UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

GABRIEL SARTORIO PANDOLFI

AVALIAÇÃO EXPERIMENTAL DA TECNOLOGIA DE COMUNICAÇÃO *ZIGBEE* NO CONTEXTO DE IIOT

VITÓRIA
2020

GABRIEL SARTORIO PANDOLFI

AVALIAÇÃO EXPERIMENTAL DA TECNOLOGIA DE COMUNICAÇÃO *ZIGBEE* NO CONTEXTO DE IIOT

Parte manuscrita do Projeto de Graduação do aluno **Gabriel Sartorio Pandolfi**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Dr. Jair A. L. Silva
Coorientador: Eng. Menno J. Faber

VITÓRIA
2020

GABRIEL SARTORIO PANDOLFI

AVALIAÇÃO EXPERIMENTAL DA TECNOLOGIA DE COMUNICAÇÃO *ZIGBEE* NO CONTEXTO DE IIOT

Parte manuscrita do Projeto de Graduação do aluno **Gabriel Sartorio Pandolfi**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Aprovada em 05 de fevereiro de 2020.

COMISSÃO EXAMINADORA:

_____

Prof. Dr. Jair A. L. Silva
Universidade Federal do Espírito Santo
Orientador

_____

Eng. Menno J. Faber
Universidade Federal do Espírito Santo
Coorientador

_____

Prof. Dr. Helder R. de O. Rocha
Universidade Federal do Espírito Santo
Examinador

_____

Eng. Wesley Costa
Universidade Federal do Espírito Santo
Examinador

RESUMO

Neste trabalho é apresentado um Projeto de Graduação do Curso de Engenharia Elétrica da Universidade Federal do Espírito Santo, cujo desenvolvimento faz parte do principal objetivo da unidade curricular denominada Projeto de Graduação II. Neste, são abordados temas referentes aos avanços na área da Tecnologia da Informação, Internet das Coisas (*IoT, do inglês Internet of Things)*, Indústria 4.0 e Redes de Sensores sem Fio, bem como suas aplicações e atual estado da arte. Também é feita uma comparação entre os novos protocolos de comunicação que emergiram com a *IoT*. O projeto de uma rede de sensores sem fio exige não somente a implementação dos módulos sensores e de comunicação, mas também a utilização de um *software* para gerenciamento dos dados coletados. Dessa forma, este trabalho contempla a implementação de uma rede *ZigBee* voltada para ambiente industrial, com possibilidade de acesso remoto das variáveis de controle. Foram avaliadas experimentalmente diferentes topologias de rede, em especial a topologia *mesh* e a comunicação ponto-a-ponto. Medidas de desempenho foram tomadas para diferentes caracterizações da rede, de modo a evidenciar a viabilidade e vantagens da utilização da tecnologia *ZigBee* em ambientes industriais. Características como robustez, baixo consumo de energia e capacidade de auto-regeneração foram comprovadas e justificam o uso desse meio de comunicação. A parceria da Universidade Federal do Espírito Santo com o Centro de Pesquisa, Inovação e Desenvolvimento (CPID) e a empresa *2Solve* facilitou a disponibilização do material necessário para a realização da pesquisa.

Palavras-chave: Internet das Coisas. Indústria 4.0. *ZigBee.* Tecnologia da Informação.

LISTA DE FIGURAS

Figura 1 – Relação entre *IoT, IIoT,* Indústria 4.0 e *CPS* 14
Figura 2 – Esquema estrutural da rede 15
Figura 3 – Representação de um sistema *IoT* 17
Figura 4 – Rede de sensores sem fio típica 22
Figura 5 – Componentes de um nó sensor 22
Figura 6 – Comparação entre *ZigBee*, *Bluetooth* e *Wi-Fi* 26
Figura 7 – Camadas de rede *ZigBee* 27
Figura 8 – Estrutura dos pacotes *ZigBee* 28
Figura 9 – Exemplo de rede *ZigBee* 30
Figura 10 – Tipos de topologia de uma rede *ZigBee* 31
Figura 11 – Tipos de antenas dos módulos *XBee* 32
Figura 12 – Estrutura do quadro de dados da *API* 33
Figura 13 – Pinagem do *XBee* 34
Figura 14 – Modos de operação do *XBee* 35
Figura 15 – Interface *X-CTU* 36
Figura 16 – Placa Arduino Uno 38
Figura 17 – Ambiente Integral de Desenvolvimento Arduino 39
Figura 18 – *Seeeduino Stalker V2.0* 40
Figura 19 – Arduino Uno com *XBee shield* 40
Figura 20 – Módulo *IC* 41
Figura 21 – Sensores utilizados para (a) humidade e temperatura, (b) temperatura e (c) som 42
Figura 22 – Módulos sensores denominados (a) kit A, (b) kit B, (c) ruído sonoro e DHT11 e (d) coordenador 43
Figura 23 – Nós remotos descobertos pelo *X-CTU* 45
Figura 24 – Painel *Network* do *X-CTU* 46
Figura 25 – Visualização da rede em modo tabela 47
Figura 26 – Nó servidor *Ethernet* 48
Figura 27 – Página *HTML* em rede local 49
Figura 28 – Ambiente de programação do *Node-RED* 50
Figura 29 – Painel de controle desenvolvido 50
Figura 30 – Topologias usadas nos testes 51

Figura 31 – Disposição dos rádios de comunicação 51
Figura 32 – Tráfego gerado na rede 54
Figura 33 – Ambiente para testes de distância 56
Figura 34 – Dados obtidos no teste RSSI 56
Figura 35 – Perda de pacotes 57
Figura 36 – Ambiente para testes de *throughput* 58
Figura 37 – *Throughput* em função da distância 59
Figura 38 – *Delay* na entrega de pacotes 60
Figura 39 – Topologia usada no teste de auto regeneração 60
Figura 40 – Chaves de configuração 62

LISTA DE QUADROS

Quadro 1 – Comparação técnica entre dispositivos de curto alcance de transmissão sem fio14
Quadro 2 – Comparação entre *IoT* e *IIoT* 19
Quadro 3 – Faixas de frequência definidas pelo padrão *IEEE* 802.15.4 24
Quadro 4 – Configuração dos nós sensores 44
Quadro 5 – Indicação da qualidade do sinal no *X-CTU* 46
Quadro 6 – Tamanho dos pacotes transmitidos 54

LISTA DE TABELAS

Tabela 1 – Consumo dos módulos 52

LISTA DE ABREVIATURAS E SIGLAS

4G	Quarta Geração da Telefonia
5G	Quinta Geração da Telefonia
AES	*Advanced Encryption System*
AODV	*Ad Hoc On Demand Distance Vector*
API	*Application Programming Interface*
AT	*Transparent Operation*
BLE	*Bluetooth Low Energy*
C1RSSF	Rede de Sensores Sem Fio de Categoria 1
C2RSSF	Rede de Sensores Sem Fio de Categoria 2
CPS	*Cyber-physical System*
FFD	*Full Function Device*
IIoT	*Industrial Internet of Things*
IoT	*Internet of Things*
LED	*Light Emitting Diode*
LoRa	*Long Range*
MAC	*Medium Access Control*
NFC	*Near Field Communication*
PAN	*Personal Area Network*
RFD	*Reduced Function Device*
RSSF	Rede de Sensores Sem Fio
SFC	*Short Field Communication*
TI	Tecnologia da Informação
UWB	*Ultra Wide Band*
WAN	*Wide Area Network*
Wi-Fi	*Wireless Fidelity*

LISTA DE SÍMBOLOS

Hz	Hertz
kbps	Quilobits por segundo
mA	Miliampère
ms	Milissegundo
V	Volt

SUMÁRIO

1 INTRODUÇÃO 12

1.1 Apresentação 12

1.2 Motivação 15

1.3 Objetivos 16

1.3.1 Objetivo Geral 16

1.3.2 Objetivos Específicos 16

2 A INTERNET DAS COISAS 17

2.1 Conceito 17

2.2 Indústria 4.0 e *Industrial Internet of Things* 18

3 REDES SEM FIO 20

3.1 Padronização das redes sem fio – *IEEE* 802.11 20

3.2 Redes de Sensores sem Fio 21

3.3 Redes sem fio pessoais de baixas taxas de transmissão - *IEEE* 802.15.4 23

***4 ZIGBEE* 25**

4.1 *ZigBee* Alliance 26

4.2 Arquitetura da pilha *ZigBee* 26

4.2.1 Camada Física (*PHY*) 27

4.2.2 Camada de controle de acesso ao meio (*MAC*) 28

4.2.3 Camada de rede (*NWK*) 28

4.2.4 Camada de aplicação (*APL*) 29

4.2.5 Segurança 29

4.3 Rede *ZigBee* 30

4.3.1 Tipos de dispositivos 30

4.3.2 Topologias da rede 31

4.3.3 Roteamento 31

4.4 Rádio *XBee* 32

4.4.1 Envio e Recepção de Dados 33

4.4.2 Portas E/S 34

4.4.3 Modos de Operação 35

4.5 *X-CTU* 36

5 PLATAFORMAS DE PROTOTIPAGEM 37

5.1 Arduino 37

5.2 *Seeeduino Stalker* V2 ..... **39**
5.3 *XBee Shield* **39**
5.4 IC **41**
6 MONITORAMENTO DA REDE DE SENSORES SEM FIO **42**
6.1 Desenvolvimento dos kits de monitoramento **42**
6.1.1 Microcontroladores e sensores 42
6.2 Configuração dos rádios *XBee* **44**
6.3 Aplicação de monitoramento **47**
6.4 Avaliação da capacidade dos rádios de comunicação **51**
7 RESULTADOS OBTIDOS **52**
7.1 Aplicação de monitoramento da rede de sensores **52**
7.1.1 Autonomia da rede 52
7.1.2 Tráfego de dados 53
7.2 Avaliação experimental do potencial da rede **54**
7.2.1 Medição de *RSSI* 55
7.2.2 Vazão de dados 57
7.2.3 Atraso na entrega de pacotes 59
7.2.4 Tempo de auto regeneração da rede 60
7.3 Desafios para a implantação do sistema **61**
8 CONCLUSÕES **63**
REFERÊNCIAS BIBLIOGRÁFICAS **64**
APÊNDICE A – CÓDIGOS EM MODO *API* **67**
APÊNDICE B – CÓDIGOS EM MODO *AT* **71**

1 INTRODUÇÃO

1.1 Apresentação

Em um futuro não muito distante, computadores poderão ser considerados onipresentes num domínio global. Uma vez que a capacidade humana seja superada pela capacidade das máquinas, um vasto leque de opções se abre, dando espaço às mais diversas aplicações provenientes dos avanços tecnológicos adquiridos com o nascimento da Internet em 1989. Com o advento das áreas da eletroeletrônica, sistemas embarcados, microeletrônica e comunicações surgiu o termo Internet das Coisas (*IoT,* do inglês *Internet of Things).* "Coisas" se refere à imensa variedade de objetos que está à nossa volta, desde os pequenos *microchips* aos maiores carros e navios. Num contexto de *IoT*, objetos inteligentes estão todos interconectados e permitem a um usuário final ter controle sobre estes dispositivos, conectando o mundo físico ao mundo da informação (*CPS*, do inglês *Cyber-physical system*). Aliada à evolução da presença do *Wi-Fi* (do inglês *Wireless Fidelity)* e das redes de dados móveis de quarta e quinta gerações (4G e 5G), a *IoT* demonstra uma tendência da informação se tornar, um dia, onipresente (PRESS, 2014).

Essa tendência é evidente quando o radical crescimento da *IoT* é avaliado quantitativamente: já no ano de 2011 o número de dispositivos interconectados no planeta se igualou à quantidade de pessoas que nele vivem, cerca de sete bilhões de pessoas. Estimativas apontam que o número de dispositivos interconectados no planeta chegará a setenta e cinco bilhões até o ano de 2030 (COLUMBUS, 2018). Segundo a *International Data Corporation* (TORCHIA; SHIRER, 2019), os gastos anuais com *IoT* vão superar a marca de um trilhão de dólares até 2022. As áreas da indústria mais propensas a investimentos em *IoT* são as áreas da manufatura e manufatura processada, com foco em soluções que ofereçam suporte aos processos de manufatura e gerenciamento de produção; transportes, especialmente relacionado ao sensoriamento de peso e controle de tráfego; e utilidades, com gastos direcionados a *smart grids* para eletricidade, gás e água.

Digitalização em massa somada aos avanços em *IoT* serão responsáveis por transformar o modelo da indústria atual. A capacidade de objetos físicos de se comunicarem e sensoriarem o ambiente ao seu redor gera inúmeras possibilidades, trazendo à tona o termo Indústria 4.0.

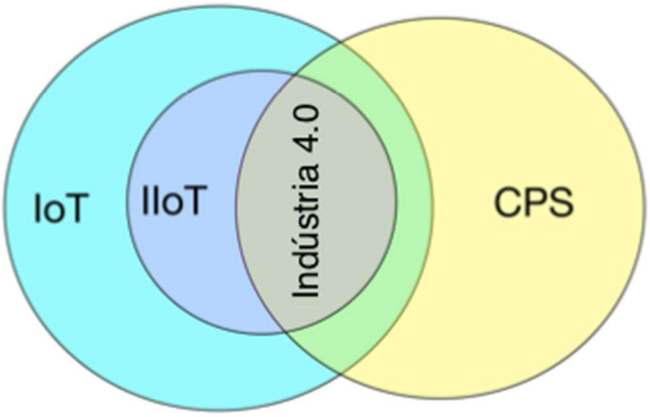
Três revoluções industriais, a saber, do uso do vapor, eletricidade e tecnologia de semicondutores progrediram ao longo de muitas décadas, a partir do final do século XVIII. Na revolução industrial mais recente, onde se desenvolveram tecnologias advindas dos semicondutores, surgiram os transistores e os circuitos integrados, seguidos pela mecanização e automação. Esses processos desenvolveram-se e avançaram a passos curtos, enquanto a Quarta Revolução Industrial, que acontece atualmente, progride exponencialmente e torna obsoletos muitos processos (muitas vezes caracterizada como uma tecnologia disruptiva) e sistemas existentes, enquanto cria soluções totalmente novas e inovadoras (GRIFFITHS; OOI, 2018).

O termo Indústria 4.0 surgiu em 2011 numa promoção do uso da digitalização na indústria pelo governo da Alemanha (BMBF, 2016). Desde então, tem sido foco dos investimentos daquele país, que mantém sua posição de líder mundial em automação. A proliferação de máquinas interconectadas, unida aos avanços na segurança das redes de comunicação e análise de dados, tem ajudado na melhor utilização de equipamentos e reduzindo o *downtime*, período em que uma fábrica está parada, sem produção, das indústrias alemãs.

A relação entre a indústria e o *IoT* deu origem a uma subdivisão da Internet das Coisas (Figura 1), denominada Internet das Coisas Industrial (*IIoT*, do inglês *Industrial Internet of Things*), que engloba os domínios da comunicação máquina-máquina (*M2M*, do inglês *machine-to-machine*) e comunicação industrial com aplicações de automatização. *IIoT* prepara o caminho para um melhor entendimento e, consequentemente, utilização mais eficiente e sustentável do processo de produção (SISINNI *et al.*, 2018).

Aplicações industriais de *IoT* requerem uma vazão de dados relativamente pequena, portanto, a capacidade de transmissão de dados não é um problema. No entanto, a necessidade de se conectar uma grande quantidade de dispositivos, com baixo custo e restrições de poder de processamento e fontes de energia, torna fatores como latência, eficiência energética, confiabilidade e segurança mais atraentes. Nesse contexto, soluções como a tecnologia *ZigBee* foram desenvolvidas para atender a essa demanda.

Figura 1 – Relação entre *IoT, IIoT,* Indústria 4.0 e *CPS*



Fonte: Sisinni e outros. (2018).
Nota: Adaptado pelo autor.

O protocolo *ZigBee* (Alliance, 2005) foi especificado pela *ZigBee Alliance* em 2006 e se baseia nas definições do protocolo *IEEE* 802.15.4 para as camadas física e MAC, adicionando as camadas de rede e aplicação da pilha de protocolos de comunicação, além da criptografia *AES* (do inglês *Advanced Encryption Standard*) de 128 bits, podendo endereçar até 65535 nós em cada sub-rede. O protocolo *ZigBee* usa modulação *DSSI* (do inglês *Direct-Sequence Spread Spectrum*), operando em um canal fixo dos 16 possíveis definidos pelo *IEEE* 802.15.4. O Quadro 1 mostra a comparação técnica do *ZigBee* com outras tecnologias de transmissão sem fio de curto alcance.

Quadro 1 – Comparação técnica entre dispositivos de curto alcance de transmissão sem fio

Características técnicas	***ZigBee***	**WiFi**	**Bluetooth**
Frequência de trabalho (MHz)	2400/868/915	2400	2400
Alcance de transmissão (m)	10~100	100	10
Velocidade de transmissão (kbps)	20/250	11000	1000
Número máximo de nós	65535	32	8
Autonomia da bateria	Muito alta	Média	Baixa
Consumo de energia	Baixo	Alto	Muito Alto
Custo ($)	30	100	60

Fonte: Tose (2012).

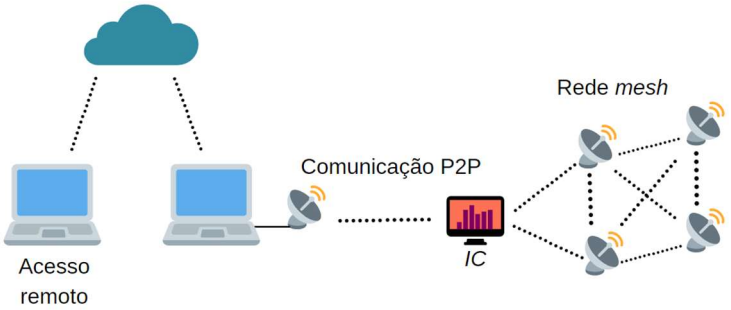
Nesta perspectiva, este trabalho se propôs a avaliar experimentalmente o protocolo *ZigBee* na comunicação de uma rede de sensores sem fio, simulando um ambiente industrial. Objetivou-se, ainda, analisar indicadores de desempenho da comunicação entre os dispositivos da rede e monitorar remotamente as variáveis de controle do sistema de sensoriamento.

1.2 Motivação

Atualmente, há uma grande quantidade de padrões e protocolos que estabelecem as características da transmissão de dados para canais de voz, vídeo, redes de computadores pessoais e industriais, entre outros. Porém, ainda são muito recentes os padrões que definem a comunicação entre sensores e controladores, seja num ambiente residencial voltado para *IoT* e automação de casas inteligentes ou num ambiente industrial, ligado ao sensoriamento e controle de plantas, automatização de processos e gerenciamento de equipamentos.

Em um ambiente industrial, confiabilidade e autonomia energética são pré-requisitos para uma rede de sensores. Tais fatores, aliados a distâncias mais curtas entre os nós sensores, fazem com que a tecnologia *ZigBee* seja a mais indicada. Ainda, para acesso remoto e controle por parte do usuário final, a informação deve percorrer um caminho mais longo, desde a planta industrial até um supervisório distante. Dessa maneira, este estudo se propôs a avaliar experimentalmente a viabilidade do padrão de comunicação *ZigBee* para os dois propósitos: rede *mesh* para menores distâncias e comunicação ponto-a-ponto para distâncias mais elevadas, conforme ilustra a Figura 2.

Figura 2 – Esquema estrutural da rede



Fonte: Produção do próprio autor.

A parceria entre o Centro de Pesquisa, Inovação e Desenvolvimento (CPID) e a empresa *2Solve* com o curso de Engenharia Elétrica da Universidade Federal do Espírito Santo encoraja e torna possível a realização de pesquisas na área de *IoT* residencial e industrial, por meio da disponibilização de módulos de comunicação *XBee* para fins acadêmicos. Este trabalho ainda servirá de base para projetos futuros a serem desenvolvidos na Universidade.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo deste trabalho foi implementar uma rede *ZigBee* de sensores sem fio com topologia *mesh* e, através de dispositivos concentradores de dados presentes na rede, monitorar as variáveis de controle e fazer um levantamento de medidas de desempenho, como vazão (*throughput*), tempo de resposta, indicação da potência do sinal recebido (RSSI, do inglês *Recieved Signal Strength Indication*) e a taxa de perda de pacotes.

1.3.2 Objetivos Específicos

Para alcançar o objetivo geral deste trabalho, os seguintes objetivos específicos foram almejados:

- Aprofundar os estudos do estado da arte do protocolo *ZigBee*, para melhor entendimento sobre o seu funcionamento, as diferenças entre as topologias da rede e as características dos tipos de dispositivos (*End Device,* Roteadores e Coordenadores);
- Implementação de uma rede *mesh* de sensores sem fio baseando-se no padrão *ZigBee* e realização de medidas dos indicadores de desempenho da rede;
- Estudos sobre o funcionamento do módulo de comunicação IC, suas configurações e os modos de operação;
- Por intermédio do dispositivo *IC,* desenvolvido pela empresa *2Solve,* desenvolvimento de uma interface de monitoramento em tempo real dos dados coletados pela rede de sensores.
- Levantamento de medidas de desempenho da comunicação na RSSF e avaliação da eficácia da tecnologia *ZigBee* em um contexto de *Industrial Internet of Things*.

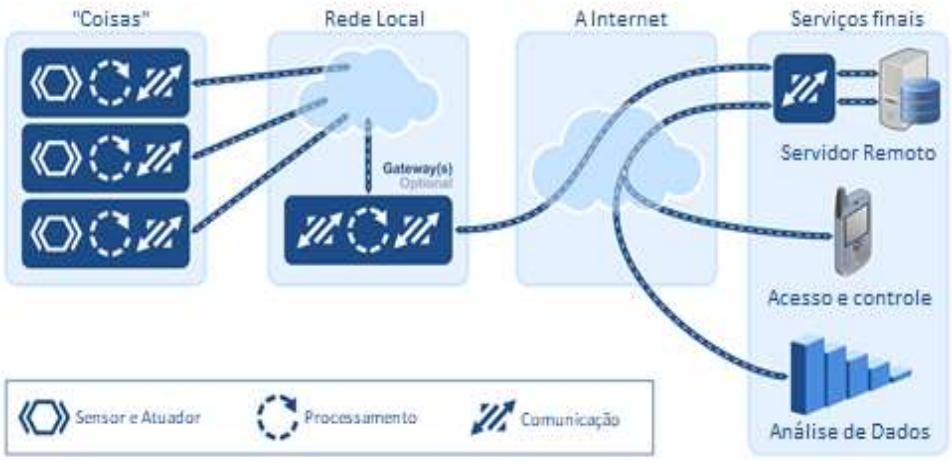
2 A INTERNET DAS COISAS

2.1 Conceito

Em um contexto de *IoT*, um objeto deve ser composto, principalmente, por uma fonte de energia, processador, memória, unidade de sensoriamento e transceptor para comunicação sem fio (ILYAS; MAHGOUB, 2004). A principal característica da *IoT* é a interconexão entre o homem e os objetos ao seu redor. Portanto, é necessária uma ponte que ligue o mundo físico com o mundo da informação (*CPS*). A unidade de sensoriamento de um dispositivo num ambiente de *IoT* é o que faz essa ligação entre os dois mundos, servindo de interface entre o usuário e o equipamento (CONNER, 2010).

O papel de um sensor é coletar variáveis físicas do mundo real e converter numa linguagem que outros equipamentos e máquinas da rede possam entender. Então, esses dados são tratados em uma unidade de processamento e preparados para serem transmitidos a outros dispositivos na rede local. Com a conexão da rede local à internet, os dados coletados pelos sensores são disponibilizados por um servidor remoto para que um usuário final possa acessar esses dados e agir em resposta. Esse processo está descrito na Figura 3.

Figura 3 – Representação de um sistema *IoT*



Fonte: Micrium (2019).
Nota: Adaptado pelo autor.

O surgimento do termo *IoT* tem feito com que o conceito de redes de computadores se altere com o passar dos anos. Para Tanenbaum (2003), uma rede de computadores é um conjunto de computadores autônomos interconectados por uma única tecnologia, tal que a conexão entre

esses computadores não é, necessariamente, via fio de cobre, podendo ser também utilizadas fibras ópticas, micro-ondas, ondas de infravermelho e satélites de comunicação. Nota-se, segundo Forouzan (2009), que o conceito de redes já não se limita a computadores, sendo um conjunto de dispositivos (também conhecidos como nós) conectados por *links* de comunicação. Um nó pode ser um computador, uma impressora ou outro dispositivo de envio e recepção de dados que estejam conectados a outros nós da rede. Kurose e Ross (2013) reforçam a ideia de Forouzan (2009), ao argumentar que o termo "Redes de Computadores" já não é o mais adequado, dada a enorme quantidade de dispositivos não tradicionais interconectados.

2.2 Indústria 4.0 e *Industrial Internet of Things*

O termo *IoT* é comumente utilizado com referência a uma utilização voltada ao ser humano, onde as "Coisas" são dispositivos eletrônicos inteligentes que auxiliam nossa percepção do ambiente ao redor. No mundo industrial, o foco é a integração entre tecnologia operacional e tecnologia da informação e um novo termo se tornou mais adequado: Internet das Coisas Industrial (*IIoT,* do inglês *Industrial Internet of Things*).

Pilar fundamental da manufatura digital, a *IIoT* trata da interconexão entre os componentes de uma indústria, como máquinas e sistemas de controle com sistemas de informação. A comunicação em *IIoT*, ao contrário da *IoT*, é orientada à máquina e abrange uma grande quantidade de setores e atividades. Uma comparação qualitativa entre as duas tecnologias pode ser observada no Quadro 2.

Em ambientes industriais, processos automatizados podem ser divididos em três subcategorias: monitoramento/supervisão, controle em malha fechada e interconexão e controle. Enquanto aplicações de monitoramento e supervisão são menos sensíveis a perdas de pacotes e toleram pequenos atrasos na transmissão, aplicações de controle em malha fechada e interconexão requerem atraso máximo de 100 ms e confiabilidade de 99,99% (AKERBERG; GIDLUND; BJORKMAN, 2011).

Comparando o volume de dados gerados, aplicações em *IoT* apresentam muita variação conforme seu tipo, enquanto em *IIoT* o objetivo é a análise dados, por exemplo, para manutenção preventiva e melhoramentos logísticos, o que implica num enorme volume de

dados. De acordo com a *Cisco Global Cloud Index*, exemplificando, uma mineradora no Rio Tinto gera cerca de 2,4 TB por minuto (CISCO, 2014).

Quadro 2 – Comparação entre *IoT* e *IIoT*

	IoT	**IIoT**
Impacto	**Revolução**	**Evolução**
Modelo de serviço	Voltado ao homem	Orientado à máquina
Estado atual	Novos dispositivos e protocolos	Dispositivos e protocolos existentes
Conectividade	*Ad hoc* (infraestrutura não tolerada; nós podem ser móveis)	Estruturada (nós fixos; gerenciamento de rede centralizado)
Criticalidade	Pouco rigoroso (exceto aplicações médicas)	Muito rigoroso (tempo de resposta, confiabilidade, segurança e privacidade)
Volume de dados	Médio - Grande	Grande - Muito Grande

Fonte: Sisinni e outros. (2018).
Nota: Adaptado pelo autor.

Uma arquitetura *IIoT* tem como destaque a extensibilidade, escalabilidade, modularidade e interoperabilidade entre diferentes dispositivos usando diferentes tecnologias. Diversas estruturas de arquiteturas para *IIoT* foram criadas e a abordagem mais comumente utilizada é uma descrição multicamadas organizada em torno dos serviços oferecidos por cada camada, dependendo da aplicação (WEYRICH; EBERT, 2016). A *ITU* (*International Telecommunication Union*) sugere que essa arquitetura possua cinco camadas: sensoriamento, acesso, rede, tecnologias intermediárias e aplicação.

Para implementação dos conceitos de *IIoT* é imprescindível o conhecimento de redes de sensores sem fio e protocolos de comunicação voltados para *IoT*.

3 REDES SEM FIO

Redes sem fio, do inglês *wireless*, baseia-se no estabelecimento da comunicação sem a utilização de fios, utilizando o ar como meio de transporte da informação. A ideia de se transmitir e receber dados pelo ar atraiu a atenção de diversas empresas, para inúmeras aplicações. Portanto, para garantir a interoperabilidade entre dispositivos, fez-se necessários padronizar as redes *wireless*.

3.1 Padronização das redes sem fio – *IEEE* 802.11

O propósito do padrão apresentado pelo *IEEE* é garantir conectividade sem fio para estações de trabalho fixas, portáteis e móveis, dentro de uma área local. Além disso, o padrão oferece a órgãos regulatórios um meio de padronizar o acesso a uma ou mais faixas de frequência para o propósito de comunicação numa área local (CHEN *et al.*, 1999).

Em virtude de seus raios de alcance, os tipos de redes sem fio foram divididos em quatro grandes grupos:

Wireless Personal Area Network (WPAN) – são as redes pessoais com pequeno alcance, que geralmente atende a um indivíduo. Desenvolvidas pelo Grupo 15 do *IEEE*, destacam-se o *Bluetooth*, Infravermelho e *ZigBee*.

Wireless Local Area Network (WLAN) – trata-se de uma rede local com alcance limitado a um raio de 100 a 300 metros, muito comum em ambientes com grande número de dispositivos conectados, como escritórios, *shopping centers*, residências e instituições de ensino. Tem como principal representante o *Wi-Fi,* padronizado pelo *IEEE* 802.11b.

Wireless Metropolitan Area Network (WMAN) – são redes metropolitanas geralmente corporativas que atravessam cidades e estados. Normatizadas segundo o *IEEE* 802.16, tem como protocolo mais conhecido o *WiMAX*.

Wireless Wide Area Network (WWAN) – também conhecida como rede continental, é uma rede de grande abrangência. Muito utilizada em serviços de voz e dados, sendo os telefones celulares os principais dispositivos presentes nessa rede.

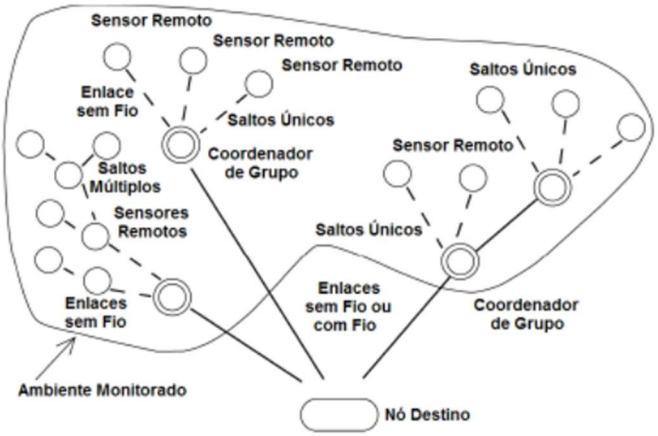
3.2 Redes de Sensores sem Fio

Redes de sensores sem fio (RSSF), devido ao seu raio de alcance, caracterizam-se como redes pessoais ou locais, dependendo da aplicação. Uma RSSF é composta de um número de nós sensores e compreende unidades de sensoriamento, processamento e comunicação que permitem ao usuário manusear, observar e agir sobre um evento ou fenômeno em um dado ambiente. O usuário pode ser um civil, entidade governamental, comercial ou industrial enquanto o ambiente trata-se do mundo físico, um sistema biológico ou uma estrutura de TI. Aplicações típicas de RSSF incluem acumulação de dados, monitoramento e vigilância, telemetria médica, controle e ativação, entre outros (SOHRABY; MINOLI; ZNATI, 2007).

Em uma RSSF há quatro componentes básicos: (1) um arranjo distribuído ou concentrado de sensores; (2) uma rede interconectada (geralmente sem fio); (3) um ponto central de agrupamento de informações; e (4) uma unidade de processamento de dados no ponto central para lidar com a correlação de dados adquiridos (SOHRABY; MINOLI; ZNATI, 2007). A Figura 4 ilustra o contexto de uma RSSF típica, onde sensores remotos desempenham a função de monitoramento e detecção de eventos e transmitem a informação coletada ao coordenador de grupo. Tal transmissão pode ocorrer por saltos únicos ou múltiplos, dependendo das condições do projeto, como distância entre os nós sensores, consumo de energia e tempo de resposta. O coordenador de grupo desempenha funções mais robustas, como o processamento dos dados de seu grupo e a transmissão a um nó destino, geralmente em maiores distâncias, dos dados processados. Um nó destino recebe informação de diversos coordenadores espalhados pela rede (SOUSA; LOPES, 2011).

Um nó sensor é composto, basicamente, por um rádio de comunicação para envio e recepção de dados, conversores analógico-digitais (ADC) que processam os sinais recebidos. Opcionalmente, um nó sensor pode ter dispositivos de acionamento, como relés ou LEDs, que são acionados conforme o sinal recebido. Por fim, um nó sensor necessita de alimentação, geralmente por bateria. A composição de um nó sensor pode ser observada na Figura 5.

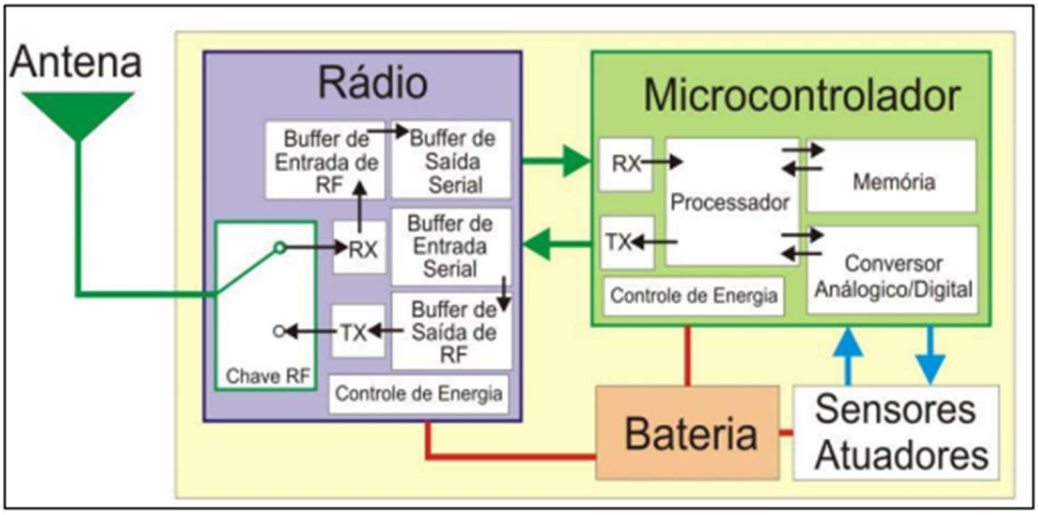
Figura 4 – Rede de sensores sem fio típica



Fonte: Sousa e Lopes (2011).

Quando um dado é coletado pelo sensor, ele passa por um conversor analógico/digital e é processado para ser enviado para o rádio (TX). No rádio há *buffers* para envio e para recepção de dados, o que permite o controle do fluxo de transmissão e recepção (TX/RX). Pelo fato de a transmissão e a recepção não ocorrerem simultaneamente, faz-se necessária uma chave RF para alternar entre os modos de operação. Ambos, rádio e microcontrolador, possuem sistemas para gerenciamento de energia (RIVERO, 2011).

Figura 5 – Componentes de um nó sensor



Fonte: Rivero (2011).

Segundo Sohraby, Minoli e Znati (2007), RSSFs comerciais podem ser divididas em duas categorias:

- RSSFs Categoria 1 (C1RSSFs): sistemas em malha com conectividade de rádio por saltos múltiplos entre os nós sensores da rede e entre redes diferentes. Utilizam roteamento dinâmico tanto na porção sem fio quanto na porção cabeada da rede. Normalmente, sistemas militares pertencem a essa categoria;
- RSSFs Categoria 2 (C2RSSFs): sistemas ponto-a-ponto ou multiponto-ponto com topologia estrela, com conectividade de rádio por saltos únicos, utilizando roteamento estático na rede sem fio. Sistemas de controle residenciais geralmente estão incluídos nessa categoria.

C1RSSFs suportam aplicações com um alto número de nós distribuídos em espaços maiores, como monitoramento ambiental e sistemas de segurança nacional; C2RSSFs comportam espaços confinados, como casas, prédios, indústrias ou o próprio corpo humano. A tendência é que C1RSSFs lidem com sistemas multiponto-ponto de larga escala, com enorme tráfego de dados, enquanto C2RSSFs tendem a focar em sistemas ponto-a-ponto de curta distância, com uma base de dados transmitidos já definida.

3.3 Redes sem fio pessoais de baixas taxas de transmissão - *IEEE* 802.15.4

Por vários anos, comerciantes utilizaram tecnologia própria para coletar dados de desempenho de dispositivos de sensoriamento e, ainda no início da década de 2000, fornecedores procuravam uma maneira de padronizá-los. Geralmente, RSSFs transmitem pequenos volumes de dados, como verificação se a temperatura de um ambiente atingiu o *set-point*. Para ambientes internos de curta distância, projetistas descartaram o padrão *Wi-Fi* por apresentar alta complexidade e suportar uma largura de banda maior do que a realmente necessária para sensores comuns. Sistemas infravermelho necessitam de um campo de visão disponível, o que nem sempre é possível. *Bluetooth* (*IEEE* 802.15.1) foi considerado uma possibilidade, porém também foi descartado pela alta complexidade e elevado custo. Isso abriu portas para o novo padrão *IEEE* 802.15.4, juntamente ao *ZigBee*. C2RSSFs têm as camadas mais inferiores da pilha de protocolos de comunicação (camadas física e controle de acesso ao meio), equivalentes a uma rede de área pessoal (*PAN*) (CHI; HATLER, 2004). O padrão *IEEE* 802.15.4 opera na faixa de frequência de 2,4 GHz (2400 MHz a 2483,5 MHz) e suporta taxa de transmissão de dados de até 250 kbps em distâncias de 10 a 60 metros, por este motivo recomendado para

RSSF. Entretanto, outras faixas de frequência são utilizadas em alguns lugares do mundo, com variação no número de canais de comunicação e na velocidade máxima da transmissão de dados, como pode ser observado no Quadro 3.

O protocolo *IEEE* 802.15.4 especifica a camada física e o controle de acesso (camadas 1 e 2 do modelo *OSI* (do inglês *Open System Interconnection*)) para redes sem fio pessoais com baixa taxa de transmissão, deixando as camadas superiores livres para implementação, dependendo da utilização, como é o caso do *ZigBee*.

O protocolo também define dois tipos de dispositivos físicos: Dispositivos de Funcionamento Completo (*FFD,* do inglês *Full Function Device)* e Dispositivos de Funcionamento Reduzido (*RFD*, do inglês *Reduced Function Device)*. Um nó FFD é responsável pelo roteamento da rede, ditando a sua topologia e fazendo o endereçamento dos outros nós conectados, cumprindo o papel, também, de um nó coordenador da rede. Já os nós RFD são dispositivos de menor complexidade e baixo custo, que só podem se conectar a um nó FFD de cada vez. Por meio do roteamento é possível uma mensagem ser enviada de um nó FFD ou RFD até outro nó FFD e chegar num coordenador da rede. Usualmente nós FFD necessitam de alimentação constante, pois precisam estar sempre ativos, enquanto nós RFD são alimentados por baterias.

Quadro 3 – Faixas de frequência definidas pelo padrão *IEEE* 802.15.4

Frequência	868,3 MHz	902-928 MHz	2400-2483,5 MHz
Número de Canais	1	10	16
Taxa de Transmissão (kbps)	20	40	250
Local	Europa	América/Austrália	Mundial

Fonte: Kohvakka e outros. (2006).
Nota: Adaptado pelo autor.

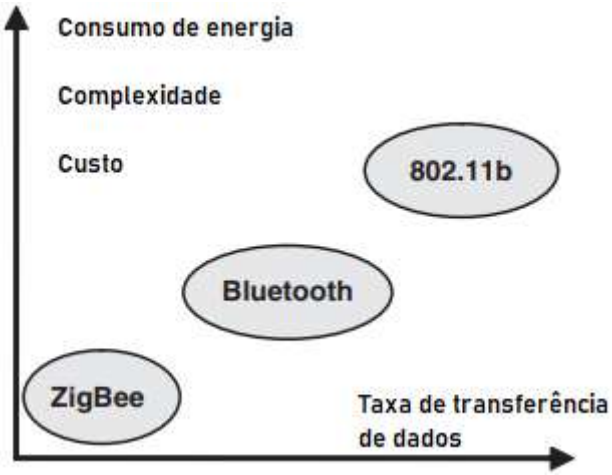
4 *ZIGBEE*

ZigBee é um padrão que define um conjunto de protocolos de comunicação para redes sem fio de baixa taxa de transmissão (*LR-WPAN,* do inglês *Low Rate Wireless Personal Area Network)*. Dispositivos baseados na tecnologia *ZigBee* podem operar em diferentes faixas de frequência, conforme estabelece a padronização *IEEE* 802.15.4 e mostrado no Quadro 3. Para aplicações alimentadas por baterias, onde há baixa taxa de transferência de dados, baixo custo e grande duração da bateria é muito recomendado o uso do padrão *ZigBee* de comunicação, uma vez que, devido às funções de economia de energia, dispositivos podem funcionar por anos sem necessitar da substituição das baterias (FARAHANI, 2011).

Um exemplo de aplicação utilizando *ZigBee* é o monitoramento de pacientes em casa. Variáveis como pressão sanguínea e batimentos cardíacos podem ser medidas por dispositivos vestíveis e, por meio de uma interface dos sensores com o transmissor *ZigBee*, essas informações podem ser armazenadas em um servidor e analisadas por um profissional competente. Outro exemplo prático é o monitoramento da estrutura de construções de grande escala, como torres e edifícios. Nesta aplicação, diversos dispositivos sensores podem ser espalhados pela estrutura da construção e, por meio de uma rede de comunicação sem fio, compartilhar informações sobre o estado de preservação da estrutura, reduzindo os gastos com inspeção humana.

A baixa taxa de transferência de dados de dispositivos *ZigBee* é a razão pela qual não se aplica na implementação de uma conexão sem fio com a *Internet* ou na transferência de áudio de alta qualidade em fones de ouvido, por exemplo. Portanto, se o intuito é a transferência de comandos simples ou adquirir informação de sensores, como temperatura, humidade e nível de ruído, como em ambientes industriais num contexto de *Industrial Internet of Things*, o padrão *ZigBee* apresenta o melhor custo benefício quando comparado ao *Bluetooth* e *Wi-Fi* (FARAHANI, 2011), como ilustra a Figura 6.

Figura 6 – Comparação entre *ZigBee*, *Bluetooth* e *Wi-Fi*



Fonte: Farahani (2011).
Nota: Adaptado pelo autor.

4.1 *ZigBee* Alliance

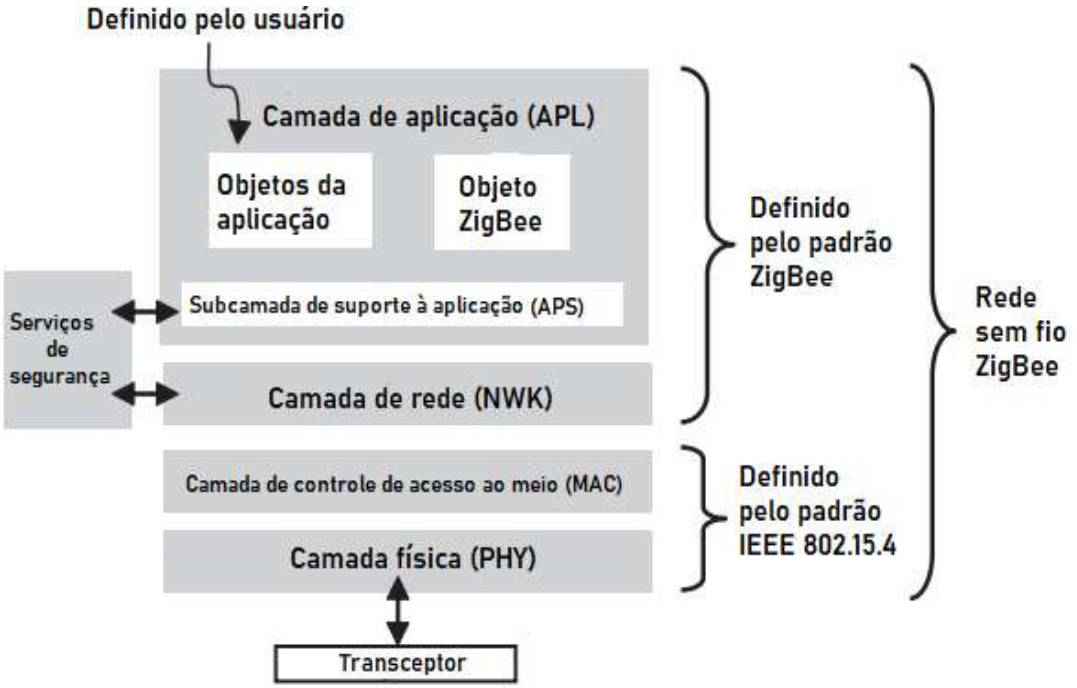
O padrão *ZigBee* foi desenvolvido pela *ZigBee Alliance*, composta por centenas de companhias de diversos ramos, desde a indústria de semicondutores, desenvolvedores de *softwares* a fabricantes e instaladores de equipamentos. Surgiu em 2002 como uma organização sem fins lucrativos aberta a todos que quiserem colaborar com o projeto, adotou a padronização *IEEE* 802.15.4 como seus protocolos de camadas física (*PHY)* e de controle de acesso ao meio (*MAC*).

4.2 Arquitetura da pilha *ZigBee*

Uma das maneiras mais comuns de se estabelecer comunicação em rede é por meio do conceito de camadas de rede. Cada camada suas funções bem definidas em uma rede e a troca de informações e comandos acontece somente entre camadas diretamente acima ou abaixo umas das outras.

As camadas de rede *ZigBee* são baseadas no modelo de referência *OSI* e são mostradas na Figura 7. Dado que o padrão *IEEE* 802.14.5 define somente as camadas física e de controle de acesso ao meio, o padrão *ZigBee* define as camadas de ordem superior, como camada de rede, de aplicação e de segurança.

Figura 7 – Camadas de rede *ZigBee*



Fonte: Farahani (2011).
Nota: Adaptado pelo autor.

4.2.1 Camada Física (*PHY*)

Como pode ser observado na Figura 7, a camada física é a mais inferior na pilha *ZigBee*. Esta camada é a mais próxima do *hardware* utilizado, controla e se comunica diretamente com o rádio transceptor. É responsabilidade da camada física ativar os rádios que transmitem e recebem pacotes, além de selecionar o canal de frequência utilizado, garantindo que ele não esteja sendo usado por outro dispositivo ou outra rede (FARAHANI, 2011).

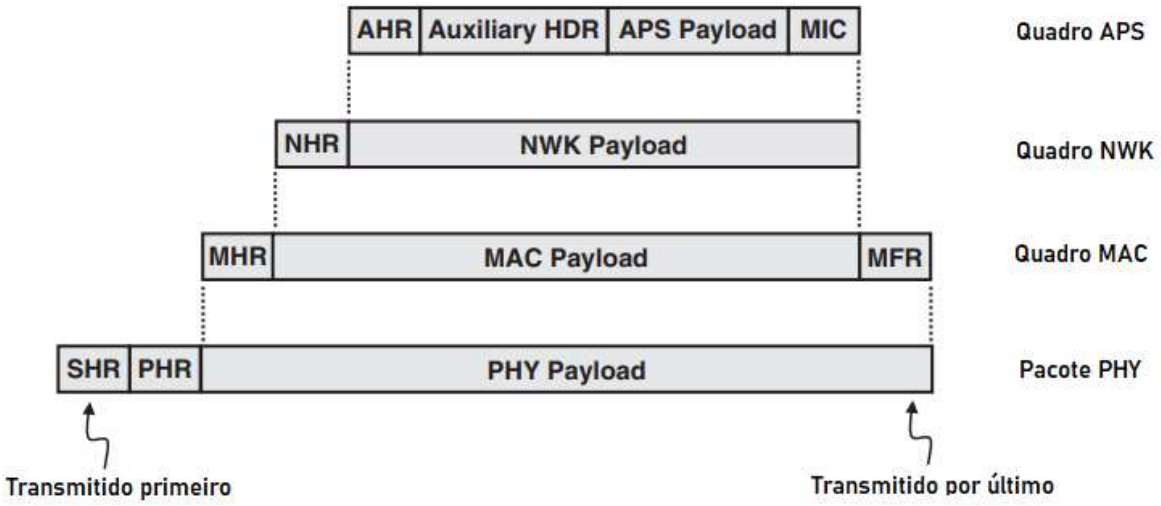
Comandos e dados são comunicados entre dispositivos *ZigBee* na forma de pacotes. A estrutura geral de um pacote é mostrada na Figura 8 e é composta por 3 partes: o cabeçalho de sincronização (*SHR,* do inglês *Synchronization header*), o cabeçalho *PHY* (*PHR*, do inglês *Physical header*) e a carga útil *PHY* (do inglês *payload*).

O *SHR* habilita o receptor a se sincronizar com o fluxo de dados. O *PHR* contém informações sobre o tamanho do quadro de dados e o *payload* é fornecido por camadas superiores, contendo dados e comandos para o dispositivo receptor.

O quadro *MAC*, transmitido a outros dispositivos como *PHY payload*, tem três seções. O cabeçalho *MAC* (*MHR,* do inglês *MAC header)* contém informações sobre endereçamento e

segurança, o *MAC payload* sobre comandos ou informações transmitidas e rodapé *MAC* (*MFR,* do inglês *MAC footer)* é composto por um quadro de 16 bits utilizado na verificação de dados.

Figura 8 – Estrutura dos pacotes *ZigBee*



Fonte: Farahani (2011).
Nota: Adaptado pelo autor.

O quadro *NWK* é composto por duas partes: cabeçalho *NWK* (*NHR, do inglês Network header)* e *NWK payload*. O cabeçalho fornece informações sobre endereçamento e controle, enquanto o *payload* é fornecido pela subcamada *APS*. No quadro *APS* estão informações sobre a segurança do pacote enviado, chaves de segurança e mecanismos de verificação da integridade do pacote (FARAHANI, 2011).

4.2.2 Camada de controle de acesso ao meio (*MAC*)

A camada *MAC* serve de interface entre a camada física e a camada de rede. É responsável por executar serviços de associação e desassociação. O *IEEE* 802.15.4 define quatro quadros *MAC*: *Beacon,* Dados, Reconhecimento e Comandos.

O quadro *beacon* é utilizado pelo coordenador da rede para sincronizar o *clock* de todos os dispositivos na mesma rede. Os quadros de dados e reconhecimento são usados para transmitir informações e verificar se os pacotes enviados foram recebidos com sucesso. Os comandos *MAC* são transmitidos por meio de um quadro de comandos.

4.2.3 Camada de rede (*NWK*)

A camada de rede serve de interface entre as camadas de controle de acesso ao meio e de aplicação, tendo como principais responsabilidades o gerenciamento da formação da rede

ZigBee e o roteamento de pacotes transmitidos. Roteamento é o processo de seleção do caminho que a mensagem percorrerá até chegar em seu destino.

Em uma rede *ZigBee*, o dispositivo coordenador e os roteadores são responsáveis pelo descobrimento de novas rotas e pela manutenção das rotas já estabelecidas na rede. É a camada de rede do coordenador que estabelece uma nova rede e seleciona a sua topologia: árvore, estrela ou *mesh*.

4.2.4 Camada de aplicação (*APL*)

A camada *APL* é a mais alta na pilha *ZigBee*. Fabricantes desenvolvem objetos de aplicação para personalizar um dispositivo à sua maneira. Estes objetos controlam e gerenciam as camadas de dispositivos.

O padrão *ZigBee* oferece a opção de uso de perfis de aplicação no desenvolvimento de produtos. Um perfil de aplicação é um conjunto de acordos sobre o formato das mensagens transmitidas na rede. O uso de um perfil de aplicação torna interoperacionais produtos fabricados por diferentes companhias.

4.2.5 Segurança

Segundo Farahani (2011), mensagens transmitidas em uma rede *ZigBee* podem ser detectadas por qualquer dispositivo próximo, incluindo um intruso. Portanto, há duas preocupações sobre a segurança da rede. A primeira delas é a confiabilidade dos dados. Para solucionar o problema da confiabilidade, as mensagens são criptografadas antes da transmissão e somente o dispositivo destino tem acesso à chave de segurança necessária para recuperar a mensagem original. O padrão *IEEE* 802.15.4 apoia o uso do Padrão de Criptografia Avançada (*AES*, do inglês *Advanced Encription Standard*) (DAEMEN; RIJMEN, 2001) para criptografar suas mensagens.

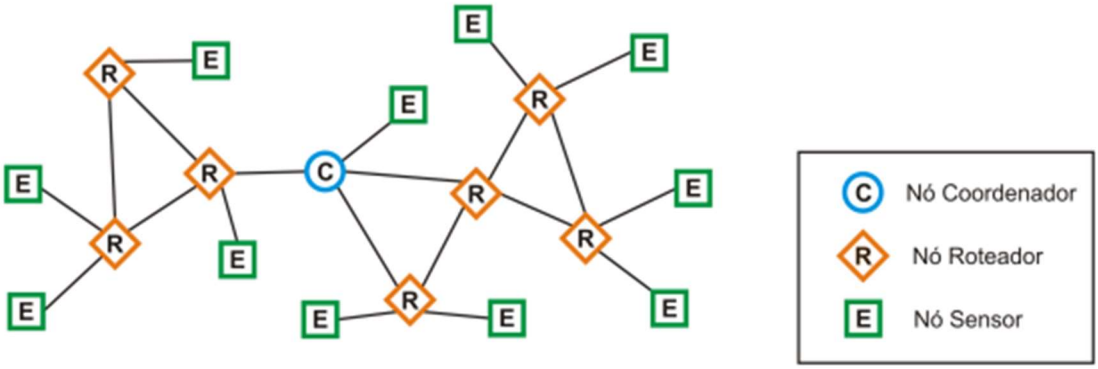
A segunda preocupação com relação à segurança da rede é caso um dispositivo intruso tente modificar e reenviar um pacote transmitido mesmo que este esteja criptografado. Para saber se a mensagem foi alterada em trânsito, é incluído um código de integridade da mensagem (*MIC*, do inglês *message integrity code*) em cada frame enviado, num processo denominado autenticação da informação.

4.3 Rede *ZigBee*

4.3.1 Tipos de dispositivos

Em uma rede *ZigBee*, há três possíveis tipos de dispositivos: coordenador e roteador *ZigBee*, implementados como nós FFD e Dispositivo Final, implementado como nó RFD. Usualmente, o dispositivo final (do inglês *End Device*) é representado por um nó sensor. As disposições dos tipos de dispositivos de uma rede *ZigBee* são mostradas na Figura 9.

Figura 9 – Exemplo de rede *ZigBee*



Fonte: Rivero (2011).

O coordenador é responsável por inicializar a rede, selecionando um canal de comunicação e uma identificação para a rede, denominada *PAN* ID. Através do nó coordenador é possível que novos roteadores e sensores sejam incorporados à rede. Portanto é preciso que opere em estado ativo, alimentado diretamente, para que efetue o controle da rede.

Os dispositivos roteadores têm a principal função de dar mais robustez à rede. Através de tabelas de roteamento, os roteadores conseguem encontrar o menor caminho até o destino de um pacote de informações. É devido aos roteadores que uma rede em malha *ZigBee* apresenta a característica de auto-regeneração, uma vez que não encontrado um destino requisitado, é feito um *broadcast* na rede de uma requisição de rota e receberá do destino uma rota eficaz.

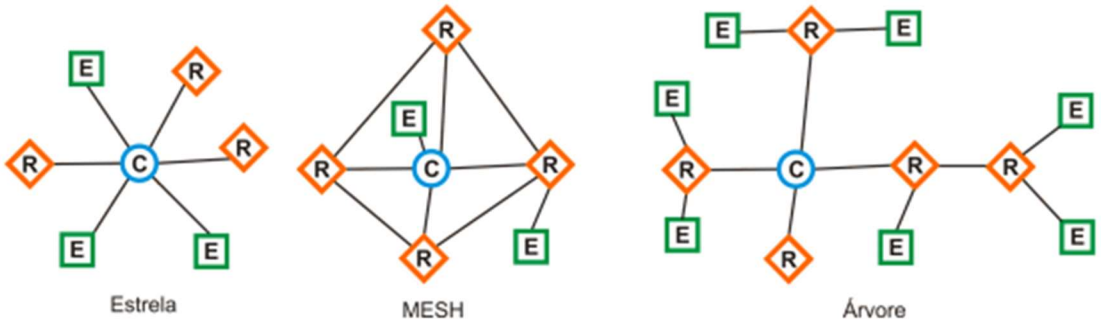
Já os dispositivos finais não fazem roteamento nem coordenam a rede *ZigBee*. Portanto passam a maior parte do tempo em estado inativo (*sleep mode*). São implementados em dispositivos RFD, geralmente sensores, atuadores e sistemas de controle. Nós sensores devem ingressar em uma rede através de um roteador ou de um coordenador, antes que comece a receber e transmitir dados. Nenhum outro dispositivo pode ser incorporado à rede por meio de um nó sensor e, ao

contrário de roteadores e do coordenador, um nó sensor só pode se comunicar com o dispositivo pelo qual teve acesso à rede, denominado nó pai.

4.3.2 Topologias da rede

Na topologia estrela, ilustrada na Figura 10, a comunicação é estabelecida entre os dispositivos da rede e um controlador central, denominado coordenador. Nesta configuração o coordenador geralmente é alimentado diretamente, enquanto os outros dispositivos da rede são alimentados por baterias. Aplicações como automação residencial e periféricos de computadores pessoais se beneficiam desse tipo de topologia.

Figura 10 – Tipos de topologia de uma rede *ZigBee*



Fonte: Rivero (2011).

Na topologia *mesh*, também existe um Coordenador da rede. Porém, neste caso, qualquer dispositivo pode se comunicar com os outros, contanto que estejam dentro do seu raio de contato. Uma rede *mesh* pode ser *ad hoc,* autoconfigurada e auto recuperada. A possibilidade de múltiplos saltos para a transmissão de dados fornece confiabilidade à rede, o que torna interessantes as aplicações em monitoramento, controle e Redes de Sensores sem Fio.

Ainda existe a topologia árvore, que é um caso especial da topologia *mesh*. Neste caso, a maioria dos dispositivos é *FFD* e um *RFD* pode se conectar à rede como uma folha no final de um galho. A vantagem dessa estrutura é a possibilidade de cobrir uma maior área, ao custo de aumento da latência das mensagens (ERGEN, 2004).

4.3.3 Roteamento

As características de roteamento das redes *ZigBee* aumentam sua autonomia energética, uma vez que as rotas são mantidas independentemente da qualidade da comunicação, evitando o uso excessivo de *broadcasts* e constante atualização da tabela de endereçamento (Tose, 2012).

O protocolo *ZigBee* utiliza o protocolo de roteamento *AODV* (do inglês *Ad Hoc On Demand Distance Vector*), um padrão reativo onde os dispositivos da rede só atualizam suas tabelas de endereçamento no caso de uma rota de transmissão de dados for perdida ou ainda não existir. Neste tipo de aplicação, onde dispositivos embarcados apresentam severas restrições com relação à alimentação, é essencial um controle sob demanda das rotas, o que eleva o tempo de vida da rede sem prejudicar a qualidade da transmissão de dados. Além da economia de energia, o sistema de roteamento *ZigBee* garante a confiabilidade necessária numa RSSF, principalmente no ambiente industrial.

4.4 Rádio *XBee*

Os módulos de comunicação *XBee* (DIGI, 2018a) adotam os protocolos *IEEE* 802.15.4 e *ZigBee*. A fabricante, *Digi International*, disponibiliza três modelos de rádio *XBee*: Série 1, que implementa estritamente o padrão *IEEE* 802.15.4; Série 2 que, além do *IEEE* 802.15.4, implementa o padrão *ZigBee*; e Série 3, uma versão melhorada e atualizada do módulo. Dos modelos citados, somente o *XBee PRO* das séries 2 e 3 permite configuração de redes do tipo *Mesh* (malha).

Existem três possíveis configurações da antena do módulo *XBee*: antena de fio, conector para antena externa no padrão U.FL (mini conector coaxial) e antena de chip, como pode ser visto na Figura 11.

Figura 11 – Tipos de antenas dos módulos *XBee*

Fonte: Digi (2018a).
Nota: Adaptado pelo autor.

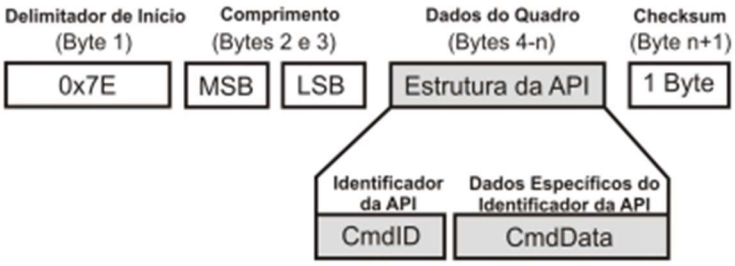
4.4.1 Envio e Recepção de Dados

Basicamente, o módulo *XBee* trabalha com dois modos de operação de transmissão e recepção de dados: *AT* (do inglês *Transparent Operation)* e *API* (do inglês *Application Programming Interface*). O primeiro modo apresenta uma interface mais simples e é mais facilmente suportado por aplicações. Toda informação é recebida através de uma porta serial e sempre será transmitida, a menos que o dispositivo esteja operando no modo de comando (*command mode*). Apesar de simples, esse modo não é escalável para enviar dados a múltiplos destinatários e não permite o envio de configurações remotas de módulos.

O segundo modo, *API*, baseia-se na transmissão de quadros (*frames*) de dados, especificando como os comandos, respostas a comandos e mensagens sobre o estado de funcionamento dos módulos devem ser enviados e recebidos. Apresenta maior velocidade, e consequentemente viabilidade, na transmissão de dados para múltiplos destinatários, dado que só é necessário alterar o campo de endereço do quadro *API*. Um exemplo da estrutura do quadro de dados da *API* é mostrado na Figura 12. Neste modo, toda informação recebida indica o endereço do dispositivo que enviou a informação, facilitando no comando de dispositivos finais.

O quadro de dados da *API* tem como delimitador inicial um byte com o valor 0x7E, seguido por dois bytes que indicam o tamanho do quadro de dados, sem contar o último byte, de *checksum*. Dependendo do tipo da *API* e da aplicação, o tamanho do quadro de dados pode variar significativamente (RIVERO, 2011).

Figura 12 – Estrutura do quadro de dados da *API*



Fonte: Digi (2018a).
Nota: Adaptado pelo autor.

4.4.2 Portas E/S

O módulo *XBee* possui 13 portas de entrada e saída (E/S), sendo elas *DIO0* a *DIO12*. Através do *software X-CTU*, desenvolvido pela própria *Digi International,* é possível configurar suas portas. Uma representação da pinagem destas é apresentada na Figura 13.

Figura 13 – Pinagem do *XBee*

Fonte: Adaptado de Digi (2018a).
Nota: Adaptado pelo autor.

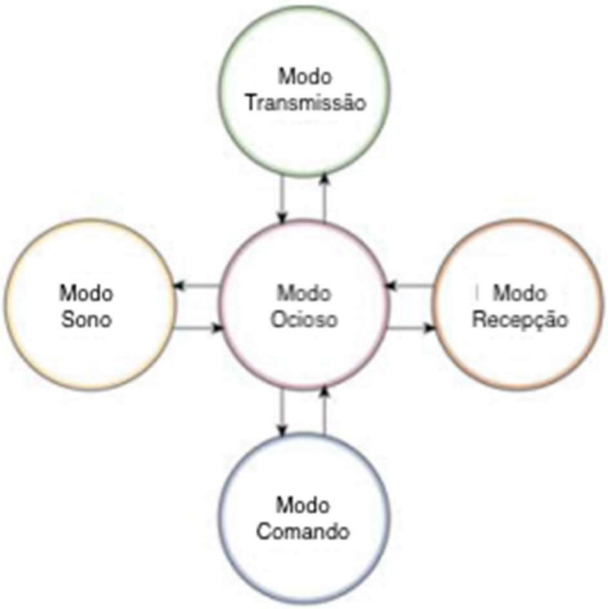
A alimentação do rádio é feita por meio dos pinos 1 (*VCC*) e 10 (*GND*) e tensão de alimentação no pino 1 deve ser de 3,3 V. A entrada e saída de dados é feita, respectivamente, nos pinos 3 (*DIN*) e 2 (*DOUT*), tal que esses pinos podem ser conectados ao microcontrolador onde estão os sensores. No pino 5 é executada a função *RESET* do módulo, devendo ser conectada uma chave ligando o pino 5 ao pino 10. O pino 6, além de E/S de sinal digital, pode ser utilizado para medir o *RSSI* da última transmissão recebida. O pino 8 é de uso reservado e não deve ser conectado pelo usuário. O pino 9, além de E/S de sinal digital, é utilizado para indicar se o módulo está em modo de economia de energia (*sleep mode*). Os pinos 4 (*DIO12*), 7 (*DIO11*) e 11 (*DIO4*) funcionam como E/S digital somente. Os pinos 12 ($\overline{CTS}$/*DIO7*) e 16 ($\overline{RTS}$/*DIO6*) podem ser utilizados para controle de fluxo *Clear-to-Send* (*CTS*) e *Request-to-Send* (*RTS*), respectivamente. O pino 13, além de E/S digital, funciona para indicar o status de funcionamento do módulo: ativo ou dormindo. O pino 15, além de E/S digital, indica o estado de associação do módulo com a rede. Caso o módulo esteja desconectado da rede, o pino 15 recebe um sinal de tensão constante. Caso contrário, recebe pulsos de tensão em intervalos regulares de 500 ms para nós coordenadores e 250 ms para roteadores e dispositivos finais (DIGI, 2018a).

Os pinos 17 (*AD3*/*DIO3*) a 20 (*AD0*/*DIO0*) funcionam como E/S digitais ou como entradas analógicas. A última porta, *AD0*/*DIO0*, também pode funcionar como um botão de comissionamento da rede.

4.4.3 Modos de Operação

O rádio *XBee* apresenta cinco modos de operação, como apresentado na Figura 14: Ocioso (do inglês *Idle)*, Transmissão (do inglês *Transmission)*, Recepção (do inglês *Recieve)*, Comando (do inglês *Command)* e Sono (do inglês *Sleep)*. Por padrão, ele permanece em Modo Ocioso sem transmitir ou receber dados. Quando dados seriais estão prontos para transmissão, no buffer do módulo, ele entra em Modo Transmissão. O Modo Recepção é ativado quando um pacote é recebido e a informação é transferida para o *buffer* de transmissão. Para modificar ou ler parâmetros de configuração do módulo, é preciso entrar em Modo Comando. Neste estado, todos os caracteres seriais recebidos são interpretados como comandos. O Modo Sono está disponível somente a dispositivos finais e permite que ele entre em um estado de baixo consumo de energia quando não está em funcionamento. Os módulos *XBee* têm suporte a dois tipos diferentes de sono: *pin sleep*, ou seja, o módulo fica "dormindo" até ser requisitado para alguma atividade; e *cyclic sleep*, no qual o módulo dorme por um tempo determinado e depois volta a seu funcionamento normal.

Figura 14 – Modos de operação do *XBee*



Fonte: Digi (2018a).
Nota: Adaptado pelo autor.

4.5 *X-CTU*

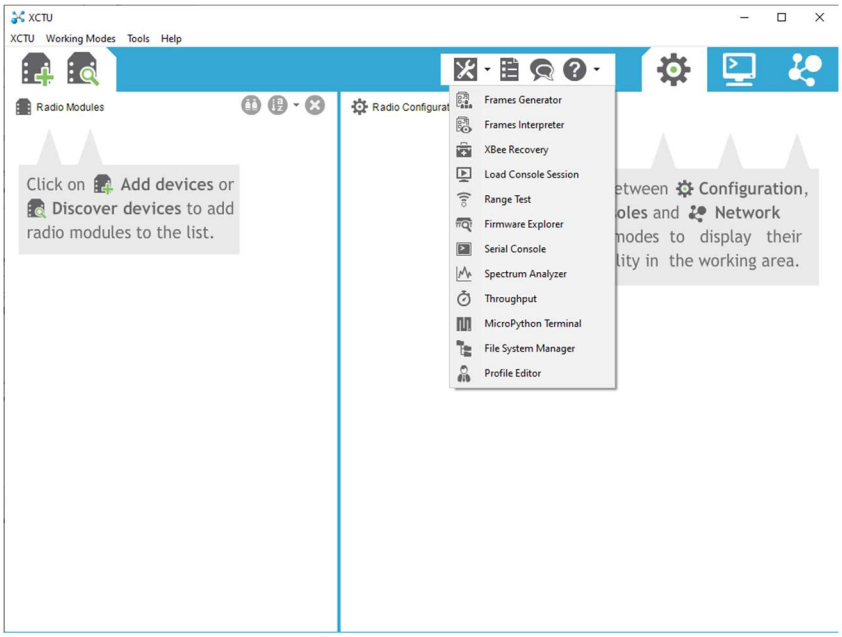
X-CTU (DIGI, 2018b) é uma aplicação multiplataforma livre desenvolvida para permitir que desenvolvedores interajam com módulos fabricados pela *Digi International* por meio de uma interface gráfica de simples uso. A aplicação inclui ferramentas que facilitam a configuração e testes de módulos *XBee*.

Estão disponíveis ferramentas gráficas como a visualização da rede, identificando a sua topologia e a potência do sinal entre as conexões. Outros destaques do *software* são:

- Gerenciamento e configuração remota de múltiplos dispositivos.
- Atualização de *firmware* e restauração das configurações de fábrica.
- Duas opções de console para comunicação com os módulos: *API* e AT.

Mais detalhes sobre a configuração dos módulos e utilização das ferramentas de testes serão dados ao decorrer do trabalho. A visualização da tela inicial do *X-CTU* pode ser observada na Figura 15.

Figura 15 – Interface *X-CTU*



Fonte: Produção do próprio autor.

5 PLATAFORMAS DE PROTOTIPAGEM

5.1 Arduino

Arduino é uma plataforma livre utilizada para construção e programação de eletrônicos. Pode receber e enviar informação para uma grande variedade de dispositivos e até mesmo usando a Internet para receber comandos. Nos últimos anos, o Arduino tem sido muito utilizado na programação de microcontroladores devido a sua interface amigável e fácil configuração.

A versão utilizada neste trabalho é denominada Arduino Uno, que, diferentemente das placas anteriores, não utiliza um *chip FTDI* (do inglês *Future Technology Devices International*) para conversão *USB* para serial. Para tal, é utilizado o *chip* Atmega16U2 programado como conversor *USB*-serial.

A alimentação do Arduino Uno pode ser tanto via *USB* quanto via fonte externa (baterias ou adaptadores CA-CC) e a seleção da fonte de alimentação é feita automaticamente. A tensão da fonte externa pode variar entre 6 e 20 V, porém, tensões abaixo de 7 V podem causar mau funcionamento e acima de 12 V podem causar aquecimento no regulador de tensão da placa.

Em suma, um Arduino pode ser classificado em duas partes:

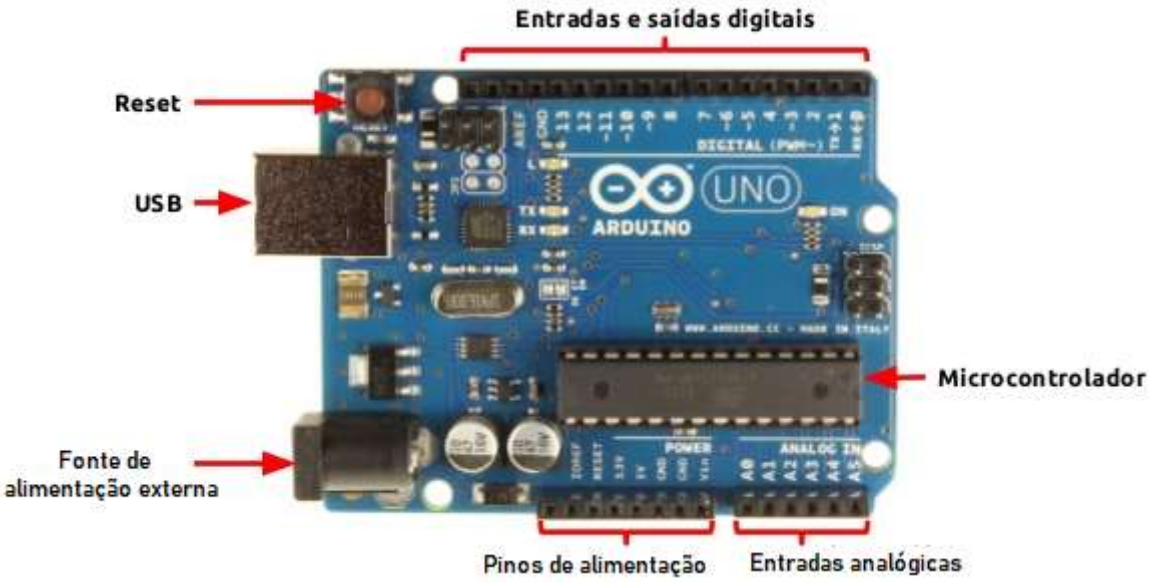
i. *Hardware*: várias componentes são combinadas para o correto funcionamento da plataforma, mas as principais, apresentadas na Figura 16, são (ARDUINO, 2019):
    - Entrada *USB*: utilizada para descarregar o código de programação de um computador para o microcontrolador da placa. Possui tensão regulada de 5 V que também alimenta o Arduino.
    - Fonte de alimentação externa: com função única de alimentar a placa, sugere-se aplicação de tensão entre 9 e 12 V. Valores fora dessa faixa podem causar mau funcionamento e/ou superaquecimento do regulador de tensão presente.
    - Botão de *reset*: reinicia o Arduino em caso de atualização do código programado ou apresentação de mau funcionamento.
    - Microcontrolador: peça principal da placa, processa, recebe e envia informações e comandos para o restante do circuito.
    - Pinos analógicos (0-5): entradas analógicas de A0 a A5
    - Pinos digitais de E/S: entradas ou saídas digitais, nos pinos D2 ao D13.

- Pinos de referência digitais e analógicos.
- Pinos de alimentação: utilizados para fornecer tensão de 3,3 ou 5 V.

ii. *Software*: o ambiente de desenvolvimento Arduino é utilizado na programação da placa, informando o *hardware* o que fazer e como fazer (BOXALL, 2013). Essa interface pode ser dividida em três partes principais, como pode ser visto na Figura 17:

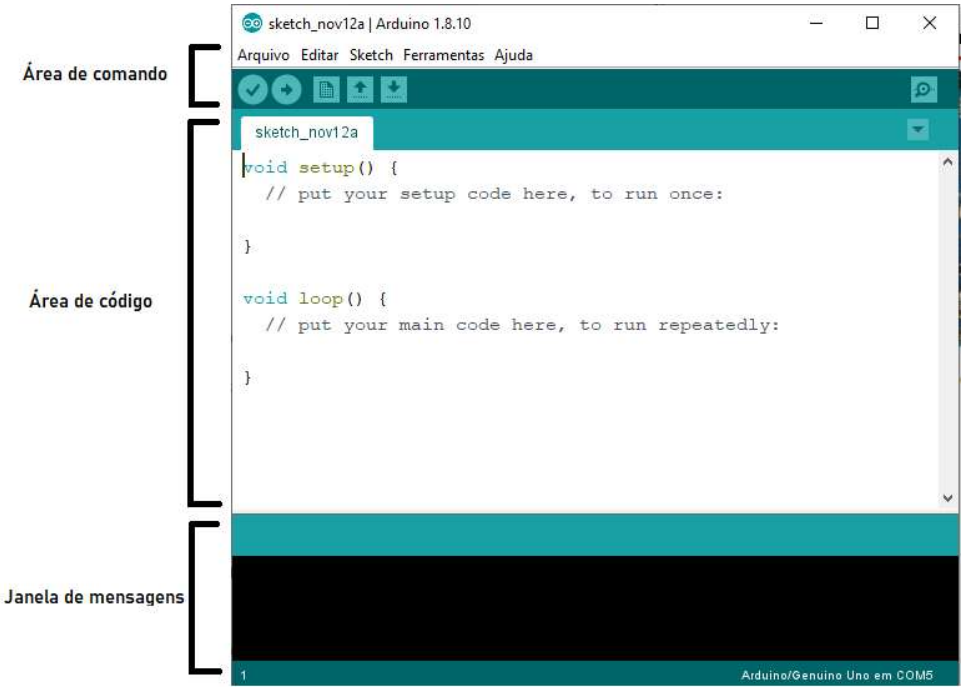
- Área de comando: nesta área estão disponíveis menus para criação de novos códigos, edição de outros já existentes, informações sobre bibliotecas instaladas, ferramentas de verificação e monitor serial.
- Área de código: é onde um código em uma versão simplificada do C++ é escrito, determinando o comportamento da placa. Esta área é dividida em outras duas: rotina de configuração, executada somente uma vez, são estabelecidas as condições iniciais das variáveis declaradas e determinadas instruções preliminares do código. A segunda parte é a rotina de *loop*, onde o código principal do usuário é executado repetidamente enquanto o Arduino estiver sendo alimentado por uma fonte de energia.
- Área da janela de mensagens: mostra mensagens do ambiente de desenvolvimento, em grande parte sobre a verificação do código.

Figura 16 – Placa Arduino Uno



Fonte: Arduino (2019).
Nota: Adaptado pelo autor.

Figura 17 – Ambiente Integral de Desenvolvimento Arduino



Fonte: Produção do próprio autor.

5.2 *Seeeduino Stalker* V2

O *Seeeduino Stalker* (SEEED, 2010) é um módulo desenvolvido para redes de sensores sem fio compatível com Arduino. Possui portas de E/S operadas em 3,3 V, o mesmo microcontrolador ATmega328P, conectores integrados para cartão de memória e para a família *Bee*. Ainda, o *Seeeduino Stalker* possui um circuito carregador de baterias embarcado, o que facilita a otimização de módulos de monitoramento remotos, evitando o uso de fontes externas de alimentação. As principais características do módulo podem ser observadas na Figura 18.

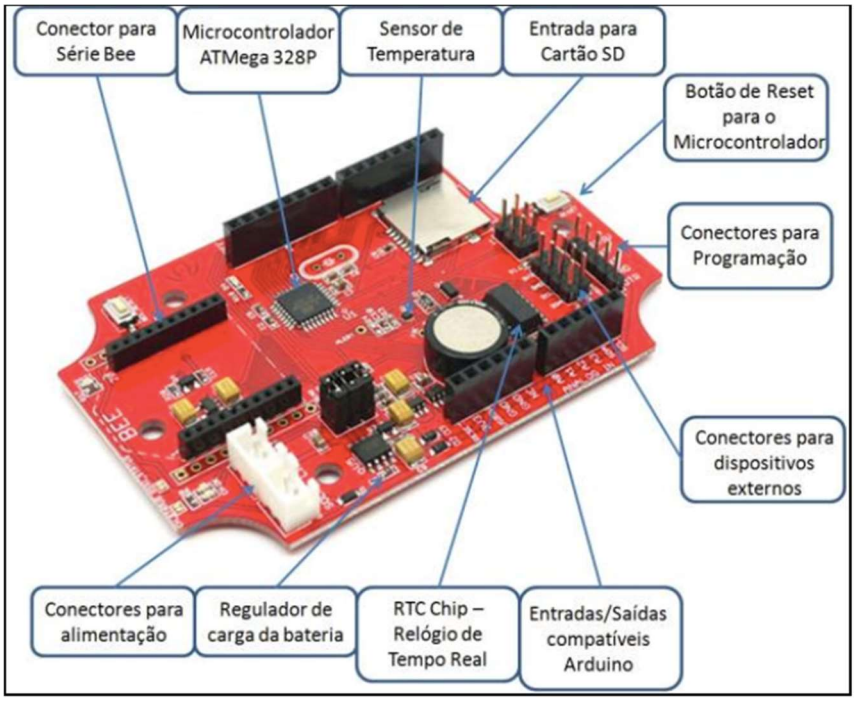
5.3 *XBee Shield*

A plataforma Arduino oferece ao usuário a possibilidade do uso de *shields,* que são módulos complementares adicionados com o objetivo de trazer novas funcionalidades a um projeto, como conexão *Bluetooth, Ethernet* e *XBee*.

O *shield XBee* permite comunicação sem fio com outros dispositivos usando *ZigBee*. A placa permite acesso às portas de E/S do Arduino, como pode-se observar na Figura 19, porém, devido ao seu formato, restringe o acesso aos pinos de alimentação. A sobreposição também

ocorre com o botão de *reset* do Arduino, portanto, o *XBee shield* possui um botão de *reset* adicional. A placa utiliza alimentação de 5 V e possui seu próprio regulador de tensão para alimentação do *XBee* com 3,3 V.

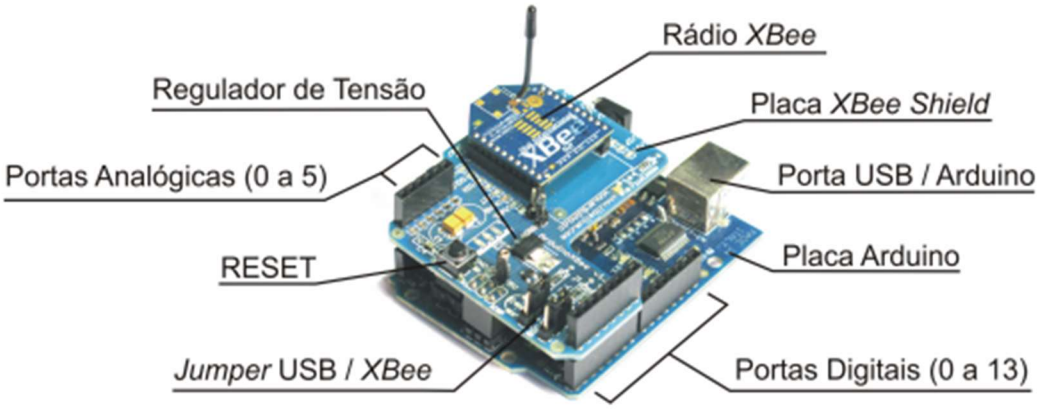
Figura 18 – *Seeeduino Stalker V2.0*



Fonte: Tose (2012).

Um conjunto de conectores determina como será a comunicação serial entre as placas: na posição *XBee*, o microcontrolador recebe dados somente do rádio *XBee*, enquanto na posição *USB*, há comunicação da placa com um computador por meio da porta *USB*.

Figura 19 – Arduino Uno com *XBee shield*



Fonte: Rivero (2011).

5.4 IC

O módulo *IC* (do inglês *Industrial Computer*) foi desenvolvido pela empresa *2Solve* para funcionar como um computador em ambientes industriais, concatenando, tratando e processando dados coletados por outros módulos sensores.

O *IC* é baseado em *Raspberry Pi* (RASPBERRY, 2019), que funciona com o sistema operacional Linux, o que o torna um computador totalmente funcional, ao invés de simplesmente um dispositivo que faz interface com outro *hardware.* Por possuir um sistema operacional, a *Raspberry Pi* possui mais poder de processamento e diversas opções de aplicação quando comparado ao *Arduino*.

Por meio dos rádios de comunicação *XBee* é possível que *Arduino* e *Raspberry Pi* se comuniquem e troquem informações sobre os dados coletados por sensores. Além dos soquetes para dispositivos da família *Bee*, o *IC* possui 12 entradas de sinal analógico, 3 entradas de sinal digital, 3 saídas digitais, porta para conexão *RS 485, HDMI*, antenas para conexão *Wi-Fi* e *Remote I/O* e ainda tem disponíveis portas *USB* e *Ethernet*, utilizadas na configuração do *Raspberry Pi.* O módulo é alimentado por fonte de tensão de 24 V. Todas as portas presentes podem ser observadas na Figura 20.

Figura 20 – Módulo *IC*

Fonte: Produção do próprio autor.

6 MONITORAMENTO DA REDE DE SENSORES SEM FIO

6.1 Desenvolvimento dos kits de monitoramento

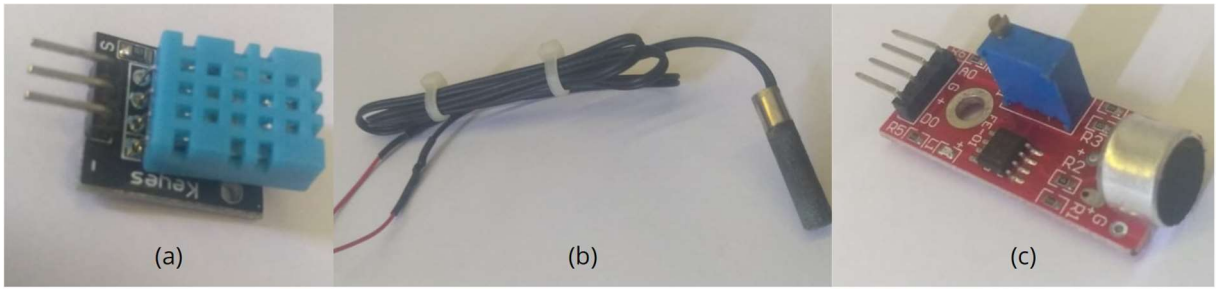
Para este trabalho, foram adquiridos três placas *Arduino Uno*, duas placas *Seeeduino Stalker*, um módulo *IC*, sete rádios *XBee*, três *XBee Shields* e 1 *Uart-Bee*, utilizado para conectar um rádio *XBee* ao computador por meio de uma porta *USB,* com o objetivo de configurar os rádios e os *Stalkers*. Adicionalmente, foram obtidos antenas, baterias, pilhas e sensores de temperatura, humidade e som.

6.1.1 Microcontroladores e sensores

Boa parte do tempo e esforço aplicados no desenvolvimento deste projeto foi relacionado à programação dos microcontroladores e preparação dos diversos sensores utilizados.

Para a programação dos microcontroladores foi necessário estudo da linguagem de programação C++, assim como a familiarização com as bibliotecas já existentes para realizar uma interface entre os dados analógicos captados pelos sensores e as entradas analógicas dos microcontroladores. Os dispositivos sensores utilizados podem ser vistos na Figura 21.

Figura 21 – Sensores utilizados para (a) humidade e temperatura, (b) temperatura e (c) som

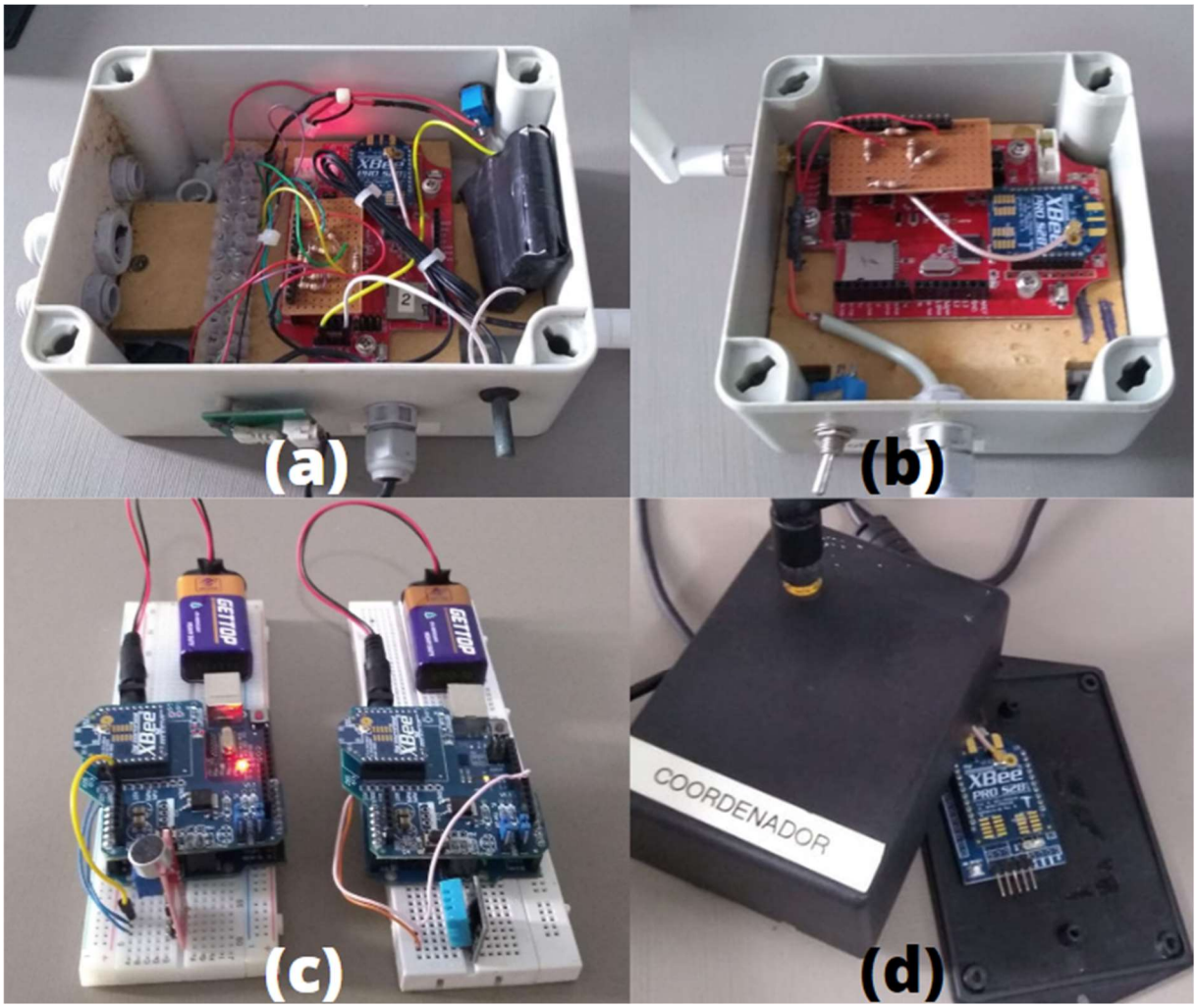


Fonte: Produção do próprio autor.

O sensor na Figura 21(a) é denominado DHT11. Este dispositivo tem saída digital calibrada e pode ser considerado bastante confiável e estável, uma vez que utiliza técnica de coleta de dados digitais e possui tecnologia de sensoriamento de humidade (DFROBOT, 2010). Cada elemento é estritamente calibrado em laboratório e os coeficientes de calibração são armazenados na memória do dispositivo. Seu tamanho pequeno, baixo consumo de energia e distância máxima de transmissão de sinais de 20 metros fazem com que esta seja uma ótima opção para a maioria das aplicações. O sensor da Figura 21(b) é denominado NTC 10K e é capaz de medir temperaturas de -55 a 125 °C (CANTHERM, 2012). Já o sensor da Figura 21(c) é denominado

KY-038 (JOY-IT, 2017) e mede a intensidade sonora do ambiente ao seu redor, variando o estado de sua saída digital caso detectado algum sinal sonoro. O limite de detecção pode ser ajustado através do potenciômetro presente no sensor que regulará a saída digital *D0*, mas ainda é possível utilizar a saída analógica do sensor para uma melhor resolução. O uso dos sensores citados com as placas *Arduino Uno* e *Seeeduino Stalker* pode ser observado na Figura 22.

Figura 22 – Módulos sensores denominados (a) kit A, (b) kit B, (c) ruído sonoro e DHT11 e (d) coordenador



Fonte: Produção do próprio autor.

No kit A estão presentes sensores de temperatura e de humidade. No kit B há um sensor de temperatura conectado ao *Stalker*. Nos módulos mostrados na Figura 22(c) estão conectados os sensores de ruído sonoro e DHT11, respectivamente. Todos os kits, além dos dados coletados pelos sensores, transmitem também a qualidade da comunicação entre os dispositivos *ZigBee*. Na Figura 22(d) observa-se o coordenador da rede, que não faz uso de um microcontrolador e está conectado a um computador via *USB*.

A programação dos microcontroladores utiliza bibliotecas livres e disponíveis para qualquer usuário. Por exemplo, para a transmissão de dados em uma rede *ZigBee*, há bibliotecas para configuração *AT* e *API.* Neste projeto foi utilizada a configuração *API*, pois desta maneira há uma quantidade maior de informações sendo transmitidas nos pacotes, o que reduz a necessidade de processamento de dados pelos microcontroladores e, consequentemente, o consumo de energia destes. Mais detalhes sobre o processo de otimização no tratamento de dados da rede *ZigBee* serão explicados no decorrer do trabalho.

6.2 Configuração dos rádios *XBee*

Os rádios *XBee* foram configurados por meio do *software X-CTU*, disponibilizado pela *Digi International.* Todos os nós foram configurados sem criptografia de dados, velocidade de comunicação com a interface serial de 9600 *bps*, 8 *bits* de dados, 1 bit de parada, sem paridade e sem controle de fluxo. A identificação da rede é 777 e o canal estabelecido automaticamente pelo coordenador da rede é 12. No Quadro 4 são apresentados os endereços de rede, com 16 bits, e *MAC*, com 64 bits. Ainda, estão representados os nomes dos nós e suas respectivas funções na rede.

Quadro 4 – Configuração dos nós sensores

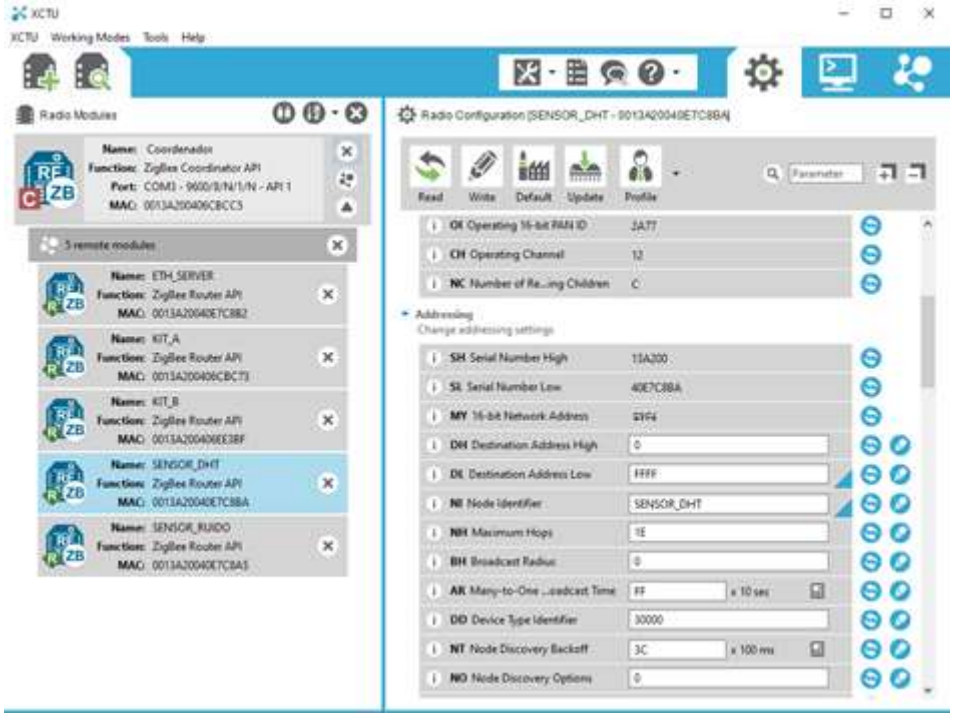
Nome do nó	End. 16 bits	End. 64 bits	Tipo do nó
Coordenador	0000	0013A200406CBCC5	Coordenador
ETH_SERVER	A6B6	0013A20040E7C8B2	Roteador
KIT_A	2341	0013A200406CBC73	Roteador
KIT_B	D635	0013A200406EE3BF	Roteador
SENSOR_DHT	E9F6	0013A20040E7C8BA	Roteador
SENSOR_RUIDO	7AC0	0013A20040E7C8A5	Roteador
IC	7E52	0013A200408C2A72	Roteador

Fonte: Produção do próprio autor.

Devido à pouca quantidade de rádios *XBee* disponíveis para a pesquisa, todos os nós foram configurados como roteadores. Como explicitado anteriormente, nós roteadores não podem ser configurados com modo de sono habilitado, pois necessitam estar sempre acordados para fazer o gerenciamento da rede *ZigBee*. Ainda, como o objetivo do estudo é avaliar a configuração *mesh* de topologia da rede, a configuração dos nós sensores como roteadores parece mais

atraente, dado que *end devices* não podem rotear um sinal de comunicação entre dois dispositivos e essa limitação pode ser expressiva em uma rede pequena. Na Figura 23 observa-se a apresentação gráfica de todos os dispositivos conectados na rede *ZigBee* por meio do *software X-CTU*. Além de listar os dispositivos conectados na rede, são disponibilizadas todas as configurações dos dispositivos e é possível a alteração dessas configurações remotamente.

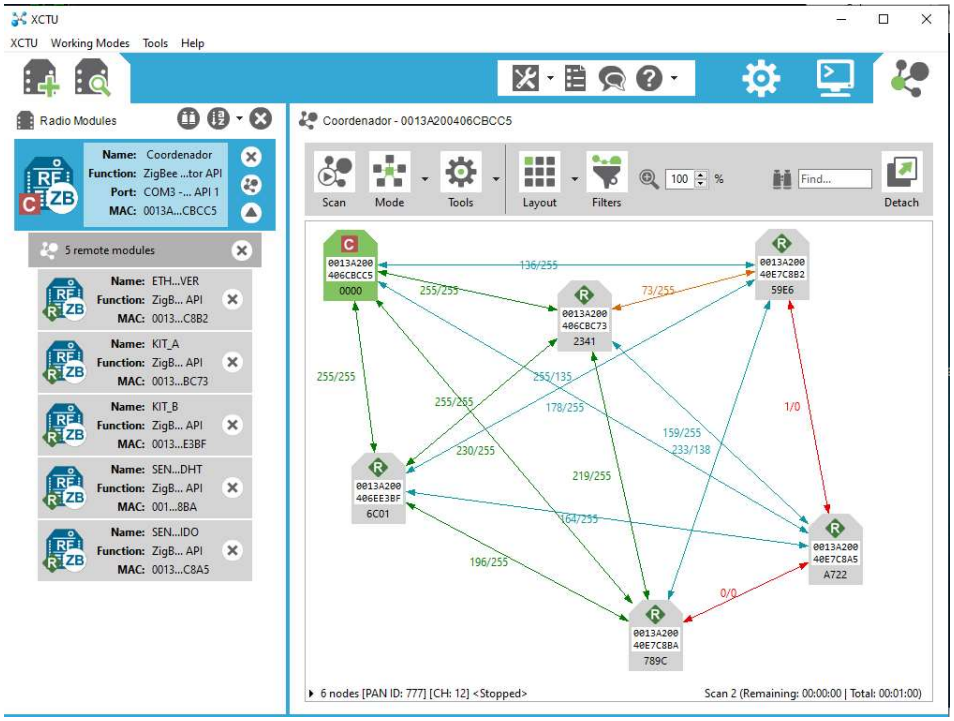
Figura 23 – Nós remotos descobertos pelo *X-CTU*



Fonte: Produção do próprio autor.

Utilizando o próprio *X-CTU* é possível, além de observar os dispositivos conectados à rede *ZigBee*, descobrir a topologia atual da rede e as conexões disponíveis entre os dispositivos. Como ilustrado na Figura 24, por meio do painel *Network* do *X-CTU* é possível observar de que forma os dispositivos da rede estão se comunicando, a qualidade da comunicação e a topologia assumida pela rede. Neste trabalho, todos os rádios foram configurados como roteadores, logo, todos dispositivos são capazes de se comunicar entre si, formando uma complexa rede com inúmeros caminhos possíveis para um pacote ser enviado de um rádio a outro.

Figura 24 – Painel *Network* do *X-CTU*



Fonte: Produção do próprio autor.

A cor das setas conectando os dispositivos está diretamente relacionada com a qualidade do sinal entre eles. O *software* usa um parâmetro denominado Indicador da Qualidade da Conexão (*LQI, do inglês Link Quality Indicator*) para classificar a qualidade das conexões entre os dispositivos da rede. Este número varia de 0 a 255 onde 0 é o mais fraco e 255 é o mais ideal para transmissão/recepção. No Quadro 5 estão dispostos os valores utilizados pelo *software* para classificar e ilustrar a qualidade da comunicação da rede.

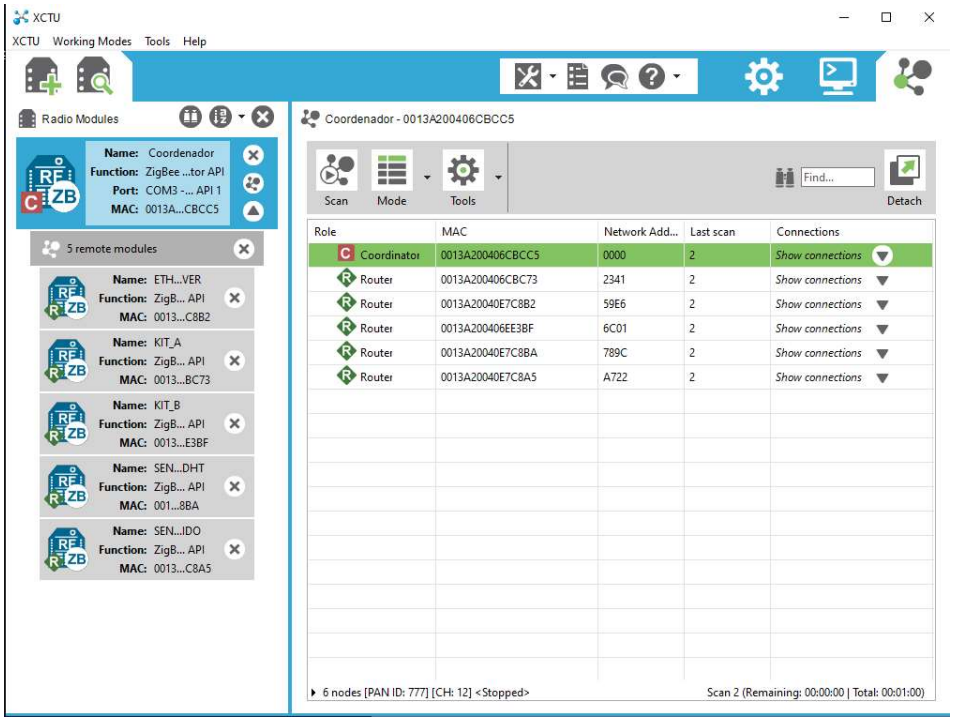
Quadro 5 – Indicação da qualidade do sinal no *X-CTU*

Cor da seta	**Qualidade da conexão**	***LQI* mínimo**	***LQI* máximo**
Verde	Muito boa	195	256
Azul	Boa	130	195
Laranja	Moderada	65	130
Vermelha	Fraca	0	65
Cinza	Desconhecida	-	-

Fonte: Produção do próprio autor.

Para redes muito grades, onde a visualização pode ser prejudicada pela grande quantidade de conexões entre os dispositivos, o *software* disponibiliza uma opção de visualização da rede por meio de uma tabela, conforme Figura 25, onde estão disponíveis informações como a função do dispositivo, seus endereços *MAC* e de rede, as conexões possíveis entre um rádio e seus adjacentes e a última vez que foi feita uma varredura da rede em busca de novas conexões.

Figura 25 – Visualização da rede em modo tabela



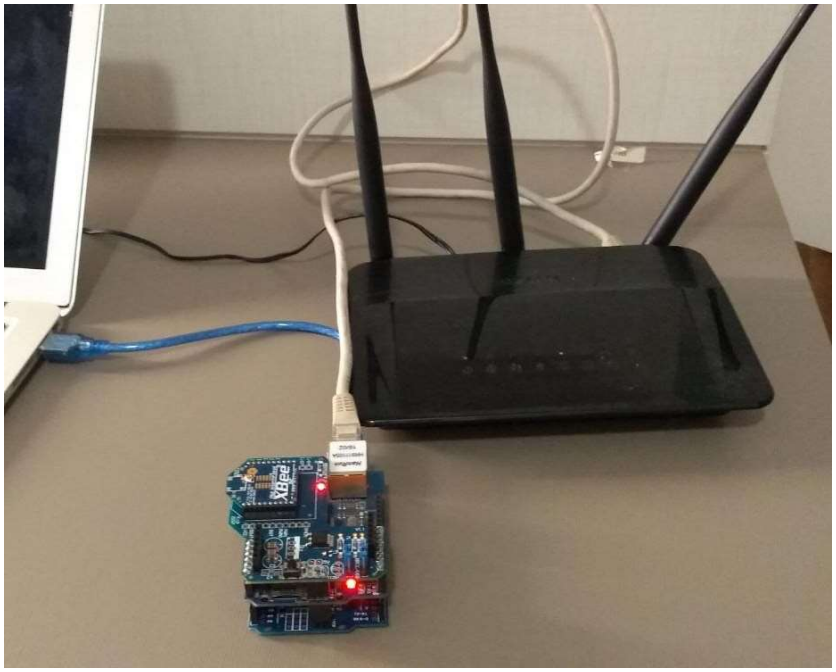
Fonte: Produção do próprio autor.

6.3 Aplicação de monitoramento

Como explicitado anteriormente, objetiva-se neste estudo o desenvolvimento de uma aplicação de monitoramento dos dados dos sensores em tempo real e avaliação do desempenho do padrão *ZigBee* na comunicação industrial.

Uma vez montados todos os kits de sensoriamento e configurados os rádios de comunicação, é necessário um meio de concentrar todos os dados trafegando na rede em um dispositivo que permita ao usuário a visualização das informações captadas dos sensores sobre o ambiente ao seu redor. Portanto, um dos nós roteadores da rede está equipado com um *shield Ethernet* W5100 (WIZNET, 2008) capaz de criar um servidor *Ethernet* em uma rede local, por meio de um roteador *Wi-Fi.* A Figura 26 mostra o equipamento utilizado.

Figura 26 – Nó servidor *Ethernet*

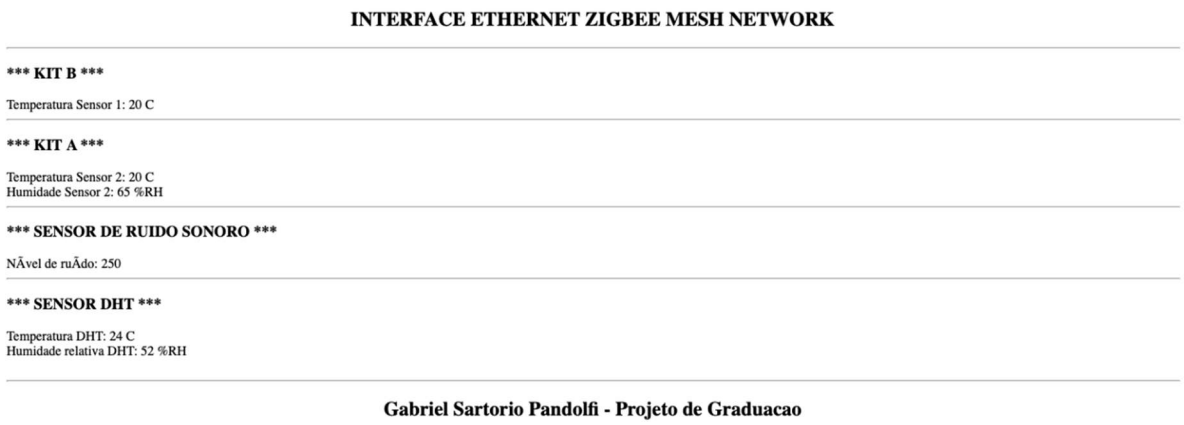


Fonte: Produção do próprio autor.

O tratamento dos dados e disponibilização em uma página *HTML* (do inglês *Hypertext Markup Language*) acontece da seguinte maneira: todos os dispositivos sensores da rede possuem como endereço destino o servidor *Ethernet*, ou seja, todas as informações coletadas pelos sensores da rede são enviadas continuamente a este dispositivo. Por meio dos dados de endereçamento contidos nos pacotes em formato *API*, é possível que o microcontrolador responsável pela concentração dos dados faça o reconhecimento da fonte que enviou aquele pacote. Dessa maneira, não há necessidade de o microcontrolador fazer o tratamento dos dados para descobrir sua origem, como exemplificado no trabalho de Tose (2012). Essa medida traz grandes melhorias à aplicação, visto que a redução do processamento do Arduino resulta na redução do seu consumo de energia e aprimoramento da gestão energética é um dos pilares da Internet das Coisas, principalmente em ambiente industrial.

Com o endereço de origem dos pacotes recebidos, fica a cargo do microcontrolador do servidor *Ethernet* separar as informações e distribuí-las visualmente na página *HTML*. Todas as configurações de endereçamento IP, portas de acesso, endereço *MAC* e tempo de atualização da página são desenvolvidas dentro da programação do Arduino, disponibilizada no APÊNDICE A juntamente com os códigos utilizados nos kits de monitoramento. A página HTML desenvolvida para a aplicação pode ser visualizada na Figura 27. Para acesso em rede local, o acesso foi realizado por meio do *IP* 192.168.1.45.

Figura 27 – Página *HTML* em rede local



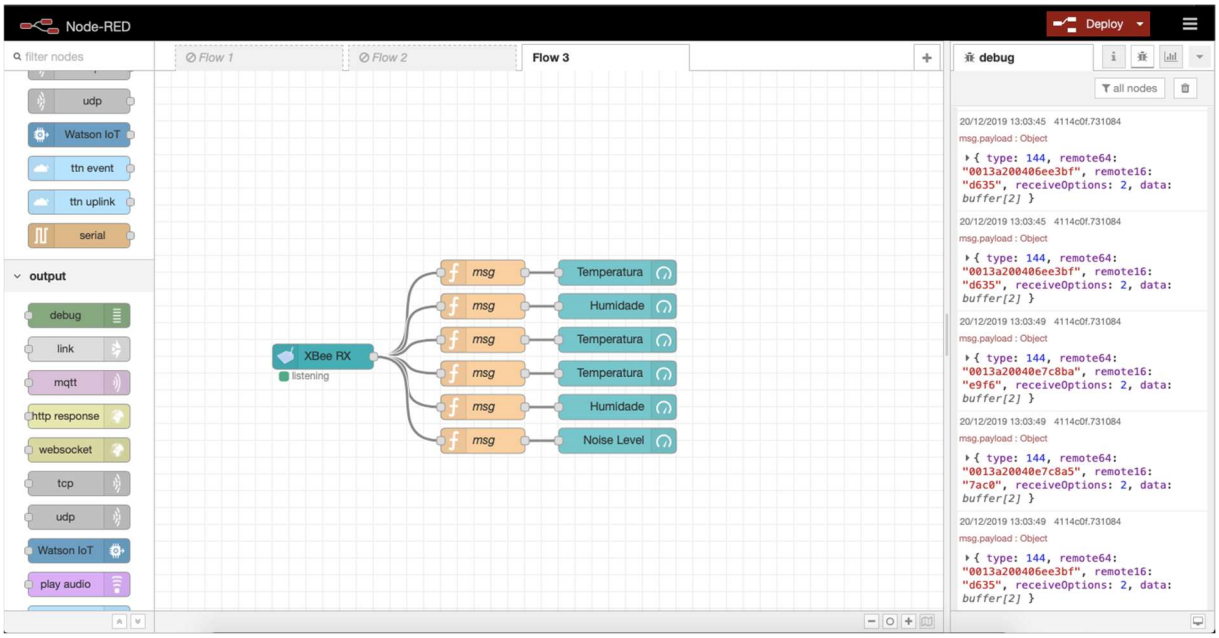
Fonte: Produção do próprio autor.

Além da visualização dos dados dos sensores por meio de um servidor *Ethernet*, foi desenvolvido um *dashboard* utilizando o módulo *IC*. Através de uma ferramenta de programação chamada *Node-RED*, é possível conectar dispositivos, interfaces de programação de aplicativos e serviços *online* sem a necessidade de escrever as linhas de código (NODE-RED, 2019). Toda a programação é feita por meio de fluxogramas, onde blocos com diferentes funções ligados entre si, conforme a necessidade da aplicação.

No desenvolvimento do painel de controle foi utilizado um bloco específico para aplicações *ZigBee*, capaz de receber os pacotes enviados ao *XBee* instalado no módulo *IC* e obter as informações contidas nos pacotes, como endereço do dispositivo que enviou o pacote e as informações contidas nele. Em seguida, por meio de uma função em *Javascript* é feito o tratamento dos dados, de maneira similar ao que foi feito utilizando os Arduinos. Conforme o endereço do nó que enviou os dados, eles são separados e utilizados como dados de entrada em um terceiro bloco, responsável por criar uma interface gráfica exibindo os valores contidos nos pacotes recebidos. A disposição dos blocos pode ser observada na Figura 28.

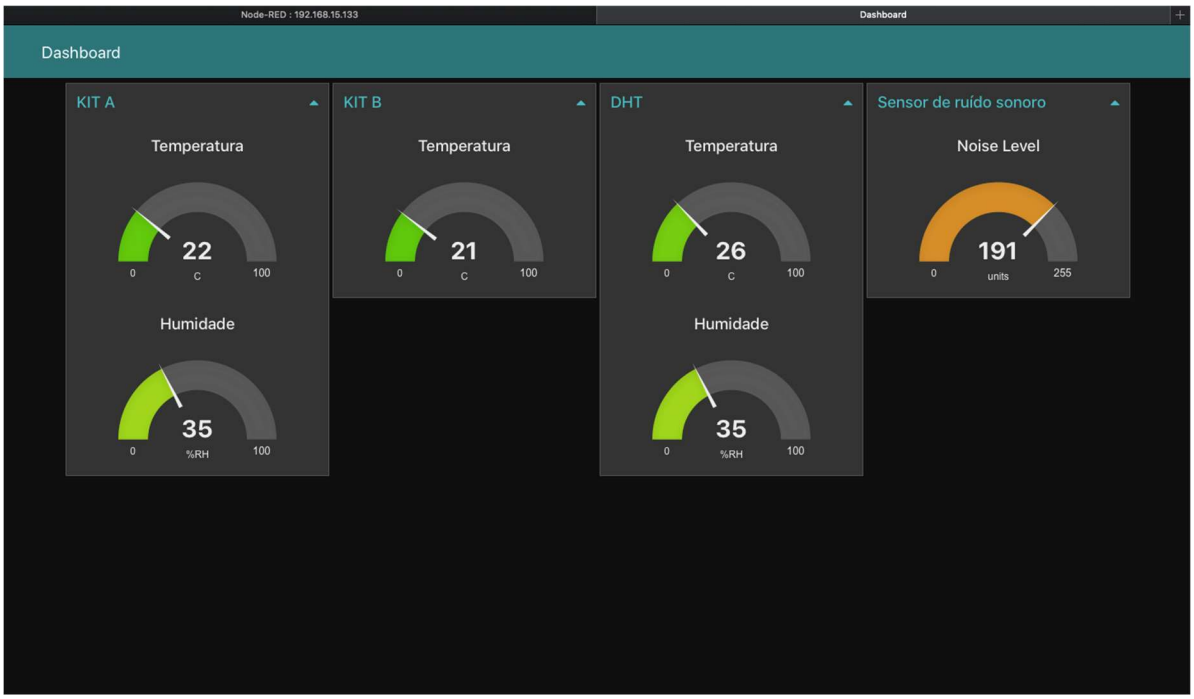
Na Figura 29 estão dispostos os dados recebidos pelo módulo *IC*, na interface criada pelo *Node-RED*. Nota-se que é possível agrupar os indicadores de acordo com o nó transmissor da mensagem, assim como estabelecer diferentes códigos de cores para dados recebidos de acordo com os limites da variável sendo monitorada.

Figura 28 – Ambiente de programação do *Node-RED*



Fonte: Produção do próprio autor.

Figura 29 – Painel de controle desenvolvido



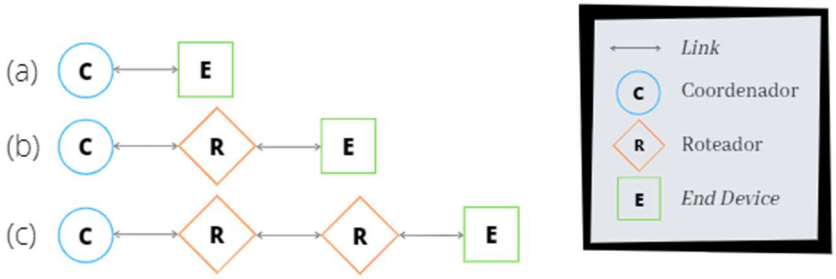
Fonte: Produção do próprio autor.

6.4 Avaliação da capacidade dos rádios de comunicação

Com o objetivo de avaliar experimentalmente os limites da comunicação *ZigBee*, foram realizados testes, em campo aberto, da vazão máxima de dados entre os dispositivos, potência do sinal recebido e distância máxima entre dois rádios adjacentes.

Para esta etapa do projeto, as ferramentas de teste do *X-CTU* foram de extrema importância, uma vez que oferecem toda a estrutura necessária em nível de *software* para o estudo da capacidade da rede. As topologias utilizadas neste estudo podem ser observadas na Figura 30.

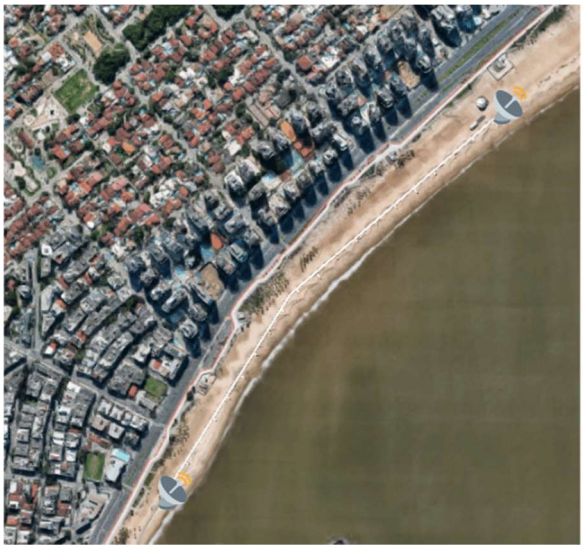
Figura 30 – Topologias usadas nos testes



Fonte: Produção do próprio autor.

Devido à necessidade de um amplo espaço livre entre os dispositivos para a realização dos testes de distância, estes ocorreram na orla da Praia de Camburi, em Vitória, Espírito Santo. A imagem de satélite do local é apresentada na Figura 31. A distância total máxima percorrida é de cerca de trezentos e cinquenta metros.

Figura 31 – Disposição dos rádios de comunicação



Fonte: Produção do próprio autor.

7 RESULTADOS OBTIDOS

7.1 Aplicação de monitoramento da rede de sensores

No desenvolvimento da aplicação de monitoramento da rede de sensores foram utilizados os kits apresentados no Capítulo 6. O foco deste experimento foi simular um ambiente industrial onde é necessário monitorar variáveis do processo como temperatura, humidade relativa do ar e nível de ruído sonoro do ambiente.

7.1.1 Autonomia da rede

Foram avaliados os kits de monitoramento alimentados por conjuntos de pilhas associadas de modo que formem uma bateria de 4,5 V e 6300 mAh. No total, foram instalados dois kits com *Seeeduino*, dois kits com Arduino, um servidor *Ethernet*, um módulo *IC* e um coordenador da rede alimentados via porta *USB.*

Neste primeiro experimento todos os módulos foram configurados como roteadores, com exceção do coordenador da rede. Os microcontroladores foram programados de modo a transmitir os dados lidos pelos sensores a cada 10 segundos. Na Tabela 1 pode ser observado o consumo de corrente dos módulos da rede.

Tabela 1 – Consumo dos módulos

Módulo	**Consumo medido (mA)**	**Consumo medido (*cyclic sleep*) (mA)**	**Autonomia (h)**	**Autonomia (*cyclic sleep*) (h)**
Coordenador	25	-	-	-
IC	300	-	-	-
KIT A	60	29	105	217,24
KIT B	58	28	108,62	225
Sensor DHT	87	44	72	143
Sensor Ruído	73	30	86	210
Servidor Eth.	250	216	-	-

Fonte: Produção do próprio autor.

Durante a realização das medições, ficou evidente que a configuração do modo *cyclic sleep* nos rádios *XBee* reduz consideravelmente o consumo de energia dos módulos. Ainda assim, os resultados referentes à autonomia dos kits não são satisfatórios levando em conta a aplicação em sistemas embarcados de difícil manutenção, onde não há possibilidade de substituição das fontes de energia com frequência. Entretanto, como o objetivo do trabalho não é o

desenvolvimento de nós sensores com baixo consumo de energia, mas o estudo do protocolo *ZigBee* para comunicação entre eles, os resultados obtidos não impactam negativamente no restante do estudo.

Os nós sensores são compostos basicamente por um microcontrolador, o rádio de comunicação e os sensores. Como já foi tratado o consumo de energia dos rádios *XBee* e os sensores são dispositivos cujas correntes estão na ordem de unidades de mA, a fonte do elevado consumo de corrente dos módulos sensores é o processamento das placas Arduino e *Seeeduino*, por eliminação. A presença de conversores analógico-digitais, reguladores de tensão e *LEDs* indicando o funcionamento da placa acaba elevando o consumo de corrente do módulo. Adicionalmente, apesar de os microcontroladores apresentarem a função de economia de energia, os outros componentes das placas de prototipagem permanecem ativos.

Portanto, de maneira a otimizar o gerenciamento de energia dos nós sensores, uma boa alternativa seria utilizar os microcontroladores isoladamente das placas de prototipagem, eliminando gastos desnecessários de energia.

7.1.2 Tráfego de dados

Para efeitos experimentais, os microcontroladores foram programados para coletar informações dos sensores conectados e enviar os dados para o rádio responsável pela concentração de informações repetidamente a cada 10 segundos.

O tamanho dos pacotes enviados por cada rádio variava com a quantidade de dados coletada e transmitida por cada nó. Além dos *bytes* de dados nos pacotes, outras informações referentes a delimitador inicial, tamanho do pacote, endereçamento do dispositivo destino, entre outros, garantem que os pacotes possuam tamanho inicial de 18 *bytes*. Os tamanhos dos pacotes enviados pelos módulos podem ser observados no Quadro 6. Não foram considerados os módulos que somente recebem os dados da rede.

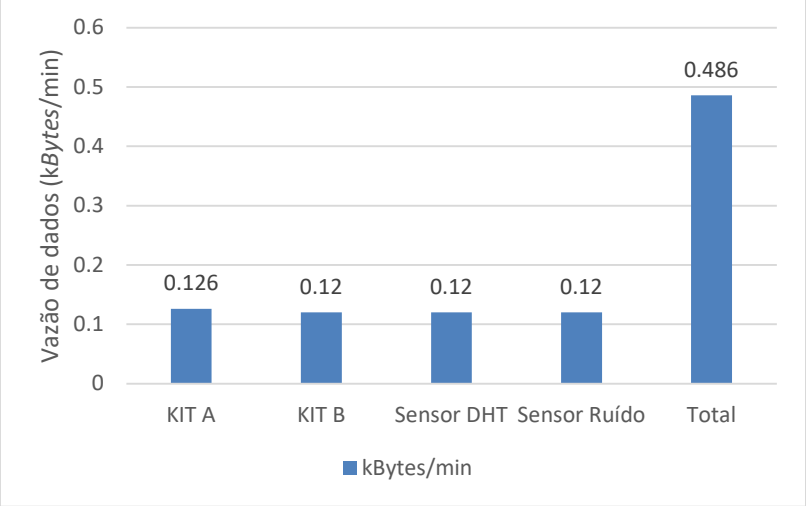
Como os módulos enviam dados a cada 10 segundos, com tamanhos especificados na **Erro! Fonte de referência não encontrada.**, tem-se que o *throughput* do KIT B, sensor DHT e sensor de ruído sonoro é de 0,12 *kBytes/min* enquanto do KIT A é de 0,126 *kBytes/min*. Um resumo dessas informações é representado na Figura 32, juntamente com o *throughput* total da rede desenvolvida.

Quadro 6 – Tamanho dos pacotes transmitidos

Módulo	***Bytes* de dados**	**Tamanho total (*bytes*)**
KIT A	3	21
KIT B	2	20
Sensor DHT	2	20
Sensor Ruído	2	20

Fonte: Produção do próprio autor.

Figura 32 – Tráfego gerado na rede



Fonte: Produção do próprio autor.

7.2 Avaliação experimental do potencial da rede

Para a realização dos experimentos, foram utilizados os mesmos módulos sensores apresentados anteriormente, com a diferença de que nesta etapa os microcontroladores servem somente para alimentação dos rádios e a topologia utilizada está representada na Figura 30. Toda a configuração dos pacotes transmitidos, realização da comunicação e medição dos parâmetros de desempenho fica a cargo do *X-CTU*.

Os rádios foram configurados com *baud rate* de 9600 bps e equipados com antenas de 2,4 GHz. A documentação do *XBee* (DIGI, 2018a) garante comunicação para até 140m em espaços externos. Com os equipamentos disponíveis, um dos objetivos dos experimentos foi ir além do oferecido.

7.2.1 Medição de *RSSI*

A indicação da intensidade do sinal relativo ao último pacote recebido pelo dispositivo é expressa em -dBm. A ideia principal por trás dessa indicação é o decaimento da potência do sinal à medida que uma distância é percorrida. No espaço livre, essa potência decai com o quadrado da distância entre transmissor e receptor (DARGIE, 2010). A partir da equação de Friis, a potência de recepção é ilustrada pela equação (1).

$$P_r = P_t \times G_t \times G_r \left(\frac{\lambda}{4\pi d}\right)^2 \tag{1}$$

Onde, G_t, G_r são, respectivamente, os ganhos de transmissão e recepção. λ é o comprimento de onda e d representa a distância entre transmissor e receptor. O *RSSI* pode ser obtido, então, a partir da equação (2).

$$RSSI = 10\ log \frac{P_r}{P_{ref}} \tag{2}$$

Onde, P_{ref} é definido por uma potência recebida de referência, tipicamente representada pelo valor $P_{ref} = 1\ mW$. Para coletar os dados experimentais da medição, foi utilizada a topologia de rede (a) da Figura 30, com um coordenador e um *end device*. O valor de *RSSI* medido é obtido após a transferência de 100 pacotes de 84 *bytes* cada e feita a média do *RSSI* de cada transmissão. A distância entre o coordenador e o *end device* foi variada para se estabelecer uma relação entre os valores de *RSSI* e distância. A interface da ferramenta de testes utilizada pode ser vista na Figura 33. Os resultados obtidos estão dispostos na Figura 34.

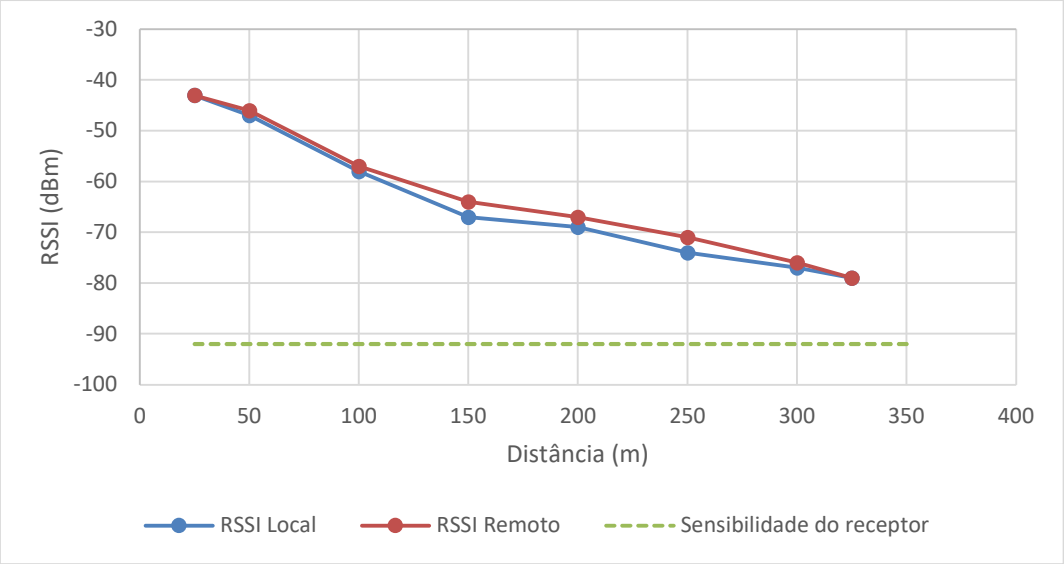
No ambiente de testes disponibilizado pelo *X-CTU*, é possível observar a identificação dos nós que estão passando pelo teste, a configuração dos pacotes transmitidos, como tamanho, número de pacotes transmitidos durante o teste, tempo máximo de transmissão e o intervalo entre pacotes. Um gráfico é atualizado em tempo real sumarizando os dados do teste e elaborando curvas de *RSSI* e porcentagem de perda de pacotes.

Pode-se observar na Figura 34 que a potência do sinal recebido é de -43 dBm para curtas distâncias de cerca de 25 metros e tende a diminuir à medida que aumenta a distância entre emissor e receptor, de modo que, com 325 metros de distância a comunicação fica comprometida e a porcentagem de erros se torna muito expressiva, se aproximando de 83% conforme mostra a Figura 35.

Figura 33 – Ambiente para testes de distância

Fonte: Produção do próprio autor.

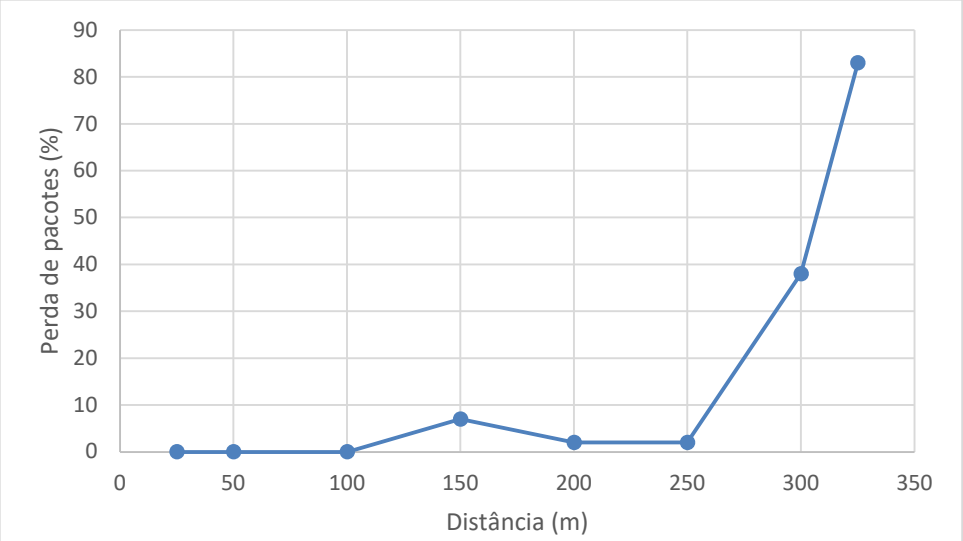
Figura 34 – Dados obtidos no teste RSSI



Fonte: Produção do próprio autor.

Na Figura 34 a legenda *RSSI* Local faz referência aos dados coletados na unidade conectada ao computador que realiza o teste enquanto *RSSI* Remoto representa o nó distante de onde é realizada a medição. Observa-se que tanto o *RSSI* Local quanto o *RSSI* Remoto não atingiram a sensibilidade máxima do receptor *ZigBee*, devido fatores como a pouca elevação dos rádios durante os testes e interferência de pessoas ou objetos na comunicação.

Figura 35 – Perda de pacotes



Fonte: Produção do próprio autor.

Vale observar que na realização de testes de *RSSI* não faz sentido utilizar uma topologia de rede com múltiplos saltos, uma vez que esta é a medida da intensidade do último pacote recebido, independentemente de quantos saltos houve anteriormente.

7.2.2 Vazão de dados

A vazão de dados em uma rede (*throughput)* é expressa como a quantidade de dados transferidos em um determinado intervalo de tempo e é influenciada pelo tamanho dos pacotes e velocidade na transmissão. Esse indicador pode ser calculado conforme a equação (3) a seguir.

$$TP = \frac{número\ de\ bytes}{tempo\ total\ de\ transmissão\ (seg)} \tag{3}$$

Para a realização deste experimento, foi utilizada a ferramenta para medição de *throughput* do *X-CTU*. Para todos os testes, foi utilizado um intervalo de um minuto de transmissão e pacotes de 84 *bytes*. Fatores como perda de pacotes e atraso na transmissão influenciam negativamente na medição. Novamente, a distância entre transmissor e receptor foi variada para estabelecer uma relação entre os valores de *throughput* e distância.

Na Figura 36 pode-se observar o ambiente de testes de *throughput* no *software X-CTU*. Nele podem ser configurados os nós usados no processo, o tipo de pacote transmitido entre eles e a duração do teste, em número de pacotes ou temporizada. Neste teste foi definido um intervalo de um minuto para cada teste realizado. Na área gráfica do ambiente observa-se a variação da

taxa de transferência ao decorrer do tempo. As regiões do gráfico com quedas súbitas na taxa de transferência representam as perdas de pacotes na transmissão.

Figura 36 – Ambiente para testes de *throughput*

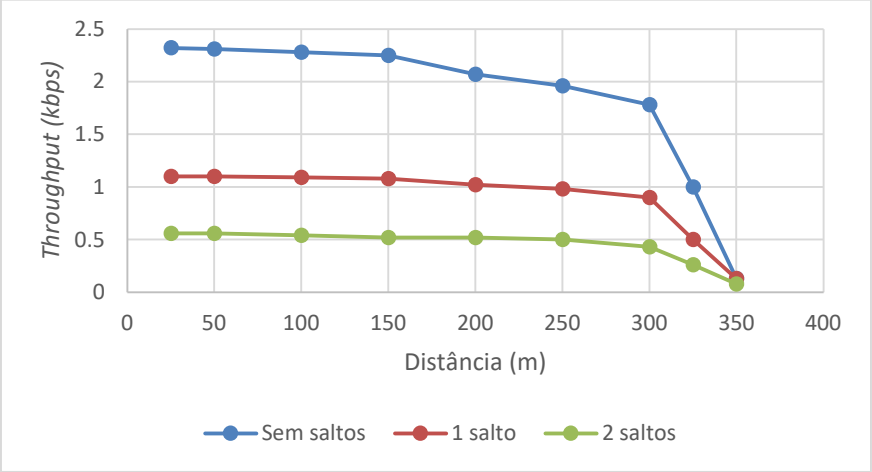
Fonte: Produção do próprio autor.

Os resultados obtidos são mostrados na Figura 37. Neste caso, para medir o desempenho na comunicação com múltiplos saltos, foram utilizadas as topologias de rede (b) e (c) da Figura 30, onde os pacotes enviados do coordenador para o *end device* passam antes pelos roteadores. Nota-se que a presença dos saltos reduz consideravelmente a vazão máxima de dados entre os dispositivos, reduzindo em cerca de 50% sua capacidade a cada salto realizado.

De acordo com a fabricante dos rádios *XBee*, a máxima distância que garante o funcionamento de acordo com os padrões estabelecidos é de 120 metros em ambientes externos, com uma taxa de 250 kbps (DIGI, 2018a). Porém, a máxima vazão obtida nos testes foi de 2,32 kbps, um valor muito abaixo do esperado.

Possivelmente, os valores estabelecidos pela *Digi* foram obtidos em ambientes ideais, onde o caminho entre os dispositivos testados está totalmente livre de qualquer interferência e a *baud rate* configurada nos rádios com valor máximo possível, o que tornaria a aplicação incompatível com os microcontroladores *Arduino* e *Seeeduino*. Em contrapartida, a máxima distância com a qual os dispositivos mantiveram conexão foi superior à esperada, atingindo cerca de 300 metros.

Figura 37 – *Throughput* em função da distância



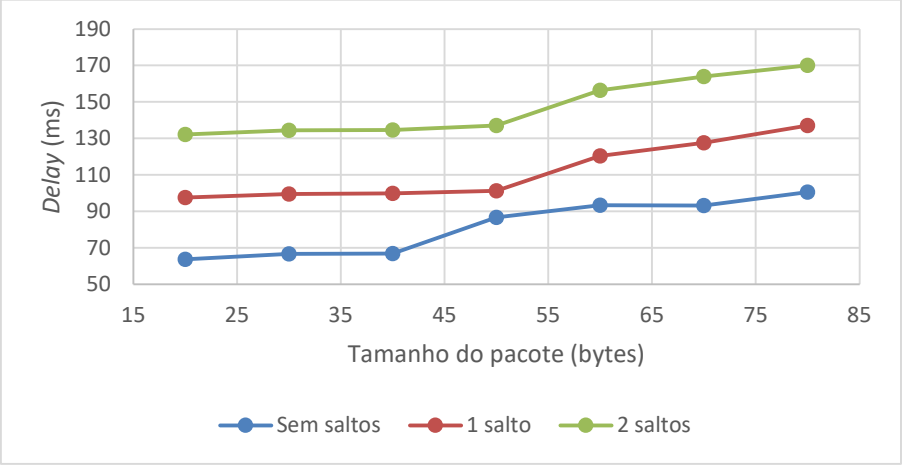
Fonte: Produção do próprio autor.

Há duas possíveis maneiras de se garantir que dados realizem dois saltos na rede *ZigBee*: aumentando a distância entre dois nós até que a conexão entre eles deixe de existe e então adicionando um terceiro nó entre os dois iniciais, fazendo que as informações passem necessariamente pelo nó central; ou, como realizado no experimento, utilizando o parâmetro *Node Join Time* na configuração dos rádios *XBee*, de maneira que, após um determinado tempo, um nó não aceitasse conexões de novos módulos e, obrigatoriamente, dispositivos que tentassem entrar na rede devem fazê-lo por meio do nó central que ainda aceita novas conexões.

7.2.3 Atraso na entrega de pacotes

Outro importante indicador de desempenho em uma rede *ZigBee* é o tempo entre o envio de um pacote e sua efetiva recepção por outro dispositivo. A distância entre os nós é de 50 metros e o tamanho dos pacotes é variado para estabelecer uma relação entre tamanho do pacote e o atraso na transmissão. A Figura 38 mostra os resultados obtidos tanto na transmissão direta entre coordenador e dispositivo final quanto na transmissão indireta por meio dos roteadores.

Figura 38 – *Delay* na entrega de pacotes



Fonte: Produção do próprio autor.

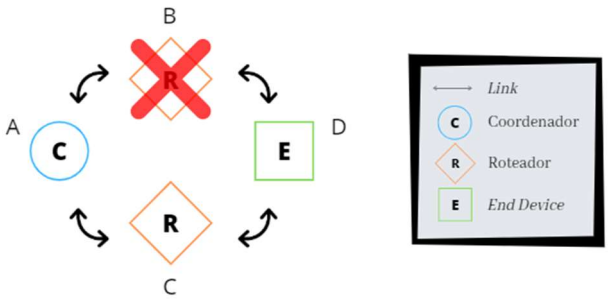
7.2.4 Tempo de auto regeneração da rede

O estabelecimento da rede *ZigBee* é feita de forma automática pelos próprios dispositivos da rede. O coordenador inicia a rede e os outros dispositivos, por meio de requerimentos de associação, entram na mesma. Uma das características mais importantes do padrão *ZigBee* é a capacidade de auto regeneração da rede quando alguma rota já estabelecida sofre danos ou deixa de existir.

Para avaliar o tempo que a rede demora para criar uma nova rota quando um nó perde a conexão, este experimento se baseia na topologia de rede mostrada na Figura 39. Foi medido o tempo entre a eliminação de um nó roteador e a criação de outra rota de comunicação entre o coordenador da rede e o *end device*.

Após diversas medidas e feita a média, foi observado um tempo máximo de reconfiguração da rede de 161 ms e mínimo de 92 ms, para pacotes contendo 84 bytes de dados.

Figura 39 – Topologia usada no teste de auto regeneração



Fonte: Produção do próprio autor.

7.3 Desafios para a implantação do sistema

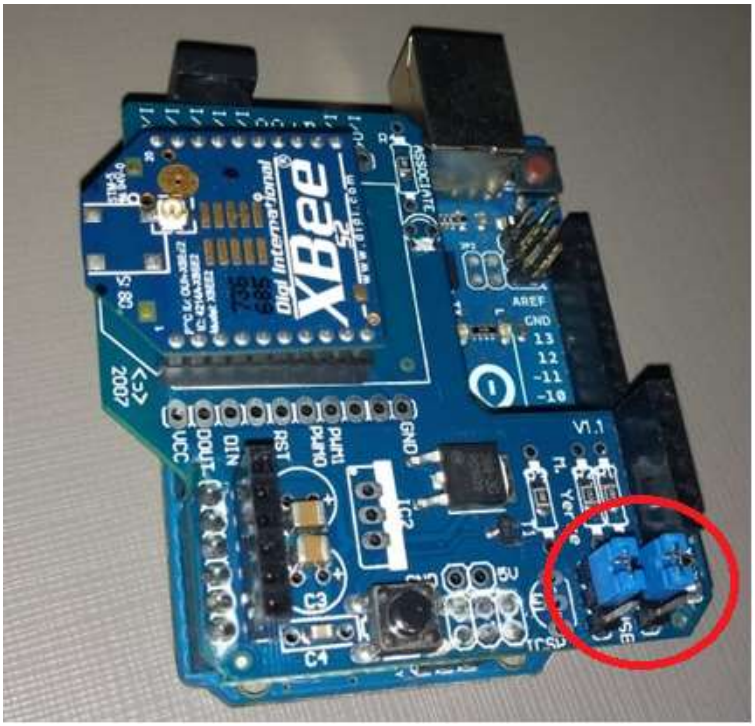
No início do projeto, os rádios *XBee* seriam utilizados com auxílio de placas desenvolvidas manualmente pelo autor, porém, devido ao seu formato compacto, tornou-se uma tarefa muito trabalhosa e que demandaria muito tempo e esforço. Portanto, foi decidido utilizar adaptadores existentes no mercado para fazer a conexão do *XBee* com o Arduino. Felizmente, algumas placas utilizadas no estudo já possuem o encaixe para dispositivos da família *Bee*.

Os *shields XBee* comprados logo foram comprados e apresentaram alguns entraves. Primeiramente, a falta de informações claras sobre o funcionamento de seus componentes tornou lento o processo de familiarização e aprendizagem. Ainda, algumas configurações da placa foram descobertas após vários dias de testes frustrados. Uma delas é o posicionamento de duas chaves presentes na placa que determinam o funcionamento do *XBee* conectado, como pode ser observado na Figura 40.

Após a montagem dos módulos sensores, iniciou-se a etapa de programação dos microcontroladores. Os rádios foram configurados em modo *AT*, pela aparente simplicidade de funcionamento. Entretanto, no momento de concatenar os dados, foi necessário desenvolver um algoritmo para recepção de dados, pois os valores lidos pelos sensores eram enviados como vetores de caracteres. Consequentemente, o microcontrolador que recebia os dados era responsável por identificar o início e fim de cada transmissão assim como a origem do pacote recebido. Assim, cada informação enviada era codificada de maneira que os pacotes contivessem a fonte que gerou aqueles dados e caracteres delimitadores que indicassem início e fim do pacote. Todo o processamento exigido pelo algoritmo de decodificação acabava por aumentar o consumo de energia do microcontrolador e congestionar os canais de comunicação. Convenientemente, com a substituição do modo *AT* para *API*, os procedimentos de codificação e decodificação dos dados são feitos automaticamente pelos rádios. Os códigos desenvolvidos para comunicação em modo *AT* estão disponíveis no APÊNDICE B.

Outro fator que trouxe dificuldades na realização do projeto foi a necessidade de utilizar módulos empilhados, no caso do módulo servidor *Ethernet*. Essas placas não foram desenvolvidas para funcionarem de tal maneira, de modo que a alimentação transmitida do Arduino para o *shield Ethernet* não era levada para o *XBee* acima. Portanto, foi necessário fazer ajustes no cabeçote de Programação Serial no Circuito (*ICSP*, do inglês *In-Circuit Serial Programming*), prolongando os pinos de alimentação para as camadas superiores.

Figura 40 – Chaves de configuração



Fonte: Produção do próprio autor.

Embora de extrema importância para aplicações embarcadas, o modo *sleep* dos rádios de comunicação pode trazer dificuldades relacionadas ao uso do *X-CTU.* Esse problema acontece, pois, enquanto um *XBee* está dormindo, o *software* não é capaz de identificar o nó e incluí-lo na rede. Consequentemente, não é possível editar as configurações do rádio remotamente nem realizar testes de desempenho com o mesmo.

8 CONCLUSÕES

Este trabalho apresentou um estudo experimental do protocolo de comunicação *ZigBee* e de suas características mais relevantes em um contexto de Internet das Coisas Industrial. Foi implementada uma pequena rede com topologia *mesh* para verificar a eficácia dessa tecnologia na prática, com sensores sem fio e acumuladores de dados.

As informações transmitidas na rede são confiáveis e a estrutura da rede é estável, uma vez que apresenta capacidade de auto-regeneração em caso de falha em algum canal de comunicação. Na ordem de milésimos de segundos a topologia da rede se altera, adaptando-se à entrada ou saída de nós.

Para melhor visualização dos dados coletados pelos sensores da rede, dois nós desempenharam o importante papel de concatenar e tratar as informações recebidas, disponibilizando de maneira segura e amigável ao usuário. Por meio de redes *Wi-Fi* locais, qualquer usuário com acesso à rede pode visualizar as informações em computadores ou celulares.

Tratando-se de ambientes industriais, onde a distância entre nós sensores não é muito grande, a tecnologia prova-se eficaz e suficiente, uma vez que essas aplicações, em geral, não envolvem grandes quantidades de dados, limitados a comandos e informações coletadas por sensores.

O tempo e velocidade da propagação de dados na rede apresentaram resultados satisfatórios, uma vez que não foram considerados muitos saltos entre os nós. Vale ressaltar que em redes com um maior número de saltos entre os nós roteadores pode ocasionar um atraso expressivo na recepção do sinal.

Como trabalhos futuros, é proposto o desenvolvimento de uma aplicação focada em baixo consumo de energia, que funcione por um grande período de tempo coletando dados e armazenando em bancos de dados para posterior análise. Pode-se ainda realizar estudos comparativos entre o protocolo *ZigBee* e outras tecnologias desenvolvidas para aplicações similares, como *Bluetooth* de Baixo Consumo (*BLE*, do inglês *Bluetooth Low Energy*), *Z-Wave*, *LoRaWAN* e Comunicação de Campo Próximo (*NFC*, do inglês *Near Field Communication*).

REFERÊNCIAS BIBLIOGRÁFICAS

AKERBERG, J.; GIDLUND, M.; BJORKMAN, M. Future research challenges in wireless sensor and actuator networks targeting industrial automation. **IEEE International Conference on Industrial Informatics**, n. 9, p. 410-415, 2011.

ALLIANCE, Z. **ZigBee Specification version 1.0**, 2005.

ARDUINO. **Arduino Uno Rev3.** 2019. Disponível em: https://store.arduino.cc/usa/arduino-uno-rev3. Acesso em: 03 nov. 2019.

BMBF-INTERNETREDAKTION. **Zukunftsprojekt Industrie 4.0 – BMBF.** 2019. Disponível em: https://www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html. Acesso em: 29 nov. 2019.

BOXALL, J. **Arduino workshop**: A Hands-On introduction with 65 projects. San Francisco: No Starch Press, 2013.

CANTHERM. **Pearl-Shaped Precision NTC Thermistor for Temperature Measurement**. 2012. Disponível em: https://img.filipeflop.com/files/download/Datasheet_termistor_mf52_1.pdf. Acesso em: 14 nov. 2019.

CHEN, J.; PANG, A.; SHEU, S.; TSENG, H. High performance wireless switch protocol for IEEE 802.11 wireless networks. **Mobile Networks and Applications**, v. 10, n. 5, p. 741-751, 2005.

CHI, C.; HATLER, M. **Wireless sensor network. Mass market opportunities**. San Diego: ON World, 2004.

COLUMBUS, L. 2018 roundup of internet of things forecasts and market estimates. **Forbes**, California, 2018. Disponível em: https://www.forbes.com/sites/louiscolumbus/2018/12/13/2018-roundup-of-internet-ofthings-forecasts-and-market-estimates/#62536c7d28c5. Acesso em: 02 jun. 2019.

CONNER, M. Sensors empower the "Internet of Things". **EDN (Electrical Design News)**, v. 55, n. 10, p. 32, 2010.

DAEMEN, J.; RIJMEN, V**. AES proposal: Rijndael**. Brussel: Proton World, 1999.

DARGIE, W.; POELLABAUER, C. **Fundamentals of wireless sensor networks: theory and practice**. Singapore: John Wiley & Sons, 2010.

DROBOTICS. **DHT11 Humidity & Temperature Sensor.** 2010. Disponível em: http://wiki.sunfounder.cc/images/c/c7/DHT11_datasheet.pdf. Acesso em: 13 nov. 2019.

DIGI. **XBee / XBee-PRO ZB RF Modules User Guide**. 2018a. Disponível em: https://www.digi.com/resources/documentation/digidocs/PDFs/90000976.pdf. Acesso em: 02 jun. 2019.

DIGI. **X-CTU User Guide.** 2018b. Disponível em: https://www.digi.com/resources/documentation/digidocs/90001458-13/default.htm. Acesso em: 12 nov. 2019.

ERGEN, S. C. **ZigBee/IEEE 802.15. 4 Summary**. California: U.C. Berkeley, 2004.

FARAHANI, S. **ZigBee wireless networks and transceivers**. San Diego: Elsevier, 2011.

FOROUZAN, B. A. **Comunicação de Dados e Redes de Computadores**. Porto Alegre: AMGH Editora, 2009.

GRIFFITHS, F.; OOI, M. The Fourth Industrial Revolution — Industry 4.0 and IoT. **IEEE Instrumentation & Measurement Magazine**, v. 21, n. 6, p. 29-43, 2018.

ILYAS, M.; MAHGOUB, I. **Handbook of sensor networks**. Boca Raton, Florida: CRC Press, v. 1, 2004.

JOY-IT. **KY-038 Microphone sound sensor module**. 2017. Disponível em: https://datasheetspdf.com/pdf-file/1402048/Joy-IT/KY-038/1. Acesso em: 13 nov. 2019.

KOHVAKKA, M.; KUORILEHTO, M.; HANNIKAINEN, M.; HAMALAINEN, T.D. Performance analysis of IEEE 802.15. 4 and ZigBee for large-scale wireless sensor network applications. *In*: PROCEEDINGS OF THE 3RD ACM INTERNATIONAL WORKSHOP ON PERFORMANCE EVALUATION OF WIRELESS AD HOC, SENSOR AND UBIQUITOUS NETWORKS, 2006, Terremolinos, Espanha. **Anais** […]. Nova York: ACM, 2006. p. 48-57.

KUROSE, J. F.; ROSS, K. W. **Computer Networking**. 6. ed. Boston: Pearson, 2013.

LIBELIUM. **XBee Shield.** 2016. Disponível em: https://www.arduino.cc/en/Main/ ArduinoXbeeShield. Acesso em: 11 nov. 2019.

MICRIUM. **How to Think about the Internet of Things (IoT)**. 2017. Disponível em: https://www.micrium.com/iot/devices/. Acesso em: 28 maio 2019.

NODE-RED. **Low-code programming for event-driven applications.** 2019. Disponível em: https://nodered.org/. Acesso em: 28 nov. 2019.

PRESS, G. Internet of Things by numbers: Market estimates and forecasts. **Forbes,** California, 2014. Disponível em: http://www.forbes.com/sites/gilpress/2014/08/22/internet-of-things-by-the-numbers-market-estimates-and-forecasts. Acesso em: 02 jun. 2019.

RASPBERRY. **Compute module 3+ Datasheet.** 2019. Disponível em: https://www.raspberrypi.org/documentation/hardware/computemodule/datasheets/rpi_DATA_CM3plus_1p0.pdf. Acesso em: 28 nov. 2019.

RIVERO, I. A. S. **Redes de Sensores sem Fio para Monitoramento de Equipamentos Eletrônicos.** 2011. Dissertação (Mestrado em Informática) – Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte, 2011.

SANTOS, S. T. D. **Redes de sensores sem fio em monitoramento e controle**. 2007. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2007.

SEEED. **Seeeduino Stalker V2.0.** 2011**.** Disponível em: https://seeeddoc.github.io/Seeeduino_Stalker_v2.0/. Acesso em: 12 nov. 2019.

SISINNI, E.; SAIFULLAH, A.; HAN, S.; JENNEHAG, U.; GIDLUND, M. Industrial Internet of Things: Challenges, Opportunities, and Directions. **IEEE Transactions on Industrial Informatics**, v. 14, n. 11, p. 4724-4734, 2018.

SOHRABY, K.; MINOLI, D.; ZNATI, T. **Wireless Sensor Networks:** technology, protocols and applications. Hoboken, New Jersey: John Wiley & Sons, 2007.

SOUSA, M. P.; LOPES, W. T. A. Desafios em Redes de Sensores sem Fio. **Revista de Tecnologia da Informação e Comunicação**, v. 1, n. 1, p. 41-47, 2011.

TANENBAUM, A. S. **Redes de computadores**. 4. ed. Amsterdam: Campus, 2003.

TOSE, T. **Redes de sensores sem fio ZigBee aplicada em uma estação de tratamento de esgoto**. 2012. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal do Espírito Santo, Vitória, 2012.

WEYRICH, M.; EBERT, C. Reference architectures for the internet of things. **IEEE Software**, v. 1, p. 112-116, 2016.

WIZNET. **W5100 Datasheet**. 2008. Disponível em: https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100_Datasheet_v1_1_6.pdf. Acesso em: 17 nov. 2019.

APÊNDICE A – CÓDIGOS EM MODO *API*

Neste apêndice são mostradas as rotinas de leitura dos sensores utilizados com *Arduinos* e *Seeeduinos*, em modo *API*, adotado como referência neste trabaho para o desenvolvimento da aplicação de monitoramento. O *software* utilizado para a compilação dos dados é o *Arduino IDE*, como citado na seção 5.1.

- **KIT 1**

```
#include <XBee.h>
#include <math.h>
#define ThermistorPIN 1 //PINO ANALOGICO 1

XBee XBee = XBee();
uint8_t payload[] = {0, 0};
// SH + SL Address of receiving XBee
XBeeAddress64 addr64 = XBeeAddress64(0x00000000, 0x0000FFFF);
ZBTxRequest zbTx = ZBTxRequest(addr64, payload, sizeof(payload));
ZBTxStatusResponse txStatus = ZBTxStatusResponse();

float pad = 9805; //valor resistencia do divisor de tensão
float thermr = 10000; //valor termistor (seeed = 10k ohms)
float Thermistor(int RawADC) {
  long Resistance;
  float Temp;
  Resistance = ((1024 * pad / RawADC) - pad);
  Temp = log(Resistance);
  Temp = 1 / (0.001129148 + (0.000234125 * Temp) + (0.0000000876741 * Temp * Temp *
          Temp));
  Temp = Temp - 273.15;
  return Temp;
}
// ********************************* KIT 1 *********************************
void setup() {
  Serial.begin(9600);
  XBee.setSerial(Serial);
}
void loop() {
  float temp;
  temp = Thermistor(analogRead(ThermistorPIN)) - 3;
  payload[0] = temp;
  XBee.send(zbTx);
  delay(10000);
}
```

- **KIT 2**

```
#include <XBee.h>
#include <math.h>
#define ThermistorPIN 1 //PINO ANALOGICO 1
XBee XBee = XBee();
uint8_t payload[] = {0, 0};
// SH + SL Address of receiving XBee
XBeeAddress64 addr64 = XBeeAddress64(0x00000000, 0x0000FFFF);
ZBTxRequest zbTx = ZBTxRequest(addr64, payload, sizeof(payload));
ZBTxStatusResponse txStatus = ZBTxStatusResponse();

float pad = 9805; //valor resistencia do divisor de tensão
int rhsense = 0;
int rh = 0;


float thermr = 10000; //valor termistor (seeed = 10k ohms)
float Thermistor(int RawADC) {
  long Resistance;
  float Temp;
  Resistance = ((1024 * pad / RawADC) - pad);
  Temp = log(Resistance);
  Temp = 1 / (0.001129148 + (0.000234125 * Temp) + (0.0000000876741 * Temp * Temp *
          Temp));
  Temp = Temp - 273.15;
  return Temp;
}

// ******************************** KIT 2 ********************************
void setup() {
  Serial.begin(9600);
  XBee.setSerial(Serial);
}
void loop() {
  float temp;
  temp = Thermistor(analogRead(ThermistorPIN))-3;
  rhsense = (analogRead(2));
  rh = ((30.855 * (rhsense / 204.6)) - 11.504);
  payload[0] = temp;
  payload[1] = rh;
  XBee.send(zbTx);
  delay(10000);
}
```

- **SENSOR DHT**

```
#include <dht.h>
#include <XBee.h>
#include <LowPower.h>
#define DHT11_PIN 7

XBee XBee = XBee();
uint8_t payload[] = {0, 0};
// SH + SL Address of receiving XBee
XBeeAddress64 addr64 = XBeeAddress64(0x00000000, 0x0000FFFF);
ZBTxRequest zbTx = ZBTxRequest(addr64, payload, sizeof(payload));

dht DHT;
int temp, humidity;


void setup(){
   Serial.begin(9600);
   XBee.setSerial(Serial);
}

void loop()
{
  int chk = DHT.read11(DHT11_PIN);
  payload[0] = DHT.temperature;
  payload[1] = DHT.humidity;
  XBee.send(zbTx);
  delay(10000);
}
```

- **SENSOR DE RUÍDO SONORO**

```
#include <XBee.h>
int sensorPin = A0; // select the input pin for the potentiometer
int sensorValue = 0; // variable to store the value coming from the sensor

XBee XBee = XBee();
uint8_t payload[] = {0, 0};
// SH + SL Address of receiving XBee
XBeeAddress64 addr64 = XBeeAddress64(0x00000000, 0x0000FFFF);
ZBTxRequest zbTx = ZBTxRequest(addr64, payload, sizeof(payload));
void setup ()
{
  Serial.begin (9600);
  XBee.setSerial(Serial);

}
```

```
void loop ()
{
  sensorValue = analogRead (sensorPin);
  payload[0] = sensorValue;
  XBee.send(zbTx);
  delay(5000);
}
```

APÊNDICE B – CÓDIGOS EM MODO *AT*

Neste apêndice são mostradas as rotinas de leitura dos sensores utilizados com *Arduinos* e *Seeeduinos*, desenvolvidas para o funcionamento em modo *AT*, posteriormente substituídas por códigos para modo *API*. O *software* utilizado para a compilação dos dados é o *Arduino IDE*, como citado na seção 5.1.

- **KIT 1**

```
#include <math.h>
#define ThermistorPIN 1 //PINO ANALOGICO 1
float pad = 9805; //valor resistencia do divisor de tensão

float thermr = 10000; //valor termistor (seeed = 10k ohms)
float Thermistor(int RawADC) {
  long Resistance;
  float Temp;
  Resistance=((1024 * pad / RawADC) - pad);
  Temp = log(Resistance);
  Temp = 1 / (0.001129148 + (0.000234125 * Temp) + (0.0000000876741 * Temp * Temp *
  Temp));
  Temp = Temp - 273.15;
  return Temp;
}
// ********************************** KIT 1 **********************************
void setup() {
Serial.begin(9600);
}
void loop() {
float temp;
temp=Thermistor(analogRead(ThermistorPIN));
Serial.print("[E:");
Serial.print(temp-3,1);
Serial.println("]");
delay(1000);
}
```

- **KIT 2**

```
#include <math.h>
#define ThermistorPIN 1 //PINO ANALOGICO 1
float pad = 9805; //valor resistencia do divisor de tensão
int rhsense = 0;
int rh = 0;
```

```
float thermr = 10000; //valor termistor (seeed = 10k ohms)
float Thermistor(int RawADC) {
  long Resistance;
  float Temp;
  Resistance=((1024 * pad / RawADC) - pad);
  Temp = log(Resistance);
  Temp = 1 / (0.001129148 + (0.000234125 * Temp) + (0.0000000876741 * Temp * Temp *
  Temp));
  Temp = Temp - 273.15;
  return Temp;
}

// ******************************** KIT 2 ********************************
void setup() {
Serial.begin(9600);
}
void loop() {
float temp;
temp=Thermistor(analogRead(ThermistorPIN));
rhsense = (analogRead(2));
rh = ((30.855*(rhsense/204.6))-11.504);

Serial.print("[T:");
Serial.print(temp-3,1);
Serial.println("]");
delay(3000);
Serial.print("[H:");
Serial.print(rh);
Serial.println("]");
delay(3000);
}
```

- **Servidor *Ethernet***

```
#include <SPI.h>
#include <Ethernet.h>
// Variaveis de configuração do server
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };//MAC ARDUINO
IPAddress ip(192, 168, 1, 45);
EthernetServer server(80);
// Variaveis gerais
const byte numChars = 10;
char receivedChars[numChars];
boolean newData = false;
float T, E;
int P, H;
```

```
void setup() {
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  Serial.println("Ethernet WebServer Example");
  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);

  // Check for Ethernet hardware present
  if (Ethernet.hardwareStatus() == EthernetNoHardware) {
    Serial.println("Ethernet shield was not found.  Sorry, can't run without hardware. :(");
    while (true) {
      delay(1); // do nothing, no point running without Ethernet hardware
    }
  }
  if (Ethernet.linkStatus() == LinkOFF) {
    Serial.println("Ethernet cable is not connected.");
  }
  // Inicia o servidor Ethernet:
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");
    bool currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        if (c == '\n' && currentLineIsBlank) {
          // send a standard http response header
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");  // the connection will be closed after completion of
the response
          client.println("Refresh: 5");  // refresh the page automatically every 5 sec
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");

          client.print("<title> REDE MESH ZIGBEE </title>");
          // Leitura da informação dentro das "<>"
          recvWithStartEndMarkers();
```

```
        if (newData == true) {
          switch (receivedChars[0]) {
            case 'T':
              T = atof(receivedChars + 2);
              break;
            case 'E':
              E = atof(receivedChars + 2);
              break;
            case 'H':
              H = atoi(receivedChars + 2);
              break;
          }
          newData = false;
        }
        client.print("<center><h2>INTERFACE ETHERNET ZIGBEE MESH
NETWORK</h2></center>");
        // KIT 1
        client.print("<HR WIDTH=100%>");
        client.print("<h3> *** KIT 1 *** </h3>");
        client.print("Temperatura Sensor 1: ");
        client.print(E, 1);
        client.print(" C");
        client.print("<br>");

        //KIT 2
        client.print("<HR WIDTH=100%>");
        client.print("<h3> *** KIT 2 *** </h3>");
        client.print("Temperatura Sensor 2: ");
        client.print(T, 1);
        client.print(" C");
        client.print("<br>");
        client.print("Humidade Sensor 2: ");
        client.print(H);
        client.print(" %RH");
        client.print("<br>");

        //FINALIZAÇÃO
        client.print("<br>");
        client.print("<HR WIDTH=100%>");
        client.print("<center><h2>Gabriel Sartorio Pandolfi – Projeto de
Graduacao</h2></center>");
        client.print("</br>");

        break;
      }
      if (c == '\n') {
        currentLineIsBlank = true;
      }
      else if (c != '\r') {
```

```
        currentLineIsBlank = false;
      }
    }
  }
  delay(1);
  client.stop();
 }
}
void recvWithStartEndMarkers() {
 static boolean recvInProgress = false;
 static byte ndx = 0;
 char startMarker = '[';
 char endMarker = ']';
 char rc;

 while (Serial.available() > 0 && newData == false) {
  rc = Serial.read();

  if (recvInProgress == true) {
   if (rc != endMarker) {
    receivedChars[ndx] = rc;
    ndx++;
    if (ndx >= numChars) {
     ndx = numChars - 1;
    }
   }
   else {
    receivedChars[ndx] = '\0'; // terminate the string
    recvInProgress = false;
    ndx = 0;
    newData = true;
   }
  }

  else if (rc == startMarker) {
   recvInProgress = true;
  }
 }
```

- **SERVIDOR *ETHERNET***

```
#include <SPI.h>
#include <Ethernet.h>
#include <XBee.h>

// Variaveis de configuração do server
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };//MAC ARDUINO
IPAddress ip(192, 168, 1, 45);
EthernetServer server(80);
```

```
// Variaveis gerais
const byte numChars = 10;
char receivedChars[numChars];
boolean newData = false;
int P, H, T, E, S, D, F, W, X, Y, Z;
XBee XBee = XBee();
XBeeResponse response = XBeeResponse();
// create reusable response objects for responses we expect to handle
ZBRxResponse rx = ZBRxResponse();
//Rx16Response rx = Rx16Response();
ModemStatusResponse msr = ModemStatusResponse();

void setup() {
  Serial.begin(9600);
  XBee.begin(Serial);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  Serial.println("Ethernet WebServer Example");
  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);

  // Check for Ethernet hardware present
  if (Ethernet.hardwareStatus() == EthernetNoHardware) {
    Serial.println("Ethernet shield was not found.  Sorry, can't run without hardware. :(");
    while (true) {
      delay(1); // do nothing, no point running without Ethernet hardware
    }
  }
  if (Ethernet.linkStatus() == LinkOFF) {
    Serial.println("Ethernet cable is not connected.");
  }
  // Inicia o servidor Ethernet:
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  XBee.readPacket();
  if (XBee.getResponse().isAvailable()) {
    if (XBee.getResponse().getAPIId() == ZB_RX_RESPONSE) {
      XBee.getResponse().getZBRxResponse(rx);

      switch (rx.getRemoteAddress16()) {
        case 0xD635:    //kitB
          T = rx.getData(0);
          H = rx.getData(1);
```

```
        //Z = rx.getRssi();
        break;
      case 0x2341:    //kitA
        E = rx.getData(0);
        //Y = rx.getRssi();
        break;
      case 0x7AC0:    //sensor ruido
        S = rx.getData(0);
        //X = rx.getRssi();
        break;
      case 0xE9F6:    //dht
        D = rx.getData(0);
        F = rx.getData(1);
        //W = rx.getRssi();
        break;
    }
  }
}

Ethernet Client client = server.available();
if (client) {
  Serial.println("new client");
  bool currentLineIsBlank = true;
  while (client.connected()) {
    if (client.available()) {
      char c = client.read();
      Serial.write(c);
      if (c == '\n' && currentLineIsBlank) {
        // send a standard http response header
        client.println("HTTP/1.1 200 OK");
        client.println("Content-Type: text/html");
        client.println("Connection: close");  // the connection will be closed after completion of the response
        client.println("Refresh: 3");  // refresh the page automatically every 5 sec
        client.println();
        client.println("<!DOCTYPE HTML>");
        client.println("<html>");

        client.print("<title> REDE MESH ZIGBEE </title>");
        client.print("<center><h2>INTERFACE ETHERNET ZIGBEE MESH NETWORK</h2></center>");

        // KIT B
        client.print("<HR WIDTH=100%>");
        client.print("<h3> *** KIT B *** </h3>");
        client.print("Temperatura Sensor 1: ");
        client.print(E);
        client.print(" C");
        client.print("<br>");
```

```
//        client.print("RSSI: ");
//        client.print(Z);
//        client.print("<br>");

        //KIT A
        client.print("<HR WIDTH=100%>");
        client.print("<h3> *** KIT A *** </h3>");
        client.print("Temperatura Sensor 2: ");
        client.print(T);
        client.print(" C");
        client.print("<br>");
        client.print("Humidade Sensor 2: ");
        client.print(H);
        client.print(" %RH");
        client.print("<br>");
//        client.print("RSSI: ");
//        client.print(Y);
//        client.print("<br>");

        // SENSOR RUIDO
        client.print("<HR WIDTH=100%>");
        client.print("<h3> *** SENSOR DE RUIDO SONORO *** </h3>");
        client.print("Nível de ruído: ");
        client.print(S);
        client.print("<br>");
//        client.print("RSSI: ");
//        client.print(X);
//        client.print("<br>");

        // SENSOR DHT
        client.print("<HR WIDTH=100%>");
        client.print("<h3> *** SENSOR DHT *** </h3>");
        client.print("Temperatura DHT: ");
        client.print(D);
        client.print(" C");
        client.print("<br>");
        client.print("Humidade relativa DHT: ");
        client.print(F);
        client.print(" %RH");
        client.print("<br>");
//        client.print("RSSI: ");
//        client.print(W);
//        client.print("<br>");

        //FINALIZAÇÃO
        client.print("<br>");
        client.print("<HR WIDTH=100%>");
        client.print("<center><h2>Gabriel Sartorio Pandolfi - Projeto de
Graduacao</h2></center>");
```

```
        client.print("</br>");

        break;
      }
      if (c == '\n') {
        currentLineIsBlank = true;
      }
      else if (c != '\r') {
        currentLineIsBlank = false;
      }
    }
  }
  delay(1);
  client.stop();
}
delay(500);
}
```