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

Henrique Oliveira Bodart Soares

DESENVOLVIMENTO DE UMA REDE *IOT*

Vitória-ES

12/2017

Henrique Oliveira Bodart Soares

DESENVOLVIMENTO DE UMA REDE *IOT*

Parte manuscrita do Projeto de Graduação do aluno Henrique Oliveira Bodart Soares, apresentada ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Vitória-ES

12/2017

Dedico este trabalho aos meus pais por desde o princípio terem me dado tanto apoio para alcançar o degrau em que hoje me encontro. Agradeço ao meu Orientador de TCC, Orientador de Iniciação Científica, Professor e Tio Marcelo Segatto por ter sido meu guia nestes anos de Universidade e ter me permitido, incentivado e ajudado a chegar onde quer que eu tentasse ir. Dedico este texto também à minha avó por ter me recebido em sua casa nesse período e cuidado de mim com tanto carinho. Dedico também ao meu amigo Matheus por ter me acompanhado em tantas madrugadas, tantos finais de semana e tantas madrugadas de finais de semana dentro do laboratório pesquisando e desenvolvendo. Agradeço ao Bob por tanto ter me ensinado, se não fosse ele teria sido tudo muito mais difícil. Agradeço ao Willian pelos bate papos de sexta feira onde tanto aprendemos sobre o assunto neste texto trabalhado. Dedico à Juliana pela companhia na alegria e na tristeza de cada dia e pelo trabalho desenvolvido que gerou este TCC. E à Marina por toda compreensão e ajuda nestes dias tão difíceis.

AGRADECIMENTOS

A Deus por ter me permitido o privilégio de cursar uma Universidade e ter me dado forças para vir até o final. Aos meus guias e protetores por cuidarem da minha saúde física e mental mesmo nos momentos mais difíceis e terem me ajudado quando achei que não havia solução. À minha família por ter me dado todo suporte emocional, incentivo e carinho para que eu pudesse me concentrar no projeto. Aos meus amigos e namorada por entenderem minha ausência, minha pouca paciência e terem me dado todo apoio que foi necessário sem nunca reclamar nem pedir nada em troca além de abraços. Aos meus colegas de trabalho pelo contínuo esforço em me ajudar a aprender. Aos professores e funcionários da UFES pela maravilhosa companhia nestes longos anos de Universidade.

RESUMO

Na atualidade em uma empresa, indústria, campo, ou até mesmo em áreas públicas com o advento do *IoT* (*Internet of Things* - Internet das coisas), é usual, e até necessário, ter o máximo possível de equipamentos conectados, direta ou indiretamente, à Internet.

Em conformidade com a ideia de *IoT*, foi desenvolvida uma rede de sensores e controladores com o objetivo de ser uma rede de fácil configuração e implementação, onde o usuário tenha apenas que criar a própria tela de monitoramento utilizando o banco de dados já desenvolvido.

Neste projeto de graduação há o desenvolvimento e especificação dos equipamentos de controle e de sensoreamento usados, especificação dos rádios usados e da configuração deles, desenvolvimento de uma estrutura de banco de dados e servidor, além de toda a programação envolvida para funcionamento da rede. Portanto o leitor poderá identificar todas as etapas de projeto e desenvolvimento de uma rede IoT completa para controle e sensoreamento de diferentes tipos de carga e variáveis.

Palavras-chave: *IOT*; *Internet of Things*; Internet das Coisas; Rede de controladores; Banco de Dados; *ZigBee*; *XBee*; *Node* JS; *MySQL*; Rede de sensores; Rede de sensores sem fio.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

A/D	*Analógio/Digital*
D/A	*Digital/Analógico*
DDS	*Serviço de Distribuição de Dados, do inglês Data Distribution Service*
E/S	*Entrada/Saída*
FTDI	*Future Technology Devices International*
HTTP	*Protocolo de Transferência de Hipertexto, do inglês Hypertext Transfer Protocol*
I2C	*Inter-Integrated Circuit*
IoT	*Internet das coisas, do inglês Internet of Things*
M2M	*máquina máquina, do inglês machine-to-machine*
TCP	*Protocolo de controle de transmissão, do inglês Transmission Control Protocol*
TTL	*Lógica Transistor Transistor, do inglês Transistor Transistor Logic*
UDP	*User Datagram Protocol*
UFES	*Universidade Federal do Espírito Santo*
USB	*Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO

Os equipamentos eletrônicos e robóticos vem evoluindo em diversos aspectos. Os mais notáveis são a redução de seu tamanho, ficando cada vez mais portáteis e a diminuição do seu custo, que os tornam mais acessíveis. Além disso, muitos desses têm incorporado interfaces de comunicação, sendo capazes de trocar informações entre si. O aumento da capacidade de processamento, agregada às informações sensoriais compartilhadas, possibilitam que esses dispositivos também tenham certo nível de inteligência. Todas essas características tornaram possível a concepção de ambientes que possam oferecer serviços ao seres humanos baseado em eventos e informações de sensores (HASHIMOTO, 2013).

A Internet permitiu aos desenvolvedores criarem soluções que produzissem dados que pudessem ser vistos por qualquer pessoa em qualquer lugar no mundo. Adaptar protótipos ou pequenas versões de soluções para incorporarem a Internet pode ser um desafio. Não é simples como pegar uma solução já funcionando em uma rede local, ou um mecanismo de comunicação similar, e adicionar conexão com a Internet. Por exemplo, aumentar sua rede de sensores, de alguns poucos sensores tendo seus dados vistos por poucas pessoas para centenas de sensores com dados sendo vistos por todo mundo, pode requerer que haja uma mudança dos seus métodos de comunicação, captação e armazenamento de dados (BELL, 2016).

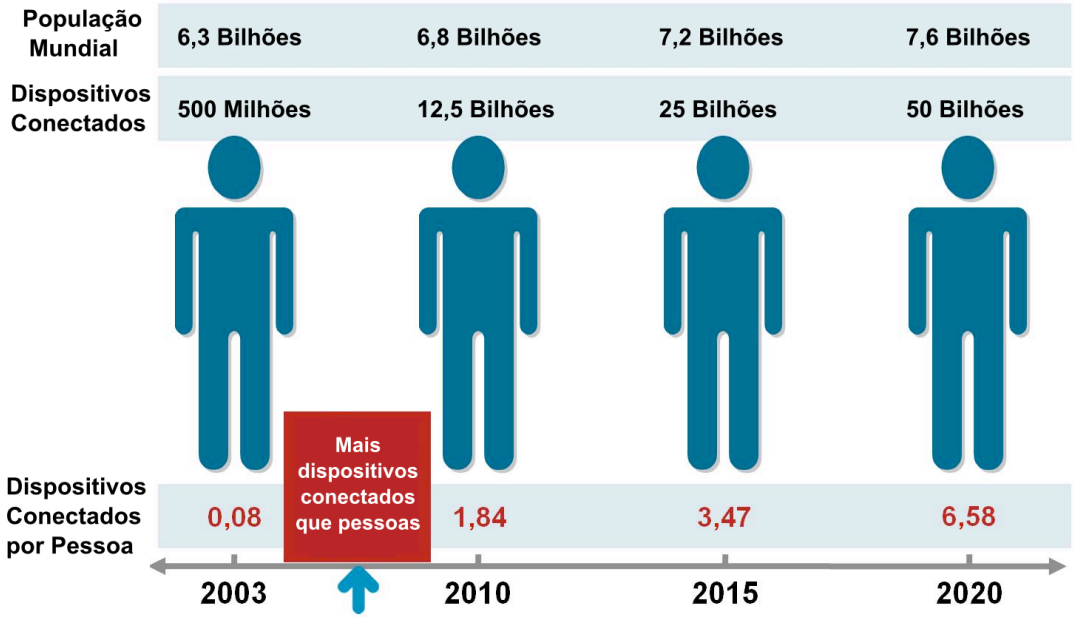
A Internet das coisas (*IoT - Internet Of Things*) é de alguma forma o melhor caminho para um mundo conectado com computação ubíqua e interligada. Ela objetiva fazer diferentes tarefas mais facilmente para os usuários e fornecer outras opções de tarefas, como monitoramento fácil de diferentes fenômenos à nossa volta. Com a computação ubíqua, a computação será embarcada em todos os lugares e programada para agir automaticamente sem ação manual, ou seja, ela será onipresente (CHAOUCHI, 2013). Lembra Queiroz (2016) que a computação ubíqua consiste em dispositivos conectados entre si e inseridos no meio de forma que não sejam percebidos pelo usuário, tornando a interação com eles o mais natural possível. Para que isso seja viável, é necessário o desenvolvimento de uma infraestrutura flexível para o espaço inteligente, no qual seja possível a substituição ou inserção de novos dispositivos sem que sejam necessárias modificações nas aplicações ou na forma de comunicação.

Esse novo paradigma muda a forma na qual as coisas relacionam-se com as pessoas, interferindo diretamente no dia-a-dia e no seu comportamento (ATZORI; IERA; MORABITO, 2010).

Ainda que o conceito de Internet das coisas seja atual, muitas de suas características já eram previstas em trabalhos desenvolvidos sobre espaços inteligentes no início da década de 1990 (WEISER, 1993). Podemos citar, por exemplo, as Redes de Sensores Sem Fio (*WSN - Wireless Sensor Network*) que já existiam antes do termo *IOT* e que hoje na verdade fazem parte do termo.

Vale ressaltar que a quantidade de dispositivos com conectividade e capacidade de processamento é crescente e tem perspectivas gigantescas, o que torna um desafio o gerenciamento, monitoramento e endereçamento desses dispositivos. Com a popularização da Internet, os dados dos dispositivos passaram a trafegar na grande rede, sendo que tais dados são provenientes de diferentes tipos de dispositivos, com tamanhos e finalidades diferentes. Podemos observar que de acordo com Evans (2011), a quantidade de dispositivos ligados à Internet tem crescido bruscamente, conforme mostrado na Figura 1.

Figura 1 – Histórico e perspectiva de crescimento da população mundial e do número de dispositivos conectados à Internet.



Fonte: Retirado de Queiroz (2016), adaptado de Evans (2011).

Portanto, nota-se que há uma grande variedade de dispositivos que geram diferentes tipos informações que estão sendo transmitidas pela Internet, e consequentemente, acarretam num grande volume de dados na rede. Além disso, esses dispositivos possuem capacidades de processamento suficientes para que, a partir de informações adquiridas através de sensores e provenientes de outros dispositivos, sejam capazes de compreender e julgar uma situação a fim de tomar uma decisão, assim como em um espaço inteligente (QUEIROZ, 2016).

1.1 Objetivos gerais

O objetivo geral do trabalho é desenvolver uma proposta de rede de controladores e sensores que serão ligados à Internet. Aqui fica descrito cada componente desta estrutura de Internet das coisas.

O desenvolvimento da rede passa por diversas etapas, definição de um *hardware* simples, de fácil manufatura e que atenda uma proposta básica de sensoreamento e controle de um ambiente qualquer, escolha de um protocolo de comunicação que atenda de pequenos ambientes até grandes distâncias, definição do *hardware* que será usado como concentrador da rede, escolha de um banco de dados apropriado, etc. Dentre todas as definições é necessário se preocupar em atender a critérios básicos como velocidade de comunicação, custo e confiabilidade.

1.2 Objetivos específicos

- Levantamento dos requisitos necessários de *hardware* e *software* para implementação da rede.
- Escolha das linguagens de programação e protocolos para cada etapa do projeto.
- Padronização da estrutura de dados para desenvolvimento de uma rede versátil.
- Criação de um banco de dados que permita inserção de diferentes tipos de equipamentos na rede.
- Estruturação de uma rede que atenda a diferentes protocolos de comunicação.
- Escolha de equipamentos que possam trabalhar com variados tipos de sensores.

1.3 Estrutura do texto

A parte escrita deste Projeto de Graduação está dividida em cinco capítulos, descritos a seguir:

- **Introdução:** Aqui é exibido em âmbito geral a ideia proposta afim de contextualizar o leitor sobre o tema do trabalho.
- **Referencial teórico:** Este capítulo terá uma breve apresentação sobre cada um dos temas abordados no decorrer do trabalho com o objetivo de mostrar ao leitor a importância de cada uma das escolhas técnicas do projeto.

- **Ponto de partida:** Será mostrado de onde surgiu a ideia de seguir com o tema proposto.
- **Ideia futura:** O capítulo mostra qual a pretensão do autor quanto à estrutura ideal que deve futuramente ser implementada.
- **Infraestrutura proposta:** Aqui será mostrado todo o trabalho desenvolvido em termos práticos.
- **Experimentos e resultados:** Após toda a construção ser discutida, será mostrada a rede funcionando.
- **Conclusão:** Este capítulo apresenta um desfecho para tudo que foi gerado como conhecimento tecnológico no decorrer do desenvolvimento do trabalho.

2 REFERENCIAL TEÓRICO

2.1 Internet das coisas

De acordo com Embratel (2017) a transformação digital implementada pelas empresas abre espaço para diversos projetos de inovação e novos negócios, que se tornam possíveis graças à Internet das Coisas. O caminho para que essas estimativas se concretizem passa pela evolução gradual de seis itens sendo eles:

- Objetos conectáveis
- A tecnologia que liga os objetos em rede
- A infraestrutura para a conectividade
- As plataformas de recepção e emissão de sinais
- Os aplicativos para os usuários (operadores e consumidores)
- As soluções de segurança

No ano de 2001 (dois mil e um) Engels et al. (2002) trazia uma ideia de arquitetura que visava integrar o mundo físico com o mundo virtual. Para cada dispositivo ou coisa seria entregue uma identificação única, nomeada de *GUIDe* que tinha como função identificar as informações sobre o objeto em questão, que poderia ser requisitada através de um serviço chamado de *ONS* (do inglês, *Object Name Service* - Serviço de Nome de Objeto). Dessa forma, os objetos passaram a pertencer a um mundo virtual, sendo possível ter aplicações nas quais há a troca de informações entre eles.

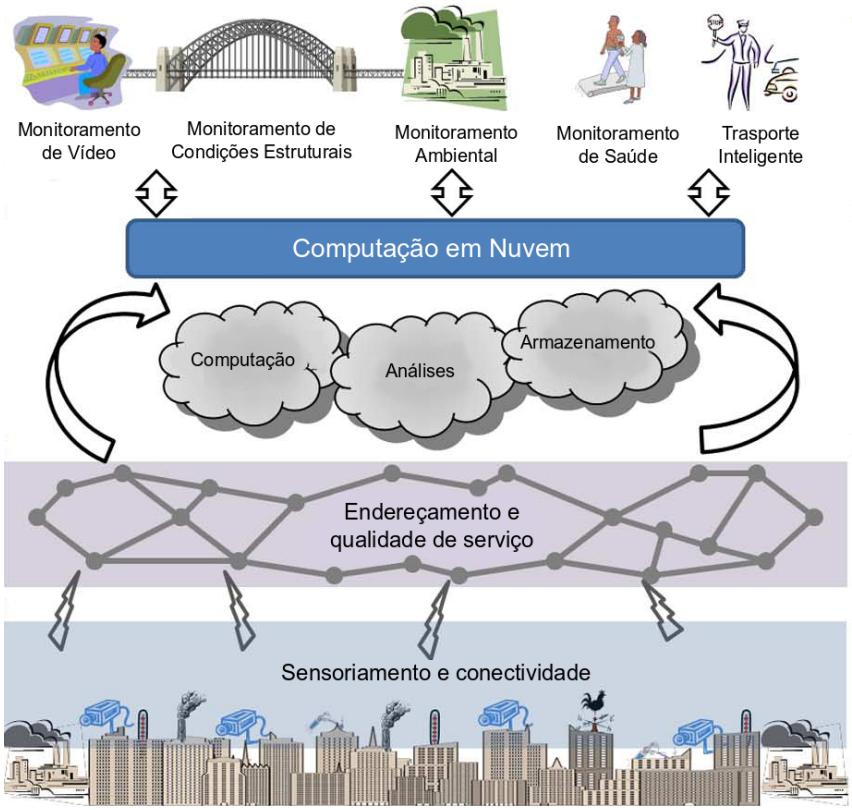
Kiljander et al. (2014) acredita que para pôr em prática a "Computação ubíqua" e a "Internet das coisas" são necessárias abordagens comuns para permitir uma interoperabilidade de alto nível entre diferentes dispositivos.

Outro aspecto relevante é a variedade de tipos de domínios de coisas que apresentam conectividade com a Internet, o que demanda requisitos diferentes para cada domínio. Além disso, interconectar dispositivos com pouca semelhança em uma mesma infraestrutura utilizando uma mesma arquitetura é um grande desafio para o projetista. Dessa forma, diversos trabalhos na área de IoT vem sendo realizados para domínios específicos de coisas (QUEIROZ, 2016).

Trabalhos cujo escopo são cidades inteligentes também utilizam o conceito de *IoT* para modular sua infraestrutura e organização de dados. Por exemplo, em cidades onde é crescente a densidade populacional, cresce também a demanda por serviços e infraestrutura. Contudo, se cidadãos e gestores tiverem acesso à dados da cidade, esses podem ser utilizados para planejamentos futuros e tomadas de decisões (QUEIROZ, 2016).

Jin et al. (2014) propõe, conforme mostrado na Figura 2, um *framework* para a realização de cidades inteligentes utilizando o conceito de *IoT* que engloba um sistema de informações desde a camada de sensoriamento até a de rede, bem como integração com serviço de armazenamento de dados na nuvem.

Figura 2 – Representação de uma arquitetura de Internet das coisas em uma cidade inteligente.



Fonte: Retirado de Queiroz (2016), adaptado de Jin et al. (2014).

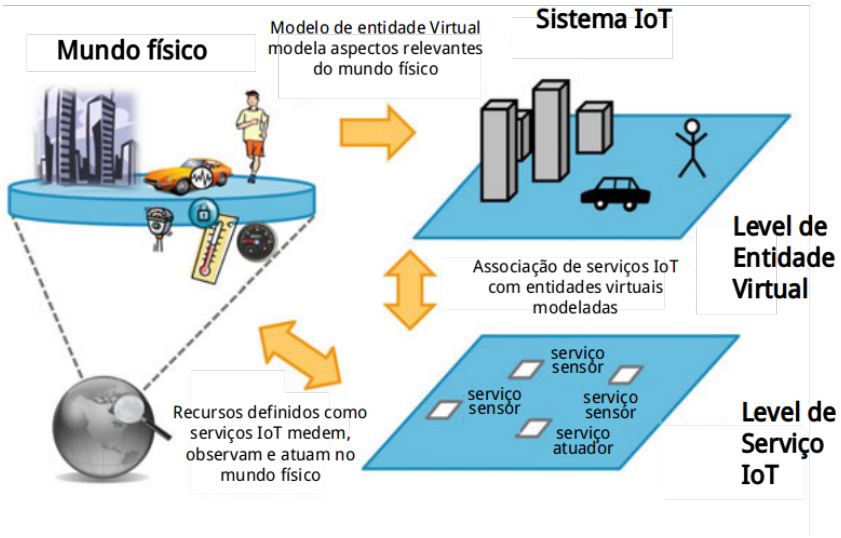
Assim fica claro que desenvolver uma rede apropriada para diferentes tipos de dados sensoreados, com diferentes modelos de sensores, ligados a diferentes microcontroladores, que na rede *LAN* (do inglês, *Local Area Network* - Rede Local) se comunicam a partir de diferentes protocolos de comunicação, é um processo desafiador mas que pode ser facilitado a partir de um projeto feito para a interoperabilidade.

Arquiteturas de referências são desenvolvidas para servirem de ponto de partida para o desenvolvimento de arquiteturas para domínios específicos de uma determinada área.

Talvez o mais conhecido na área de computação seja o modelo *OSI* (do inglês, *Open Systems Interconnection* - Interconexão de sistemas abertos), que serve de referência para o desenvolvimento de protocolos de comunicação para os mais diversos projetos. Modelos de referência facilitam o desenvolvimento de sistemas, reduzindo tempo e custo, além de serem guias para a evolução de sistemas já existentes. Com o crescimento da Internet das coisas, surge a necessidade de se ter uma arquitetura de referência para o desenvolvimento de soluções nesse tema (QUEIROZ, 2016).

Um referencial usado para o projeto é aquele proposto por Bassi et al. (2013), uma simplificação do modelo pode ser encontrada na Figura 3.

Figura 3 – Níveis de abstração "Serviço *IoT*" e "Entidade virtual".



Fonte: Adaptado de Bassi et al. (2013).

Seguindo a arquitetura proposta, é possível observar que o ponto ideal é "expor" os mais diferentes dados captados pela rede em um banco de dados externo. Para que de tal forma as entidades virtuais possam ser criadas uma camada abaixo ou uma camada acima do banco de dados. Assim, os dados sensoreados ou os dados de comando podem ser enviados diretamente entre a entidade virtual solicitante e os serviços *IoT* através de uma rede interna (*LAN*), neste caso fora da Internet, o que é ideal para casos de tempo real e para casos onde há uma tela de monitoramento localizada geograficamente no mesmo local onde há o ambiente físico sensoreado. A outra possibilidade é que a comunicação entre a entidade virtual e os serviços *IoT* aconteçam através do banco de dados utilizando conexão Internet. Transformando assim o mundo virtual em uma "cópia" computacional do mundo físico.

2.2 Comunicação

A rede a ser montada deve atender ao pré-requisito de interoperabilidade, ela deve portanto se capaz de se comunicar em diversos tipos de protocolos, sendo eles cabeados ou não cabeados. No entanto, foi pensado em um protocolo que atenda a maioria dos dispositivos *IoT*, além de tentar lidar com critérios importantes como custo, gasto de energia, distância de comunicação, velocidade, etc.

A camada de comunicação é responsável por prover a conectividade entre os dispositivos do espaço inteligente. Existem diversos protocolos, baseados em diferentes padrões, que são utilizados para a implementação de plataformas *IoT*. Um exemplo de padrão é o *request/response*, utilizado no protocolo *HTTP* (do inglês, *Hypertext Transfer Protocol*). Nesse padrão, existe um servidor que fica esperando uma requisição para então emitir uma resposta. Atualmente, existem microcontroladores com interfaces de comunicação com a Internet, que são capazes de suportar um servidor *HTTP*. Dessa forma, pode-se por exemplo enviar uma requisição para realizar a leitura dos dados de um sensor (QUEIROZ, 2016).

Existem também protocolos conhecidos como *m2m* (máquina maquina, do inglês *machine-to-machine*), que possibilitam a comunicação direta de dois dispositivos através de uma infraestrutura de rede (QUEIROZ, 2016).

Queiroz (2016) lembra ainda que existem também os protocolos baseados em mensagens. Esses protocolos normalmente trabalham associados à um *broker*, que tem a função de receber e encaminhar as mensagens para seus respectivos destinos. Dessa forma, para um dispositivo se comunicar com outro qualquer, basta saber o endereço do *broker* e uma identificação do destinatário da mensagem.

Um outro tipo de funcionamento de protocolo *m2m* é o *DDS* (Data Distribution Service - Serviço de Distribuição de Dados) citado por Peniak e Franekova (2015), onde *Publishers* e *subscribers* se conectam através de um "barramento de dados" e neste espaço de dados declaram sua intenção de se tornar *publisher* ou *subscribers*. Assim o dado pode ser publicado por qualquer um e pode enfim ser lido por quem se interessar pela mensagem.

Para o projeto aqui citado, foi escolhido o modelo de protocolos baseados em mensagens para que fosse possível lidar, de forma mais efetiva, com o fluxo intenso de mensagens da rede que existe em mais de uma direção.

O custo de ligar via cabo uma grande rede de sensores e controladores normalmente torna os projetos inviáveis. Por isso, as soluções cabeadas são usadas geralmente quando a

não cabeada é impossibilitada (segurança, interferência, distância) ou quando já existe uma rede cabeada instalada no local. No entanto, Li (2016) avisa que os dispositivos de interface de rede das linhas existentes tendem a ser volumosos e caros. Além disso, nem todos os dispositivos de sensoreamento e controle são mais eficazes nos terminais das linhas existentes. Portanto, a maioria dos dispositivos *IoT* tendem a se comunicar sem fio, não através de linhas existentes.

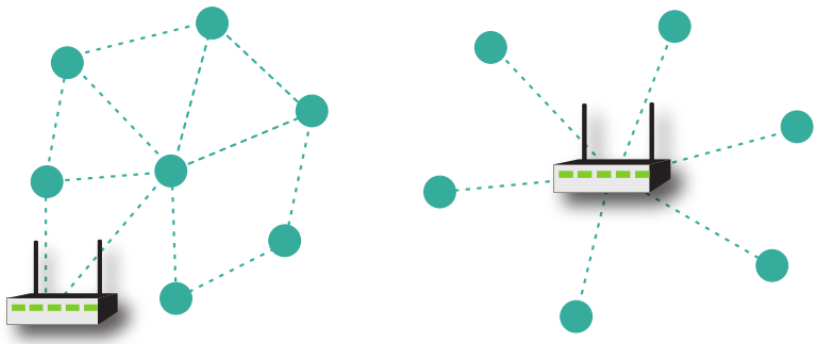
É pensando nisso que foi decidido usar um protocolo de comunicação sem fio. No entanto existe uma infinidade deles (*Wi-Fi, Bluetooth, ZigBee, 6LoWPAN, LoRa, 2G, 3G, 4G, 5G, RFID*, etc), então é preciso verificar alguns outros critérios para escolher um principal.

Como distância alcançada e gasto de energia são variáveis normalmente inversamente proporcionais, Lethaby (2017) propõe como solução exemplo a rede ZigBee por conta dos benefícios de uma topologia malha, mostrada na Figura 4, pois ela pode estender o alcance da rede através de múltiplos saltos, mantendo a baixa potência de transmissão.

Ainda assim devemos lembrar da topologia árvore, usada em redes como *2G*, *3G* e *4G*, em que é possível acessar a Internet mesmo em locais remotos. Tendo como problema o alto custo de instalação dos roteadores (ERB - Estação Rádio Base), sendo um problema para locais onde a rede ainda não está instalada.

Temos também a topologia estrela, mostrada na Figura 4, usada por exemplo no *Wi-Fi* e no *LoRa*, que alcança uma maior distância de operação quanto mais concentradores houverem. O protocolo *LoRa* apresenta ainda a vantagem de que um nó pode se conectar com mais de um concentrador automaticamente, fazendo com que o movimento do dispositivo não o faça perder conexão. Porém a desvantagem das redes em topologia estrela é a necessidade de muitos pontos de conexão com a Internet para manter uma conexão a longas distâncias.

Figura 4 – Topologias Malha (esquerda) e Estrela (direita).



Fonte: Lethaby (2017).

Uma topologia de malha também pode obter uma melhor confiabilidade ao permitir mais de um caminho para transmitir uma mensagem através da rede (LETHABY, 2017).

Assim, foi escolhido como protocolo de rede a ser aplicado neste projeto o *ZigBee* que, além dos benefícios citados anteriormente, funciona em uma faixa de frequências aberta ($2.4GHz$) e o laboratório onde o estudante trabalha já possui diversos rádios que comunicam no protocolo citado, tornando mais rápida a implementação no projeto pois não há a necessidade de compra e teste de novos rádios.

2.3 *Hardware*

Um microcontrolador é um computador em um chip ou, se você preferir, um computador de chip único. A palavra "micro" sugere que o dispositivo é pequeno e o "controlador" diz que o dispositivo pode ser usado para controlar objetos, processos ou eventos. Outro termo para descrever um microcontrolador é o controlador embarcado, porque o microcontrolador e seus circuitos de suporte são geralmente incorporados ou embarcados nos dispositivos que controlam (AXELSON, 1994).

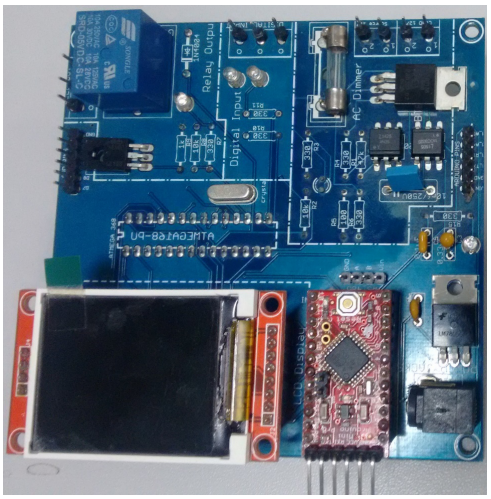
Mesmo no momento em que a *Intel* apresentou o primeiro microprocessador com o *4004*, houve uma demanda de microcontroladores: o *TMS1802* contemporâneo da *Texas Instruments*, projetado para uso em calculadoras, foi publicado no final de 1971 para aplicações em caixas registradoras, relógios e medições instrumentos. O *TMS 1000*, que foi introduzido em 1974, já incluiu *RAM*, *ROM* e *E/S embarcado* e pode ser visto como um dos primeiros microcontroladores, embora seja chamado de microcomputador (GRIDLING; WEISS, 2007).

Apesar do *ATmega328P* ser um microcontrolador 8 bits, bem atrás dos microcontroladores da atual geração (16 e 32 bits), lembra Klubnikin (2017) que o *ATmega328P* facilita aquisição de dados dos sensores e a comunicação serial.

O *ATmega328P* é usado na plataforma *Arduino*, que conforme cita Gerber (2017) é uma plataforma de dispositivos de código aberto, com uma comunidade ativa que está criando placas de desenvolvimento e ferramentas compatíveis. As capacidades dos dispositivos variam em todos os modelos oficiais da *Arduino* e também entre as dezenas de placas compatíveis com terceiros.

Foi utilizado portanto como controlador para a rede uma placa inicialmente desenhada por Flávio Valentim e modificada por Juliana Mattos e Henrique Bodart, conforme mostra a Figura 5.

Figura 5 – Controlador usado.

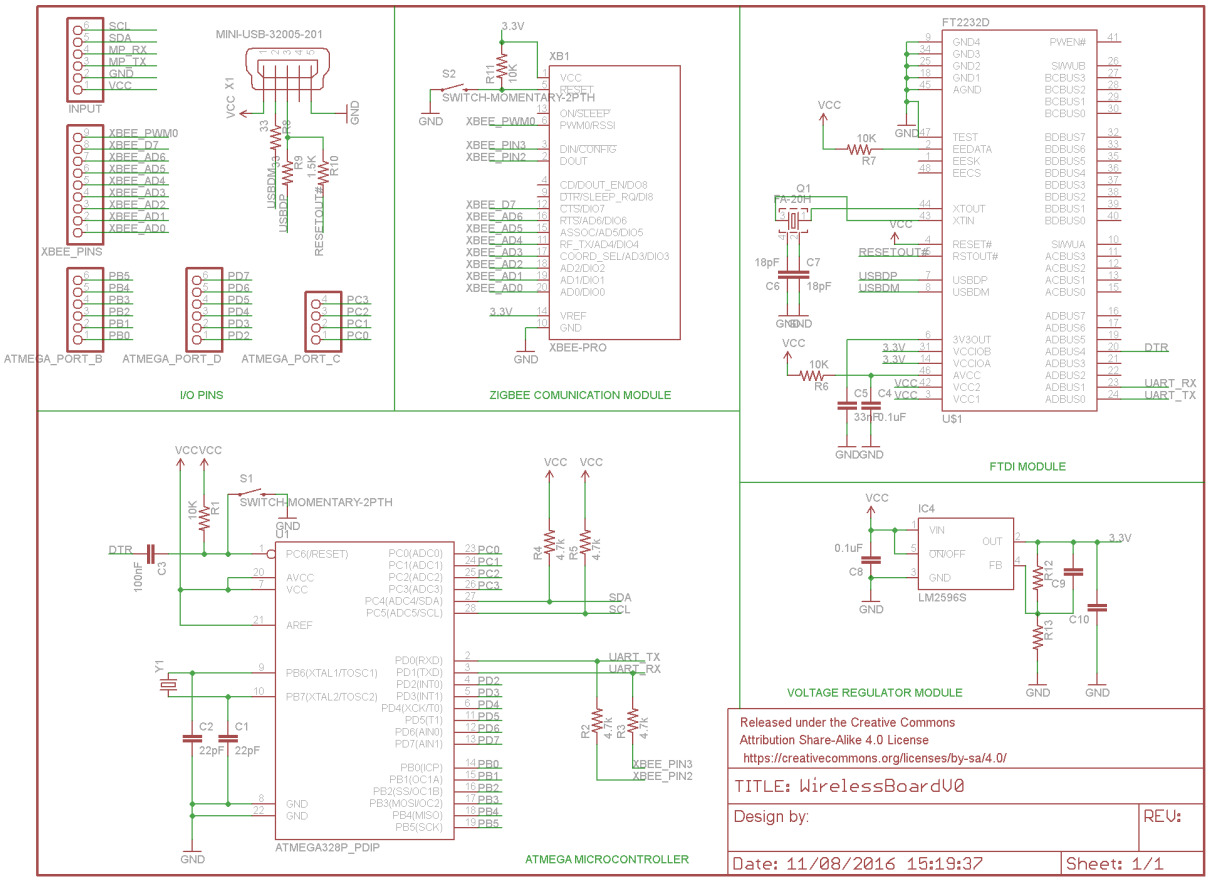


Fonte: Produção do próprio autor.

O controlador possui entradas e saídas digitais, um relé para chaveamento de cargas, um *dimmer* para controle de potência em cargas, entrada para uma pequena tela de monitoramento, entrada para um *Arduino Minipro* e é preparado para se comunicar com outros controladores via protocolos *Rs485* e *I2C*.

O projeto inicial desta placa teve como objetivo ser modular, assim foi desenhada separadamente uma placa para habilitar comunicação sem fio, conforme mostra a Figura 6. Tal placa, no entanto, ainda não foi produzida. Para substituí-la foi escolhida a placa *Seeeduino Stalker* (ver Figura 7) que também tem como microcontrolador o *ATmega328P*, e que já possui toda a estrutura de *Hardware* necessária para se comunicar com um rádio *XBee* (Rádio habilitado para funcionar com o protocolo *ZigBee*), além de ser de baixo custo e baixo consumo de energia.

Figura 6 – Placa desenhada para a comunicação sem fio.



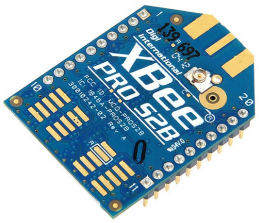
Fonte: Produção do próprio autor.

Figura 7 – Placa usada para a comunicação sem fio (*Seeeduino Stalker*).

Fonte: (SEEEDSTUDIO, 2015)

O dispositivo usado como rádio que se comunica via protocolo *ZigBee* foi *XBee* (ver Figura 8). De acordo com Digi International (2017) os módulos são desenvolvidos para operar com o protocolo *ZigBee* e suprem as necessidades de baixo custo e baixo consumo das redes de sensores sem fio. Os módulos requerem mínima alimentação e proporcionam entrega confiável dos dados entre dispositivos distantes.

Figura 8 – Rádio usado para comunicação entre dispositivos da rede (*XBee*).



Fonte: (DIGI INTERNATIONAL, 2017)

2.3.1 *Software* embarcado

A abordagem padrão para desenvolver softwares para executar em microcontroladores compatíveis com *Arduino* é usar *C* ou *C++* e a *Arduino IDE* (GERBER, 2017).

C++, Criada por Bjarne Stroustrup em 1983, inicialmente conhecida como "*C* with classes" (C com classes), é uma linguagem para uso genérico. Como *C*, *C++* é uma linguagem compilada, oferecendo as mesmas vantagens de acesso à hardware, flexibilidade e performance. No entanto, ela traz consigo conceitos nativos de orientação a objeto, programação genérica e metaprogramação, o que, a nível de codificação, proporciona um sistema de mais fácil manutenção, reuso e ampliação (CURVELLO et al., 2015). Assim, a linguagem escolhida foi C*++*.

2.4 Concentrador

Para conectar uma placa qualquer à Internet é necessária uma pilha *TCP/IP* (ou *UDP/IP* caso não haja preocupação com a garantia da chegada do dado ao destino final). De acordo com SunSoft (1994), o *TCP/IP* é reconhecido como padrão pelas principais organizações de padrões internacionais e é usado no mundo todo. Como é um conjunto de padrões, é executado em vários tipos diferentes de computadores.

Existem portanto três diferentes opções para o uso da pilha. Segue:

- **Implementá-la:** O que demanda um tempo maior que as duas outras opções, além dos inúmeros erros que podem ser causados por adotar tal opção.
- **Usar uma biblioteca pronta:** Muitos desenvolvedores se juntaram e montaram bibliotecas que implementam a pilha em diversas linguagens, inclusive em *C*. No

entanto problemas de confiabilidade e segurança são muitos, já que são bibliotecas abertas e com poucas possibilidades de implementação de segurança.

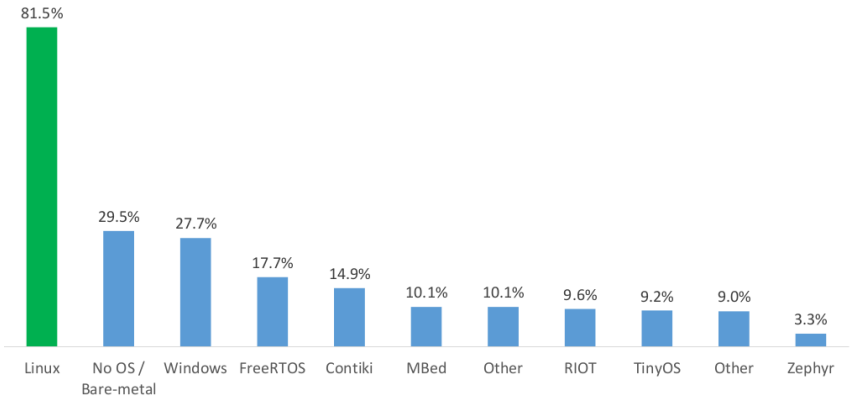
- **Usar um microcontrolador com sistema operacional embarcado:** Esta opção é a que possui maior custo financeiro, no entanto acelera o tempo de entrada do produto no mercado e trás com ela as facilidades que um computador trás sem que seja perdido o acesso ao *hardware* que os microcontroladores comuns têm.

Assim, foi decidido, que o concentrador a ser usado futuramente terá um microcontrolador com sistema operacional embarcado. Para o presente projeto, foi usado um computador para facilitar a implementação, o que facilita também no futuro a troca para o concentrador desejado. Nele foi adicionado um rádio *Xbee* conforme mostra a Figura 8.

2.4.1 Software

Conforme citado, o concentrador usado para este projeto será um computador. Portanto a gama de possibilidades de linguagens de programação é muito maior. A opção pelo uso do sistema operacional *Linux* é por uma questão de preço (já que Linux é o único sistema operacional gratuito entre os mais usados) e vasta disseminação, pois de acordo Group et al. (2017) a maioria dos sistemas *IoT* usam *Linux* (ver Figura 9).

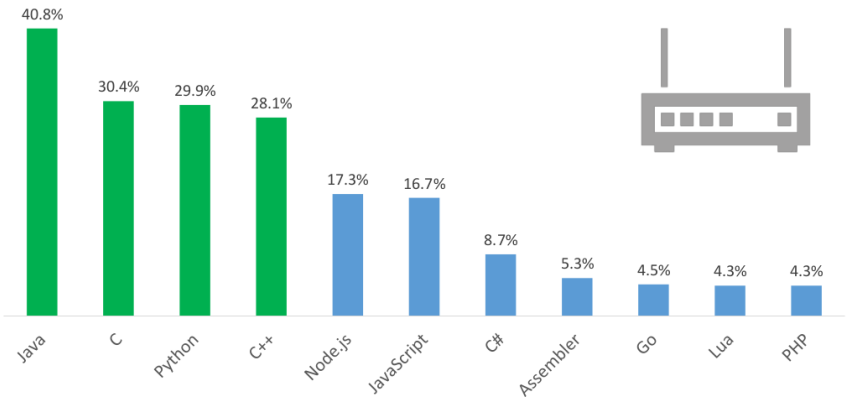
Figura 9 – Sistemas Operacionais mais usados em dispositivos *IoT*.



Fonte: (GROUP et al., 2017)

Podemos ver na Figura 10 as linguagens de programação mais usadas para os concentradores de redes *IoT*.

Figura 10 – Linguagens de programação mais usadas em concentradores de redes *IoT*.



Fonte: (GROUP et al., 2017)

É possível observar que *Node JS* é a quinta mais utilizada, com 17.3% e . No entanto *Node JS* não é uma linguagem de exatamente programação, e sim um sistema que usa como linguagem o JavaScript (que é a sexta linguagem mais usada, com 16% dos usuários).

Node JS é um sistema de tempo de execução que facilita a criação de uma rede ou outros servidores de aplicativos orientados a eventos (WEN, 2013). I2C.INFO (2017) diz que *Node JS* é um *JavaScript* que roda em tempo de execução, é conduzido por eventos assíncronos e que foi projetado para criar aplicativos de rede escaláveis.

Todos os métodos de E/S na biblioteca padrão *Node JS* fornecem versões assíncronas, que são não bloqueadoras e aceitam funções de retorno de chamada (LINUX, 2017a). Assim, podemos acessar o banco de dados sem que o resto do código *JavaScript* pare e fique esperando.

Ainda assim o conjunto *Node JS* e *Javascript* somam uma porcentagem menor do que o *JAVA*, no entanto as páginas *WEB* são escritas, na maioria dos casos, em *Javascript* o que torna o código mais otimizado se o *front-end* e o *back-end* forem programados na mesma linguagem, permitindo o uso das mesmas funções nos dois setores. Pensando nessas vantagens, *Node JS* foi escolhido para fazer o concentrador funcionar.

2.5 Banco de dados

Para a escolha do banco de dados, foram levadas em consideração alguns itens conforme segue:

- Preço.

- Velocidade.
- Possibilidade de funcionamento em um microcontrolador.
- Funcionamento em um sistema operacional *Linux*.

Tahaghoghi e Williams (2007) dizem que o *MySQL* pode estar em um equipamento bem modesto e põe pouquíssimo atraso no sistema. Muitos pequenos usuários alimentam suas organizações com informações através do *MySQL* em modestas áreas de trabalho. A velocidade com que ele pode recuperar informações fez dele um favorito de longa data dos administradores *WEB*.

Ainda de acordo com Welling e Thomson (2016) uma das melhores características do *MySQL* é que ele funciona com qualquer um dos maiores sistemas operacionais e com vários dos menores.

Assim, mesmo havendo outros bancos de dados que funcionam conforme as especificações necessárias para o projeto, o escolhido foi o *MySQL*.

3 PONTO DE PARTIDA

Conforme citado na sessão 1.3 aqui será mostrado de onde surgiu a ideia de seguir com o tema proposto.

Este projeto de graduação começou a partir de um projeto de iniciação científica desenvolvido no ano de 2016 pelos alunos Henrique Bodart e Juliana Mattos onde havia sido montada uma rede de controladores que será mostrada nas sessões a seguir.

3.1 Estrutura de Hardware

Ao final do projeto, haviam sido utilizadas três placa de controle conforme mostra a Figura 5, cada uma delas ligadas a uma placa usada apenas para comunicação (ver Figura 7) e em cada uma destas há um rádio conforme mostra a Figura 8. Nos controladores não foram incluídas as telas de monitoramento que ficavam na placa. A comunicação interna da placa era muito lenta quando dotada de tal equipamento, o que prejudicava o controle das cargas inserindo atrasos.

Um dos três controladores se comunicava com um computador, para tal ele possuía ainda uma placa *Arduino MiniPro*, que servia para tirar do microcontrolador principal (*ATmega328P*) a carga de comunicação com o computador.

O *Arduino MiniPro* não pode se comunicar diretamente com o computador, pois ele não é preparado para uma comunicação *USB*, por isto foi usado um conversor *FTDI* (Future Technology Devices International - Conversor TTL/USB) para implementar a comunicação.

3.2 Estrutura de software

Nos microcontroladores *ATmega328P*, dos controladores (Figura 5), todo o código foi implementado em *C++*, foi inclusive gerada uma biblioteca própria para os controladores.

Em cada uma das placas de comunicação sem fio (Figura 7) o código foi também desenvolvido em *C++* para facilitar o acesso às placas *XBee* (Figura 8).

O *Arduino MiniPro*, usado para comunicar um dos controladores ao computador, também foi programado em *C++*.

Por fim, no computador, a programação usada foi *MATLAB*, de onde foi feita a interface usuário. Nele o usuário era capaz de enviar comandos a qualquer um dos controladores da rede.

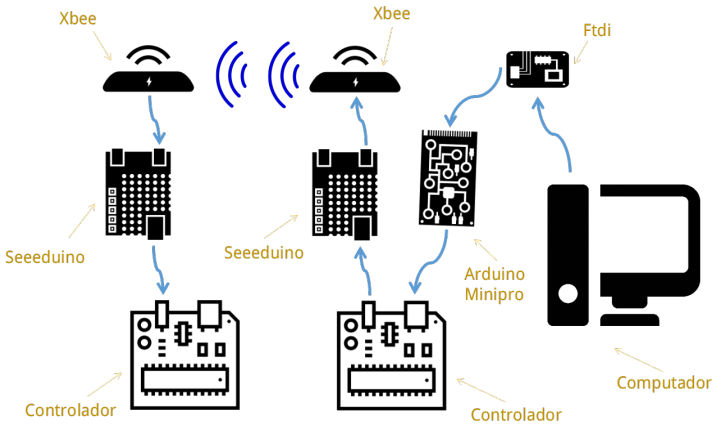
3.3 Estrutura de telecomunicações

Aqui será mostrada desde a estrutura de comunicação da rede toda até a estrutura de comunicação entre equipamentos.

3.3.1 Rede de controladores

Nesta rede, o conjunto Controlador (Figura 5), placa de comunicação sem fio (Figura 7) e rádio (Figura 8) formavam um **roteador**. Enquanto o conjunto Controlador, placa de comunicação sem fio, rádio, placa *Arduino MiniPro*, *FTDI* e computador formavam o conjunto **concentrador**. A estrutura da rede fica mostrada na Figura 11.

Figura 11 – Rede anteriormente desenvolvida.



Fonte: Produção do próprio autor.

3.3.2 Comunicação entre dispositivos

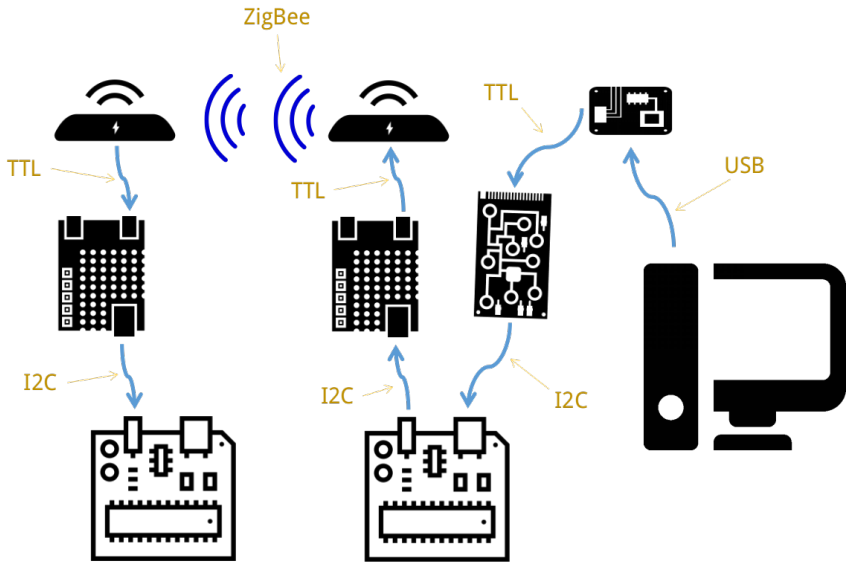
Do lado do concentrador, a comunicação acontece entre o computador e o *FTDI* via *USB* (Universal Serial Bus), entre *FTDI* e *Arduino MiniPRO* via protocolo *TTL* (Transistor Transistor Logic). O *Arduino MiniPRO*, por sua vez, fazia a interação com o controlador (Figura 5) utilizando o protocolo de comunicação *I2C* (Inter-Integrated Circuit) que de acordo com (LINUX, 2017b) é um protocolo serial de dois fios usado para conectar dispositivos de baixa velocidade como alguns tipos de microcontroladores, conversores A/D e D/A, interfaces de E/S entre outros periféricos semelhantes em sistemas embarcados. O controlador se comunica com a placa de comunicação sem fio (Figura 7) utilizando também o protocolo *I2C*. A placa de comunicação sem fio se comunica com o rádio *ZigBee* (Figura 8) utilizando mais uma vez o protocolo *TTL*.

Entre concentrador e roteador, os protocolos usados para comunicação eram *ZigBee* (comunicação sem fio) ou *RS485* (Comunicação com fio). Quando a comunicação era sem fio os dados passavam pela placa de comunicação sem fio, quando era cabeada os dados saíam diretamente do controlador.

Do lado dos roteadores, os dados vão do rádio *ZigBee*, para a placa de comunicação sem fio utilizando a comunicação *TTL* e da placa de comunicação sem fio para o controlador utilizando o canal *I2C*.

Na Figura 12 todo os protocolos de comunicação são mostrados onde foram utilizados.

Figura 12 – Uso dos protocolos de comunicação na rede anteriormente desenvolvida.



Fonte: Produção do próprio autor.

3.3.3 Pacote de dados

Para entender como a rede funciona, é preciso entender como são formados os pacotes de dados que por ela passam. Portanto, nesta sessão será mostrado todo o processo de formação de pacotes e como eles são compreendidos.

No *MATLAB* são criados cinco bytes, onde os três primeiros significam sempre a mesma coisa e os dois últimos podem mudar conforme variam os três primeiros. Segue no quadro 1 um exemplo da formação dos três primeiros bytes.

Quadro 1 – Significado dos três primeiros bytes que circulavam na rede anteriormente desenvolvida.

Posição	**Significado**	**Exemplo**
Byte 1	Tipo de comunicação	*RS485*, *ZigBee*
Byte 2	Qual o endereço de destino	Controlador 1 ... n
Byte 3	Qual a funcionalidade do controlador	Relé, Dimer, etc

Fonte: Produção do próprio autor.

Os próximos dois bytes dependem da função escolhida (terceiro byte), e então o pacote com os cinco bytes é enviado para o *Arduino Minipro*.

Nas partes da rede onde é usado o protocolo *I2C*, conforme mostra na Figura 12, o "mestre" da comunicação *I2C*, de tempos em tempos, pede os dados para os "escravos", assim um dado antigo que esteja em um dos escravos acaba sendo reenviado toda vez que o mestre o pede. Para solucionar tal problema, foi acrescentado um byte no início pacote, incluído no *Arduino Minipro*. Sempre que um dado é novo (que acabou de ser enviado pelo computador) este byte recebe o valor um (1), quando tal dado é requisitado pelo mestre da comunicação *I2C* (o controlador, ver Figura 5), o byte é novamente convertido para (0). Desta maneira o controlador saberá quando o dado é novo ou antigo.

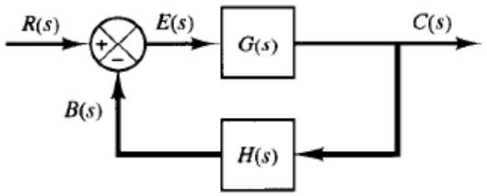
É o pacote com seis bytes que circulará por toda a rede. Tal pacote atravessa a rede sem que seja necessário implementar na programação a real formação do pacote para cada tipo de linguagem, pois foram usadas bibliotecas prontas que implementam os protocolos. Assim, tais bibliotecas, enviam os dados dentro das regras de cada protocolo.

3.4 Observações sobre a rede antiga

Apesar do objetivo da rede ter sido uma rede de baixo custo, para tal foram desenvolvidas e usadas placas de baixo custo, o concentrador da rede havia sido mal projetado.

É possível observar, conforme consta na sessão 3.3.1, que o concentrador teve um objetivo duplo de controlar a rede e controlar cargas. Para tal foi preciso adicionar nele uma capacidade computacional enorme (4 microcontroladores além de um computador), pois em um sistema comum de controle, conforme mostra a Figura 13, existe uma realimentação de dados. Por exemplo a temperatura do ambiente seria usada para determinar a potência que o dimer aplicaria na carga.

Figura 13 – Sistema realimentado.



Fonte: (OGATA, 2001)

No entanto, a partir de uma perspectiva de controle de dados amostrados, é natural amostrar a saída do processo equidistantemente com um período de amostra "h". Também é natural manter o atraso de controle o mais curto possível. A razão é que os atrasos no tempo dão origem a um atraso de fase, que geralmente degeneram a estabilidade e o desempenho do sistema (NILSSON, 1998).

Por isso foi feita a opção de distribuir as tarefas do concentrador em diferentes microcontroladores, fato tal que teria sido evitado se o concentrador tivesse como função única controlar a rede.

Além disso, a ideia de ter uma rede de baixo custo fica comprometida com o uso do *software MATLAB*, uma vez que o mesmo é proprietário e tem um custo de assinatura muito maior que o custo de todo o restante da rede. Além de ser um programa que exige uma capacidade computacional maior que aquela dos microcontroladores com sistema operacional embarcado, o que impediria uma futura substituição do computador por um *hardware* menor, que consuma menos energia e menos dispendioso.

4 IDEIA FUTURA

Neste projeto se fez necessário apresentar o que havia antes dele ser iniciado e onde se quer chegar com ele para que por fim seja possível compreender os resultados atuais.

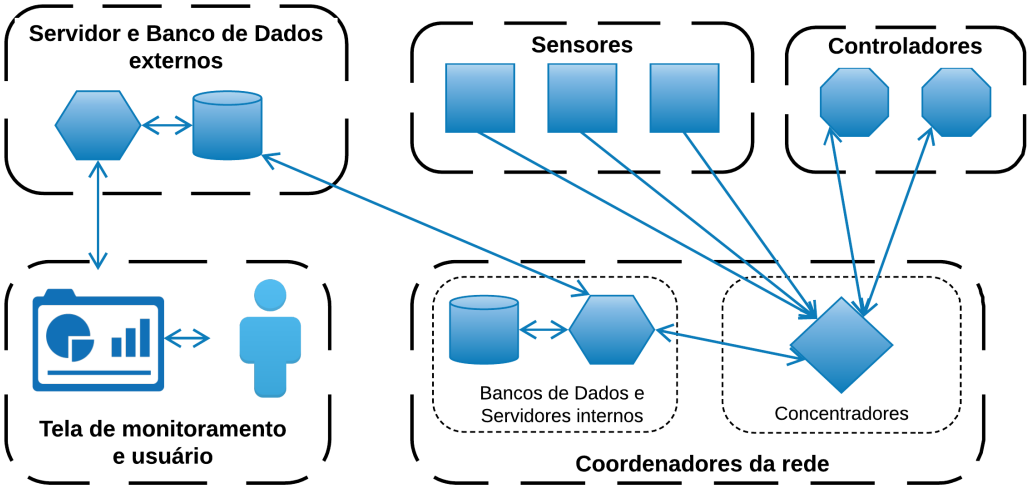
No presente capítulo é apresentada a ideia futura, sendo consideradas as possíveis aplicações.

4.1 Ideia conceitual

Para que seja possível aplicar a arquitetura mostrada na Figura 3, é preciso antes desenvolver toda uma estrutura de interligação entre o mundo físico e o mundo virtual, equivalente a ligar a rede à Internet.

A Figura 14 mostra em forma de diagrama a ligação citada anteriormente.

Figura 14 – Diagrama conceitual da ideia futura da rede.



Fonte: Produção do próprio autor.

É possível observar na Figura 14 que supõe-se que haverá mais de um concentrador na rede, pois em uma rede IoT real podem haver inúmeros dispositivos captando dados. Para não afunilar os dados em um único concentrador distribuem-se mais alguns em locais físicos que tenham maior concentração de dispositivos conectados.

Haverá então mais de um acesso ao banco de dados principal (externo), além deste banco também ser usado para levar informação à tela de monitoramento do usuário. Para que não

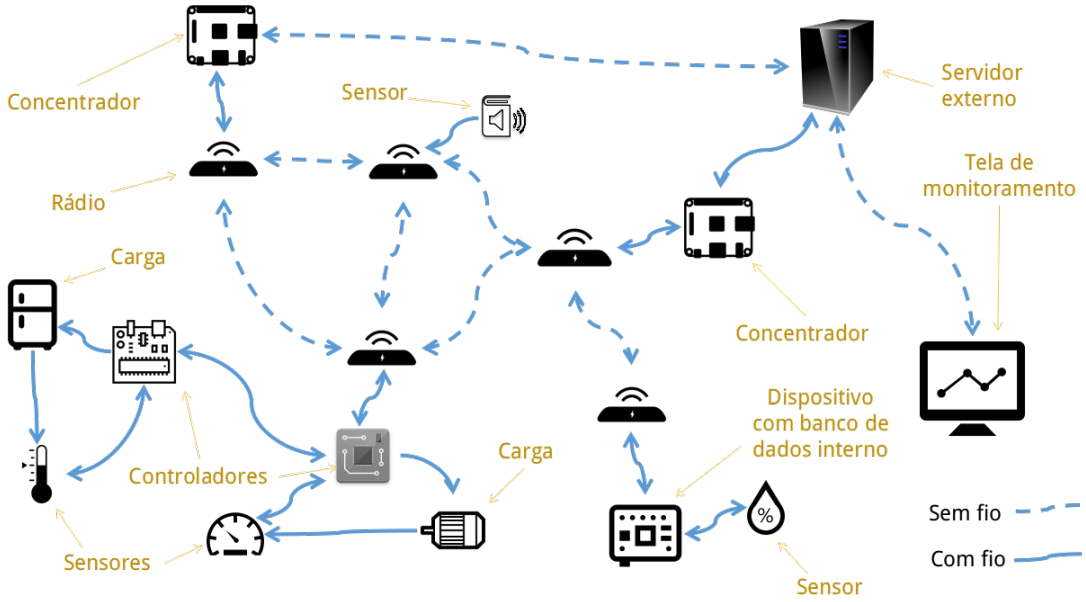
hajam muitos dados perdidos por excesso de acessos ao banco de dados externo, sugere-se que haja um banco de dados por concentrador, assim até mesmo uma possível perda de conexão com a Internet seria algo contornável, já que os dados salvos no banco de dados interno poderiam ser reenviados à rede para que alcançassem outro concentrador.

Será feita portanto uma coordenação entre diversos bancos de dados. Para não sobrecarregar o servidor externo que já é responsável pela tela de monitoramento, a sugestão é que os servidores internos, contidos em cada concentrador, cuidem de acessar o banco de dados externo para salvar os dados gerados pela rede e para captar os dados destinados à rede.

4.2 Exemplo de funcionamento

Para que fique mais clara a ideia, nesta sessão será mostrado um exemplo de funcionamento prático da rede tendo em mente o diagrama conceitual da Figura 14. Segue na Figura 15 o exemplo sugerido.

Figura 15 – Exemplo prático da ideia futura da rede.



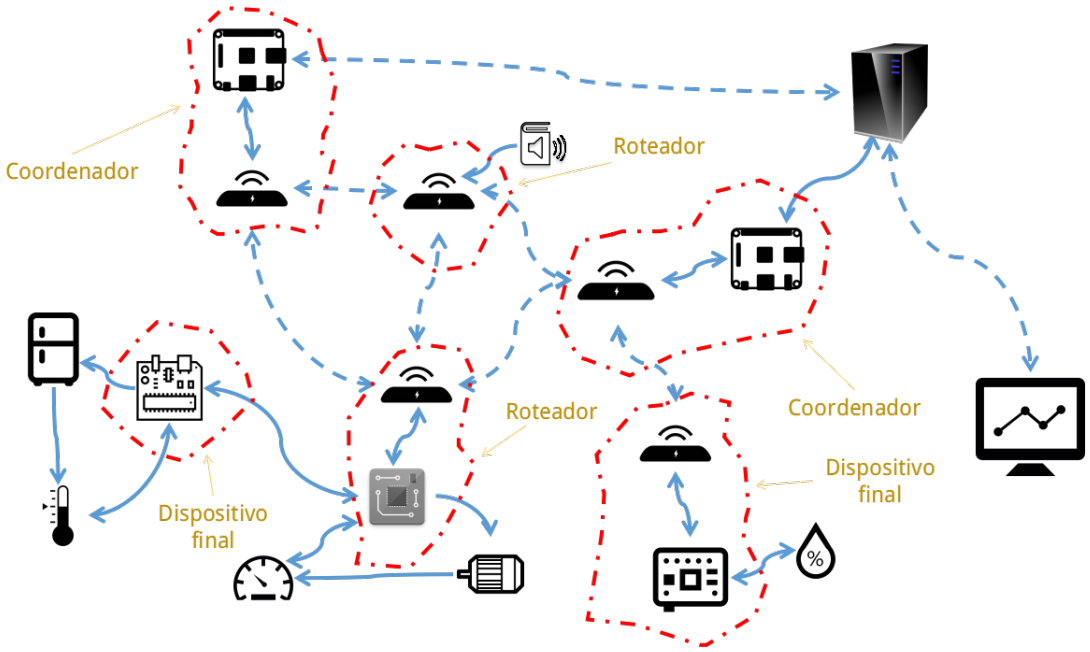
Fonte: Produção do próprio autor.

É possível observar que o objetivo é conectar diversos tipos diferentes de controladores e sensores em uma mesma rede. Para tal serão utilizados diferentes protocolos de comunicação sem fio e com fio. O objetivo é que a rede tenha mais de um concentrador, para permitir um maior dinamismo, velocidade e invulnerabilidade a falhas, pois com apenas um concentrador a queda dele desconectaria toda a rede.

A Figura 16 mostra do ponto de vista de redes qual a função de cada conjunto de

equipamentos.

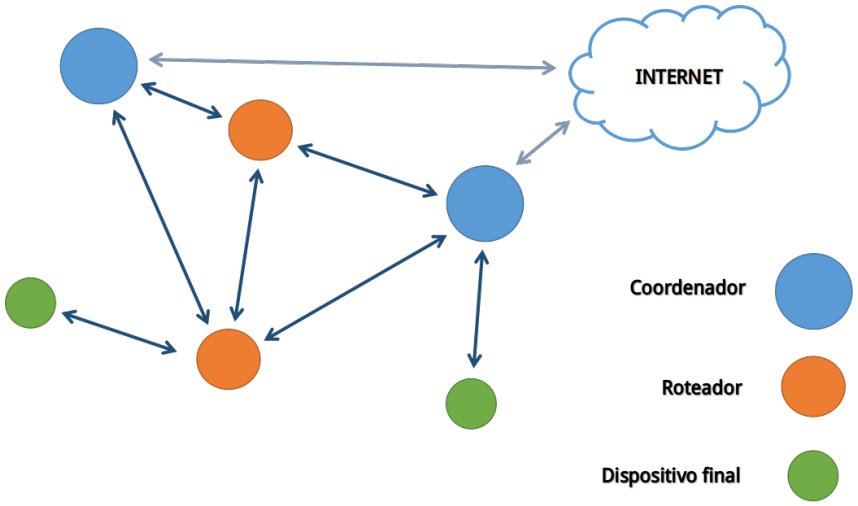
Figura 16 – Exemplo prático da ideia futura da rede do ponto de vista de comunicação.



Fonte: Produção do próprio autor.

Para facilitar a visualização da rede mostrada na Figura 16, é disponibilizada na Figura 17 a visualização da mesma em forma de grafo, onde as áreas contornadas em vermelho representam os nós e as ligações entre elas (comunicação com fio ou sem fio) representam as arestas.

Figura 17 – Grafo da rede citada na Figura 16.



Fonte: Produção do próprio autor.

Nota-se que havendo uma falha na comunicação (falha de arestas) apenas os dispositivos finais possivelmente se desconectariam da rede. A rede é resistente a até duas falhas de

equipamentos (falha de nós). Redes em malha são mais tolerantes a falhas e com tal tipo de diagrama podemos ter apenas alguns dispositivos conectados à rede externa (Internet), podendo assim concentrar os esforços de segurança a invasões e ataques apenas nestes pontos. Outro fator a ser considerado é que os dispositivos capazes de se conectar à Internet são geralmente mais caros, assim é preciso fazer um equilíbrio quanto ao preço da rede e sua confiabilidade.

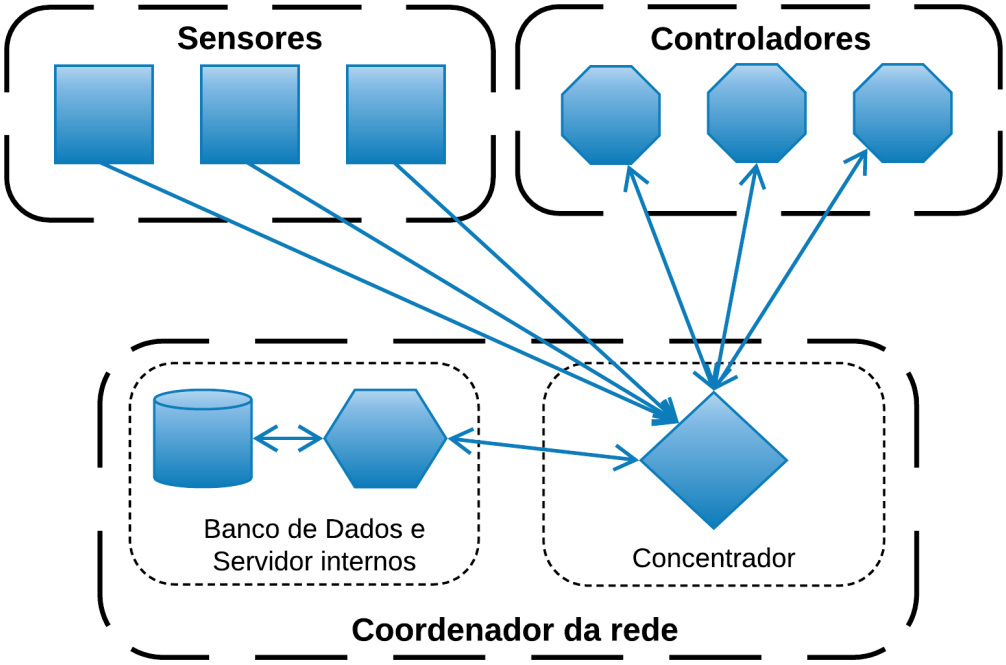
5 INFRAESTRUTURA DESENVOLVIDA

Aqui será apresentado o projeto prático desenvolvido e os conceitos ligados a ele.

5.1 Ideia conceitual

Na Figura 18 podemos ver conceitualmente como funciona a rede desenvolvida.

Figura 18 – Diagrama conceitual da rede desenvolvida.



Fonte: Produção do próprio autor.

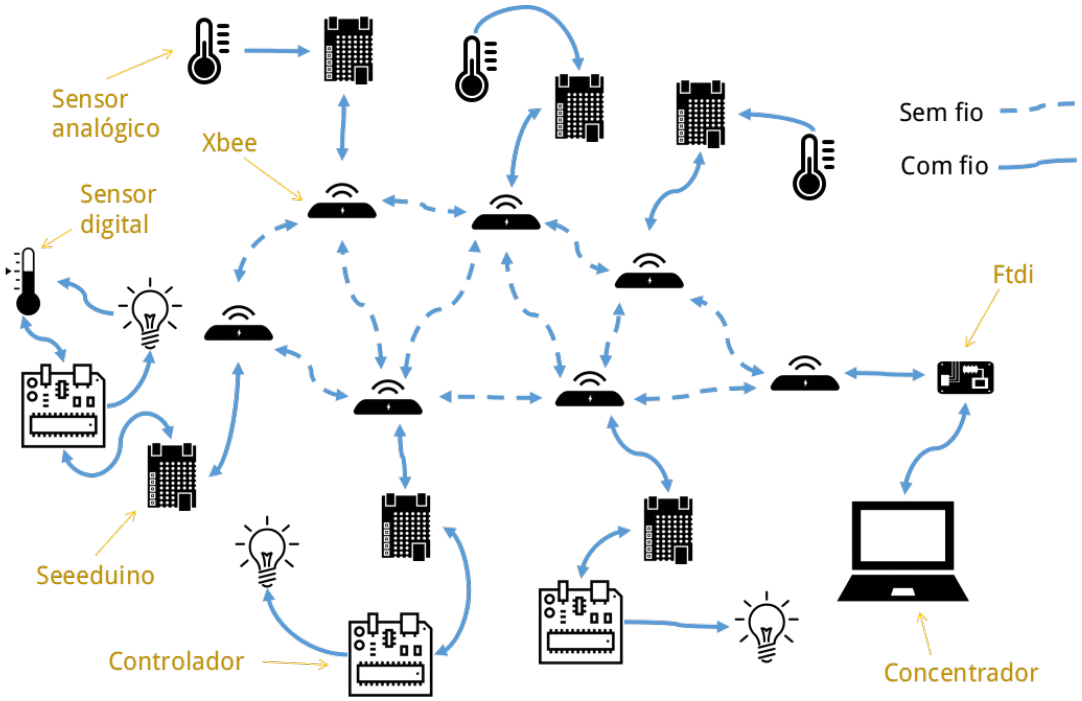
É possível perceber que comparativamente com o diagrama conceitual apresentado na Figura 14 da sessão 4.1, o que foi retirado é tudo aquilo que é externo à rede.

Esta escolha foi feita para ser fiel à ideia de desenvolver algo genérico, de modo a tornar fácil a implantação de uma rede mais específica. Conforme mostrado na Figura 3 e explicado na sessão 2.1 para cada rede faz-se um grupo de "entidades virtuais" que são senão uma cópia do mundo físico e, portanto, dependem de onde a rede será aplicada e de suas características particulares.

5.2 Funcionamento

A Figura 19 mostra a rede desenvolvida.

Figura 19 – Rede desenvolvida.



Fonte: Produção do próprio autor.

Observa-se que existe uma malha formada entre os rádios *Xbee*, no entanto esta malha, no protocolo *Zigbee*, varia dinamicamente conforme a necessidade da rede de se adaptar a possíveis falhas, a menos que o usuário a torne estática.

5.3 Controlador

Observa-se na Figura 19 que existe apenas um modelo de controlador sendo usado, que é aquele mostrado na Figura 5 e explicado na sessão 2.3.

Existem dois modelos de pacotes de dados que podem correr pelo controlador. Um que é recebido, um que é enviado.

5.3.1 Pacote de dados recebido pelo controlador

O pacote de dados recebido pelo controlador tem quatro bytes (os quatro primeiros) de significado fixo, os outros dois bytes dependem do quarto.

O Quadro 2 mostra os possíveis significados dos quatro primeiros bytes.

Quadro 2 – Significado dos quatro primeiros bytes da rede desenvolvida.

Posição	**Significado**	**Exemplo**
Byte 0	Informação nova?	Sim ou não
Byte 1	Tipo de comunicação	*RS485*, *ZigBee*, etc.
Byte 2	Qual o endereço de destino	Controlador 1 ... n
Byte 3	Qual a funcionalidade do controlador	Relé, Dimer, etc.

Será listada algumas considerações sobre alguns dos bytes citados no Quadro 2.

- **Byte 0:** Define se o pacote recebido será lido ou não, funciona para evitar que pacotes que tenham ficado salvos em algum *buffer* sejam relidos e afetem negativamente o funcionamento do controlador.
- **Byte 1:** Para esta rede em específico está sendo usada apenas a comunicação via protocolo *ZigBee*, portanto o único valor possível é 2.
- **Byte 2:** Como os rádios tem endereço próprio e cada rádio se encontra ligado a apenas um controlador, este campo ficou inutilizado, o valor padrão adotado para ele é 2.

A partir daqui serão mostrados os possíveis pacotes de dados dependendo do valor contido no **Byte 3**.

5.3.1.1 Relé

Quando o valor do **Byte 3** for um (1), significa que será acessado o relé do controlador. Para tal seguem os significados dos últimos dois bytes.

- **Byte 4:** Temperatura para abertura do relé.
- **Byte 5:** Temperatura para fechamento do relé.

Assim um controle qualquer de carga pode ser feito com relação a uma determinada faixa. Segue na Figura 20 um exemplo de pacote que poderia ser recebido pelo controlador.

No caso citado na Figura 20, usando uma lâmpada incandescente como aquecedor, toda vez que a temperatura chegasse a vinte e sete graus Celsius (27^oC), o relé seria ativado e a lâmpada acenderia, aquecendo o ambiente. Quando a temperatura chegasse a trinta graus Celsius (30^oC), o relé desligaria e a temperatura no ambiente voltaria a diminuir naturalmente.

Figura 20 – Exemplo de pacote para controle de carga via relé.

Fonte: Produção do próprio autor.

5.3.1.2 Dimer

Quando o valor do **Byte 3** for dois (2), significa que será acessado o dimer do controlador. Segue a estrutura dos dois últimos bytes.

- **Byte 4:** 1 para um processo da incrementação no valor do sensor e 2 para decremento. (esquentar ou esfriar).
- **Byte 5:** Valor para controle.

Como nesta rede estamos usando apenas o processo de aquecimento através das lâmpadas incandescentes, o **Byte 4** será sempre um (1).

Este tipo de controle faz um corte na onda de alimentação da carga objetivando controlar a potência inserida. No entanto não é um processo indicado para controle de cargas de muita potência pois o corte na onda gera harmônicos que podem danificar o equipamento e diminuir sua vida útil.

Como o controlador desenvolvido é apenas proporcional, há um erro estático no controle.

Segue na Figura 21 um exemplo de pacote que atuaria sobre o dimer.

Figura 21 – Exemplo de pacote para controle de carga via dimer.

Fonte: Produção do próprio autor.

Neste caso a lâmpada teria sua potência variada pelo dimer objetivando manter a temperatura ambiente em trinta graus Celsius (30^oC).

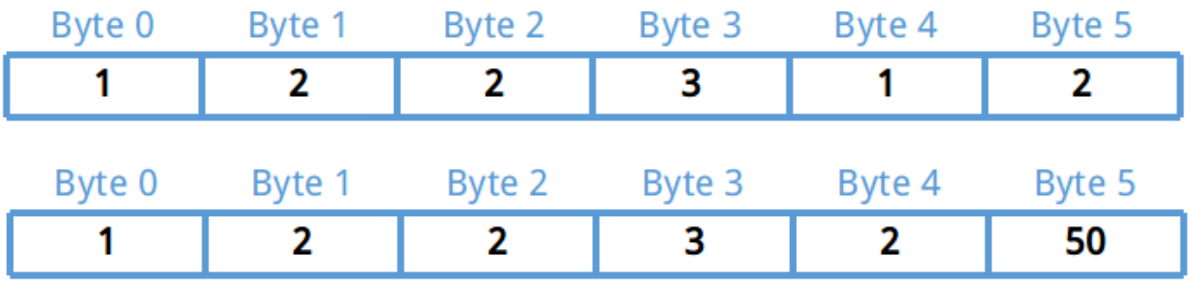
5.3.1.3 Controle manual do relé ou dimer

Quando o valor do **Byte 3** for três (3), significa que será feito um acesso ao relé ou ao dimer, no entanto não será um controle realimentado usando valores vindos do sensor. O controle será feito manualmente. Seguem os significados relativos aos dois últimos bytes.

- **Byte 4 = 1** - *Controle manual do Relé:*
  - **Byte 5:** 2 para ligar o relé, 1 para desligar.
- **Byte 4 = 2** - *Controle manual do Dimer:*
  - **Byte 5:** Valor em porcentagem da potência inserida na carga.

Na Figura 22 são mostrados dois possíveis pacotes de dados que utilizariam esta função.

Figura 22 – Exemplo de pacote para controle manual de carga via dimer ou relé.



Fonte: Produção do próprio autor.

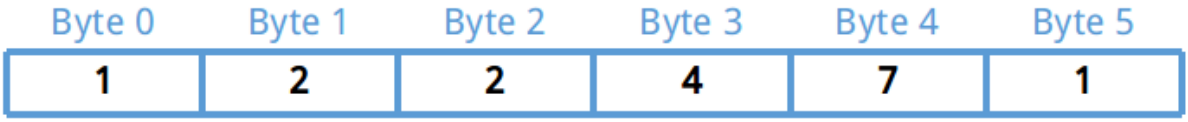
O primeiro pacote é um comando de ligar o relé. O segundo pacote é um comando de controlar a carga através do dimer em cinquenta por cento (50%) da onda total de alimentação.

5.3.1.4 Pinos digitais de entrada

Quando o valor do **Byte 3** for quatro (4), significa que será acessado algum pino digital de entrada. Assim, o **Byte 4** deverá ser o nome (número) do pino e o **Byte 5** será sempre de valor um (1).

Segue na Figura 23 um exemplo de pacote para tal função.

Figura 23 – Exemplo de pacote para leitura dos pinos digitais de entrada.



Fonte: Produção do próprio autor.

Neste caso, o pino 7 será lido.

5.3.1.5 Pinos digitais de saída

Se o valor do **Byte 3** for cinco (5), então será promovida alguma ação sobre algum dos pinos digitais de saída. Segue a função dos últimos dois bytes.

- **Byte 4:** Número do pino sobre o qual se quer atuar.
- **Byte 5:** 2 para aplicar tensão de $5V$ e 1 para aplicar tensão de $0V$.

A Figura 24 mostra um exemplo de pacote para o caso especificado.

Figura 24 – Exemplo de pacote para atuação sobre os pinos digitais de saída.

Fonte: Produção do próprio autor.

Neste caso, o pino oito (8) receberá uma alimentação de cinco volts ($5V$).

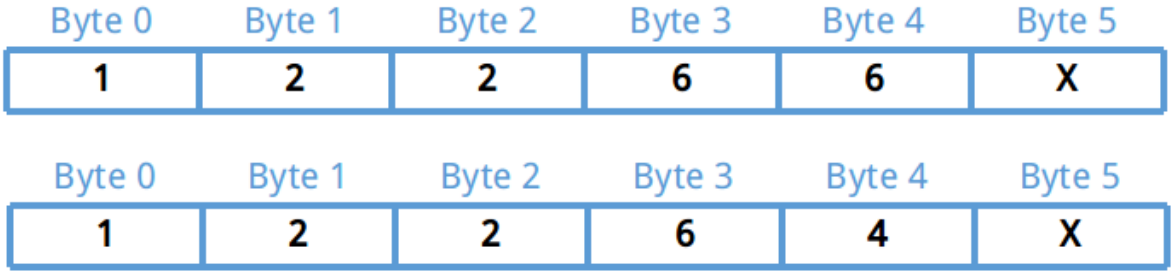
5.3.1.6 Captura de temperatura

Se o valor do **Byte 3** for seis (6), então o objetivo é fazer com que o controlador passe a simplesmente captar dados de temperatura local e enviá-los para o concentrador, ou o contrário. Segue explicação da função.

- **Byte 4:** 6 para permitir o envio de dado de temperatura, 4 para cancelar permissão.

Segue na Figura 25 um exemplo de pacote para a função citada.

Figura 25 – Exemplo de pacote para atuação sobre o sensor de temperatura.



Fonte: Produção do próprio autor.

O primeiro pacote inicia o envio de dados de temperatura ao concentrador, o segundo finaliza este envio.

5.3.2 Pacote de dados enviado pelo controlador

O pacote de dados enviado pelo controlador ao coordenador da rede possui dois bytes e tem como conteúdo o valor de temperatura do sensor.

A temperatura é adquirida via protocolo *I2C* e contém um resultado em *float*. Para melhorar o envio e gastar menos memória no banco de dados do coordenador, foi optado por enviar o valor em dois bytes.

Usando como exemplo um dado de temperatura fictício, será mostrado nas equações a seguir o processo de adaptação do valor até o envio do dado.

- Valor fictício.

$$T_{(^oC)} = 27.3235$$

- Multiplicação por 100.

$$Tx100 = 2732.35$$

- Transformação de *float* para *inteiro*.

$$T_{(int)} = 2732$$

- Separação em dois bytes.

– Transformando para binário.

$$2732_{(dec)} = 0000101010101100_{(bin)}$$

– Separando de 8 em 8 bits.

$$Byte1 = 00001010_{(bin)}$$

$$Byte2 = 10101100_{(bin)}$$

– Transformando para decimal

$$00001010_{(bin)} = 10_{(dec)}$$

$$10101100_{(bin)} = 172_{(dec)}$$

Assim o valor de temperatura é enviado em forma de dois bytes.

5.4 Placa de comunicação sem fio

Como mostra na Figura 19 foi usado apenas um tipo de placa para acoplar o controlador ao rádio. A placa usada foi a *Seeeduino Stalker* conforme mostrada na Figura 7 e explicada na sessão 2.3.

5.4.1 Pacote de dados recebido pela placa de comunicação sem fio

Quando o dado recebido vem do concentrador, o que significa que é um dado de comando direcionado ao controlador, o próprio protocolo *ZigBee* faz a verificação se o dado é para ele mesmo ou não, portanto a placa de comunicação sem fio dispensa verificação quanto à destinação do dado e dentro do ciclo principal do código faz uma verificação de chegada de dados. Quando os dados chegam, eles são salvos em seis (6) bytes.

Quando o dado recebido vem do controlador, o que significa que é um dado de temperatura destinado ao concentrador, uma interrupção é ativada na placa de comunicação sem fio, então os dados são salvos em uma variável de dois bytes. além disso um marcador é ativado para que no ciclo principal do código estes dados de temperatura sejam enviados para o radio *XBee*.

5.4.2 Pacote de dados enviado pela placa de comunicação sem fio

Quando o dado a ser enviado vem do concentrador (destinado ao controlador), este é enviado e então o primeiro byte é modificado de um (1) para zero (0), pois conforme descrito no Quadro 2, o primeiro byte serve para dizer se a informação é nova ou não. Então, ela deixa de ser nova logo após ter sido enviada ao controlador.

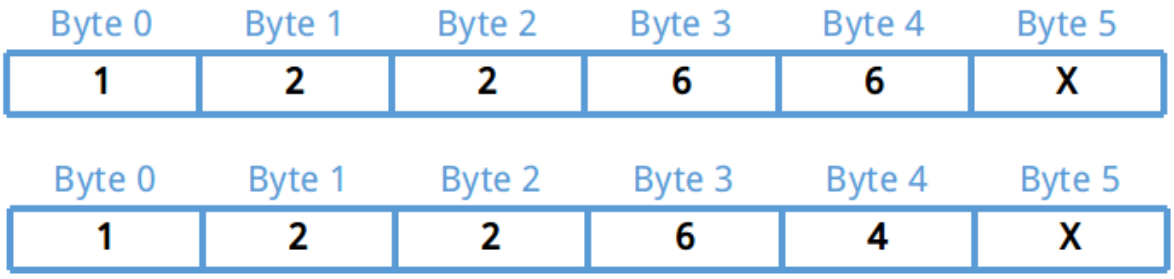
Quando o dado a ser enviado vem do controlador (destinado ao concentrador), este é enviado e então o marcador é desativado para que o mesmo dado não volte a ser enviado.

5.5 Placa de sensoreamento

É possível observar na Figura 19 que existem três placas *Seeeduino* (mostradas na Figura 7 e explicadas na sessão 2.3) que não estão ligadas a controlador nenhum e estão ligadas a sensores. Apesar da placa ser a mesma utilizada como placa de comunicação sem fio (ver sessão 5.4) suas funções são diferentes.

Quando ela recebe um pacote destinado a ela mesma, que tem a composição mostrada na Figura 26, ela passa a captar dados de temperatura do ambiente e enviá-los ao concentrador, ou para de captar dados.

Figura 26 – Pacote destinado à placa de sensoreamento.



Fonte: Produção do próprio autor.

O primeiro pacote mostrado na Figura 26 habilita a placa a captar dados, o segundo a desabilita.

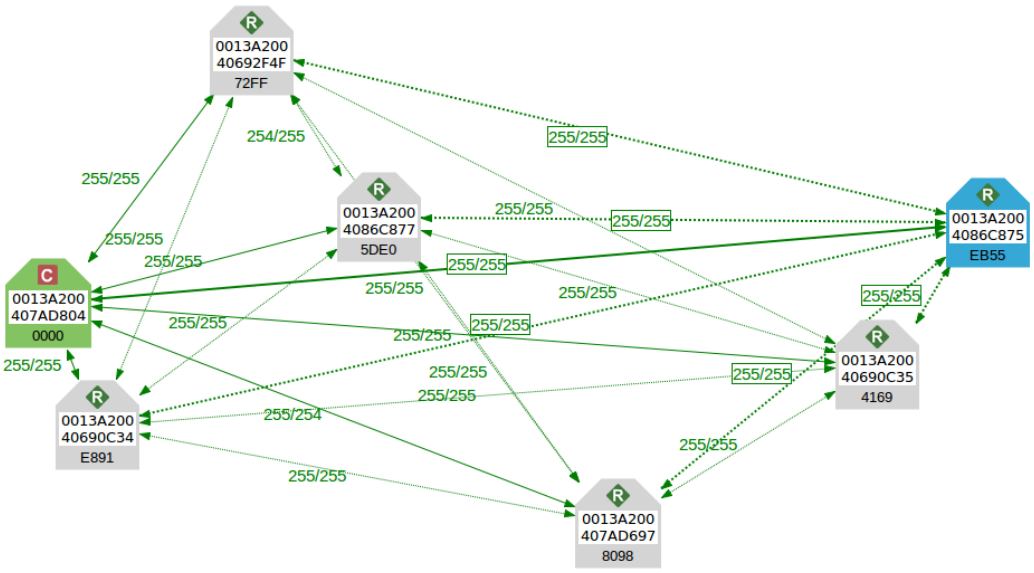
5.6 Rede sem fio

O protocolo de comunicação sem fio usado é o *ZigBee*, conforme explicado na sessão 2.2, o rádio usado para se comunicar através deste protocolo é o *XBee* (ver Figura 8 da sessão

2.3). O aplicativo usado para fazer a conFiguração dos rádios foi o *XCTU* (software da empresa *DIGI* usado para conFiguração dos rádios por ela produzida).

O grafo da rede, gerado pelo *XCTU*, é mostrado na Figura 27.

Figura 27 – Grafo da rede gerado pelo *XCTU*.

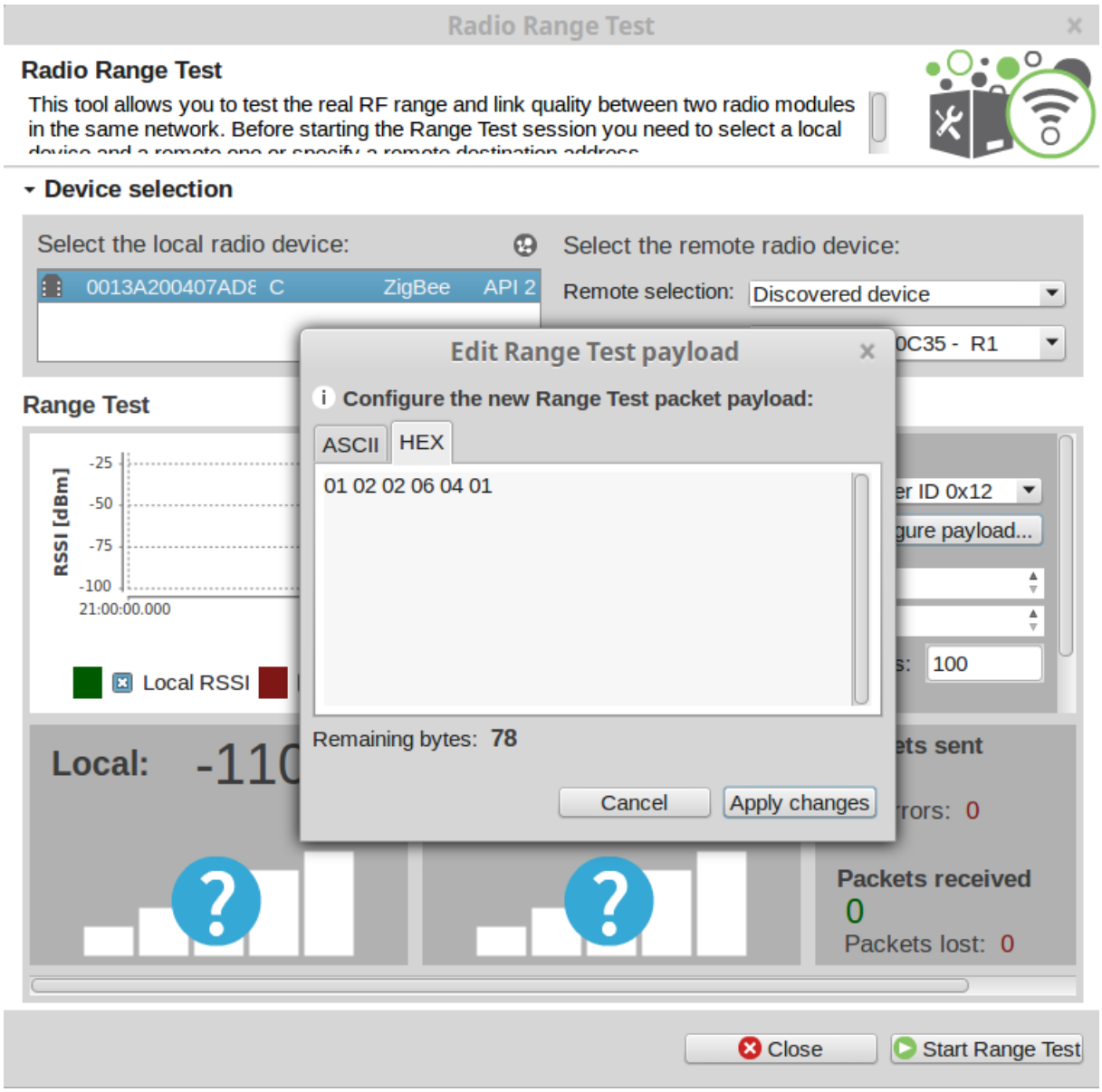


Fonte: Produção do próprio autor.

Na Figura 27 percebe-se uma ligação entre todos os rádios, no entanto esta ligação mostra apenas que todos os rádios estão em possível comunicação. Na realidade a rota é fixa, mas pacotes de teste circulam entre a rede verificando sempre a continuidade e a qualidade das ligações entre rádios, assim se uma outra rota se torna melhor, o desenho da rede muda. Portanto um pacote enviado de um rádio não segue dois caminhos, ele segue apenas o melhor caminho, evitando assim o desperdício de energia.

Foi feito também um teste de alcance (ou potência). Conforme mostra a Figura 28, para ele foi usado um pacote de dados de seis bytes que na rede desenvolvida serviriam para acionar a leitura de temperatura em algum controlador (ver sessão 5.3.1.6) ou para acionar a leitura de temperatura em alguma placa de sensoreamento (ver sessão 5.5).

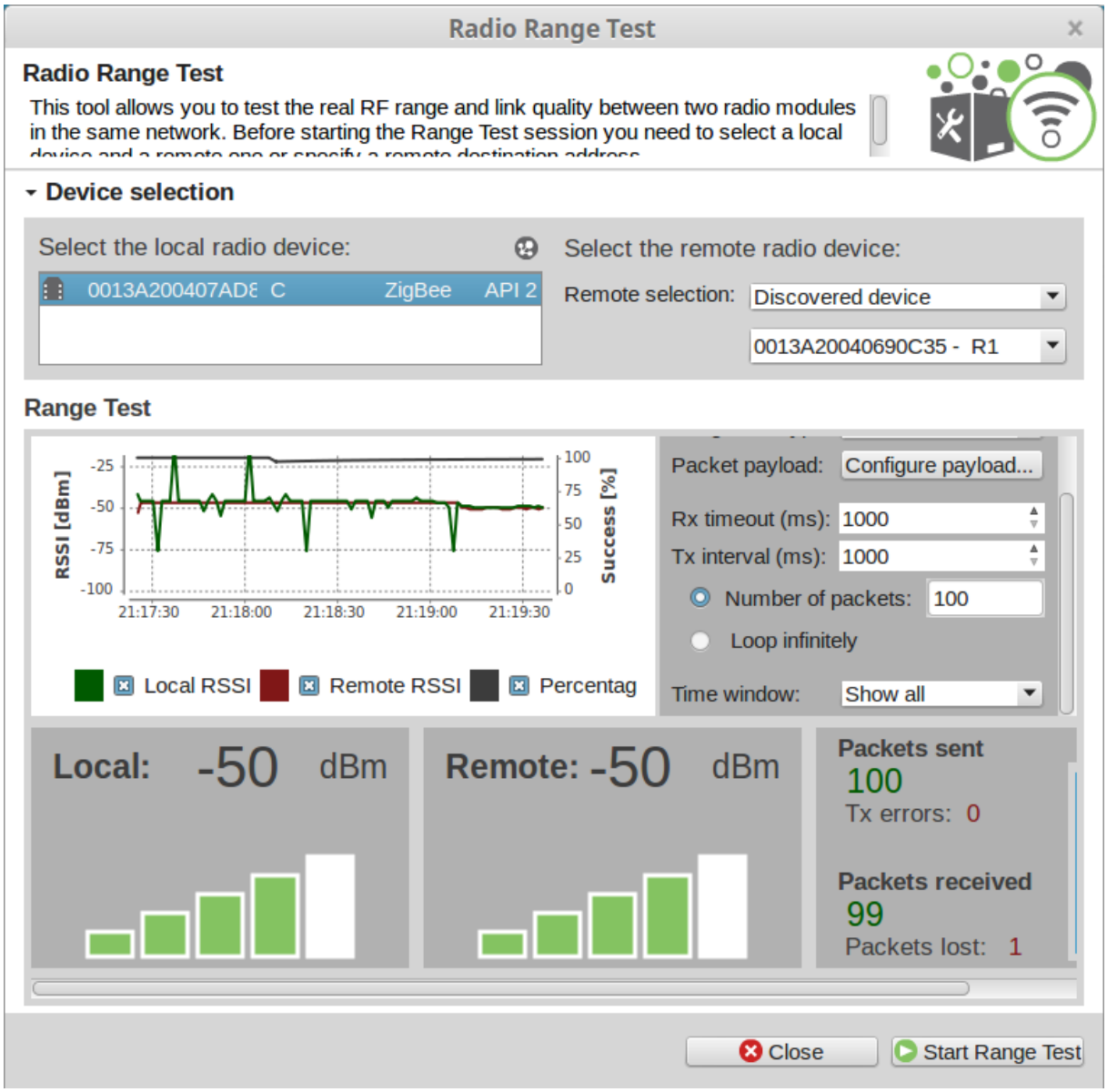
Figura 28 – Pacote usado no teste de alcance.



Fonte: Produção do próprio autor.

Para o teste, os rádios estavam a uma distância de apenas um metro, com um obstáculo de madeira de $10cm$ entre eles, usando uma antena $2dB_i$ no coordenador e uma antena $8dB_i$ no roteador. Foram enviados cem pacotes do rádio local (coordenador) para o rádio remoto (roteador). Para cada pacote enviado foi gerada também uma resposta que foi do rádio remoto para o local. A Figura 29 mostra o resultado do teste.

Figura 29 – Teste de alcance de rádios.



Fonte: Produção do próprio autor.

Conforme mostra a Figura 29 a potência no roteador varia entre $-42dB_m$ e $-52dB_m$ (equivalente a um sinal entre $63.1\eta W$ e $6.31\eta W$). No coordenador as variações foram mais bruscas, indo desde $-19.5dB_m$ até $-75dB_m$ (entre $11.22\mu W$ e $31.6\rho W$). É possível observar que uma antena melhor promove uma recepção mais estável, a antena menor além de gerar um sinal instável na recepção, gera vales no gráfico que significam um sinal de potência muito baixa recebido.

É necessário lembrar que o sinal *RSSI* usado para medição da intensidade do sinal, é medido utilizando apenas o preâmbulo, portanto se o sinal no rádio remoto se manteve sempre numa faixa muito próxima de sinal, é razoável que a perda de um dos pacotes

(que acontece quando nada do pacote chega ou quando o pacote não possui o conteúdo enviado) tenha sido causada pelo emissor (rádio local) que esteve conectado a uma antena de $2dB_i$. Assim, fica decido que na rede, mesmo que os rádios remotos (roteadores) estejam conectados a antenas menores, o coordenador deve ser conectado a uma antena maior e mais confiável, pois terá de lidar com um fluxo muito maior de dados.

5.7 Concentrador

Conforme mostra a Figura 19, o que liga o concentrador à rede é um rádio *XBee* e um *FTDI* (conversor *TTL-USB*). Como a conversão dos dados desde o *XBEE* até o computador é automática, nesta sessão falaremos apenas do servidor e no banco de dados contidos do concentrador.

5.7.1 Banco de dados

O banco dados é composto por sete tabelas *MySQL*. A ideia é que as tabelas ajudem a criar as entidades virtuais futuramente. Para tal, é preciso que haja uma cópia das tabelas que ficará no banco de dados externo representado na Figura 14.

As sete tabelas são:

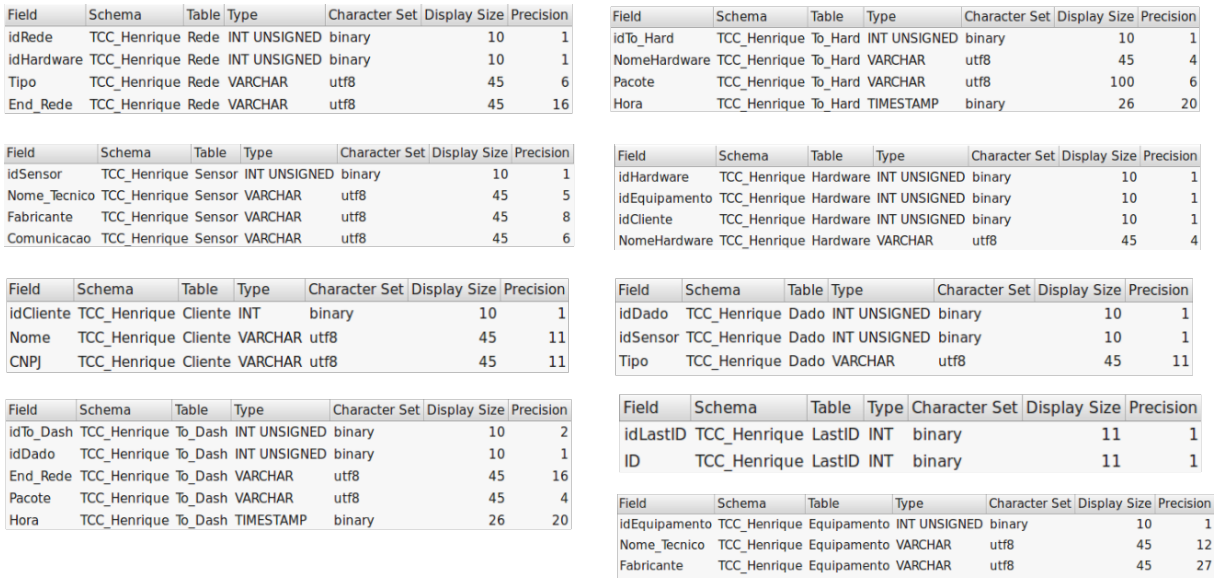
- **Cliente:** Define quais os clientes que usam este modelo de rede *IoT* (Tabela opcional).
- **Equipamento:** Fala sobre características técnicas dos equipamentos controladores e aqueles que são usados como placa de sensoreamento.
- **Sensor:** Mostra quais sensores são usados no projeto e como é a comunicação com eles.
- **Dado:** Como existem sensores que captam mais de um tipo de dado, esta tabela serve para dizer quais os possíveis dados medidos pela rede.
- **Hardware:** Especifica o nome de cada hardware usado para captação de dados ou controle.
- **Rede:** Apresenta características de rede, inclusive o endereço.
- **To_Dash:** São os dados que vêm da rede e que possam ser usados para geração de gráficos por exemplo.
- **To_Hard:** Dados enviados do concentrador para algum dos equipamentos.

- **LastID:** Mostra o último comando da tabela **To_Hard** que foi enviado e é usado para que o servidor saiba qual foi o último comando enviado para a rede.

As tabelas funcionam dentro de uma dinâmica de busca de dados. A dinâmica dos dados que vão do concentrador pra algum outro dispositivo da rede é diferente daquela relativa aos dados que vão de algum dispositivo da rede para o concentrador. Por isto, as duas próximas sessões tratam do assunto separadamente.

As características específicas de cada tabela (modelo de entidade) são mostradas na Figura 30.

Figura 30 – Modelo de entidade das tabelas do Banco de Dados.



Field	Schema	Table	Type	Character Set	Display Size	Precision
idRede	TCC_Henrique	Rede	INT UNSIGNED	binary	10	1
idHardware	TCC_Henrique	Rede	INT UNSIGNED	binary	10	1
Tipo	TCC_Henrique	Rede	VARCHAR	utf8	45	6
End_Rede	TCC_Henrique	Rede	VARCHAR	utf8	45	16

Field	Schema	Table	Type	Character Set	Display Size	Precision
idTo_Hard	TCC_Henrique	To_Hard	INT UNSIGNED	binary	10	1
NomeHardware	TCC_Henrique	To_Hard	VARCHAR	utf8	45	4
Pacote	TCC_Henrique	To_Hard	VARCHAR	utf8	100	6
Hora	TCC_Henrique	To_Hard	TIMESTAMP	binary	26	20

Field	Schema	Table	Type	Character Set	Display Size	Precision
idSensor	TCC_Henrique	Sensor	INT UNSIGNED	binary	10	1
Nome_Tecnico	TCC_Henrique	Sensor	VARCHAR	utf8	45	5
Fabricante	TCC_Henrique	Sensor	VARCHAR	utf8	45	8
Comunicacao	TCC_Henrique	Sensor	VARCHAR	utf8	45	6

Field	Schema	Table	Type	Character Set	Display Size	Precision
idHardware	TCC_Henrique	Hardware	INT UNSIGNED	binary	10	1
idEquipamento	TCC_Henrique	Hardware	INT UNSIGNED	binary	10	1
idCliente	TCC_Henrique	Hardware	INT UNSIGNED	binary	10	1
NomeHardware	TCC_Henrique	Hardware	VARCHAR	utf8	45	4

Field	Schema	Table	Type	Character Set	Display Size	Precision
idCliente	TCC_Henrique	Cliente	INT	binary	10	1
Nome	TCC_Henrique	Cliente	VARCHAR	utf8	45	11
CNPJ	TCC_Henrique	Cliente	VARCHAR	utf8	45	11

Field	Schema	Table	Type	Character Set	Display Size	Precision
idDado	TCC_Henrique	Dado	INT UNSIGNED	binary	10	1
idSensor	TCC_Henrique	Dado	INT UNSIGNED	binary	10	1
Tipo	TCC_Henrique	Dado	VARCHAR	utf8	45	11

Field	Schema	Table	Type	Character Set	Display Size	Precision
idTo_Dash	TCC_Henrique	To_Dash	INT UNSIGNED	binary	10	2
idDado	TCC_Henrique	To_Dash	INT UNSIGNED	binary	10	1
End_Rede	TCC_Henrique	To_Dash	VARCHAR	utf8	45	16
Pacote	TCC_Henrique	To_Dash	VARCHAR	utf8	45	4
Hora	TCC_Henrique	To_Dash	TIMESTAMP	binary	26	20

Field	Schema	Table	Type	Character Set	Display Size	Precision
idLastID	TCC_Henrique	LastID	INT	binary	11	1
ID	TCC_Henrique	LastID	INT	binary	11	1

Field	Schema	Table	Type	Character Set	Display Size	Precision
idEquipamento	TCC_Henrique	Equipamento	INT UNSIGNED	binary	10	1
Nome_Tecnico	TCC_Henrique	Equipamento	VARCHAR	utf8	45	12
Fabricante	TCC_Henrique	Equipamento	VARCHAR	utf8	45	27

Fonte: Produção do próprio autor.

5.7.1.1 Do concentrador para algum dispositivo

A tabela principal de envio de comandos do concentrador para a rede é a tabela **To_Hard**, o Quadro 3 mostra a estrutura de campos da tabela **To_Hard** e um exemplo de dado.

Quadro 3 – Tabela **To_Hard** para dados direcionados a um dispositivo da rede.

idTo_Hard	**NomeHardware**	**Pacote**	**Hora**
1	CP_1	060606	2017-12-27 18:55:57.589344

Fonte: Produção do próprio autor.

Observa-se que o pacote contém informações em hexadecimal, usado para facilitar a construção do pacote final que será enviado (ver sessão 5.7.2). Temos como pacote uma

informação com apenas três bytes, enquanto na verdade o pacote que é enviado até o controlador ou a placa de sensoreamento possui seis bytes, o motivo é explicado na sessão 5.3.1.

O campo *NomeHardware* é usado para a formação do pacote, pois uma informação necessária para enviar o pacote é qual o destino. Portanto para o caso deste exemplo, o nome do hardware de destino é CP_1. Então a tabela **Hardware** (Quadro 4) é acessada.

Quadro 4 – Tabela **Hardware** para dados direcionados a um dispositivo da rede.

idHardware	**idEquipamento**	**idCliente**	**NomeHardware**
1	1	2	CP_1

Fonte: Produção do próprio autor.

O Quadro 4 mostra uma tabela de pouco dinamismo, a cada novo equipamento inserido na rede é necessário adicioná-lo nesta tabela. Os campos *idEquipamento* e *idCliente* são usados para relacionar o envio de dados a um tipo de equipamento e a um cliente específico, porém não são campos vitais para o envio do pacote. os campo *idHardware* é usado para localizar o endereço de rede em uma outra tabela. O *idHardware* do exemplo dado é 1. Esta informação será agora procurada na tabela **Rede** (Quadro 5).

Quadro 5 – Tabela **Rede** para dados direcionados a um dispositivo da rede.

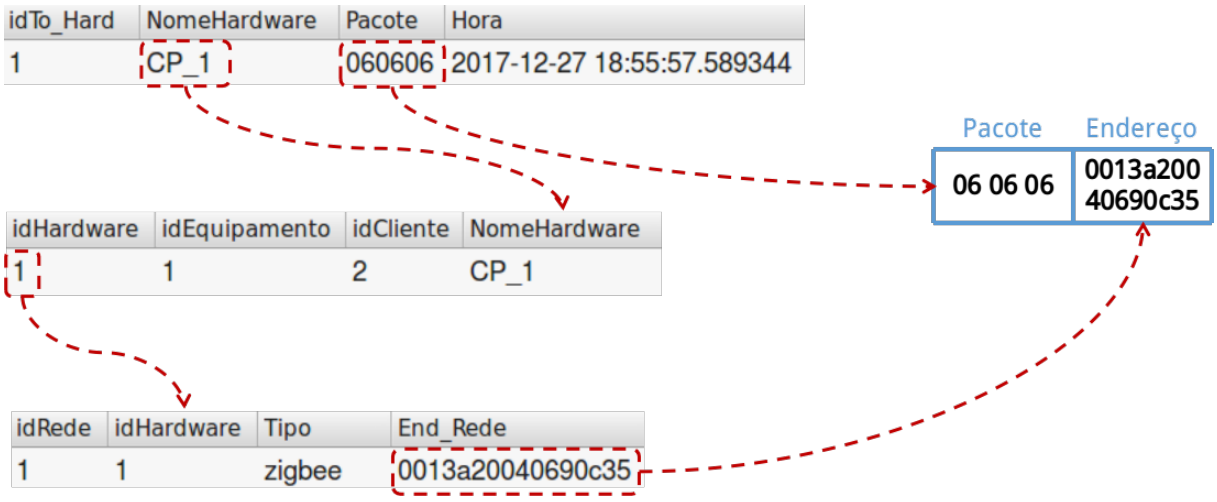
idRede	**idHardware**	**Tipo**	**End_Rede**
1	1	zigbee	0013a20040690c35

Fonte: Produção do próprio autor.

A tabela **Rede** tem também pouco dinamismo, a cada novo equipamento inserido na rede é necessário inserir suas informações de rede, no caso da rede desenvolvida o protocolo de comunicação usado foi *ZigBee* e o endereço de rede do exemplo dado é $0013a20040690c35$. Assim, em posse do dado a ser enviado e do endereço a quem é objetivado enviar, o pacote pode então ser formado pelo servidor e enviado ao controlador da rede.

A Figura 31 mostra de forma didática o processo de busca do dado e do endereço de destino do pacote.

Figura 31 – Tabelas *MySQL* para envio de dados aos dispositivos da rede.



Fonte: Produção do próprio autor.

O pacote a ser enviado para a rede alcançará portando o controlador CP_1 e fará com que ele comece a enviar dados de temperatura para o servidor (ver sessão 5.3.1.6).

Ao fim de cada novo dado enviado à rede, a tabela **LastID** (ver Figura 41) atualiza para o ID do último comando enviado da tabela **To_Hard**.

5.7.1.2 De algum dispositivo para o concentrador

A tabela principal de envio de dados dos dispositivos para o concentrador é a tabela **To_Dash**. É considerado que os dados nela salvos poderão ser, em algum momento, exibidos em uma tela de monitoramento.

O Quadro 6 mostra a estrutura dos campos da tabela e um exemplo de dado.

Quadro 6 – Tabela **To_Dash** para dados direcionados ao concentrador da rede.

idTo_Dash	idDado	End_Rede	Pacote	Hora
63	1	0013a20040690c35	2697	2017-12-27 18:56:37.887275

Fonte: Produção do próprio autor.

Na tabela **To_Dash** (Quadro 6) já pode ser obtido o conteúdo do pacote e o endereço de rede de quem o enviou. Os campos **idTo_Dash** e **Hora** servem para que seja possível identificar a posição dos valores nos gráficos a serem gerados em uma tela de monitoramento por exemplo. O Campo *End_Rede* também é usado para captar informações na tabela **Rede** (Quadro 7).

Quadro 7 – Tabela **Rede** para dados direcionados ao concentrador da rede.

idRede	**idHardware**	**Tipo**	**End_Rede**
1	1	zigbee	0013a20040690c35

Fonte: Produção do próprio autor.

Em posse do endereço de rede, sabemos a partir do Quadro 7 qual o protocolo de rede pelo qual o pacote circula (*ZigBee*) e qual o *idHardware*, que será utilizado para captar informações na tabela **Hardware**, o exemplo é mostrado no Quadro 8 .

Quadro 8 – Tabela **Hardware** para dados direcionados ao concentrador da rede.

idHardware	**idEquipamento**	**idCliente**	**NomeHardware**
1	1	2	CP_1

Fonte: Produção do próprio autor.

Do Quadro 8 podemos tirar de informação o nome do hardware (CP_1 para o exemplo dado), além dos campos *idEquipamento* e *idCliente* terem uso para acessar outros dados. As tabelas **Equipamento** e **Cliente** (ver Quadros 9 e 10) mostram informações sobre o equipamento que enviou o referido dado e a qual usuário (cliente) ele pertence (ver sessão 5.7.1).

Quadro 9 – Tabela **Equipamento** para dados direcionados ao concentrador da rede.

idEquipamento	**Nome_Tecnico**	**Fabricante**
1	PLC	Flávio, Juliana e Henrique

Fonte: Produção do próprio autor.

Na tabela **Equipamento** (Quadro 9) é possível haver mais informações sobre o dispositivo de sensoreamento ou o controlador.

Quadro 10 – Tabela **Cliente** para dados direcionados ao concentrador da rede.

idCliente	**Nome**	**CNPJ**
2	Ufes	98885465480

Fonte: Produção do próprio autor.

Na tabela **Cliente** (Quadro 10) é possível que sejam incluídas mais informações a respeito de quem usa a rede.

Ainda no Quadro 6 o campo *idDado* é usado para captar informações na tabela **Dado**, um exemplo é mostrado no Quadro 11.

Quadro 11 – Tabela **Dado** para dados direcionados ao concentrador da rede.

idDado	**idSensor**	**Tipo**
1	1	temperatura

Fonte: Produção do próprio autor.

A partir do Quadro 11 é possível perceber que o dado recebido corresponde a um dado de temperatura. Outra informação importante nesta tabela é o *idSensor*, que será usado na tabela **Sensor**, segue exemplo no Quadro 12.

Quadro 12 – Tabela **Sensor** para dados direcionados ao concentrador da rede.

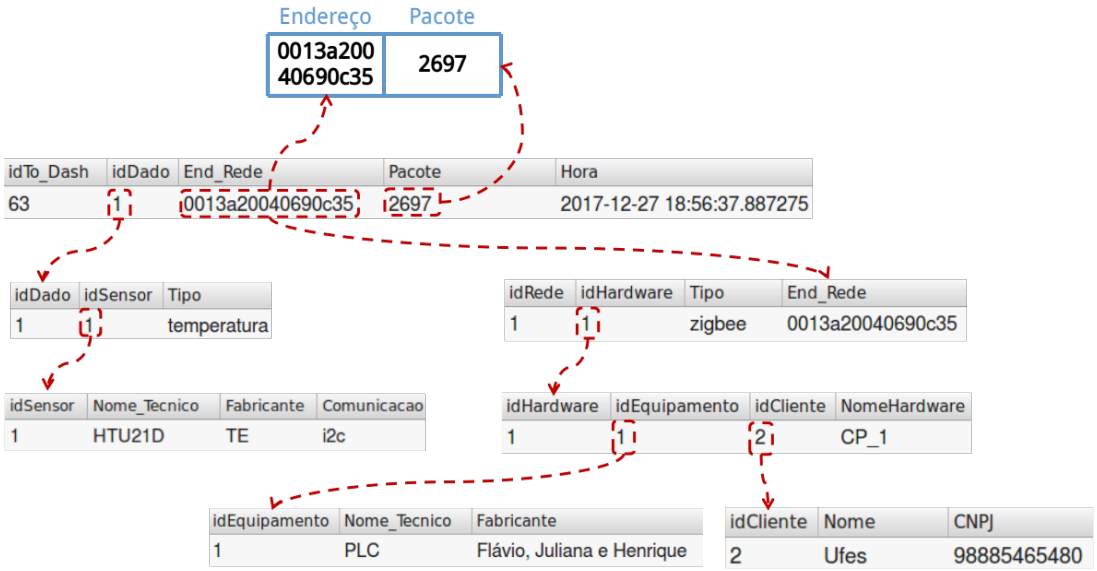
idSensor	**Nome_Tecnico**	**Fabricante**	**Comunicação**
1	HTU21D	TE	i2c

Fonte: Produção do próprio autor.

Com a tabela **Sensor** (Quadro 12) podemos obter mais informações relativas ao sensor usado para a captação do dado estudado.

A Figura 32 mostra de forma didática a estrutura de relação das tabelas para obtenção das informações necessárias.

Figura 32 – Tabelas *MySQL* para envio de dados ao concentrador da rede.



Fonte: Produção do próprio autor.

É possível perceber pela Figura 32 que com as informações que são salvas na tabela **To_Dash**, é possível acessar qualquer informação que tenha sido salva em alguma outra tabela, portanto se as informações serão usadas para serem exibidas, para gerarem um gráfico, ou até mesmo para criar uma entidade virtual, é importante que o banco de dados tenha tabelas relacionadas que deem esta possibilidade às pessoas que vão criar e conFigurar toda rede.

5.7.2 Servidor

Neste caso, o servidor mostrado na Figura 18 tem duas funções, que são explicadas a seguir:

- Tornar os pacotes vindos da rede compreensíveis e salvá-los no banco de dados
- Captar os dados que foram inseridos no banco de dados com destino a algum dos dispositivos da rede e torná-los compreensíveis para o rádio que vai enviá-los.

Assim, esta sessão fica dividida em duas partes para explicar as funções separadamente.

5.7.2.1 Pacotes destinados à rede

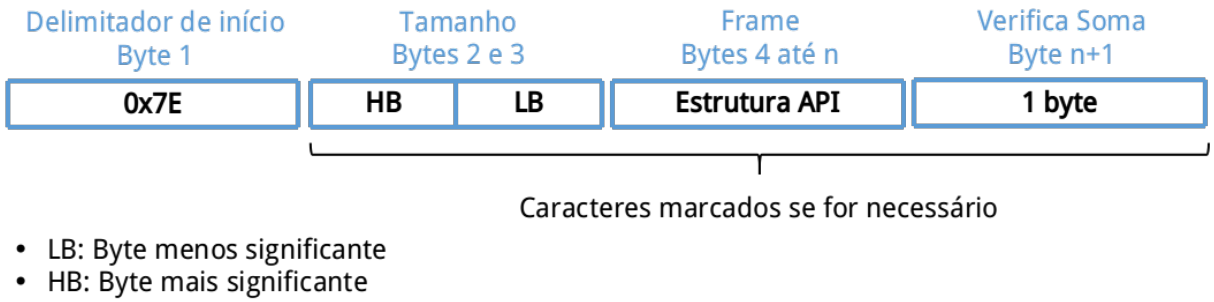
Um pacote destinado a algum dispositivo da rede deve possuir um conjunto específico de características para que o rádio possa enviá-lo corretamente.

De acordo com Digi International (2017) ao enviar ou receber um pacote de dados *UART*, os valores de dados específicos devem ser marcados para que eles não interfiram com o sequenciamento do pacote de dados. Para marcar um byte de dados interferente, insira *0x7D* e siga-o com o byte para ser marcado passando por uma operação de soma com *0x20*. Observe que, se não for API2, os caracteres *0x11* e *0x13* são enviados como são.

Para tal, é utilizado nos rádios o modo API2 (API com marcação).

A estrutura de um pacote segue na Figura 33.

Figura 33 – Estrutura de um pacote no modo API - 2.



Fonte: Adaptado de Digi International (2017)

Segue uma explicação de cada campo do pacote de dados, de acordo com Digi International (2017).

- **Delimitador de início:** Mostra que é um início de pacote.
- **Tamanho:** Número de bytes entre o campo **Tamanho** e o campo **Verifica Soma**.
- **Frame:**
  - *Tipo de frame:* Identifica se é um pacote recebido, ou a ser enviado, etc.
  - *ID Frame:* Identifica o pacote *UART* para o transmissor se correlacionar com um *ACK* subsequente (confirmação). Se definido como 0, nenhuma resposta é enviada.
  - *Endereço de destino (64 bits):* Defina o endereço de 64 bits do dispositivo de destino. Os seguintes também são suportados: 0x0000000000000000 (Endereço reservado de 64 bits para o coordenador) e 0x000000000000FFFF (Endereço de *broadcast*).
  - *Endereço de destino (16 bits):* Defina para o endereço de 16 bits do dispositivo de destino, se conhecido. Defina para *0xFFFE* se o endereço for desconhecido ou se estiver enviando um *broadcast*.

  - *Raio do* broadcast*:* Define o número máximo de saltos que pode ocorrer um *broadcast*. Se definido como 0, o raio de transmissão será ajustado para o valor máximo.
  - *Opções:* 0x00 (Nenhum) - 0x01 (Desativar tentativas de reparar rotas) - 0x20 (Ativar criptografia APS) - 0x40 (Use o tempo limite de transmissão prolongado).
  - *Dado:* A informação que realmente devemos enviar para o receptor.
- **Verifica soma:** 0xFF menos os 8 bits menos significativos da soma desde os bytes de deslocamento 3 até este byte.

No exemplo a seguir é mostrada a formação de um pacote que vai desde o concentrador até um dos controladores com o objetivo de autorizar o início do envio de pacotes.

Com as informações adquiridas a partir das tabelas *MySQL*, o campo *Pacote* captado na tabela **To_Hard** (Quadro 3) com o valor 060606 concatenando com 010202 (ver sessão 5.3.1) e o campo *End_Rede* da tabela **Rede** (Quadro 5) com o valor 0013a20040690c35, é possível começar a construção do pacote conforme segue:

- **Delimitador de início:** 7E
- **Tamanho:** 00 14
- **Frame:**
  - *Tipo de frame:* 10
  - *ID Frame:* 00
  - *Endereço de destino (64 bits):* 00 13 A2 00 40 69 0C 35
  - *Endereço de destino (16 bits):* FF FE
  - *Raio do* broadcast*:* 00
  - *Opções:* 01
  - *Dado:* 01 02 02 06 06 06
- **Verifica soma:** 3B

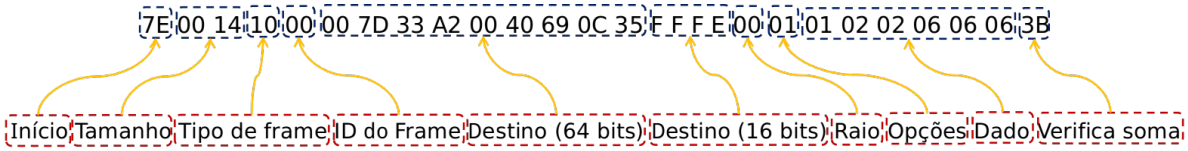
O pacote fica com a seguinte formação:

$$7E\ 00\ 14\ 10\ 00\ 00\ 13\ A2\ 00\ 40\ 69\ 0C\ 35\ FF\ FE\ 00\ 01\ 01\ 02\ 02\ 06\ 06\ 06\ 3B$$

No entanto o sétimo byte da esquerda para a direita possui valor 13 que é um valor a ser marcado, para tal adiciona-se 7D e soma 13 com 20.

Então o pacote final a ser enviado é mostra no na Figura 34

Figura 34 – Estrutura de um dado a ser enviado do concentrador para a rede.



Fonte:

Assim, o servidor fica capacitado para enviar pacotes de dados de uma rede *ZigBee*.

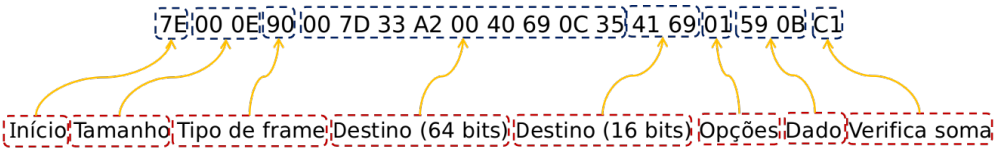
5.7.2.2 Pacotes vindos da rede

Quando um pacote vem de algum ponto da rede destinado ao concentrador para que os dados sejam armazenados, o servidor precisa entender quais partes do pacote representam informação útil (dados).

Para a recepção do pacote é usada a mesma estrutura mostrada na Figura 33 e explicada na sessão 5.7.2.1.

Para tal a Figura 35 mostra um exemplo de pacote recebido.

Figura 35 – Estrutura de um dado enviado pela rede e recebido pelo concentrador.



Fonte:

Primeiro, já que os pacotes estão no modo *API2*, é necessário desmarcá-lo. Para tal, onde há $7D$ deve-se excluí-lo e retirar 20 do próximo byte. Segue o pacote corrigido:

$$7E\ 00\ 0E\ 90\ 00\ 13\ A2\ 00\ 40\ 69\ 0C\ 35\ 41\ 69\ 01\ 59\ 0B\ C1$$

Agora é preciso verificar se a soma está correta, assim soma-se desde o quarto byte (90) até o penúltimo ($0B$).

$$(90+00+13+A2+00+40+69+0C+35+41+69+01+59+0B)_{(16)} = (33E)_{(16)} \quad (5.1)$$

Usa-se FF menos os oito bits menos significativos do resultado da equação 5.1 (3E).

$$(FF-3E)_{(16)} = (C1)_{(16)} \quad (5.2)$$

Assim verifica-se que o pacote não foi corrompido.

Como são enviados apenas dois bytes de cada dispositivo da rede para o concentrador, de acordo com o posicionamento das informações especificado por Digi International (2017) e mostrado na subsubsec:pacotesdestinadosarede, sabemos que os dois bytes de informação estão contidos nos campos $n-1$ e $n-2$. Para o exemplo dado eles são 59 $0B$. No entanto os dados são salvos pela rede invertidos no frame, além de estarem em valores hexadecimais.

Invertendo os bytes:

$$59\ 0B \implies 0B\ 59 \quad (5.3)$$

Transformando o resultado da equação 5.3 de hexadecimal para decimal:

$$(0B59)_{(16)} = (2905)_{(10)} \quad (5.4)$$

O resultado final obtido na equação 5.4 deve ser compreendido como 29.05^oC, no entanto o valor será armazenado no banco de dados como se encontra (2905) para poupar espaço de armazenamento já que uma variável *float* ocupa mais espaço que uma variável *int*.

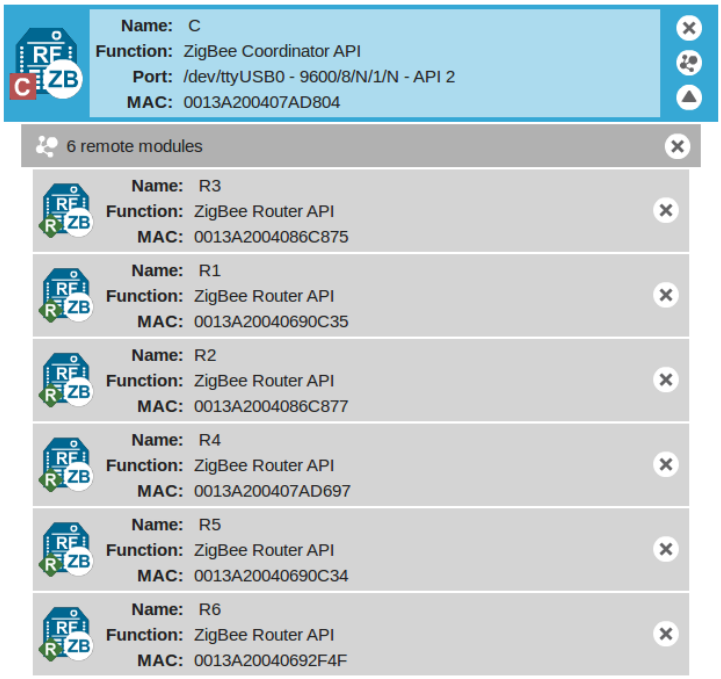
Outra informação importante no pacote é o endereço de rede (64 bits) de quem enviou, tal informação é obtida do byte cinco ao byte doze (00 13 $A2$ 00 40 69 $0C$ 35)

Assim, o servidor fica capacitado para receber pacotes de dados de uma rede *ZigBee*.

6 EXPERIMENTOS E RESULTADOS

Os testes foram feitos com o coordenador da rede *ZigBee* configurado em conexão com todos os roteadores, conforme mostra a figura 36.

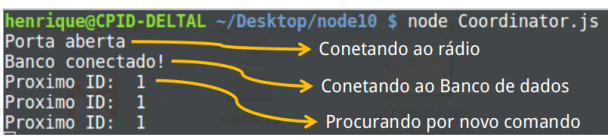
Figura 36 – O coordenador enxergando todos os roteadores (*XCTU*).



Fonte: Produção do próprio autor.

Assim, logo que o Servidor é inicializado, ele se conecta ao Rádio *XBee* e ao Banco de dados *MySQL*, e já começa a procura pelo último ID enviado à rede escrito na Tabela **LastID**. Então vai até a tabela **To_Hard** e procura se há algum comando na posição encontrada em LastID + 1. O processo de procura de comandos é temporizado. A figura 37 mostra o processo descrito.

Figura 37 – Início de conexão do servidor.



Fonte: Produção do próprio autor.

Então foram usadas seis linhas de comando para inserir dados no banco de dados, mais especificamente na tabela **To_Hard**. Os comandos são mostrados na figura 38

Figura 38 – Linhas de comando para adicionar dados na tabela **To_Hard**.

```
INSERT INTO TCC_Henrique.To_Hard (NomeHardware, Pacote) VALUES ('CP_1','060606'); -- Começa envio de temp CP1
INSERT INTO TCC_Henrique.To_Hard (NomeHardware, Pacote) VALUES ('CP_2','060606'); -- Começa envio de temp CP2
INSERT INTO TCC_Henrique.To_Hard (NomeHardware, Pacote) VALUES ('CP_3','060606'); -- Começa envio de temp CP3
INSERT INTO TCC_Henrique.To_Hard (NomeHardware, Pacote) VALUES ('CS_1','060606'); -- Começa envio de temp CS1
INSERT INTO TCC_Henrique.To_Hard (NomeHardware, Pacote) VALUES ('CS_2','060606'); -- Começa envio de temp CS2
INSERT INTO TCC_Henrique.To_Hard (NomeHardware, Pacote) VALUES ('CS_3','060606'); -- Começa envio de temp CS3
```

Fonte: Produção do próprio autor.

De tal forma, seis novas linhas foram adicionadas à tabela **To_Hard**, conforme mostra a figura 39, e cada comando representa solicitar aos componentes da rede que envie os valores de temperatura por eles captados.

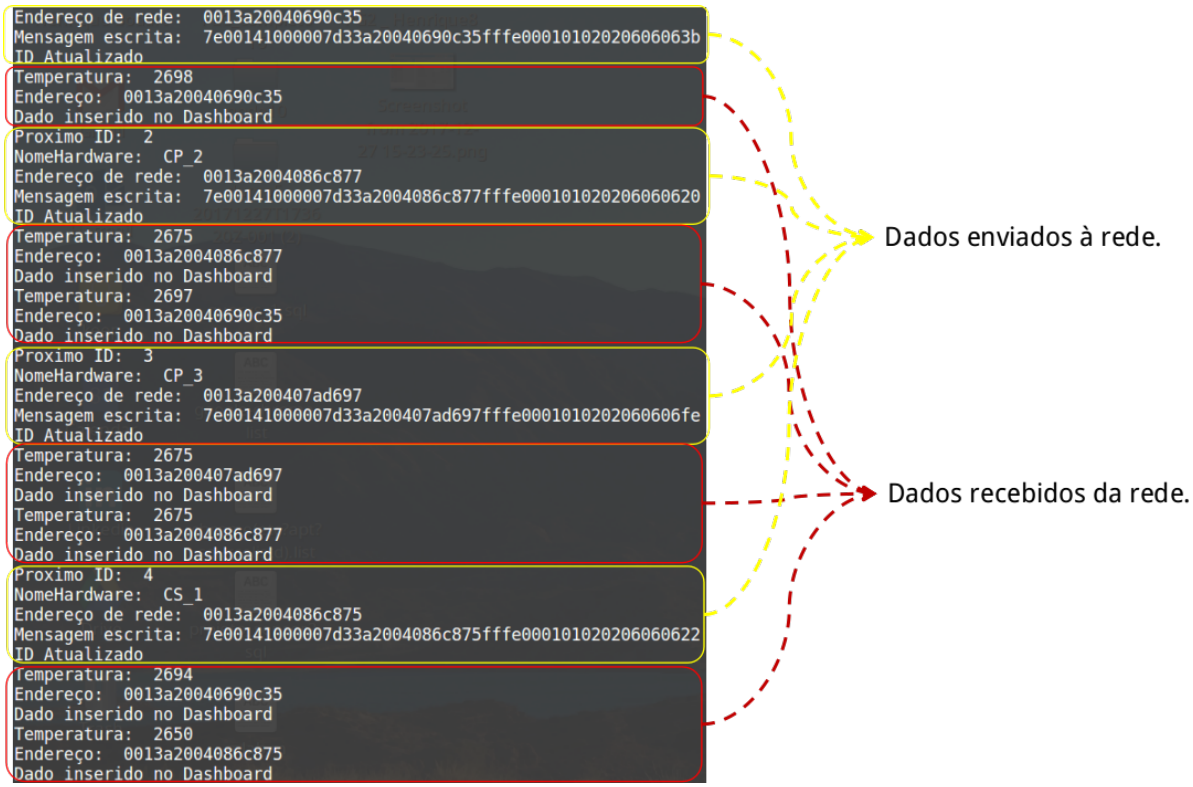
Figura 39 – Linhas adicionadas à tabela **To_Hard**.

idTo_Hard	NomeHardware	Pacote	Hora
1	CP_1	060606	2017-12-27 18:55:57.589344
2	CP_2	060606	2017-12-27 18:55:57.647879
3	CP_3	060606	2017-12-27 18:55:57.689722
4	CS_1	060606	2017-12-27 18:55:57.773308
5	CS_2	060606	2017-12-27 18:55:57.815075
6	CS_3	060606	2017-12-27 18:55:57.856953

Fonte: Produção do próprio autor.

O servidor irá então passar por cada uma das linhas e gerar um pacote que será enviado ao hardware de destino. No entanto mesmo durante o envio de pacotes, o servidor já começa a receber pacotes vindos da rede com os dados demandados daqueles que já receberam seus comandos. A figura 40 mostra o que foi descrito.

Figura 40 – Servidor envia pacotes para a rede enquanto recebe pacotes vindos da rede.



Fonte: Produção do próprio autor.

A cada dado enviado para a rede há uma atualização no campo *ID* da tabela **Last_ID**, após o envio dos seis dados, espera-se que o *ID* seja seis. Como mostra a figura 41

Figura 41 – Tabela **Last_ID** após o envio dos dados.

idLastID	ID
1	6

Fonte: Produção do próprio autor.

Por fim, os dados recebidos são salvos na Tabela **To_Dash** conforme mostra a figura 42.

Figura 42 – Tabela **To_Dash** com os dados de temperatura vindos da rede.

idTo_Dash	idDado	End_Rede	Pacote	Hora
58	1	0013a200407ad697	2675	2017-12-27 18:56:36.070765
59	3	0013a2004086c875	2650	2017-12-27 18:56:36.381772
60	3	0013a20040690c34	2600	2017-12-27 18:56:36.992295
61	1	0013a2004086c877	2675	2017-12-27 18:56:37.480783
62	3	0013a20040692f4f	2800	2017-12-27 18:56:37.661951
63	1	0013a20040690c35	2697	2017-12-27 18:56:37.887275
64	3	0013a2004086c875	2650	2017-12-27 18:56:38.604029
65	3	0013a20040690c34	2600	2017-12-27 18:56:39.220521
66	3	0013a20040692f4f	2800	2017-12-27 18:56:39.903336
67	1	0013a200407ad697	2675	2017-12-27 18:56:40.237621
68	3	0013a2004086c875	2650	2017-12-27 18:56:40.831692
69	3	0013a20040690c34	2625	2017-12-27 18:56:41.469864
70	1	0013a2004086c877	2675	2017-12-27 18:56:41.646980
71	1	0013a20040690c35	2697	2017-12-27 18:56:41.996719
72	3	0013a20040692f4f	2800	2017-12-27 18:56:42.144548

Fonte: Produção do próprio autor.

Assim se completa o ciclo de conexão de toda a rede.

7 CONCLUSÃO

Neste projeto é mostrado um diagrama conceitual para redes IoT idealizado pelo aluno (ver figura 14) a partir de outros diagramas encontrados na literatura.

Como o diagrama é aplicavel para redes IoT reais e varia de acordo com cada rede, foi desenvolvida apenas uma parte do conceito (ver figura 18). A rede desenvolvida é portanto uma parte de uma rede IoT real. A ideia do aluno é que esta parte seja genérica para toda rede, de tal forma que a implementação de uma rede real seja rápida e que a parte específica de cada rede aconteça apenas a partir do servidor e banco de dados externos.

Portanto o projeto aqui desenvolvido é uma prova de caso para uma parte do diagrama conceitual. O funcionamento do projeto desenvolvido mostra que o conceito idealizado funciona na prática.

Para redes mais simples, onde não há a necessidade de criar entidades virtuais por exemplo, a simples passagem do banco de dados interno para uma máquina externa e a criação de uma tela de monitoramento simples torna a rede aqui desenvolvida em uma rede IoT completa.

Para próximos passos do projeto considera-se montar uma rede IoT aplicada a um campo de irrigação, pois em tal caso será possível desenvolver o diagrama conceitual mostrado na Figura 14.

REFERÊNCIAS BIBLIOGRÁFICAS

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Comput. Netw.*, v. 54, p. 2787–2805, 2010.

AXELSON, J. *The Microcontroller Idea Book*. 2209 Winnebago St., Madison, WI 53704, USA: Lakeview Research, 1994.

BASSI, A. et al. *Enabling Things to Talk - Designing IoT solutions with the IoT Architectural Reference Model*. New York: Springer, 2013.

BELL, C. *MySQL for the Internet of Things*. 1st. ed. Berkely, CA, USA: Apress, 2016. 1–28 p.

CHAOUCHI, H. *The Internet of Things*. UK, London: John Wiley and Sons, Inc., 2013. 1–33 p.

CURVELLO, A. et al. *Editorial: Linguagens de Programação para Sistemas Embarcados*. 2015. Disponível em: <https://www.embarcados.com.br/editorial-linguagens-para-sistemas-embarcados/?utm_content=buffercd903&utm_medium=social&utm_source=linkedin.com&utm_campaign=buffer>. Acesso em: 2017-12-17.

DIGI INTERNATIONAL. *ZigBee RF Modules XBEE2, XBEEPRO2, PRO S2B*. 5. ed. [S.l.], 2017.

EMBRATEL. *Internet of Things - A Internet fora das telas*. 2017. Disponível em: <http://portal.embratel.com.br/embratel/ebook/assets/files/ebooks/ebook_embratel-iot_internet_fora_das_telas.pdf>. Acesso em: 2017-12-16.

ENGELS, D. W. et al. The networked physical world: An automated identification architecture. In: *In Proceedings of of the Second IEEE Workshop on Internet Applications*. [S.l.: s.n.], 2002.

EVANS. *The Internet of Things - How the Next Evolution of the Internet Is Changing Everything*. 2011.

GERBER, A. *Choosing the best hardware for your next IoT project*. 2017.

GRIDLING, G.; WEISS, B. *Introduction to Microcontrollers*. [S.l.]: Vienna University of Technology, 2007.

GROUP, E. I. W. et al. *IOT Developer Survey Results*. 2017.

HASHIMOTO, H. Intelligent space: Interaction and intelligence. *Artificial Life and Robotics*, v. 7, n. 3, p. 79–85, 2013. ISSN 1614-7456.

I2C.INFO. *I2C Info – I2C Bus, Interface and Protocol*. 2017. Disponível em: <http://i2c.info/>. Acesso em: 2017-12-18.

JIN, J. et al. An information framework for creating a smart city through internet of things. *IEEE Internet of Things Journal*, v. 1, n. 2, p. 112–121, April 2014. ISSN 2327-4662.

KILJANDER, J. et al. Semantic interoperability architecture for pervasive computing and internet of things. *IEEE Access*, v. 2, p. 856–873, 2014.

KLUBNIKIN, A. *IoT Boards Comparison: Choosing Hardware for IoT Project*. 2017. Disponível em: <http://r-stylelab.com/company/blog/iot/iot-boards-comparison-choosing-hardware-for-iot-project>. Acesso em: 2016-12-17.

LETHABY, N. *Wireless connectivity for the Internet of Things: One size does not fit all*. [S.l.], 2017.

LI, S. *Interview: Wired vs. Wireless in the IIoT*. 2016. Disponível em: <http://www.machinedesign.com/iot/interview-wired-vs-wireless-iiot>. Acesso em: 2017-12-17.

LINUX. *About Node.js*. 2017. Disponível em: <https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/>. Acesso em: 2017-12-18.

LINUX. *About Node.js*. 2017. Disponível em: <https://nodejs.org/en/about/>. Acesso em: 2017-12-18.

NILSSON, J. *Real-Time Control Systems with Delays*. Box 118, S-221 00 LUND, Sweden, 1998.

OGATA, K. *Modern Control Engineering*. 4th. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN 0130609072.

PENIAK, P.; FRANEKOVA, M. Open communication protocols for integration of embedded systems within industry 4. In: *Applied Electronics (AE), 2015 International Conference on*. [S.l.: s.n.], 2015. p. 181–184. ISSN 1803-7232.

QUEIROZ, F. M. de. *Desenvolvimento da Infraestrutura de um Espaço Inteligente baseado em Visão Computacional e IoT*. Monografia (TCC) — Universidade Federal do Espírito Santo, Vitória - ES, 2016.

SEEEDSTUDIO. *Seeeduino Stalker v2.3*. 2015. Disponível em: <http://wiki.seeedstudio.com/wiki/Seeeduino_Stalker_v2.3>. Acesso em: 2017-12-17.

SUNSOFT. *TCP/IP Network Administration Guide*. 2550 Garcia Avenue, Mountain View, CA 94043, USA: Sun Microsystems, 1994.

TAHAGHOGHI, S. M. S.; WILLIAMS, H. E. *Learning MySQL*. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O Reilly Media, Inc, 2007.

WEISER, M. Some computer science issues in ubiquitous computing. *Communications of ACM*, v. 36, n. 7, p. 74–83. In Special Issue, Computer–Augmented Environments, Jul 1993.

WELLING, L.; THOMSON, L. Php and mysql web development. In: _____. 5st. ed. Berkely, CA, USA: Addison-Wesley, 2016.

WEN, B. *6 things you should know about Node.js*. 2013. Disponível em: <https://www.javaworld.com/article/2079190/scripting-jvm-languages/6-things-you-should-know-about-node-js.html>. Acesso em: 2017-12-18.