo proposto para a aplicação da IIoT (Figura 11) são: o servidor da aplicação (módulo central); o módulo gateway; os módulos remotos. É importante esclarecer que esta dissertação se concentra no desenvolvimento dos módulos remotos e gateway. Para a parte relacionada ao módulo central, somente são apresentadas as propostas para futuro desenvolvimento.

Toda comunicação realizada pela arquitetura tem como base os protocolos Ethernet, WiFi (IEEE 802.11b, g, n) juntamente com o TCP e UDP/IP, o que fornece grande interconectividade para a solução. A comunicação entre o módulo gateway e o módulo central pode ser feita tanto de forma cabeada (Ethernet) quanto sem fio (WiFi). A comunicação entre os módulos remotos e o módulo gateway utiliza rede sem fio WiFi, proporcionando maior liberdade quanto ao layout da planta industrial e a ausência de cabos facilita a manutenção e substituição de máquinas e ferramentas. Apesar disso, é possível a conexão cabeada de um dispositivo compatível com um dos protocolos suportados, sendo necessário apenas que o dispositivo em questão e o gateway estejam conectados na mesma rede.

Os protocolos de comunicação definidos para a arquitetura são: Modbus TCP/IP, proporcionando conectividade com equipamentos que utilizam esse protocolo de Ethernet Industrial (equipamentos industriais como CLPs), MQTT e CoAP para conexão com dispositivos da IoT (sensores, atuadores e dispositivos mais simples) e protocolos da Internet como HTTP para transmissão dos dados presentes no gateway. Em se tratando do lado do ambiente de TI e Internet, definiu-se a utilização do protocolo HTTP sobre o Ethernet TCP/IP. Para que esses dados sejam acessíveis para aplicações de TI e Web de forma padronizada e interoperável, utilizou o padrão REST no formato JSON.

O servidor da aplicação (módulo central) é representado por um computador ou PC industrial, podendo fazer parte de uma nuvem local (industrial), rodando uma distribuição Linux. Para a criação da nuvem local, uma proposta é usar computadores em rede rodando Linux juntamente com um módulo DRBD (*Distributed Replicated Block Device*) para disponibilizar um cluster de dispositivos de armazenamento distribuídos, gerenciados automaticamente e proporcionando alta disponibilidade para a solução (REISNER, 2002).

Dependendo do processo e requisitos onde seria utilizado o modelo para aplicação da IIoT proposto, na parte relacionada ao servidor da aplicação, seria possível simplesmente haver um aplicativo dashboard (*front-end*) para visualização dos dados no módulo gateway via comunicação REST. Adicionalmente seria possível a comunicação entre o módulo gateway e uma nuvem comercial (*Amazon Web Services, IBM Bluemix, Microsoft Azzure* e etc) usando o protocolo MQTT. Para isso, a aplicação na nuvem comercial deve operar como um *subscriber* dos dados disponibilizados no broker MQTT do gateway.

O módulo gateway é representado por um computador de placa única (Raspberry PI 3, Cubieboard ou a Beaglebone) rodando uma distribuição de Linux embarcado. O módulo gateway roda aplicações em ambiente Node.js, como broker MQTT, clientes CoAP e Modbus TCP/IP, além de um banco de dados MySQL para armazenamento das informações coletadas.

O módulo gateway é responsável por gerenciar a comunicação com os demais módulos remotos e possibilitar a interface com o mundo externo. O módulo gateway também hospeda o servidor do OpenPLC, que é uma aplicação que possibilita a programação dos módulos remotos em linguagens padronizadas pela norma IEC-61131-3 (Ladder - LD, Diagrama de Blocos Funcionais - FBD, Sequenciamento Gráfico de Funções – SFC, Lista de Instruções – IL e Texto Estruturado – STL). A programação é realizada através de um aplicativo chamado PLCOpen Editor. A programação é enviada ao sistema através de um upload feito em uma página Web, hospedada e provida pelo próprio sistema embarcado do módulo gateway.

Os módulos remotos funcionam como uma interface de I/O com o mundo externo, através dos sinais de entrada e saída do sistema. Podem ser adicionados quantos módulos forem

necessários e como a comunicação é sem fio, eles podem estar distribuídos pela linha de produção ou malha de processo. Estes módulos são representados por hardware livre de baixo custo (ESP8266 ou ESP32) ou equipamentos industriais (CLP, PAC, SCADA, SDCD ou sistemas de supervisão e gerenciamento) que se comunicam com o módulo gateway usando um dos protocolos de comunicação suportados (Modbus TCP/IP, MQTT e CoAP).

Em se tratando do uso do hardware livre ESP32 ou ESP8266, uma solução remota de programação e atualização de firmware OTA (*Over The Air programming*) via comunicação sem fio, proporcionará flexibilidade para o modelo proposto. Dessa forma, o mesmo hardware poderá ser configurado e atualizado online com o firmware padrão requerido conforme o protocolo de comunicação escolhido (Modbus TCP/IP, MQTT ou CoAP).

4.2 Ferramentas Utilizadas

4.2.1 Hardware

Um SBC (*Single Board Computer*, inglês de Computador de Placa Única) é um computador completo construído sobre uma única placa de circuito eletrônico, com microprocessador, memória, I/O (portas de entrada e saída) e outras especificações necessárias para o funcionamento de um computador (ATWELL, 2013). Existem diversos hardwares *Single Boards* disponíveis no mercado, como computadores (SBC), microprocessadores e microcontroladores que podem ser usados para IoT aplicada a Indústria 4.0. Existem também diversos projetos open hardware cujo objetivo é o de substituir um CLP através da utilização de um microprocessador ou microcontrolador com a mesma funcionalidade (GOMES, 2014).

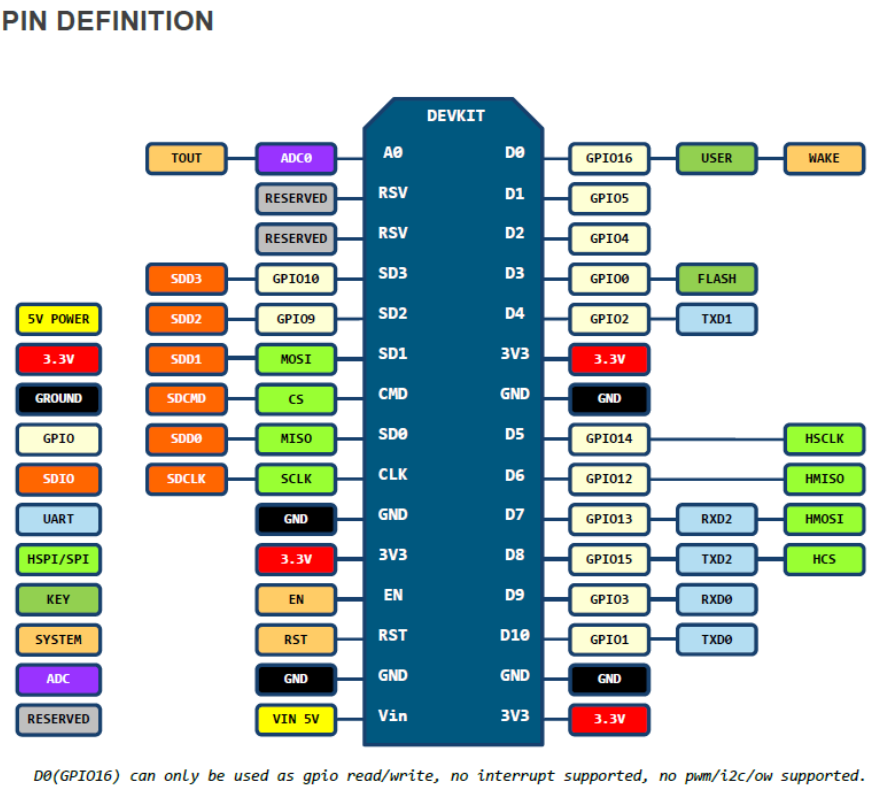
Atwell (2013) apresenta um estudo sobre diversas soluções de hardware livre SBC, entre as quais destacam-se: o Raspberry PI, que é um computador de placa única com conexões populares de periféricos. O hardware mais recente é baseado num processador ARM de 64 bits e 1GB de memória RAM, com suporte a diversos sistemas operacionais como o Linux, Android ou Windows e podendo ser programado usando linguagens de programação como o Python e C++. O projeto não inclui uma memória não-volátil (memória de armazenamento), mas possui uma entrada de cartão SD para armazenamento de dados. Para este projeto, foi escolhido o Raspberry PI 3B como hardware para o módulo gateway. Utilizando o processador BCM2837 da *Broadcom*, que é um processador de quatro núcleos com arquitetura ARM Cortex A53 (ARMv8) com clock de 1.2GHz, foi obtida uma performance cerca de 50% melhor do que o Raspberry PI 2, que utiliza um processador ARMv7 a 900MHz (BCM2837, 2018). Além disso, possui WiFi e Bluetooth LE de forma nativa, graças ao chip BCM43143, quarenta pinos de I/O, quatro portas USB 2.0, um conector HDMI, um conector DSI para conexão de display com

tecnologia *touch screen*, um conector de quatro polos para saída de áudio estéreo e vídeo composto, um conector CSI para câmera e conector para cartão SD.

O Arduino é uma plataforma de prototipagem eletrônica de hardware livre e pode ser programado através de uma IDE gratuita com linguagem de programação muito parecida com C/C++. O ESP8266 e o ESP32 são microcontroladores da fabricante chinesa *Espressif* com suporte nativo à comunicação por redes sem fio. Estes hardwares surgem como uma alternativa de desenvolvimento ao Arduino, com a vantagem de possuir conectividade sem fio, suporte à programação usando a IDE do Arduino. Além disso, esse hardware tornou-se popular devido ao seu preço, inferior a 10 dólares, valor semelhante ao de um microcontrolador sem interface de rede (ESP8266, 2017).

Devido à sua conectividade nativa com redes sem fio, foi escolhido para esse projeto como hardware para os módulos remotos o ESP8266, sendo utilizada a placa de desenvolvimento NodeMCU, na Figura 12, que já conta com periféricos como reguladores de tensão e conversor USB para Serial, utilizado para programação do módulo. O ESP8266 possui um microcontrolador de 32 bits da *Tensilica* com clock de 80MHz, 17 pinos de I/O digital e uma entrada analógica (utiliza um conversor AD) e suporta os modos cliente e ponto de acesso WiFi, sendo possível que ambos os modos sejam utilizados simultaneamente (ESPRESSIF, 2017a; ESPRESSIF, 2017b).

Figura 12 - Mapeamento dos pinos de entrada e saída da placa ESP8266 modelo NodeMCU.



Fonte: NodeMCU Devkit (2018).

A placa de desenvolvimento NodeMCU possui um chip de memória flash embutido, que utiliza até seis pinos de I/O do ESP8266 para comunicação dependendo do modo utilizado pela memória, que pode ser DIO (Dual I/O) ou QIO (Quad I/O). Caso seja configurada no modo DIO, dois pinos tornam-se disponíveis. Caso contrário, apenas 11 pinos de I/O estão disponíveis para uso (NODEMCU DEVKIT, 2018). A memória flash é utilizada para armazenar arquivos de sistema responsáveis pelo funcionamento do WiFi, o programa que será executado na placa e pode ser utilizada para armazenar arquivos adicionais de configuração ou até mesmo conteúdo para ser fornecido por um servidor web. A placa utilizada nesse projeto tem uma memória flash de 4Mb, sendo que 3Mb podem ser utilizados para armazenar arquivos e possui a pinagem mostrada na Figura 12.

4.2.2 Software

4.2.2.1 Node.js

JavaScript é uma linguagem de programação interpretada muito utilizada no *front-end*, que trata da criação de interfaces de sites da Internet e seus comportamentos (CANTELON et al., 2014). Essa linguagem é executada diretamente no navegador do cliente, sendo necessário um interpretador embutido no navegador. No Google Chrome, a responsável por executar o JavaScript é a *engine* (ou interpretador JavaScript) V8, que é open source (CHROME V8, 2017).

A adoção do JavaScript no *front-end* das aplicações *web* se deve a uma evolução natural dessas aplicações. Os primeiros *web sites* eram compostos por uma coleção de páginas HTML e arquivos estáticos, como texto e imagens. Linguagens interpretadas e executadas no servidor, como o PHP ou o ASP, tornaram os sites dinâmicos. Com a introdução do JavaScript em 1996, a criação de sites dinâmicos utilizando uma linguagem de scripts interpretada no navegador do cliente tornou-se viável. A popularização do JavaScript ocorreu após o surgimento de tecnologias como o Ajax, jQuery e HTML5. (FINK & IDO, 2014)

O JavaScript permite que uma função seja atribuída a uma variável e, até mesmo, que seja passada como parâmetro para outra função. Isso só é possível, pois no JavaScript funções são tratadas como "*first-class objects*". Essa característica do JavaScript é a base para a programação baseada em eventos. Quando uma condição é satisfeita em uma função, outra função pode ser chamada para tratar aquela condição (JUNIOR, 2012).

Apesar de sua aplicação inicial ter sido no *front-end*, o JavaScript é utilizado atualmente para desenvolvimento de aplicações para dispositivos móveis, *desktop* (para Windows, Linux ou Mac) e até mesmo em servidores web (JAIMINI & DHANIWALA, 2016).

A grande inovação do Node.js é utilizar um interpretador, baseado no V8 do Chrome, para executar JavaScript sem a necessidade de um navegador, particularmente em servidores web. A programação baseada em eventos, com métodos assíncronos, torna o servidor mais eficiente e altamente escalável, sendo possível tratar as requisições no próprio programa, ao invés de delegar essa responsabilidade ao sistema operacional. Essa eficiência pode ser notada em operações que costumam ser relativamente lentas, como por exemplo em uma operação de acesso ao disco. Neste cenário, o programa não precisa aguardar uma resposta do sistema de arquivos, pois quando o acesso é obtido, uma função é chamada para tratar aquele evento. Enquanto isso, o programa pode atender a outras requisições e executar outras tarefas (HUGHES-CROUCHER & WILSON, 2012; FLANAGAN, 2011).

Existe um acrônimo para esse tipo de aplicação em que o Node.js possui vantagem: DIRT (*Data-Intensive Real-Time application* – Aplicação de tempo real e intensiva em dados). Como o Node.js é eficiente nessas aplicações que envolvem I/O, ele é utilizado em diversas aplicações como proxy ou gateway, funcionando como um intermediário ou concentrador na transmissão de dados. Ele permite que o servidor mantenha diversas conexões abertas, enquanto trabalha com várias requisições, sem causar um grande impacto no uso da memória RAM (CANTELON et al., 2014).

4.2.2.2 OpenPLC

Para pequenas indústrias e países menos desenvolvidos, controladores industriais ainda são vistos como dispositivos de alto custo. Além disso, as empresas não fornecem informações detalhadas sobre como esses controladores trabalham internamente, pois são todos de código fechado (ALVES et al., 2014). O OpenPLC é um projeto de integração de software e hardware criado para quebrar esse paradigma. É open source e pode ser utilizado com sistemas de hardware aberto de custo reduzido como Arduino, ESP8266 e Raspberry PI (ALVES et al., 2014).

O projeto OpenPLC segue as premissas da norma IEC 611312 e da PLCOpen XML3, permitindo a programação do hardware livre de acordo com o padrão IEC 61131-3, o qual define cinco linguagens padrão para programação de CLPs:

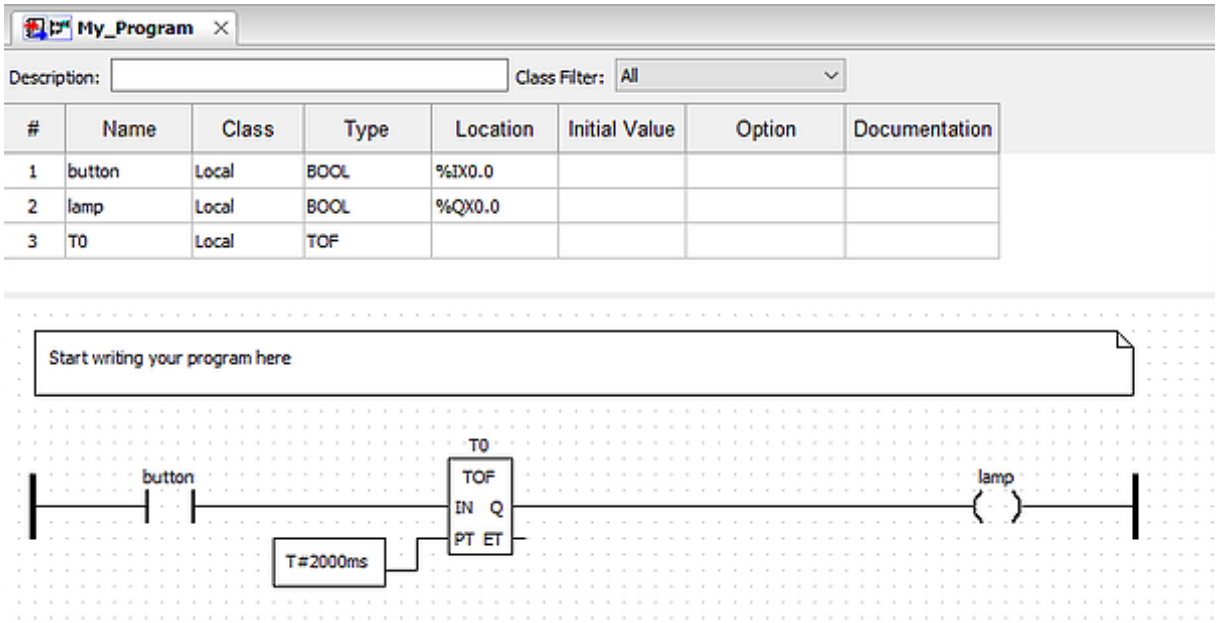
- Diagrama de Blocos Funcionais (*FBD - Function Block Diagram*);
- Diagrama Ladder (*LD - Ladder Diagram*);

[2] IEC 61131 é um padrão internacional para controladores programáveis industriais, contemplando o projeto completo desde hardware, instalação, testes, documentação, programação e comunicação.
[3] PLCOpen XML é um padrão aberto e não proprietário para especificação em formato XML de programas de controladores programáveis padronizados pela IEC 61131-3, para integração com outras ferramentas.

- Texto Estruturado (*STL - Structured Text*);
- Lista de Instruções (*IL - Instruction List*);
- Sequenciamento Gráfico de Funções (*SFC - Sequential Function Chart*);

A programação em uma das linguagens do padrão IEC 61131-3 é feita no *PLCOpen Editor*, mostrado na Figura 13, um software de código aberto que faz parte do projeto Beremiz (BEREMIZ, 2017). Posteriormente, esse programa é convertido para Texto Estruturado pelo *PLCOpen Editor*, sendo gerado um arquivo com extensão "st", que deve ser enviado para ser executado pelo servidor do OpenPLC através de uma interface Web.

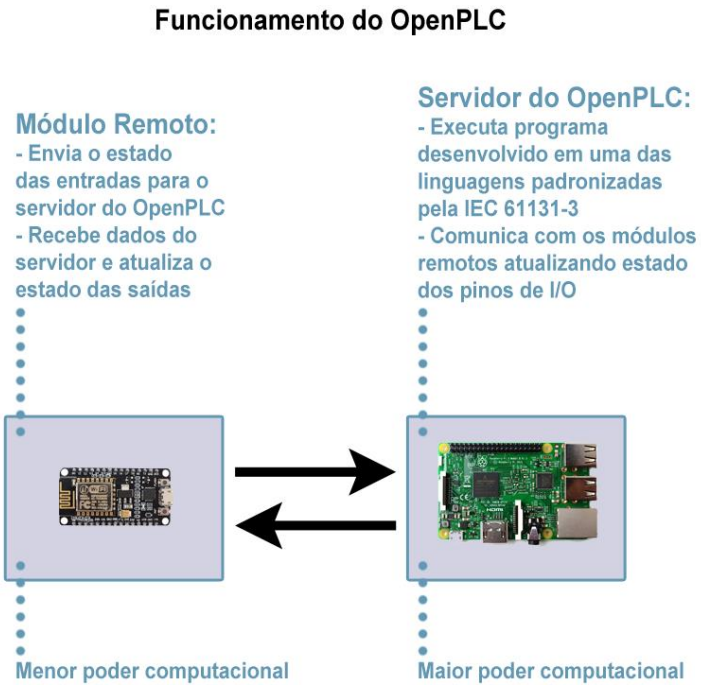
Figura 13 - Programação em Ladder, feita utilizando o PLCOpen Editor.



Fonte: OpenPLC (2017).

O servidor do OpenPLC roda sobre o ambiente Node.js, sendo necessário um sistema operacional como Windows ou Linux para sua execução. Para a utilização de hardware mais simples com o OpenPLC, o servidor deve ser executado remotamente e se comunicar com o hardware pela rede local. Além disso, o hardware deve ser programado com um firmware padrão, fornecido no site do Projeto OpenPLC (OPENPLC, 2017). Conforme o programa gerado pelo *PLCOpen Editor* é executado no servidor do OpenPLC, o estado atualizado dos pinos de entrada e saída é transmitido para o hardware remoto. A Figura 14 ilustra esse cenário.

Figura 14 – Funcionamento do OpenPLC.



Fonte: Autoria própria.

O servidor do OpenPLC exige um hardware com poder computacional o suficiente para rodar um sistema operacional completo. Entretanto, um hardware com menor poder computacional pode ser utilizado para se comunicar com o servidor do OpenPLC remotamente e aumentar a quantidade de pinos de I/O disponíveis.

4.2.2.3 Broker MQTT

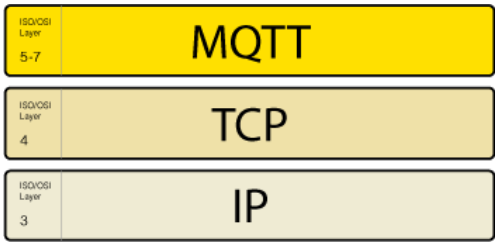
O protocolo MQTT fornece um método leve de transmitir mensagens usando o modelo de *publish/subscribe*. Isso o torna adequado para sensores de baixa potência ou dispositivos móveis, como smartphones, sistemas embarcados como o Raspberry PI ou microcontroladores como o Arduino (MOSQUITTO, 2017).

O broker é o coração de qualquer protocolo que implementa o padrão *publish/subscribe*. Dependendo de sua configuração, o broker pode gerenciar centenas de clientes (*subscribers*) simultâneos. A função primária do broker é receber todas as mensagens, filtrá-las com base nos tópicos definidos pelo publicador e encaminhar as mensagens para os clientes que estão interessados em cada tópico. Além disso, o broker deve gerenciar as conexões de todos os clientes, inclusive em relação à autenticação e autorização (HIVEMQ, 2017).

Mosca é um broker de código aberto, escrito sobre o Node.js, que implementa o protocolo MQTT versões 3.1 e 3.1.1. Pode ser utilizado tanto como uma aplicação isolada, como embutido em outra aplicação escrita com o Node.js (MOSCA, 2017).

A conexão através do MQTT é intermediada sempre pelo broker, nunca um cliente se conectando diretamente a outro cliente. O MQTT depende do protocolo TCP/IP, sendo necessário que o cliente e o broker tenham uma pilha TCP/IP implementada, conforme mostrado na Figura 15.

Figura 15 - MQTT é implementado sobre TCP/IP.



Fonte: HiveMQ (2017).

O desenvolvimento dos dispositivos remotos e do *gateway*, integrando o broker MQTT, o OpenPLC e um *web service* criado com o NodeJS, é descrito no próximo capítulo.

5 DESENVOLVIMENTO

5.1 Implementação dos Dispositivos Remotos

Os dispositivos remotos do modelo proposto para aplicação da IIoT podem ser dispositivos de mercado que se comuniquem através de um dos protocolos de comunicação suportados, como por exemplo os módulos de aquisição de dados da Advantech com comunicação Modbus TCP/IP ou MQTT (ADVANTECH, 2018), ou podem ser os módulos remotos desenvolvidos nesse projeto. Para a implementação dos módulos remotos foram utilizadas placas ESP8266 modelo NodeMCU, que representam interfaces prontas para a prototipagem do sistema. Foi utilizada a IDE do Arduino para programação das placas ESP8266 com um código customizado, desenvolvido para possibilitar a comunicação com o gateway em uma das quatro formas de comunicação suportadas:

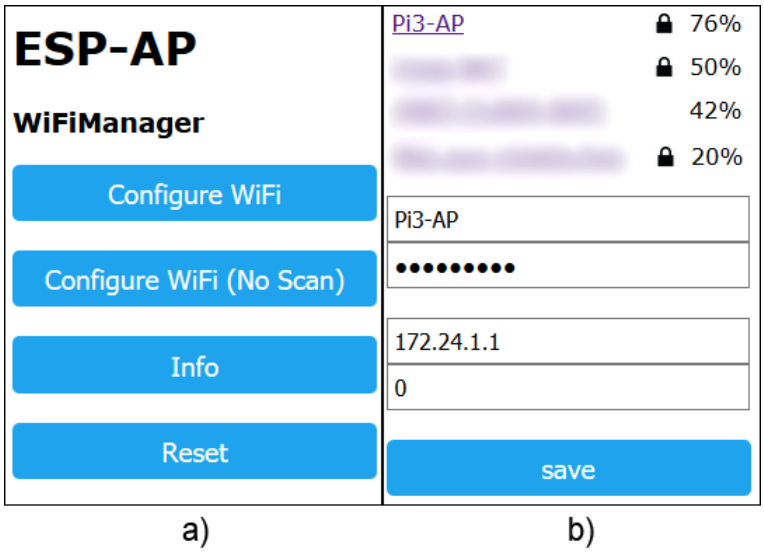
- Modbus TCP/IP;
- MQTT;
- CoAP;
- OpenPLC.

O código necessário para a comunicação com o servidor do OpenPLC é disponibilizado no próprio site do projeto OpenPLC. Foram necessárias pequenas alterações nesse código padrão fornecido no site, pois existem alguns parâmetros que precisam ser editados, como nome e senha da rede WiFi, endereço IP do servidor do OpenPLC e ID do dispositivo, sendo esse último parâmetro o responsável por permitir que várias placas ESP8266 sejam utilizadas simultaneamente.

Para evitar que um novo código precise ser compilado a cada alteração de um desses parâmetros, foi realizada uma customização, que torna possível a utilização do mesmo código em diversas placas, sendo a diferenciação das placas feita através de uma tela de configuração, conforme mostrado na Figura 16. Isso foi feito utilizando a biblioteca WiFi-Manager (WIFIMANAGER, 2018), que faz com que as placas ESP8266 funcionem temporariamente como ponto de acesso, criando uma página web do tipo *captive portal* que é exibida para quem se conectar ao dispositivo.

Deste modo, é possível se conectar às placas, acessar esse portal e configurar os parâmetros dos módulos remotos, na tela de configuração (Figura 16b) antes da execução do programa principal. Na tela de configuração é realizado: Seleção da rede WiFi a se conectar, configuração do endereço IP do servidor do OpenPLC e ID do dispositivo (necessário para o OpenPLC diferenciar as placas).

Figura 16 - Página Web para configuração dos Módulos Remotos: a) Tela inicial; b) Tela de Configuração



Fonte: Autoria própria.

Utilizando o mesmo princípio, onde uma tela de configuração precede a execução do programa principal, foram criados firmwares padrão para cada um dos protocolos suportados. A tela de configuração nesses casos não se limita apenas a parâmetros como SSID e senha da rede WiFi, também possibilita a configuração dos pinos de I/O da placa como entrada ou saída. Graças a isso, qualquer placa NodeMCU pode ser programada com o firmware padrão sem a necessidade de adaptações do código.

Para a comunicação através do protocolo CoAP, foi utilizada a biblioteca ESP-CoAP (ESP-COAP, 2018). Porém, a biblioteca utilizada ainda possui alguns itens que não foram implementados por completo, como as transferências *block-wise*. Isso impacta diretamente na funcionalidade de descoberta de recursos dos dispositivos, causando perda de informações caso existam muitos recursos a serem transmitidos como resposta a uma requisição de descoberta. Além disso, existem limitações quanto ao uso dos métodos POST e DELETE.

A comunicação utilizando o protocolo MQTT foi feita através da biblioteca *PubSubClient* (PUBSUBCLIENT, 2018), desenvolvida para o Arduino. Ela também possui algumas limitações, como apenas publicar mensagens com QoS 0 e possibilitar a inscrição apenas em tópicos com QoS 0 ou 1. O firmware desenvolvido permite que na tela de configuração seja realizada a diferenciação de cada I/O como entrada ou saída. Caso determinado I/O seja configurado como entrada, é realizada uma leitura de seu nível lógico e esse valor é publicado a cada segundo em um tópico específico identificado pelo endereço MAC do dispositivo seguido pelo endereço do I/O (como por exemplo: 18:FE:34:D4:4E:30/D0). Se um pino de I/O for configurado como saída, o dispositivo se inscreve em um tópico identificado pelo endereço MAC do dispositivo seguido pelo endereço do I/O. Os valores booleanos publicados no tópico

são recebidos pelo dispositivo e o nível lógico do pino de I/O especificado é atualizado com base nesses valores.

No caso da comunicação através do protocolo Modbus TCP/IP, foi desenvolvido um código próprio para esse projeto, proporcionando maior flexibilidade quanto as funcionalidades do protocolo. Um exemplo disso é que não foi feita diferenciação entre *coil* e *discrete input*, com a finalidade de facilitar a implementação.

5.2 Implementação do Gateway

As principais funcionalidades do módulo gateway são: a aquisição de dados de módulos remotos, armazenamento desses dados em um banco de dados e a disponibilização desses dados de uma forma padronizada para serem consumidos por outras aplicações ou dispositivos. Foi utilizada uma Raspberry PI 3 como hardware do módulo gateway, por já possuir WiFi de forma nativa e possibilitar a criação de um ponto de acesso para conexão dos módulos remotos através de funcionalidade nativa do Linux, não sendo necessário nenhum hardware adicional para prover essa conectividade.

O sistema operacional utilizado foi o *Raspbian Stretch*, uma distribuição Linux baseada no Debian 9. O Node.js já vem instalado por padrão, sendo necessário apenas instalar o servidor do OpenPLC e o MySQL (MYSQL, 2018). Foi desenvolvido um programa para implementar as funcionalidades de aquisição de dados dos módulos remotos, armazenamento desses dados em um banco de dados e disponibilização desses dados de uma forma padronizada, que roda sobre o Node.js. Esse programa cria uma interface entre módulos remotos funcionando como sensores/atuadores e consumidores de dados, através de uma API REST, criando uma forma padronizada de acesso aos dados e uma abstração dos protocolos utilizados.

Para isso, foi utilizado o módulo *Express* (EXPRESS, 2018) do Node.js para criar um servidor Web contendo a API REST. Foram utilizados os módulos *jsModbus* (JSMODBUS, 2018) para comunicação com módulos remotos através do protocolo Modbus TCP/IP, node-coap (NODE-COAP, 2018) para comunicação utilizando o protocolo CoAP e o Mosca (MOSCA, 2017), funcionando como broker MQTT e possibilitando o acesso aos dados. Os dados dos módulos remotos são coletados e armazenados num banco de dados MySQL. Utilizou-se o Sequelize (SEQUELIZE, 2018) para facilitar o desenvolvimento da comunicação com o banco de dados, de modo que a sintaxe específica para consulta e escrita no banco de dados fosse abstraída e, em seu lugar, utilizada a linguagem JavaScript.

O banco de dados está organizado em tabelas que se relacionam. Possui uma tabela de dispositivos Modbus TCP/IP, contendo as informações passadas no momento do cadastro do dispositivo a ser monitorado, como ID do dispositivo, endereço IP, porta de comunicação e

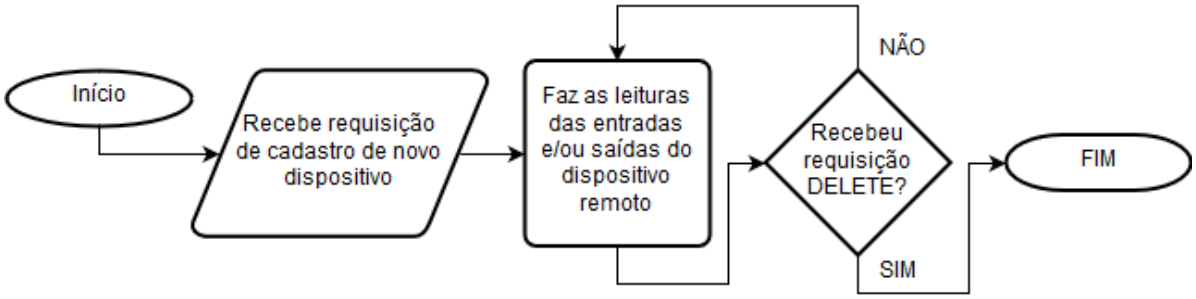
período de atualização. A tabela de dispositivos Modbus TCP/IP está relacionada com tabelas específicas para cada tipo de I/O de um dispositivo, como *coil*, *discrete input*, *holding register* e *input register*. Essas tabelas possuem o ID do dispositivo, utilizado para o relacionamento com a tabela de dispositivos, o endereço de I/O, o valor daquele I/O e a estampa de tempo indicando o momento da aquisição daquele valor.

Os tópicos MQTT cadastrados para serem monitorados são armazenados em uma tabela, que é consultada toda vez que algum dado é publicado na rede. Caso o tópico em que o dado foi publicado esteja nessa tabela, o tópico, a mensagem publicada e o *timestamp* são armazenados em uma outra tabela, relacionada com a tabela de tópicos pelo id do tópico.

Os dados de dispositivos que se comunicam através do protocolo CoAP, como endereço IP, porta de comunicação e período de atualização ficam armazenados em outra tabela. Essa tabela de dispositivos CoAP se relaciona com a tabela de *endpoints* através do ID do dispositivo. A tabela de *endpoints* contém as rotas para os recursos do dispositivo CoAP cadastrado, além de armazenar o valor, o *timestamp* e a descrição de cada recurso.

Para os protocolos de comunicação Modbus TCP/IP e CoAP, que funcionam no modelo requisição/resposta, o consumidor da API REST deve enviar uma requisição para o gateway contendo um JSON informando os dados do dispositivo que deve ser monitorado. Estes dados incluem o endereço IP do dispositivo, a porta de comunicação e o período de atualização, que é o espaço de tempo entre cada requisição ao dispositivo. O gateway atribui um identificador numérico (ID) ao dispositivo, que é enviado na resposta e começa o ciclo de requisições ao dispositivo, conforme pode ser visto representado no fluxograma da Figura 17.

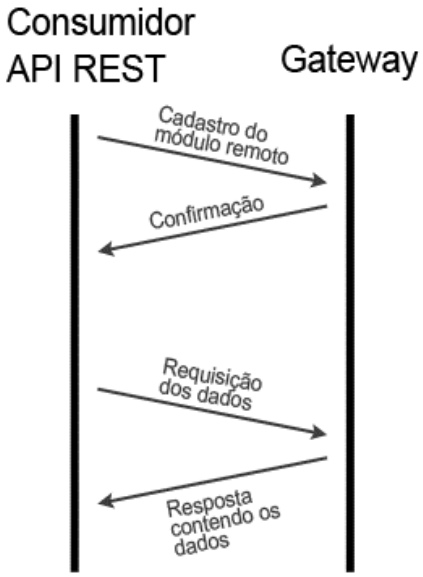
Figura 17 - Fluxograma de funcionamento do gateway quando um novo dispositivo é cadastrado.



Fonte: Autoria própria.

Para a obtenção dos dados de um dispositivo monitorado, envia-se através da API REST uma requisição informando o ID do dispositivo, para que seja realizada uma busca no banco de dados. Detalhes do procedimento e troca de informações dessa comunicação com o gateway através da API REST pode ser vista na Figura 18.

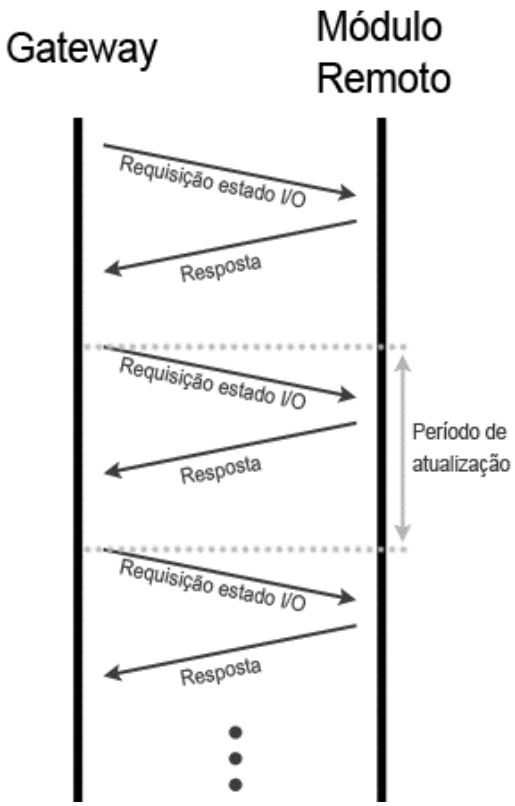
Figura 18 - Comunicação entre consumidor da API REST e gateway para dados monitorados via Modbus TCP/IP ou CoAP



Fonte: Autoria própria.

A comunicação do gateway com os módulos remotos se dá de forma cíclica, utilizando o modelo requisição/resposta, conforme ilustrado na Figura 19. O intervalo entre cada requisição é determinado pelo período de atualização, informado ao gateway na requisição de cadastro de cada dispositivo.

Figura 19 - Comunicação entre gateway e módulo remoto utilizando Modbus TCP/IP ou CoAP

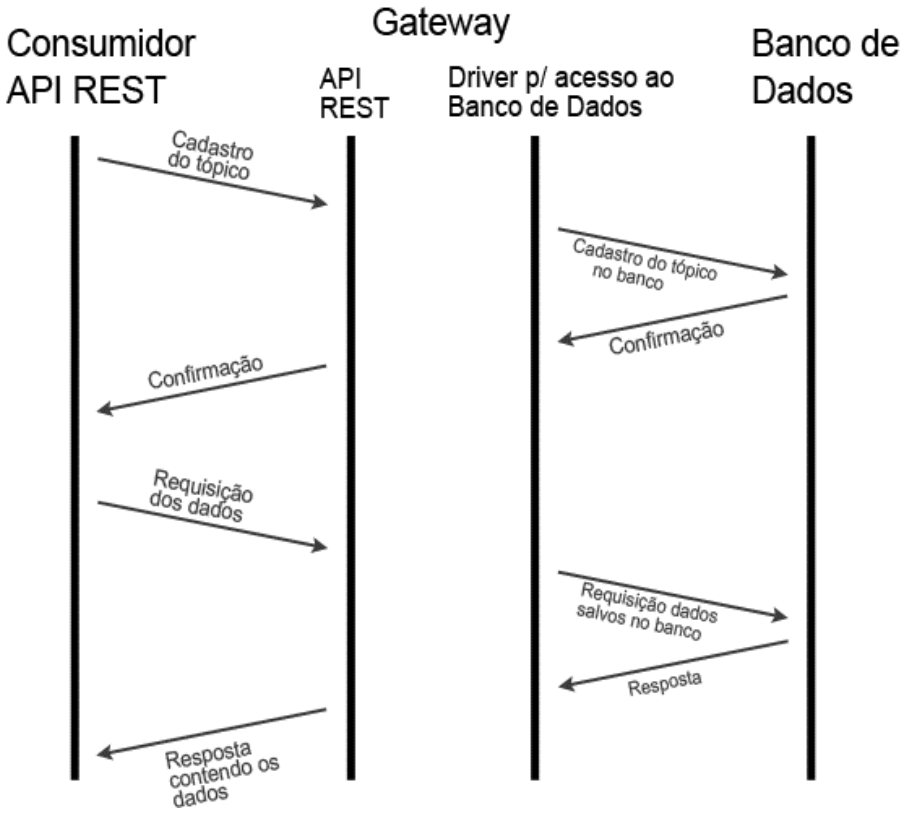


Fonte: Autoria própria.

No caso do protocolo MQTT, não há requisição direta aos módulos remotos como na Figura 19, sendo a comunicação centralizada no broker. Como o broker utilizado é o Mosca, que pode ser incluído como um módulo do programa, a informação é acessada diretamente por ele. Portanto, a comunicação entre o consumidor da API REST e o gateway acontece em duas etapas, conforme mostrado na Figura 20.

Na primeira etapa, o cliente da API REST deve enviar uma requisição indicando o tópico MQTT de interesse a ser monitorado. Deste modo, toda vez que houverem dados publicados nesse tópico, estes dados serão armazenados na memória e posteriormente no banco de dados. Na segunda etapa, quando houver uma requisição buscando dados publicados em determinado tópico MQTT, os dados armazenados no banco de dados do gateway serão retornados ao cliente da API REST, conforme exibido na Figura 20.

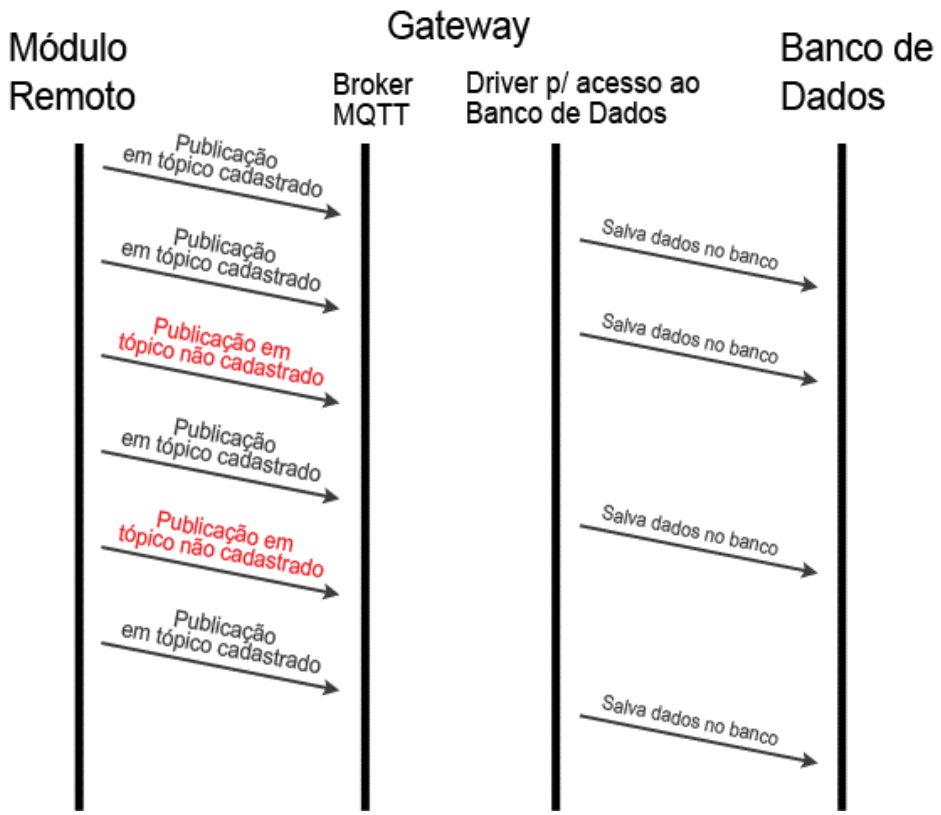
Figura 20 – Comunicação entre consumidor da API REST e gateway para dados monitorados via MQTT



Fonte: Autoria própria.

O gateway somente armazena no banco de dados, os dados relativos a tópicos MQTT previamente cadastrados no broker via API REST (informação de tópicos de interesse), conforme mostrado na Figura 21.

Figura 21 – Armazenamento no banco de dados do gateway de tópicos MQTT cadastrados via API REST



Fonte: Autoria própria.

5.3 Implementação da API REST

A API REST foi implementada de modo a distinguir os protocolos utilizados para se comunicar com os módulos remotos diretamente na URL da requisição. Para o Modbus TCP/IP, por exemplo, a URL se inicia com o IP ou endereço do servidor (gateway) seguido de "/api/modbus/", enquanto que para acesso de dados provenientes da comunicação por MQTT a URL que deve ser acessada é composta pelo endereço do servidor seguido de "/api/mqtt/", regra que se repete para a comunicação utilizando o protocolo CoAP e qualquer outro protocolo que venha a ser implementado.

5.3.1 Modbus TCP/IP

No caso da comunicação com dispositivos remotos Modbus TCP/IP, o módulo gateway se comunica de forma cíclica com os módulos remotos após uma configuração inicial. Estes dados ficam armazenados no módulo gateway para que possam ser consultados através da API REST. A comunicação com os dispositivos remotos ocorre da seguinte maneira:

1. O cliente que está consumindo os dados pela API REST deve enviar uma requisição ao gateway para cadastrar o dispositivo remoto no banco de dados. Essa requisição deve conter

dados como o endereço IP do dispositivo e a porta utilizada, além do período de atualização e os pinos de entrada ou saída a serem monitorados, de modo que o gateway saiba para onde enviar as requisições Modbus TCP/IP para a leitura e escrita no dispositivo e qual o intervalo de tempo entre as requisições;

2. Após cadastrado, o gateway enviará requisições de leitura de modo cíclico para o dispositivo remoto com o período entre cada requisição estipulado no momento do cadastro do dispositivo;

3. Os dados obtidos pelo gateway são salvos na memória e, posteriormente, salvos num banco de dados, com *timestamp*[4], endereço do I/O e valor;

4. Esses dados podem ser acessados através da mesma API REST, acessando uma URL específica para a quantidade e o tipo de dados que se deseja obter.

Essa comunicação com a API REST ocorre utilizando os métodos HTTP e troca de arquivos JSON entre o cliente e o gateway. Em mais detalhes, para cadastro de um novo dispositivo remoto Modbus TCP/IP, deve-se enviar uma requisição utilizando o método POST para a URL "ip-do-gateway:porta/api/modbus/". Essa requisição deve conter em seu corpo um JSON contendo o endereço IP do dispositivo remoto, a porta utilizada pelo protocolo Modbus TCP/IP, uma flag indicando se a conexão deve ser do tipo keep alive ou não, o período de atualização em segundos, uma descrição opcional e os endereços de memória ou pinos de entrada ou saída que devem ser monitorados pelo gateway, conforme Tabela 3.

Esses endereços ou pinos de entrada ou saída são passados como um array de objetos (Tabela 4), onde cada item do array contém um endereço inicial e a quantidade de endereços que deve ser monitorada, sendo possível informar ao gateway que ele deve monitorar diversas faixas de endereços de um determinado tipo de entrada ou saída.

Tabela 3 - Atributos e formato esperados do JSON enviado na requisição para cadastro de um novo dispositivo.

Atributo	**Formato do dado**	**Opcional**	**Descrição**
host	Endereço IP	Não	Endereço IP do dispositivo
port	Número inteiro	Não	Porta de comunicação do dispositivo
keep_alive	Valor booleano	Não	Conexão com keep alive (true/false)
update	Número inteiro	Não	Valor em segundos relativo ao período de atualização dos dados
descricao	Texto	Não (deixar em branco se não houver)	Descrição do dispositivo em texto
coil, discrete_input, holding_register e/ou input_register	Array de objetos	Sim	Array com um ou mais objetos, que descrevem os intervalos a serem monitorados (Tabela 4)

Fonte: Autoria própria.

[4] Timestamp ou estampa de tempo é uma cadeia de caracteres denotando a hora ou data que certo evento ocorreu. A cadeia é geralmente apresentada num formato consistente, permitindo comparação entre duas marcas temporais distintas. Elas são padronizadas pela Organização Internacional para Padronização (ISO) através da ISO 8601.

Tabela 4 - Atributos e formato dos objetos que descrevem as faixas de endereço de I/O a serem monitoradas.

Atributo	**Formato do dado**	**Opcional**	**Descrição**
address	Número inteiro	Não	Endereço inicial de uma faixa de endereços a ser monitorada
quantity	Número inteiro	Não	Quantidade de endereços a serem monitorados (endereços consecutivos ao endereço inicial)

Fonte: Autoria própria.

Após receber a requisição, se não houver erros, o gateway deve responder com um JSON contendo a informação enviada na requisição com o acréscimo do ID do dispositivo remoto adicionado e as URLs que podem ser acessadas para as demais funções da API, conforme Tabela 5 e Tabela 6.

Tabela 5 - Atributos e formato do JSON de resposta à requisição de cadastro de um novo dispositivo.

Atributo	**Formato do dado**	**Opcional**	**Descrição**
dados_do_dispositivo	Objeto	Não	Contém o JSON enviado na requisição, com adição do atributo “id”, usado para identificar o dispositivo cadastrado.
links	Array de objetos	Não	Cada objeto é composto por uma URL, o método que deve ser utilizado e a descrição do resultado esperado como resposta a essa requisição, informando as possíveis interações que o cliente pode fazer através da API (Tabela 6).

Fonte: Autoria própria.

Tabela 6 - Atributos e formato dos objetos que compõe o array dos links.

Atributo	**Formato do dado**	**Opcional**	**Descrição**
href	URL	Não	URL do recurso (dispositivo).
rel	Texto	Não	Descrição da funcionalidade.
Method	Texto	Não	Método a ser utilizado na requisição.

Fonte: Autoria própria.

Para interromper a comunicação entre o gateway e um determinado dispositivo, deve-se enviar uma requisição com o método DELETE para “ip-do-gateway:porta/api/modbus/id-do-dispositivo”. A resposta do gateway deve ser “OK”, informando que a requisição foi recebida com sucesso.

Para escrita nos endereços de memória ou saídas do dispositivo remoto, é necessário enviar uma requisição para “ip-do-gateway:porta/api/modbus/id-do-dispositivo”. Essa requisição utilizando o método POST deve conter em seu corpo um JSON informando o tipo de I/O (coil ou registrador) como um atributo e seu valor deve ser um array de objetos (Tabela 7), onde cada objeto possui um endereço inicial e um array com os valores a serem escritos (Tabela 8). Deste modo é possível informar ao gateway que haverá escrita em várias faixas de endereços de I/O, separando cada faixa em um objeto do array de objetos. Não é necessário passar nenhum endereço para escrita além do endereço inicial de cada faixa. O gateway deve responder com “OK”, se nenhum erro ocorrer.

Tabela 7 - Atributos e formato do JSON enviado para escrita.

Atributo	Formato do dado	Opcional	Descrição
coil/register	Array de objetos	Sim	Cada objeto corresponde a uma faixa de endereços e os respectivos valores que serão escritos nesses endereços (Tabela 8).

Fonte: Autoria própria.

Tabela 8 - Atributos e formato do objeto que representa cada faixa de endereços para escrita.

Atributo	Formato do dado	Opcional	Descrição
Address	Número inteiro	Não	Endereço inicial para escrita.
value	Array com valores a serem escritos	Não	Sequência de valores a serem escritos, sendo que o endereço onde será escrito cada valor é tomado a partir da posição do valor dentro do array e do endereço inicial.

Fonte: Autoria própria.

Para ler os valores de um determinado tipo de entrada ou saída, deve-se enviar uma requisição com o método GET para a URL “ip-do-gateway:porta/api/modbus/id-do-dispositivo/tipo-de-entrada-ou-saída”, recebendo como resposta os valores de todos os endereços monitorados daquele tipo especificado de entrada ou saída, com seus respectivos timestamps. Os atributos e o formato do JSON recebido como resposta a essa requisição estão na Tabela 9.

Tabela 9 - Atributos e formato do JSON recebido como resposta à requisição de leitura dos dados.

Atributo	Formato do dado	Opcional	Descrição
host_id	Número inteiro	Não	ID do dispositivo.
host_ip	Endereço IP	Não	Endereço IP do dispositivo.
host_port	Número inteiro	Não	Porta usada na comunicação com o dispositivo.
update	Número inteiro	Não	Valor em segundos, relativo ao período de atualização dos dados.
keep_alive	Valor booleano	Não	Conexão com keep alive (true/false).
coil, discrete_input, holding_register ou input_register	Array de objetos	Não	Array de objetos contendo timestamp, valor e endereço dos I/Os observados (Tabela 10).

Fonte: Autoria própria.

Tabela 10 - Atributos e formato dos objetos contendo os valores obtidos no tempo.

Atributo	Formato do dado	Opcional	Descrição
timestamp	Unix Timestamp	Não	Timestamp no formato Unix Timestamp
address	Número inteiro	Não	Endereço da entrada ou saída
value	Número inteiro ou valor booleano	Não	Valor apresentado pela entrada ou saída

Fonte: Autoria própria.

Para se obter apenas os valores relativos a determinado endereço de um tipo especificado de entrada ou saída, deve-se enviar uma requisição com o método GET para “ip-do-gateway:porta/api/modbus/id-do-dispositivo/tipo-de-entrada-ou-saída/endereço-da-entrada-ou-saída”. A resposta do servidor conterá um JSON com formato semelhante ao recebido como

resposta à requisição anterior (Tabela 9), porém com dados referentes apenas ao endereço de entrada ou saída informado na URL da requisição.

Pode ser necessário limitar a quantidade de valores retornada, principalmente se o período de atualização do dispositivo for muito pequeno. Para isso, basta acrescentar mais um nível na URL da requisição, sendo a nova URL "ip-do-gateway:porta/api/modbus/id-do-dispositivo/tipo-de-entrada-ou-saída/endereço-da-entrada-ou-saída/quantidade-de-valores".

Será retornado pelo servidor um JSON com os atributos exibidos na Tabela 9, contendo um array de objetos com tamanho igual à quantidade de valores informada na URL, sendo que cada objeto do array contém endereço, valor e timestamp, conforme mostrado na Tabela 10.

5.3.2 MQTT

Como o broker MQTT foi implementado como um módulo do software desenvolvido nesse projeto, é possível acessar toda a informação que trafega por ele. Nesse caso, ao invés de cadastrar um dispositivo específico, o consumidor dos dados deve cadastrar o tópico de interesse informando ao gateway que os dados enviados a esse tópico devem ser armazenados no banco de dados para posterior consulta pela API REST.

Para cadastro de um tópico a ser monitorado, uma requisição HTTP com método POST deve ser enviada para a URL http://ip-do-gateway:porta/api/mqtt/ contendo em seu corpo um JSON com o tópico a ser monitorado no parâmetro "topic". O tópico cadastrado pode conter wildcards do MQTT, caso haja interesse em monitorar também os seus subtópicos. A Tabela 11 apresenta esse formato.

Tabela 11 – Atributo necessário para cadastro de tópico no *gateway*.

Atributo	**Formato do dado**	**Opcional**	**Descrição**
topic	Texto	Não	Tópico que será monitorado

Fonte: Autoria própria.

Caso não ocorra nenhum erro no processamento da requisição, o gateway deve responder com um JSON informando o id do tópico cadastrado, o tópico enviado na requisição e as demais URLs aceitas pela API REST relativa ao protocolo MQTT. Para visualizar todas as mensagens publicadas em um tópico cadastrado, deve-se enviar uma requisição HTTP com o método GET para http://ip-do-gateway:porta/api/mqtt/id-do-topico.

O servidor deverá responder com um JSON contendo um array de objetos, onde cada objeto é uma publicação no tópico monitorado e contém um id (relativo à publicação), um timestamp, o tópico em que foi realizada a publicação (que não precisa ser necessariamente o valor exato do tópico cadastrado caso tenha sido utilizado um wildcard, podendo ser registrada

a publicação feita nos subtópicos), o dado publicado e o id do tópico. A Tabela 12 apresenta esse formato.

Tabela 12 – Formato do objeto contendo informações publicadas em um tópico cadastrado.

Atributo	**Formato do dado**	**Opcional**	**Descrição**
id	Número inteiro	Não	ID da mensagem
timestamp	Unix Timestamp	Não	Timestamp no formato Unix Timestamp
topic	Texto	Não	Tópico em que a informação foi publicada
value	Texto	Não	Informação publicada
MQTTId	Número inteiro	Não	ID do tópico cadastrado

Fonte: Autoria própria.

É possível publicar dados em determinado tópico através da API REST, mesmo que esse tópico não tenha sido previamente cadastrado. Uma requisição HTTP com método POST deve ser enviada para http://ip-do-gateway:porta/api/mqtt/publish, contendo um JSON indicando o tópico, o dado a ser publicado, o nível de QoS e se a informação deve ser retida no tópico (uso da flag Retain). A Tabela 13 apresenta esse formato.

Tabela 13 – Atributos necessários para publicar informações em um tópico.

Atributo	**Formato do dado**	**Opcional**	**Descrição**
topic	Texto	Não	Tópico em que serão publicados os dados contidos no payload
payload	Texto	Não	Dados que serão publicados no tópico
qos	Texto	Sim	Nível de QoS da mensagem (0, 1 ou 2)
retain	Valor booleano	Sim	Flag indicando se a mensagem deve ser retida no tópico (true ou false)

Fonte: Autoria própria.

Em caso de sucesso no processamento da requisição, o gateway deve responder com "OK". Para parar de registrar as publicações feitas em determinado tópico, deve-se enviar uma requisição HTTP com método DELETE para http://ip-do-gateway:porta/api/mqtt/id-do-topico. O gateway deve responder com "OK", caso a requisição seja bem-sucedida.

5.3.3 CoAP

Com funcionamento similar ao da API para o protocolo Modbus TCP/IP, a API REST para comunicação com dispositivos compatíveis com o protocolo CoAP depende do cadastro prévio de dispositivos para permitir a leitura ou escrita de dados no dispositivo.

Para cadastro de um novo dispositivo, o consumidor da API REST deve enviar uma requisição HTTP com método POST para http://ip-do-gateway:porta/api/coap/ contendo um JSON no corpo da requisição com os parâmetros apresentados na Tabela 14.

Tabela 14 – Formato do JSON para cadastro de um dispositivo CoAP.

Atributo	Formato do dado	Opcional	Descrição
host	Texto	Não	Endereço IP do dispositivo
port	Inteiro	Sim	Porta de comunicação do dispositivo
update	Inteiro	Não	Período entre cada requisição ao dispositivo
descricao	Texto	Sim	Descrição em texto do dispositivo
resources	Array de objetos	Sim	Rotas para acesso aos recursos do dispositivo

Fonte: Autoria própria.

Caso o parâmetro *resources* não seja enviado, uma requisição será enviada à rota /.well-known/core para descoberta das rotas disponíveis. A resposta do gateway deve conter os parâmetros da Tabela 14, com os dados enviados ou os valores padrão de cada parâmetro, com o acréscimo do parâmetro id, que informa o id do dispositivo cadastrado.

Para visualização dos dados de um dispositivo, uma requisição HTTP com método GET deve ser enviada para http://ip-do-gateway:porta/api/coap/id-do-dispositivo. Se o dispositivo existir e a requisição for bem-sucedida, o gateway deve responder com os parâmetros apresentados na Tabela 15.

Tabela 15 – Atributos da resposta do *gateway* contendo dados de um dispositivo cadastrado.

Parâmetro	Formato do dado	Opcional	Descrição
dados_do_dispositivo	Objeto	Não	Contém os parâmetros de configuração do dispositivo (Tabela 16)
links	Array de objetos	Não	Cada objeto é composto por uma URL, o método que deve ser utilizado e a descrição do resultado esperado como resposta a essa requisição, informando as possíveis interações que o cliente pode fazer através da API (Tabela 6)

Fonte: Autoria própria.

Tabela 16 – Objeto contendo dados do dispositivo.

Parâmetro	Formato do dado	Opcional	Descrição
id	Número Inteiro	Não	Identificador único do dispositivo
host	Endereço IP	Não	Endereço IP do dispositivo
port	Número Inteiro	Não	Porta de cominicação do dispositivo
update	Número Inteiro	Não	Período entre cada requisição ao dispositivo
descricao	Texto	Não	Texto descrevendo o dispositivo
createdAt	Data	Não	*Timestamp* indicando data e hora do cadastro do dispositivo
updatedAt	Data	Não	*Timestamp* indicando data e hora em que dados do dispositivo foram alterados

Fonte: Autoria própria.

Para obtenção dos valores armazenados no banco de dados para determinada rota do dispositivo, deve-se enviar uma requisição HTTP com método GET para http://ip-do-gateway:porta/api/coap/id-do-dispositivo/rota. A resposta do gateway deve possuir no corpo da mensagem um *array* de objetos que contém os atributos apresentados na Tabela 17.

Caso deseje-se obter uma quantidade especificada de valores armazenados no banco de dados para determinada rota do dispositivo, deve-se enviar uma requisição HTTP com método GET para http://ip-do-gateway:porta/api/coap/id-do-dispositivo/rota/quantidade-de-valores. A resposta do gateway será similar ao caso anterior, apenas diferindo no tamanho fixado do *array* de objetos retornado, definido pela quantidade de valores especificada na URL.

Tabela 17 - Formato do objeto contendo dados de um recurso disponível em uma rota do dispositivo CoAP.

Parâmetro	**Formato do dado**	**Opcional**	**Descrição**
id	Número Inteiro	Não	Identificador único do dado registrado
href	Texto	Não	Endereço da rota
title	Texto	Sim	Nome do recurso
rt	Texto	Sim	Tipo do recurso
ct	Texto	Sim	Tipo de conteúdo do recurso
if	Texto	Sim	Descrição da interface
sz	Texto	Sim	Tamanho máximo estimado do recurso
value	Texto	Não	Valor do recurso
timestamp	Unix Timestamp	Não	Timestamp indicando data e hora do registro do dado
CoapId	Número Inteiro	Não	Identificador único do dispositivo

Fonte: Autoria própria.

Para escrita de valores em determinada rota do dispositivo, deve-se enviar uma requisição HTTP com método POST para http://ip-do-gateway:porta/api/coap/id-do-dispositivo/rota com um JSON no corpo da requisição contendo o valor no parâmetro *value*. Para interromper a comunicação com um dispositivo, deve-se enviar uma requisição HTTP com método DELETE para http://ip-do-gateway:porta/api/coap/id-do-dispositivo. Caso a requisição seja bem-sucedida, o gateway deve responder com “OK”.

5.3.4 Informações do Sistema

Foi criada uma rota especial na API REST para obtenção de dados sobre a utilização de recursos do sistema. Uma requisição HTTP com método GET deve ser enviada para http://ip-do-gateway:porta/api/system/. A resposta do gateway contém um JSON com a porcentagem de utilização de CPU e memória RAM, além da temperatura do processador, conforme mostrado na Tabela 18.

Tabela 18 – Parâmetros retornados pelo gateway informando nível de utilização dos recursos do sistema.

Atributo	**Formato do dado**	**Opcional**	**Descrição**
cpu	Número real	Não	Nível de utilização da CPU em porcentagem
memory	Número real	Não	Nível de utilização da memória RAM em porcentagem
temperature	Número real	Não	Temperatura do processador em graus Celsius

Fonte: Autoria própria.

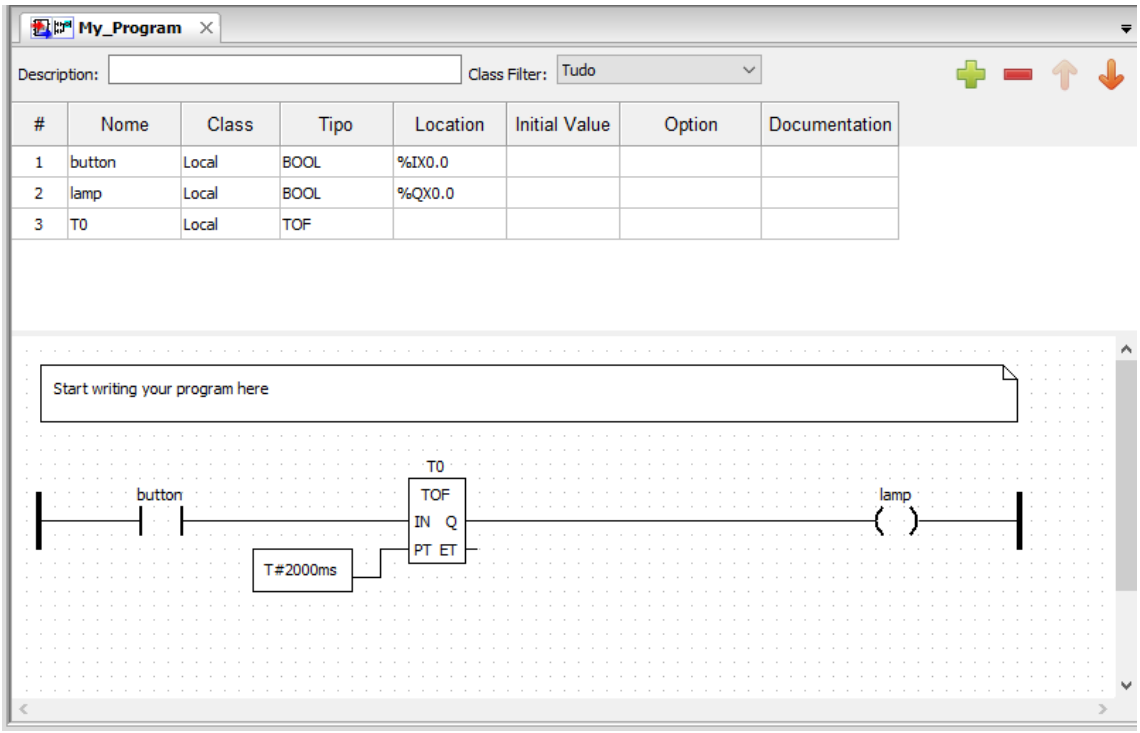
Esta rota foi utilizada nos testes de operação do gateway para análise do consumo de recursos do sistema em diferentes cenários de operação. A descrição dos experimentos e os resultados obtidos são apresentados no capítulo 6.

6 RESULTADOS

6.1.1 Validação do Módulo Remoto

Experimentos iniciais foram realizados para teste de operação do modelo para aplicação da IIoT proposto. Para isso utilizou-se o PLCOpen Editor do OpenPLC e um programa em linguagem Ladder fornecido pelo site do Projeto OpenPLC, mostrado na Figura 22. Esse programa faz com que um led permaneça aceso por 2 segundos e depois se apague quando um botão é pressionado.

Figura 22 - Programa em Ladder utilizado para testar a comunicação entre o servidor do OpenPLC e os módulos remotos.

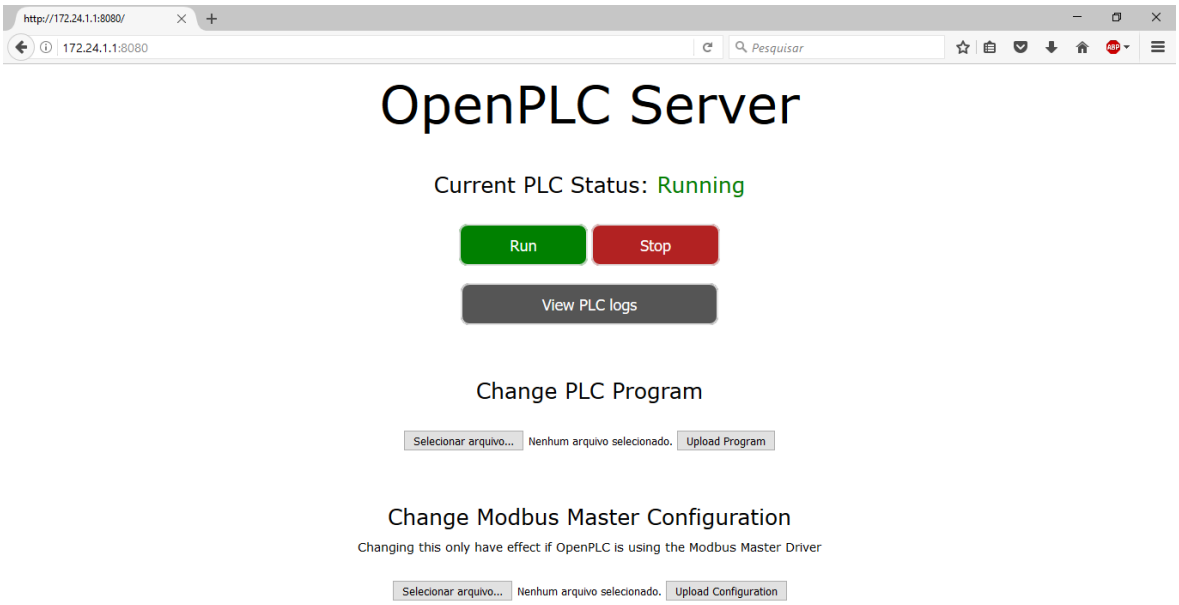


Fonte: Autoria própria.

Este programa em Ladder foi carregado e executado no servidor do OpenPLC (Figura 23), que utiliza as placas ESP8266 como interface com o mundo externo, sendo possível monitorar todos os dados (ex: estado das portas de I/O) da placa no programa desenvolvido em Node.js, que se comunica via Modbus TCP/IP com o servidor do OpenPLC. Além da comunicação com o servidor do OpenPLC, foi testada também a comunicação direta com os módulos remotos utilizando o protocolo Modbus TCP/IP com o recebimento dos dados via software ScadaBR (SCADA, 2016) e LabVIEW.

Com a utilização do Mosca, introduzido como um módulo no programa desenvolvido em Node.js, foi possível observar e validar a comunicação entre dois módulos remotos através do protocolo MQTT.

Figura 23 - Servidor do OpenPLC em execução.



Fonte: Autoria própria.

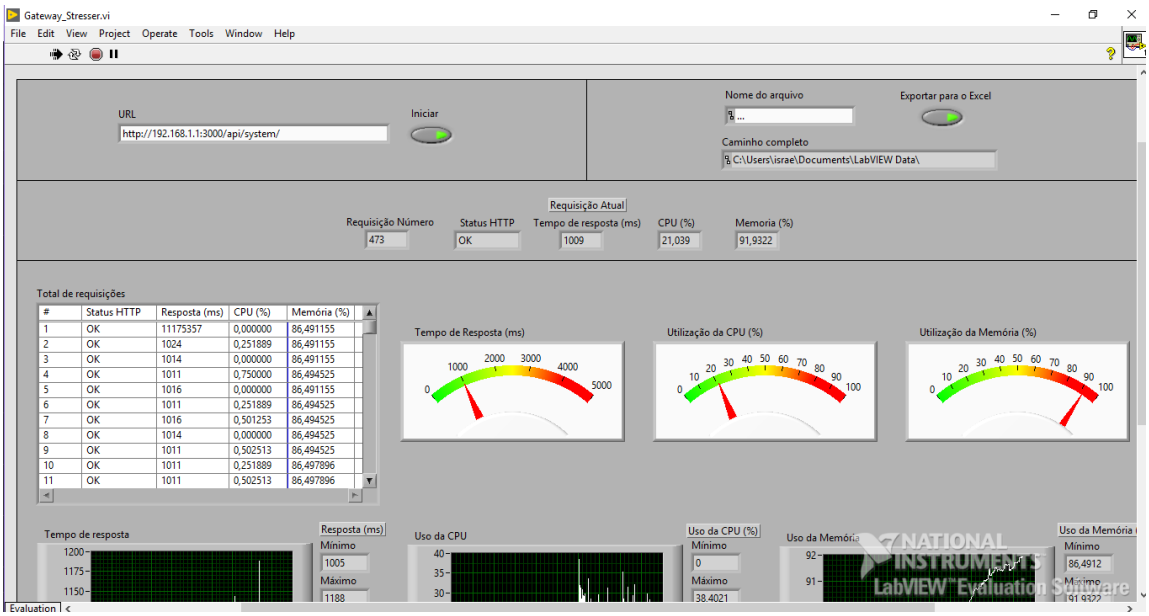
A validação da comunicação por protocolo CoAP ocorreu com a utilização de uma extensão desenvolvida por Matthias Kovatsch para o navegador Mozilla Firefox chamada Copper (COPPER, 2017). Os primeiros testes foram feitos em um notebook rodando a distribuição Linux Kubuntu.

6.1.2 Validação do Módulo Gateway

Para a validação do desenvolvimento do módulo gateway, configurou-se a Raspberry PI 3 rodando uma distribuição personalizada do Linux chamada Raspbian (baseada no Debian), para funcionar como ponto de acesso WiFi. Deste modo, os módulos remotos (ESP8266) podem se conectar a ela diretamente via WiFi (sem a necessidade de um hardware adicional como um roteador sem fio) conforme Figura 25, para troca de informações e conectividade com a rede, que pode também ser fornecida através da porta Ethernet da Raspberry PI.

O programa desenvolvido nesse projeto, juntamente com o servidor do OpenPLC e os demais recursos necessários para o funcionamento do sistema proposto foram testados na Raspberry Pi 3. Para isso foi criada uma rota na API REST que retorna um JSON contendo informações sobre o uso de recursos do sistema, como uso de memória, CPU e temperatura do processador. Foi desenvolvido um programa no software LabVIEW da National Instruments, que envia sucessivas requisições HTTP para essa rota do gateway e armazena esses valores, juntamente com o tempo entre cada requisição e sua respectiva resposta, em uma planilha do Microsoft Excel para registro desses dados no computador cliente da API REST.

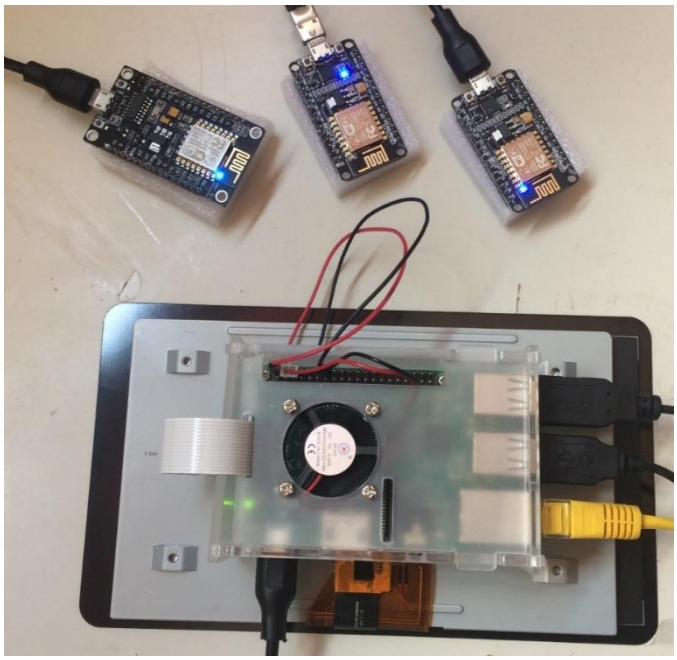
Figura 24 - Software desenvolvido no LabVIEW para coleta de dados do gateway.



Fonte: Autoria própria.

A comunicação entre os módulos remotos e o módulo gateway foi testada (Figura 25). Os estados das portas de I/O foram requisitados pela aplicação desenvolvida em Node.js e armazenados no banco de dados MySQL. Para a realização de diferentes experimentos e verificação da operação do módulo gateway desenvolvido, comparou-se o uso de recursos do sistema em diversas situações, desde o módulo gateway ligado sem executar o servidor do OpenPLC e com nenhum módulo remoto conectado, até uma verificação da quantidade máxima de módulos remotos conectados simultaneamente.

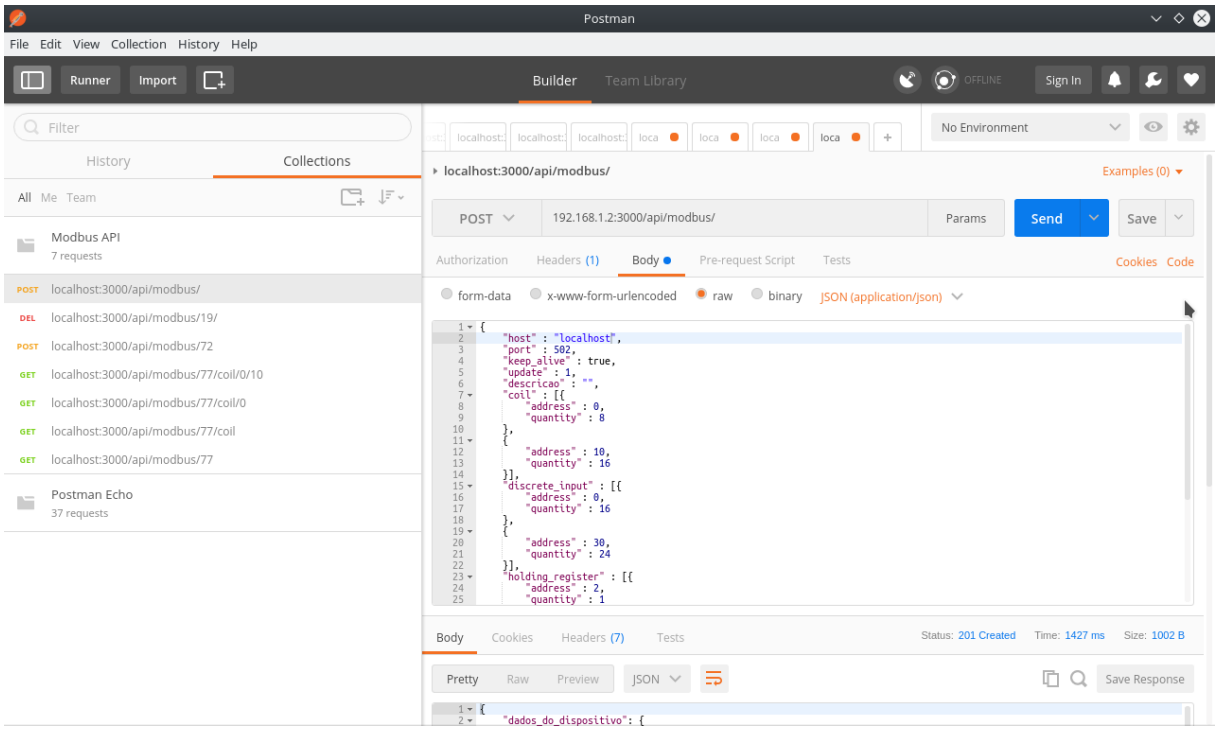
Figura 25 - Módulos remotos (ESP8266) se comunicando por WiFi com o módulo gateway (Raspberry PI).



Fonte: Autoria própria.

O software Postman (Figura 26), que facilita o teste de APIs REST, foi utilizado para enviar a requisição responsável por cadastrar um novo dispositivo no gateway (POSTMAN, 2017). Os experimentos realizados estão descritos na Tabela 19, enquanto os resultados obtidos podem ser vistos na Tabela 20.

Figura 26 - Interface do software Postman utilizado para teste de APIs REST.



Fonte: Autoria própria.

Tabela 19 - Descrição de Experimentos realizados com o Módulo Gateway.

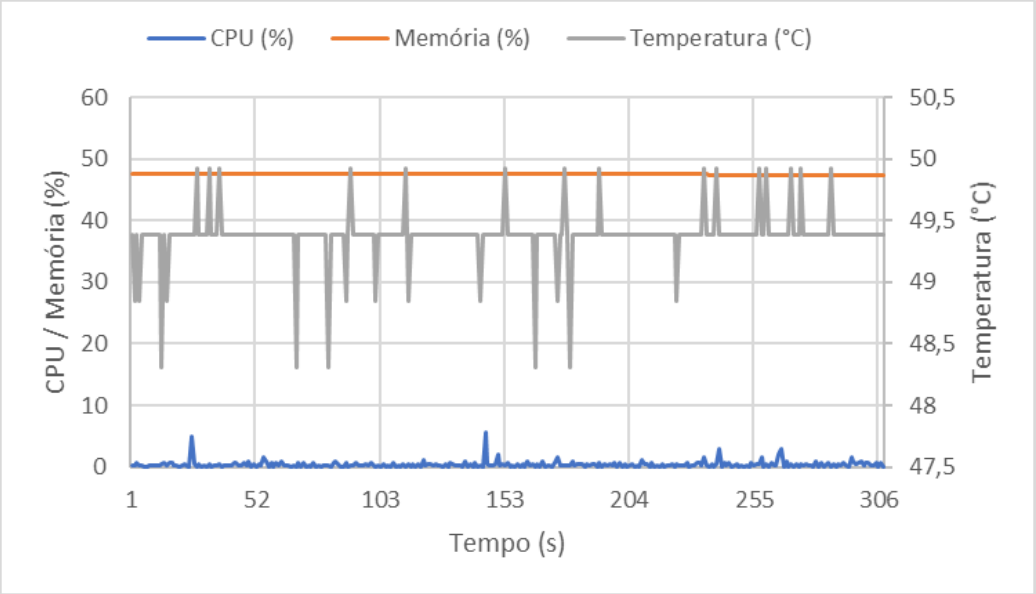
Experimento	**Descrição**	**Parâmetros**
1	Módulo gateway em espera.	Apenas o programa desenvolvido em Node.js em execução.
2	OpenPLC em execução.	Programa desenvolvido em Node.js e OpenPLC em execução. Nenhum módulo remoto conectado.
3	Módulo remoto se comunicando com servidor do OpenPLC.	Um módulo remoto conectado com o servidor do OpenPLC, com programa desenvolvido no PLCOpen Editor sendo executado pelo OpenPLC.
4	Diversas conexões com o servidor do OpenPLC sendo adicionadas sucessivamente ao gateway, até esse apresentar falha.	Período de atualização de 10s.
5	Diversas conexões com o servidor do OpenPLC sendo adicionadas sucessivamente ao gateway, até esse apresentar falha.	Período de atualização de 1s.
6	Conexões com módulos remotos através do protocolo Modbus TCP/IP sendo adicionadas sucessivamente ao gateway, até esse apresentar falha.	Período de atualização de 1s.

7	Conexões com módulos remotos através do protocolo CoAP sendo adicionadas sucessivamente ao gateway, até esse apresentar falha.	Período de atualização de 1s.
8	Diversos módulos remotos adicionados sucessivamente, comunicando-se através do protocolo MQTT.	Nenhum tópico sendo monitorado.
9	Diversos módulos remotos adicionados sucessivamente, comunicando-se através do protocolo MQTT.	Adição sucessiva de tópicos a serem monitorados.

Fonte: Autoria própria.

O experimento 1 foi realizado com a Raspberry PI 3 em repouso (o servidor do OpenPLC não havia sido inicializado e não havia nenhum módulo remoto conectado com a Raspberry PI 3). A utilização média de CPU foi de 0,41%, enquanto a utilização média de memória RAM foi de 47,55% e a temperatura média do processador nesse período foi de 49,38 °C (Figura 27).

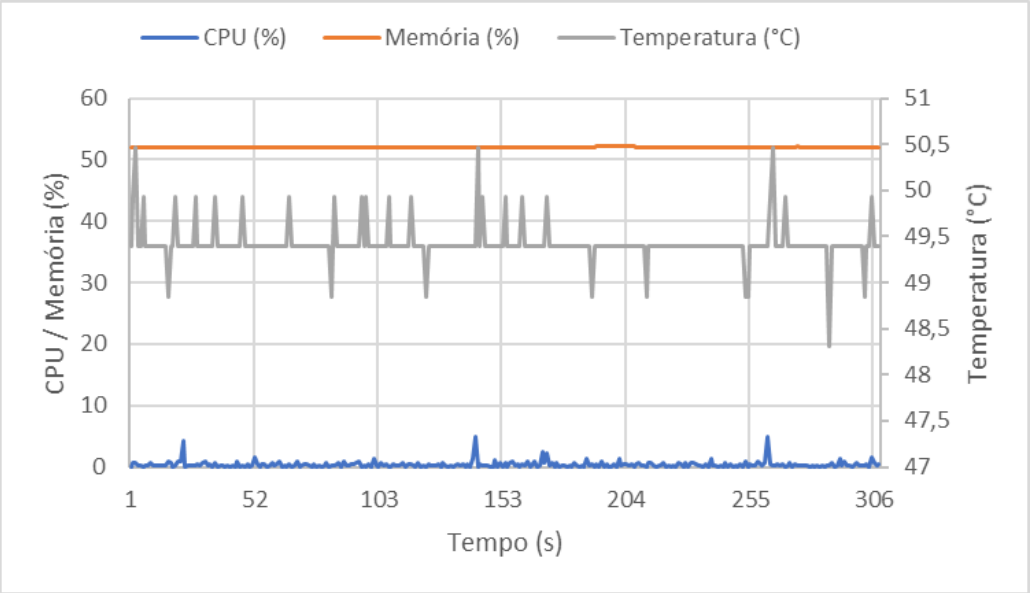
Figura 27 - Variação da Temperatura, Utilização de CPU e Memória RAM do Gateway para Experimento 1.



Fonte: Autoria própria.

O experimento 2 iniciou-se o servidor do OpenPLC, ainda sem a conexão de nenhum módulo remoto com o módulo gateway. Não houve grande impacto na utilização da CPU, sendo a média nesse experimento igual a 0,41%, enquanto em média a utilização da memória RAM ficou em 52,03% e a média da temperatura no período foi de 49,41 °C (Figura 28).

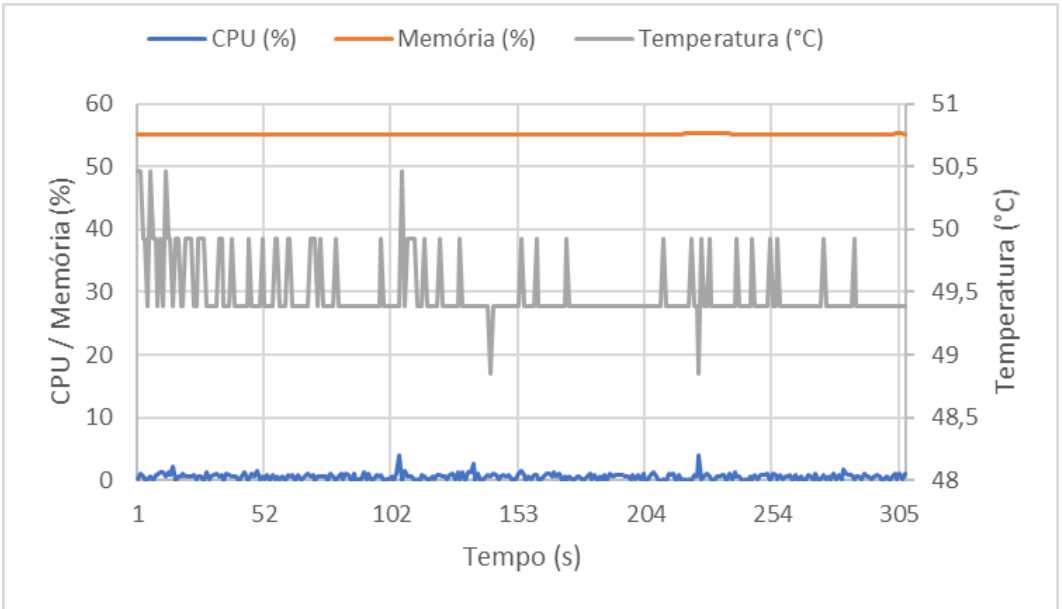
Figura 28 - Variação da Temperatura, Utilização de CPU e Memória RAM do Gateway para Experimento 2.



Fonte: Autoria própria.

Foi adicionado um módulo remoto conectado ao servidor do OpenPLC e executou-se um programa desenvolvido em Ladder no PLCOpen Editor para a realização do experimento 3. Neste caso, a utilização média dos recursos do sistema foi de 0,55% de CPU e 55,17% de memória RAM, enquanto a temperatura média da CPU foi de 49,49 °C (Figura 29).

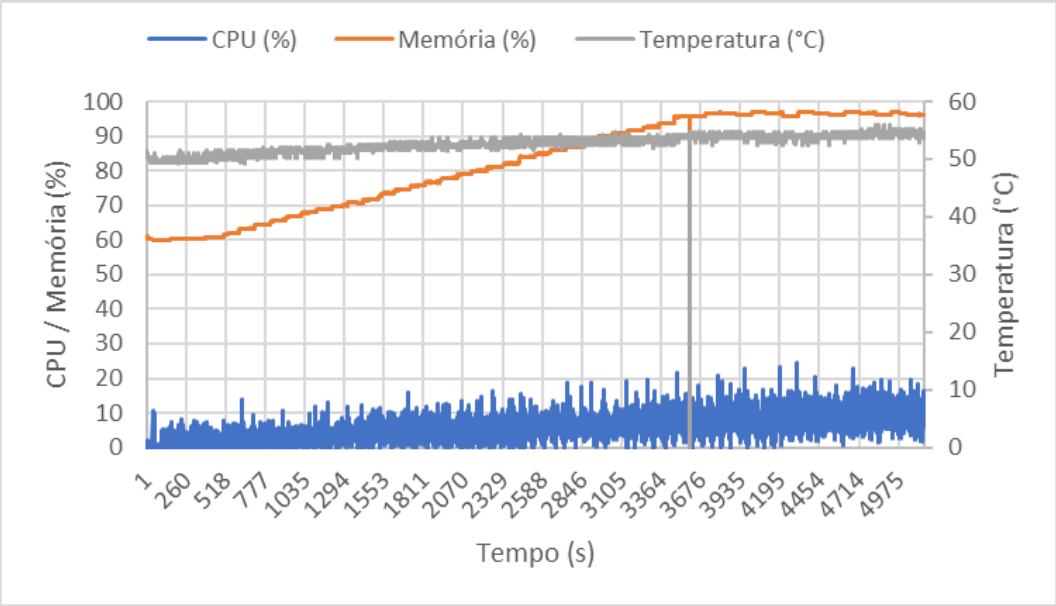
Figura 29 - Variação da Temperatura, Utilização de CPU e Memória RAM do Gateway para Experimento 3.



Fonte: Autoria própria.

Para o experimento 4, adotou-se a seguinte metodologia: a cada 100 requisições à rota que retorna o status da utilização dos recursos do sistema e temperatura, uma nova conexão com o servidor do OpenPLC foi criada. Isso foi feito cadastrando um novo dispositivo no gateway, porém sempre com os mesmos dados: endereço IP e porta do servidor do OpenPLC e sempre requisitando dados de 8 saídas digitais (coils) com período de atualização de 10 segundos. Os resultados são mostrados na Figura 30.

Figura 30 - Variação da Temperatura, Utilização de CPU e Memória RAM do Gateway para Experimento 4.

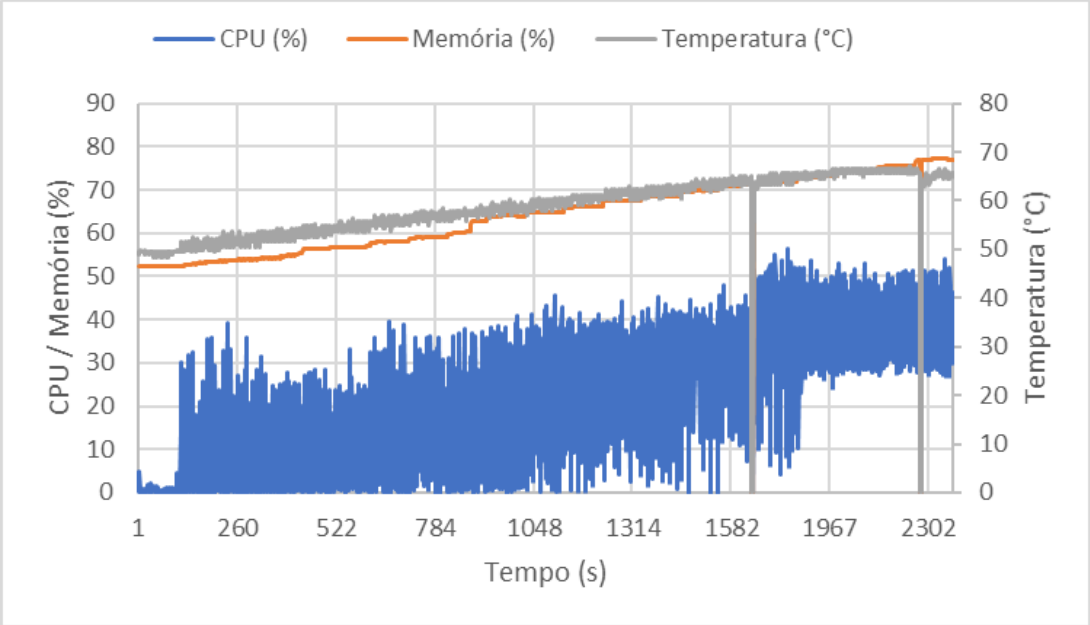


Fonte: Autoria própria.

Apesar de alguns problemas de comunicação com o gateway terem ocorrido no decorrer do experimento 4, conforme pode ser visto na Figura 30 como uma descontinuidade no gráfico, não houve nenhuma falha severa e foi possível adicionar 50 conexões simultâneas. A descontinuidade é causada pelo modo como o LabVIEW interpreta um erro de *timeout*, que ocorre quando não se obtém uma resposta a uma requisição num determinado intervalo de tempo e a conexão é encerrada. Nestes casos, o LabVIEW atribuiu valor zero para as variáveis monitoradas.

Visando encontrar os limites de operação do módulo gateway o experimento 4 foi repetido requisitando 32 endereços de I/O de cada tipo. Nesse segundo cenário o número máximo de conexões simultâneas caiu para 14, sendo que a partir da 15ª conexão o módulo gateway encontrou problemas para acessar o banco de dados (Figura 31). Assim como no caso anterior, falhas de comunicação com a API REST ocorreram e foram apresentadas na Figura 31 como descontinuidades no gráfico.

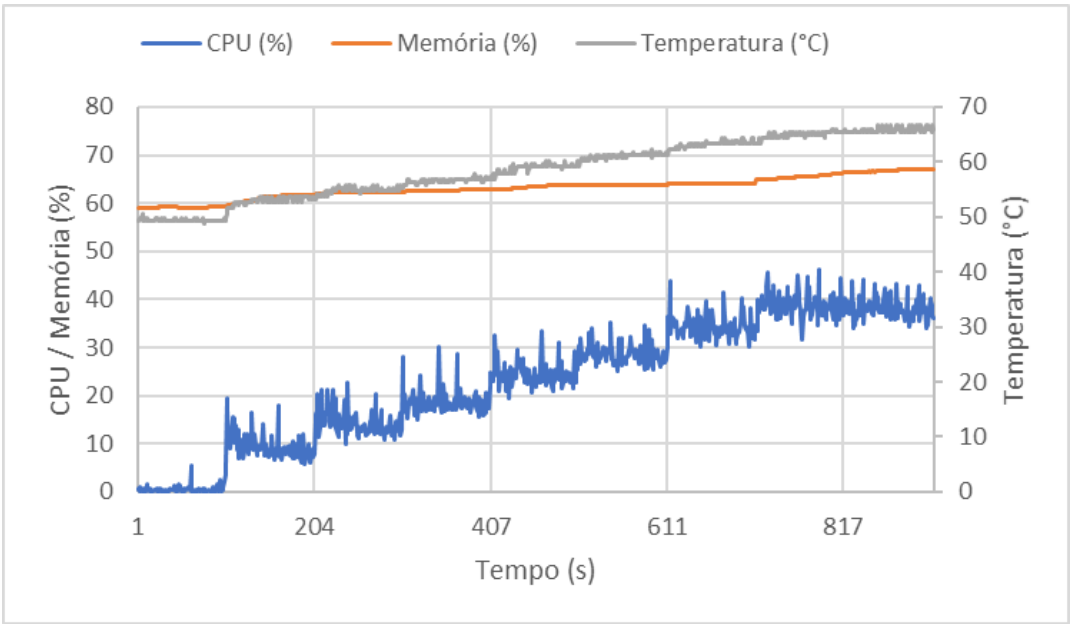
Figura 31 - Variação da Temperatura, Utilização de CPU e Memória RAM do Gateway para Complemento do Experimento 4.



Fonte: Autoria própria.

O experimento 5 segue a mesma metodologia do experimento 4, agora com período de atualização de 1 segundo e requisitando 8 endereços de I/O de cada tipo. O módulo gateway apresentou problemas de acesso ao banco de dados a partir da adição do 7º dispositivo. Os resultados são mostrados na Figura 32.

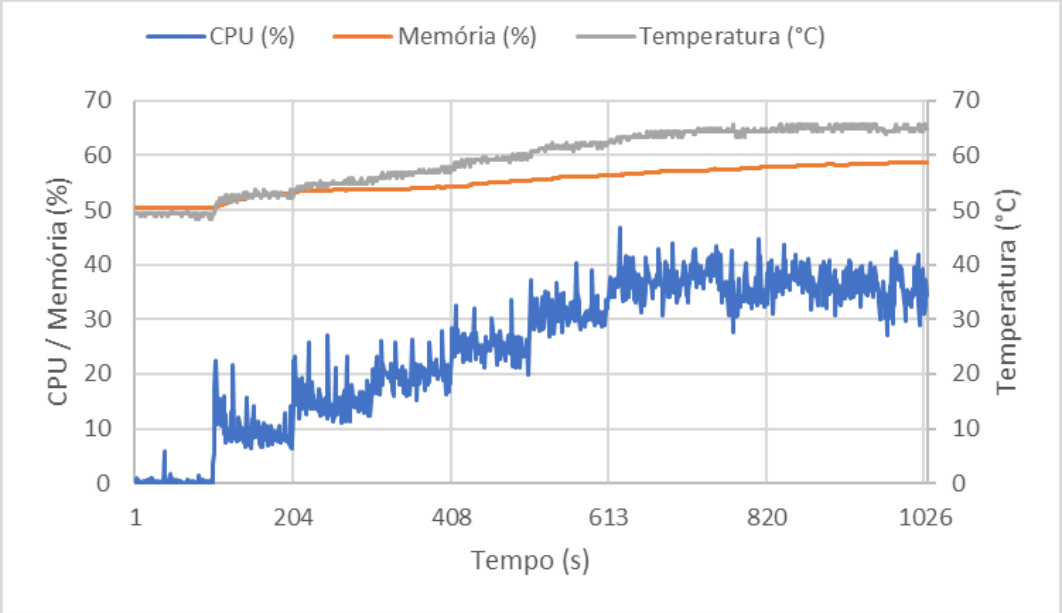
Figura 32 - Variação da Temperatura, Utilização de CPU e Memória RAM do Gateway para Experimento 5.



Fonte: Autoria própria.

O experimento 6 foi realizado seguindo a mesma metodologia do experimento 5, requisitando dados dos dispositivos remotos com período de atualização de 1 segundo e 8 endereços de I/O de cada tipo. A grande diferença entre esse experimento e o anterior é o cadastro de módulos remotos distintos ao invés de várias conexões com o mesmo módulo. Foram utilizados módulos remotos com o firmware Modbus, desenvolvido nesse projeto. Novamente os problemas de acesso ao banco surgiram a partir do 7º dispositivo cadastrado. Os resultados são mostrados na Figura 33.

Figura 33 - Variação da Temperatura, Utilização de CPU e Memória RAM do Gateway para Experimento 6.

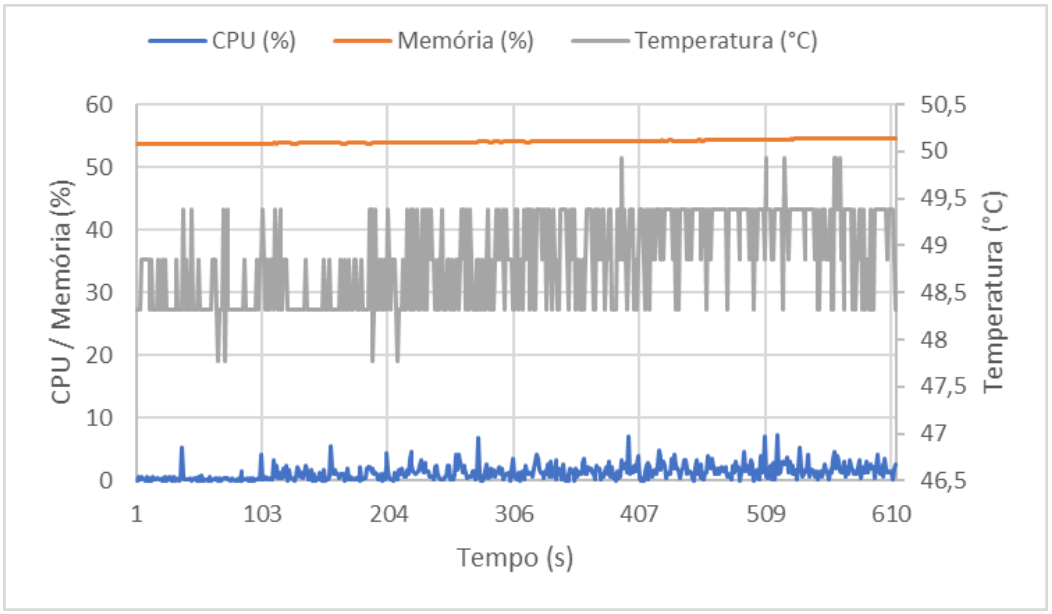


Fonte: Autoria própria.

Seguindo a metodologia do experimento 6, para o experimento 7 foram adicionadas conexões sucessivas com módulos remotos distintos, comunicando-se com o gateway pelo protocolo CoAP com período de atualização de 1 segundo. Diferentemente do experimento anterior, o ponto de falha nesse experimento foi o firmware dos módulos remotos, que não foram capazes de responder a requisições sucessivas e reiniciaram depois de algum tempo.

Como os módulos remotos deixaram de responder às requisições realizadas pelo módulo gateway, poucos valores foram armazenados no banco de dados e a utilização de recursos de sistema pelo gateway ficou abaixo do experimento anterior. Os resultados são mostrados na Figura 34.

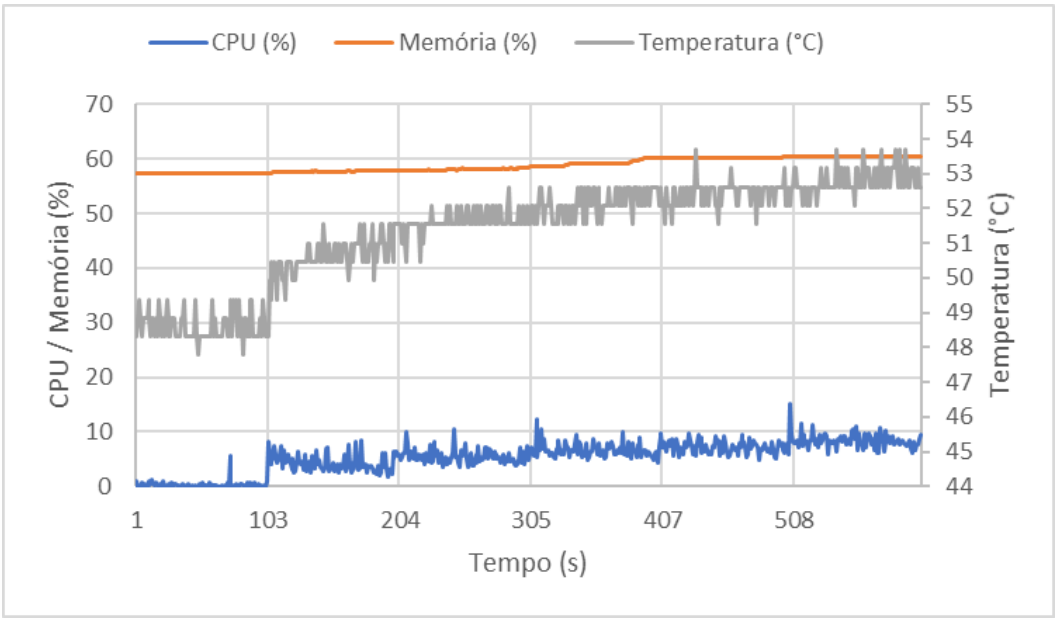
Figura 34 - Variação da Temperatura, Utilização de CPU e Memória RAM do Gateway para Experimento 7.



Fonte: Autoria própria.

No experimento 8 módulos remotos com o firmware de comunicação MQTT se conectaram ao gateway de forma sucessiva. A cada 100 requisições, um novo módulo era conectado. Cada módulo foi configurado com todos os seus 8 pinos de I/O digital como entrada, de modo que esses dados fossem enviados constantemente ao broker MQTT. Inicialmente não haviam tópicos cadastrados, ocorrendo somente a comunicação entre os módulos remotos e o broker MQTT. Os resultados são mostrados na Figura 35.

Figura 35 - Variação da Temperatura, Utilização de CPU e Memória RAM do Gateway para Experimento 8.



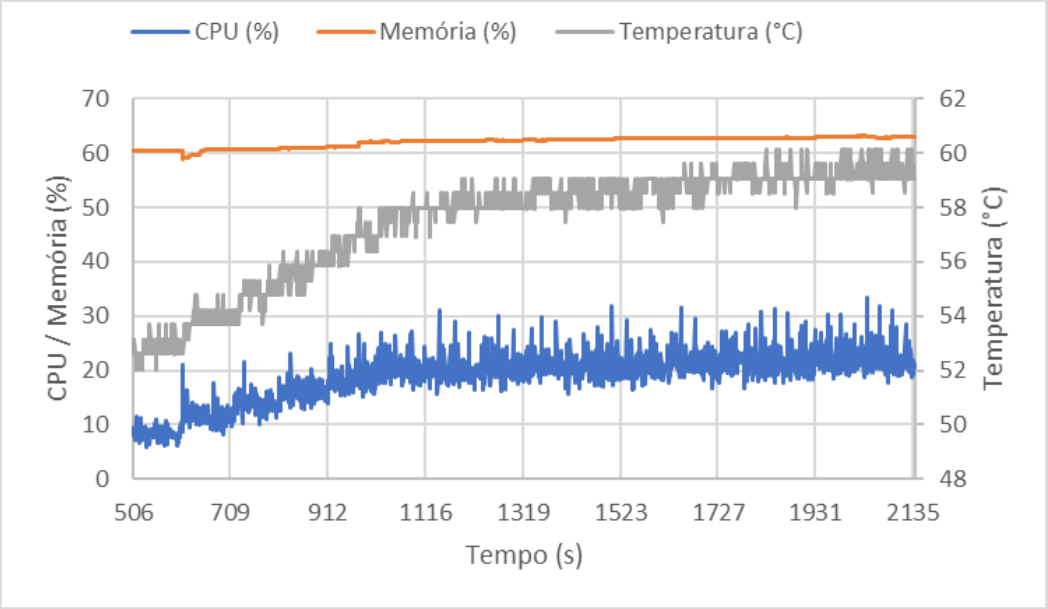
Fonte: Autoria própria.

Ao final desse experimento, 5 módulos remotos estavam conectados ao módulo gateway, sendo que cada um estava publicando dados de 8 entradas digitais a cada segundo. Nessa situação, atingindo, portanto, uma média de 40 publicações por segundo trafegando pelo broker MQTT, utilização média (últimas 100 requisições) de 8,4% de CPU, 60,34% de memória RAM e uma temperatura de 52,82 °C.

O experimento 9 é complementar ao experimento 8, sendo realizado juntamente com ele. Após os 5 módulos remotos se conectarem ao módulo gateway e enviarem os dados das entradas digitais, foram cadastrados os tópicos relativos a cada dispositivo para monitoramento pelo módulo gateway. A cada 100 requisições todos os tópicos relacionados com um dos módulos remotos eram adicionados através da utilização do wildcard "#" do MQTT.

Isso foi feito através do cadastro no tópico "endereço-mac-do-dispositivo/#", possibilitando que todos os dados publicados nos tópicos relativos às entradas digitais do dispositivo identificado pelo endereço MAC cadastrado fossem armazenados no banco de dados. Após todos os tópicos seguindo a metodologia apresentada serem cadastrados, foi realizado o cadastro de tópicos individuais de cada entrada digital de um determinado módulo remoto, para verificação do impacto causado na utilização dos recursos do sistema do módulo gateway. Os resultados são mostrados na Figura 36.

Figura 36 - Variação da Temperatura, Utilização de CPU e Memória RAM do Gateway para Experimento 9.



Fonte: Autoria própria.

Os valores médios de uso de CPU (%), memória RAM (%) e temperatura do processador (°C) para o período compreendido entre as últimas 100 requisições de cada experimento são apresentados na Tabela 20.

Tabela 20 - Comparação de Resultados de Operação do Módulo Gateway.

Experimento	Parâmetro Analisado						
	Utilização de CPU		Utilização de memória		Temperatura		Falha no módulo Gateway?
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão	
1	0,44%	0,53	47,46%	0,09	49,42°C	0,15	Não
2	0,39%	0,55	52,07%	0,02	49,38°C	0,20	Não
3	0,52%	0,50	55,21%	0,03	49,43°C	0,17	Não
4. a)	9,65%	3,34	96,25%	0,08	54,62°C	0,44	Não
4. b)	37,11%	8,61	76,88%	0,45	65,09°C	0,93	Sim
5	38,32%	2,23	66,71%	0,26	65,84°C	0,39	Sim
6	35,73%	3,16	58,44%	0,13	65,00°C	0,39	Sim
7	1,98%	1,26	54,50%	0,07	49,17°C	0,42	Não
8	8,40%	1,35	60,34%	0,02	52,82°C	0,37	Não
9	22,28%	2,74	62,92%	0,08	59,43°C	0,43	Não

Fonte: Autoria própria.

Analisando os resultados da Tabela 20 é possível observar um padrão nos casos de falha do módulo gateway que é a maior utilização de CPU e consequentemente uma temperatura do processador mais elevada, possivelmente ocasionada por uma quantidade maior de requisições por segundo ao banco de dados nesses experimentos. Até mesmo com 50 conexões simultâneas, como visto no experimento 4.a), o módulo gateway não apresentou sinais de falha e executou suas funções com normalidade, apesar da maior utilização de memória RAM registrada nesse experimento. Com exceção do experimento 7, onde a falha ocorreu no firmware dos módulos remotos e não no gateway, nos experimentos onde ocorreram falhas estas foram relacionadas ao banco de dados e à taxa de armazenamento de dados.

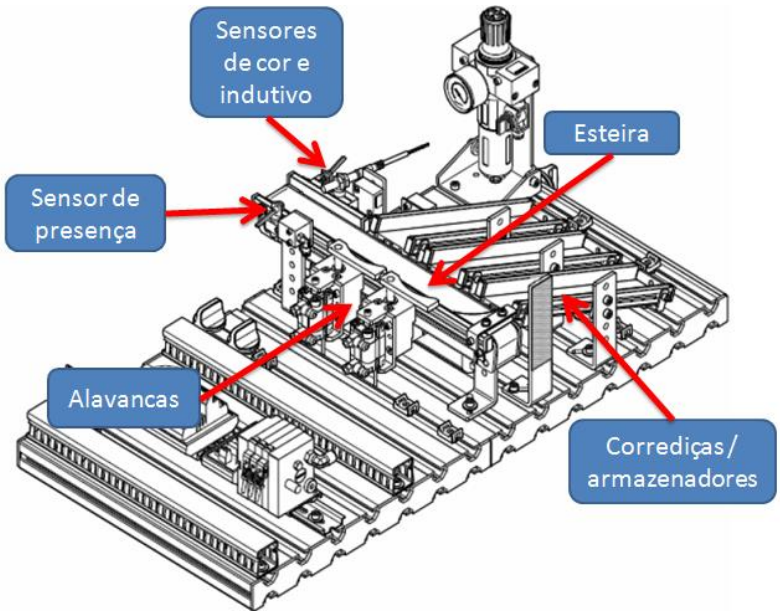
6.1.3 Prova de Conceito da Arquitetura para Aplicação da IIoT

Após a realização dos experimentos descritos anteriormente para validação dos dispositivos remotos e do módulo gateway, um caso de estudo real foi realizado como prova de conceito para validação da arquitetura para aplicação de IIoT desenvolvido. Este caso de estudo foi realizado numa configuração compatível com uma aplicação industrial, através do uso de uma estação do Sistema Modular de Produção (*MPS – Modular Production System*) da Festo do Laboratório de Automação da UNESP Sorocaba.

O MPS representa um grupo de estações de trabalho controladas por Controladores Lógicos Programáveis (CLPs) e manuseadas por sistemas automáticos (manipulador robótico) que podem produzir diferentes produtos de forma automática. Cada estação do MPS possui sensores, motores e atuadores eletropneumáticos, sendo possível a produção de diferentes peças, através da realização de diferentes etapas de manufatura. Além disso, é possível utilizar cada estação individualmente. A operação das estações é feita através da programação dos CLPs. As principais estações do MPS são: Estação de Distribuição (*Distributing*), Estação de Testes (*Testing*), Estação de Manipulação (*Handling*), Estação de Processamento (*Processing*), Estação de Montagem – Braço Robótico (*Assembly*) e a Estação de Separação (*Sorting*).

No caso de estudo deste trabalho foi utilizado a Estação de Separação, mostrada na Figura 37, para controle e monitoramento utilizando a arquitetura para aplicação de IIoT desenvolvido. A estação de separação recebe as peças em uma esteira que identifica se a peça é metálica ou não, ou se a peça é preta ou rosa, através dos sensores ópticos e magnéticos e as separa em diferentes armazenadores através de alavancas eletropneumáticas que são acionadas conforme a necessidade. Em sua configuração original, a estação possui um CLP que é o responsável pela interface de entradas e saídas digitais e programação da lógica que controla a operação da estação.

Figura 37 – Detalhes dos Componentes da Estação de Separação.



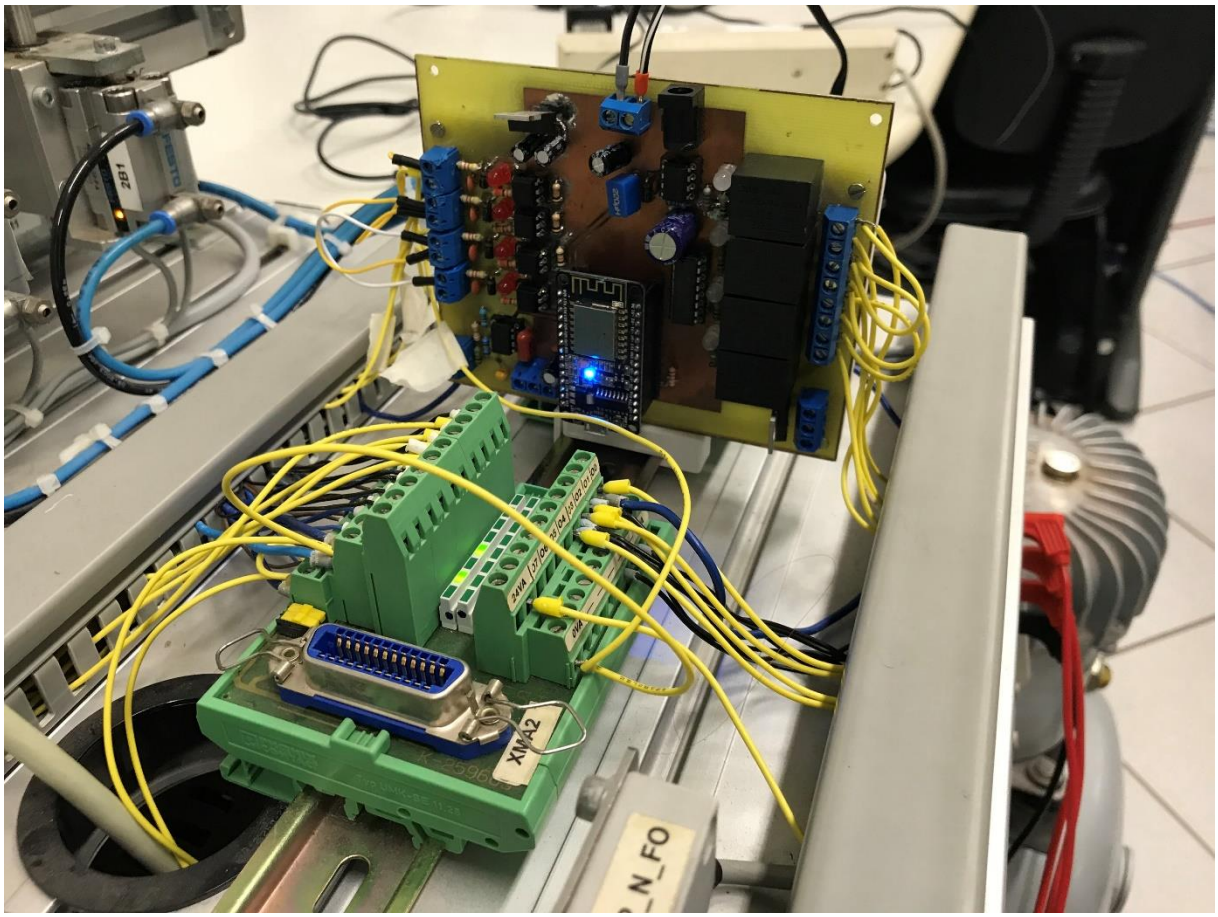
Fonte: Autoria própria.

Nesta prova de conceito, o dispositivo CLP da estação não foi usado. Para a realização das suas funções dentro da operação da estação, o módulo remoto desenvolvido neste trabalho foi utilizado. Para interface entre o módulo remoto, que consiste no dispositivo NodeMCU com o microcontrolador ESP8266, e os sensores e atuadores da estação de separação que usam

padrão industrial de 24VDC para entradas e saídas digitais, uma placa com circuitos eletrônicos para adaptação de níveis de tensão e corrente foi utilizada (Figura 38).

As entradas e saídas (I/Os) dessa placa são conectados a uma borneira de conexão dos dispositivos da estação de separação. Essa placa foi usada como módulo remoto, executando o firmware desenvolvido neste trabalho para comunicação com o servidor do OpenPLC e operação de acordo com o modelo para IIoT desenvolvido.

Figura 38 – Placa com Módulo Remoto NodeMCU conectado ao Borne de I/Os da Estação de Separação



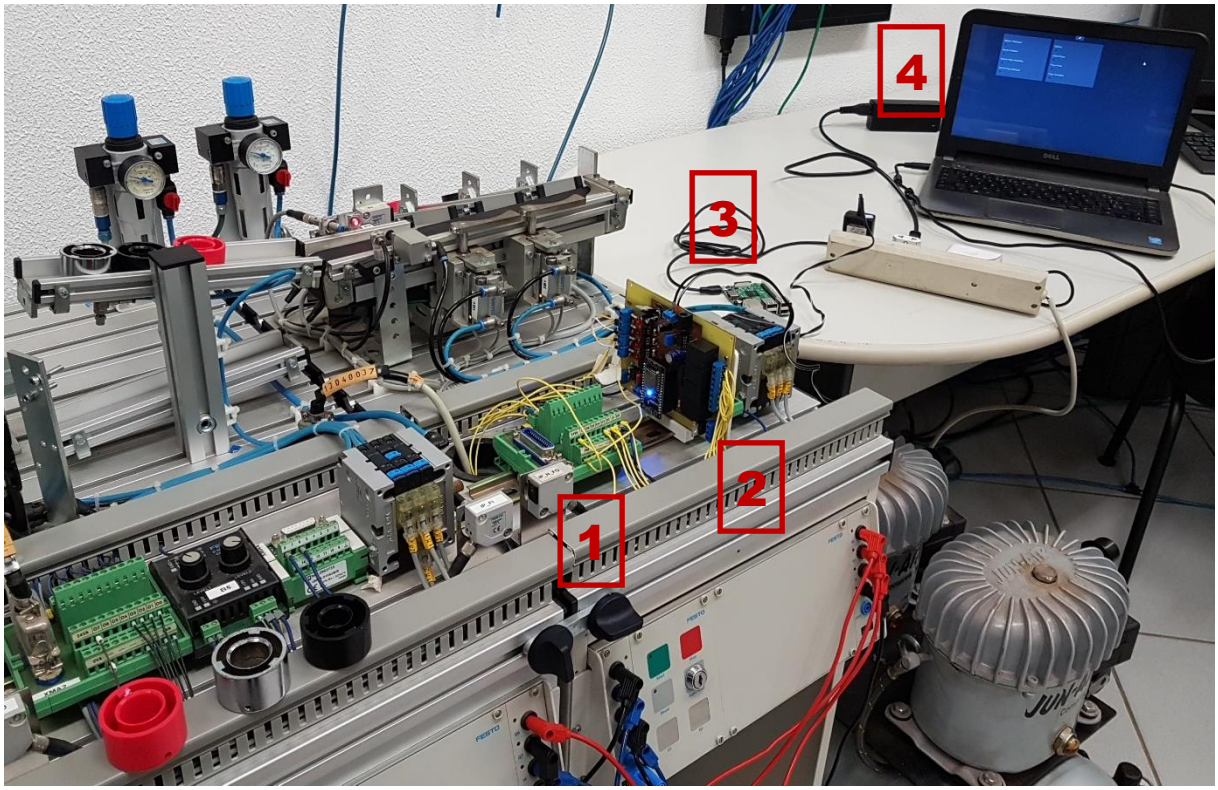
Fonte: Autoria própria.

Um programa escrito em linguagem Ladder com a utilização do PLCOpen Editor foi criado para a estação e enviado para o servidor do OpenPLC. Este programa controla as funções da estação de separação de peças (Figura 39 Legenda 1) segundo sua cor, através das entradas e saídas do módulo remoto (Figura 39 Legenda 2).

O módulo gateway (Raspberry PI 3 na Figura 39 Legenda 3) é responsável por rodar o servidor do OpenPLC e o programa criado para o gateway. O programa foi executado e, através de comunicação sem fio Modbus TCP/IP do módulo remoto com o módulo gateway, foi possível monitorar e controlar o processo.

Representado o servidor da aplicação do modelo para IIoT, nesse estudo de caso foi utilizado o aplicativo Freeboard (FREEBOARD, 2018) como interface gráfica (Figura 39 Legenda 4). Essa interface gráfica criada nesse aplicativo funciona como consumidor da API REST do gateway, tornando possível a visualização dos dados de forma amigável.

Figura 39 - Separação de peças por cor, utilizando um módulo remoto conectado ao gateway.



Fonte: Autoria própria.

Testes de operação, controle e monitoramento da estação de separação realizados com a aplicação da arquitetura para aplicação da IIoT desenvolvido comprovaram a eficácia da solução. O desempenho e tempo de resposta/atuação sobre o processo da estação de manufatura se mostram adequados em relação ao requisito de operação. Durante este teste não foram encontrados erros de operação e comunicação da solução, de forma que a arquitetura desenvolvida apresentou confiabilidade compatível com a aplicação.

A replicação do experimento desta prova de conceito para outras estações do Sistema Modular de Produção poderia ser facilmente implementada. Para isso seria necessário a construção de mais unidades da interface de I/O para os sensores e atuadores as outras estações e o desenvolvimento dos programas ou lógica de controle de operação das outras estações nas linguagens de programação de CLP permitidas pelo projeto OpenPLC. Com a criação desses programas, cada uma das novas variáveis (sensores e atuadores) das outras estações seria monitorada e seu dado transmitido para o módulo gateway via protocolo de comunicação definido (Modbus TCP/IP).

Os resultados dos diversos experimentos realizados com o gateway atrelado aos dessa prova de conceito usando um sistema de manufatura real comprovam o correto desenvolvimento da arquitetura e seu potencial para aplicações em soluções de IIoT.

7 CONCLUSÕES

Este trabalho apresentou e descreveu o desenvolvimento de um modelo para aplicação da Internet das Coisas Industrial. O modelo para aplicação da IIoT é fundamentado numa arquitetura de camadas de dispositivos e protocolos aderentes aos requisitos da aplicação da IoT no meio industrial, sendo possível categorizar as camadas dessa arquitetura de forma hierárquica, baseando-se na diferença de complexidade dos dispositivos e protocolos utilizados. O diferencial do modelo é a grande integração entre as tecnologias de informática e automação, de forma que são permitidas não somente funcionalidades de aquisição e análise dos dados industriais, mas também a programação e supervisão dos processos de acordo com padrões industriais (IEC 61131-3).

Um ponto forte do modelo desenvolvido é a flexibilidade e versatilidade de comunicação proporcionada pelos protocolos suportados pelo modelo de aplicação. O Modbus TCP/IP é utilizado de forma a representar a demanda crescente no uso da Ethernet Industrial em aplicações de automação. O suporte ao MQTT e ao CoAP, que são protocolos otimizados para comunicação em baixa largura de banda, fornecem maior eficiência na distribuição de informações, reduzindo o consumo de banda na rede. Além disso, outro ponto forte é que o modelo é totalmente baseado em soluções de hardware e software de código aberto, possibilitando a personalização e expansão do sistema para qualquer aplicação.

Os resultados apresentados validaram o desenvolvimento e demonstram o potencial do modelo para aplicações de IIoT e Industria 4.0. O hardware embarcado foi capaz de executar o software desenvolvido para este projeto, juntamente com o OpenPLC e gerenciar a comunicação entre os módulos remotos e o gateway de forma eficiente e confiável na maior parte dos cenários analisados. Nos casos em que houve falha, ela estava relacionada com o armazenamento de dados. Um estudo mais aprofundado sobre essa questão poderia melhorar o desempenho desse dispositivo, pois não foram encontrados indícios de problemas com outras funções. Isso é sugerido como um potencial tema para um trabalho futuro.

8 TRABALHOS FUTUROS

Como possíveis trabalhos futuros, pode-se citar:

- Estudo da dinâmica de armazenamento de dados em sistemas embarcados e a relação entre a tecnologia utilizada de banco de dados e o tipo de memória presente no dispositivo.
- Aplicação do conceito de *fog computing*, utilizando diversos módulos gateway conectados e trocando informações entre si.
- Implementação do módulo central e proposta de uma arquitetura orientada a serviços.

9 REFERÊNCIAS

A VOZ DA INDÚSTRIA, **Ebook: Manufatura Avançada**, 2016.

Advantech. **Enabling an Intelligent Planet.** Disponível em: <http://www.advantech.com.br/>. Acesso em: Janeiro, 2018.

ALFA INSTRUMENTOS. **Protocolo de Comunicação Modbus RTU/ASCII**, Alfa Instrumentos, 2000.

ALVES, T. R. et al. **OpenPLC: An Open Source Alternative to Automation**. In: IEEE Global Humanitarian Technology Conference (GHTC 2014). IEEE, 2014. p. 585-589.

Arduino. Wikipédia, a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/Arduino>. Acesso em: Janeiro, 2017.

ATWELL, C. **The Biggest-Little Revolution: 10 Single-Board Computers for Under $100**. EETimes Open Source. 2013. Disponível em: <http://www.eetimes.com/document.asp?doc_id=1319262?>. Acesso em: Janeiro, 2017.

BASTOS, L.O.; LADEIRA, A. C. **Protocolo HTTP**. 2001.

BCM2837. Disponível em: <https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837/README.md>. Acesso em: Janeiro, 2018.

Beremiz**.** Disponível em: <http://www.beremiz.org/>. Acesso em: Janeiro, 2017.

BERGER, R. **Industry 4.0 – The new industrial revolution**. Strategy Consultants, 2014. Disponível em: <https://www.rolandberger.com/media/pdf/Roland_Berger_TAB_Industry_4_0_20140403.pdf>. Acesso em: Janeiro, 2017.

BORMANN, C. **COAP - Constrained Application Protocol**. 2014. Disponível em: <http://coap.technology/>. Acesso em: Janeiro, 2017.

BORMANN, C.; SHELBY, Z. **Block-Wise Transfers in the Constrained Application Protocol (CoAP).** 2016. Disponível em: <https://tools.ietf.org/html/rfc7959>. Acesso em: Janeiro, 2017.

CALDIÉRI, M.R. **Implementação do Modbus para Aplicação em Sistema de Controle via Rede sem Fio**. Dissertação (Mestrado em Engenharia Elétrica) - Faculdade de Engenharia Elétrica/Universidade Estadual Paulista “Júlio de Mesquita Filho”, Bauru, 2016.

CÂNDIDO, G. et al. **SOA at device level in the industrial domain: Assessment of OPC UA and DPWS specifications.** In: Industrial Informatics (INDIN), 2010 8th IEEE International Conference on. IEEE, 2010. p. 598-603.

CANTELON, M. et al. **Node.js in Action**. Manning, 2014.

CASTELEYN, S. et al. **Engineering Web Applications**. Springer, 2009.

CERTI. **Iniciando com o ScadaBR.** CERTI - ScadaBR. 2016. Disponível em: <https://sites.google.com/a/certi.org.br/certi_scadabr/home/minicursos/iniciando-scadabr>. Acesso em: Agosto, 2016.

Chrome V8. 2017. Disponível em: <https://developers.google.com/v8/>. Acesso em: Outubro, 2017.

COPPER. 2017. Disponível em: <https://github.com/mkovatsc/Copper>. Acesso em: Outubro, 2017.

DECOTIGNIE J. D. **Ethernet-Based Real-Time and Industrial Communications.** Proceedings of the IEEE, v. 93, n. 6, p. 1102-1117, 2005.

DUSSEAULT, L.; SNELL, J. **PATCH method for HTTP.** 2010.

ESP8266. Wikipédia, a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/ESP8266>. Acesso em: Janeiro, 2017.

ESP-COAP. **CoAP protocol for ESP-12E.** Disponível em: <http://github.com/automote/ESP-CoAP>. Acesso em: Janeiro, 2018.

ESPRESSIF, **ESP8266EX Datasheet.** 2017a. Disponível em: <http://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf>. Acesso em: Janeiro, 2018.

ESPRESSIF, **ESP8266 Technical Reference.** 2017b. Disponível em: <http://espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf>. Acesso em: Janeiro, 2018.

EXPRESS. **Framework web rápido, flexível e minimalista para Node.js.** Disponível em: <http://expressjs.com/>. Acesso em: Janeiro, 2018.

FIELDING, R. et al. **Hypertext transfer protocol--HTTP/1.1.** 1999.

FIELDING, R. T.; TAYLOR, R. N. **Architectural styles and the design of network-based software architectures**. PhD dissertation: University of California, Irvine, 2000.

FIELDING, R.; RESCHKE, J. **RFC 7231-Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.** Internet Engineering Task Force (IETF), 2014.

FINK, G. et al. **Pro Single Page Application Development: Using Backbone. Js and ASP. Net.** Apress, 2014.

FLANAGAN, D. **JavaScript: The definitive guide: Activate your web pages.** O'Reilly Media, Inc., 2011.

FREEBOARD, **Dashboard for the Internet of Things.** Disponível em: <https://freeboard.io/>. Acesso em: Agosto, 2018.

GOMES, J. R. R., **Automação Industrial com recurso a ferramentas "Open Source"**, Dissertação (Mestrado Integrado em Engenharia Eletrotécnica e de Computadores) - Faculdade de Engenharia da Universidade do Porto, Portugal. 2014.

GREHS, D.H. **Sistema de Irrigação Doméstico Baseado em Internet das Coisas**. Trabalho de Conclusão de Curso em Engenharia da Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2016.

GRÜNER, S.; PFROMMER, J.; PALM, F. **RESTful Industrial Communication With OPC UA**. IEEE Transactions On Industrial Informatics, v. 12, n. 5, p. 1832-1841, 2016.

GUBBI, J et al. **Internet of Things (IoT): A vision, architectural elements, and future directions.** Future generation computer systems, v. 29, n. 7, p. 1645-1660, 2013.

HEGAZY, T.; HEFEEDA, M. **Industrial Automation as a Cloud Service.** IEEE Transactions on Parallel and Distributed Systems, v. 26, n. 10, p. 2750-2763, 2015.

HiveMQ. **MQTT Essentials Part 8: Retained Messages**. Disponível em:<http://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages>. Acesso em: Novembro, 2016.

HiveMQ. **MQTT Essentials Part 3: Client, Broker and Connection Establishment**. Disponível em: <http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment>. Acesso em: Março, 2017.

HUI, J. W.; CULLER, D. E. **IP is dead, long live IP for wireless sensor networks.** In: Proceedings of the 6th ACM conference on Embedded network sensor systems. ACM, 2008. p. 15-28.

HUGHES-CROUCHER, T.; WILSON, M. **Node: Up and running: Scalable server-side code with JavaScript**. "O'Reilly Media, Inc.", 2012.

HOPPE, S. **There Is No Industrie 4.0 without OPC UA.** 2017**.** Disponível em: <http://opcconnect.opcfoundation.org/2017/06/there-is-no-industrie-4-0-without-opc-ua/>. Acesso em: Novembro, 2017.

IETF, **About the IETF**. Disponível em: <https://www.ietf.org/about/>. Acesso em: Janeiro, 2017.

IZAGUIRRE, M. J. A. G.; LOBOV, A.; LASTRA, J. L. M. **OPC-UA and DPWS interoperability for factory floor monitoring using complex event processing.** In: Industrial Informatics (INDIN), 2011 9th IEEE International Conference on. IEEE, 2011. p. 205-211.

JAFFEY, T. **MQTT and CoAP, IoT Protocols.** Fevereiro, 2014. Disponível em: <http://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php>. Acesso em: Janeiro, 2018.

JAIMINI, U.; DHANIWALA, M. **JavaScript empowered Internet of Things.** In: Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on. IEEE, 2016. p. 2373-2377.

JAMIE M, R. et al. **Sensor Networks and Grid Middleware for Laboratory Monitoring**. In: e-Science and Grid Computing, 2005. First International Conference on. IEEE, 2005. p. 8 pp.-569.

JAMMES, F. et al. **Promising Technologies for SOA-based Industrial Automation Systems**. In: Industrial Cloud-Based Cyber-Physical Systems. Springer, Cham, 2014. p. 89-109.

JSMODBUS. **Simple Modbus TCP Master/Slave implementation for Node.js.** Disponível em: <http://github.com/dcvice1967/jsModbus>. Acesso em: Janeiro, 2018.

JSON. **Introducing JSON.** 2017. Disponível em: <http://www.json.org>. Acesso em: Outubro, 2017.

JUNIOR, F. A. R., **Programação Orientada a Eventos no lado do servidor utilizando Node.js.** 2012.

KAGERMANN, H.; WAHLSTER, W.; HELBIG, J. **Recommendations for implementing the strategic initiative INDUSTRIE 4.0**. 2013.

KOVATSCH, M. **Scalable Web Technology for the Internet of Things.** 2015. Tese de Doutorado. ETH Zurich.

KULKARNI, A., **IoT and SCADA: Complimentary technologies for Industry 4.0**. Altizon Systems, 2016. Disponível em: <http://altizon.com/iotandscadacomplimentarytechnologiesthataresteppingstonetoindustry40/>. Acesso em: dezembro de 2016.

LERCHE, C.; HARTKE, K.; KOVATSCH, M. **Industry adoption of the Internet of Things: A constrained application protocol survey.** In: Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference On. IEEE, 2012. p. 1-6.

LI, D.; SERIZAWA Y.; KIUCHI M. **Concept Design for a Web-Based Supervisory Control and Data Acquisition (SCADA) System**. In: Transmission and Distribution Conference and Exhibition 2002: Asia Pacific. IEEE/PES. IEEE, 2002. p. 32-36.

LIN, Z.; PEARSON, S. **An inside look at industrial Ethernet communication protocols.** Texas Instruments White Paper, 2013.

LYDON, B. **Internet of Things Industrial automation industry exploring and implementing IoT.** InTech Magazine, v. 2, 2014. Disponível em: <https://www.isa.org/standards-and-publications/isa-publications/intech-magazine/2014/mar-apr/cover-story-internet-of-things/>. Acesso em: Agosto, 2016.

MARTINS, I.R.; ZEM, J.L. **Estudo dos Protocolos de Comunicação MQTT e CoAP para Aplicações Machine-To-Machine e Internet das Coisas**. Revista Tecnológica da Fatec Americana, v. 3, n. 1, p. 24, 2016.

MANYIKA, James. **The Internet of Things: Mapping the value beyond the hype.** McKinsey Global Institute, 2015.

MEMCACHED. **A distributed memory object caching system.** Disponível em: <http://memcached.org/>. Acesso em: Janeiro, 2018.

MODBUS. **Modbus Organization**. Disponível em: <http://www.modbus.org/> Acesso em Janeiro de 2017.

MOSCA. **MQTT broker as a module.** Disponível em: <https://github.com/mcollina/mosca>. Acesso em: Novembro, 2017.

Mosquitto. Disponível em: <https://mosquitto.org/>. Acesso em: Janeiro, 2017.

MYSQL. **Sistema de gerenciamento de banco de dados relacional.** Disponível em: <http://www.mysql.com/>. Acesso em: Janeiro, 2018.

NODE-COAP. **A client and server library for CoAP modelled after the http module.** Disponível em: <http://github.com/mcollina/node-coap>. Acesso em: Janeiro, 2018.

NODEMCU DEVKIT. Disponível em: <https://github.com/nodemcu/nodemcu-devkit-v1.0>. Acesso em: Janeiro, 2018.

OASIS, **MQTT Version 3.1.1 Plus Errata 01**. 2015. Disponível em: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>. Acesso em: Fevereiro, 2017.

OpenPLC. 2017. Disponível em: <http://www.openplcproject.com/>. Acesso em: Março, 2017.

OPC. 2017. Disponível em: < https://en.wikipedia.org/wiki/Open_Platform_Communications>. Acesso em: Março, 2017.

OPC FOUNDATION, **What is OPC?** Disponível em: <https://opcfoundation.org/about/what-is-opc/>. Acesso em: Janeiro de 2017.

PAPAZOGLOU, M. P. **Web Services: Principles and Technology**. Prentice-Hall, 2008.

POSTMAN. Disponível em: <https://www.getpostman.com/>. Acesso em: Novembro, 2017.

PUBSUBCLIENT. **A client library for the Arduino Ethernet Shield that provides support for MQTT.** Disponível em: <http://github.com/knolleary/pubsubclient>. Acesso em: Janeiro, 2018.

REISNER, P. **Distributed replicated block device.** In: Int. Linux System Tech. Conf. 2002.

RICHARDSON, L.; RUBY, S. **RESTful Web Services**. Sebastopol, O'Reilly, 2007.

SAUTER, T. **The three generations of field-level networks—Evolution and compatibility issues.** IEEE Transactions on Industrial Electronics, v. 57, n. 11, p. 3585-3595, 2010.

SAUTER, T. et al. **The Evolution of Factory and Building Automation**. IEEE Industrial Electronics Magazine, v. 5, n. 3, p. 35-48, 2011.

SCADABR. **Sistema supervisório Web em licença Open Source.** 2016. Disponível em: < http://www.scadabr.com.br/>. Acesso em: Agosto, 2016.

SCADA-LTS. **Open Source software for Supervisory Control and Data Acquisition.** 2017. Disponível em: < http://scada-lts.org/>. Acesso em: Março, 2017.

SEQUELIZE. **Promise-Based Object-Relational Mapping.** 2018. Disponível em: < http://docs.sequelizejs.com/>. Acesso em: Agosto, 2018.

SHELBY, Z. **Constrained RESTful environments (CoRE) link format.** 2012.

SHELBY, Z.; HARTKE, K.; BORMANN, C. **The Constrained Application Protocol (CoAP)**. 2014.

STANKOVIC, J. A. **Research Directions for the Internet of Things**. IEEE Internet of Things Journal, v. 1, n. 1, p. 3-9, 2014.

SUNDMAEKER, H. et al. **Vision and challenges for realising the Internet of Things**. Cluster of European Research Projects on the Internet of Things, European Commision, v. 3, n. 3, p. 34-36, 2010.

UNGUREAN, I.; GAITAN, N. C.; GAITAN, V. G. **A Middleware Based Architecture for the Industrial Internet of Things.** KSII Transactions on Internet & Information Systems, v. 10, n. 7, 2016.

WIFIMANAGER. **ESP8266 Wi-Fi Connection manager with fallback web configuration portal.** Disponível em: <http://github.com/tzapu/WiFiManager>. Acesso em: Janeiro, 2018.

WOLLSCHLAEGER, M.; SAUTER, T.; JASPERNEITE, J. **The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0.** IEEE Industrial Electronics Magazine, v. 11, n. 1, p. 17-27, 2017.

ZUEHLKE, D. **Smart Factory—Towards a factory-of-things**. Annual Reviews in Control, v. 34, n. 1, p. 129-138, 2010.