UNIVERSIDADE FEDERAL DO MARANHÃO

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA DE ELETRICIDADE

CURSO DE ENGENHARIA ELÉTRICA

MARCOS AURÉLIO SAMINEZ DA SILVA

AFERIÇÃO E MONITORAMENTO REMOTO DE TEMPERATURA E UMIDADE EM DATACENTER.

SÃO LUÍS

2018

MARCOS AURÉLIO SAMINEZ DA SILVA

AFERIÇÃO E MONITORAMENTO REMOTO DE TEMPERATURA E UMIDADE EM DATACENTER.

Monografia apresentada ao Curso de Engenharia Elétrica da Universidade Federal do Maranhão como parte dos requisitos para obtenção do grau de Bacharel em Engenharia Elétrica.

Orientador: Dr. José de Ribamar Braga Pinheiro Junior

SÃO LUÍS

2018

Saminez da Silva, Marcos Aurélio
Aferição e monitoramento remoto de temperatura e umidade em datacenter / Marcos Aurélio Saminez da Silva. – São Luís, 2018.
50p.

Orientador: Dr. José de Ribamar Braga Pinheiro Junior

Monografia (Graduação em Engenharia Elétrica) – Universidade Federal do Maranhão, 2018.

1. Automação 2. Sensores 3. Ambiente 4. Arduino I. Título

CDU 621.3

MARCOS AURÉLIO SAMINEZ DA SILVA

AFERIÇÃO E MONITORAMENTO REMOTO DE TEMPERATURA E UMIDADE EM DATACENTER.

Monografia apresentada ao Curso de Engenharia Elétrica da Universidade Federal do Maranhão **como parte dos requisitos** para obtenção do grau de Bacharel em Engenharia Elétrica.

Aprovada em: 05 / 02 / 2018.

BANCA EXAMINADORA

Profº Dr. José de Ribamar Braga Pinheiro Junior (Orientador)

Doutor em Engenharia Elétrica

Universidade Federal do Maranhão

Profº Msc. Marcos Tadeu Rezende De Araujo

Mestre em Engenharia Elétrica

Universidade Federal do Maranhão

Profº José Roberto Quezada Pena

Especialista em Engenharia Elétrica

Universidade Federal do Maranhão

Aos meus pais,

Que me apoiaram todos estes anos.

AGRADECIMENTOS

Agradeço primeiramente a Deus sem o qual nada seria possível.

Agradeço a meus pais e minha irmã que sempre me apoiaram mesmo nas situações mais difíceis.

Agradeço a minha namorada Elis, que também cursou comigo este curso e dividiu comigo todas as dificuldades, sem ela não seria possível ter terminado esta jornada.

Aos meus professores pelos conhecimentos e experiências repassadas nesses anos de curso.

Ao meu orientador, que me ajudou na conclusão deste trabalho.

Aos meus colegas de curso que me ajudaram na conclusão desse curso.

Aos meus colegas de trabalho que sempre que puderam, me ajudaram na conclusão deste curso, em especial meu amigo Leônidas.

Agradeço a todos que me ajudaram nesta conquista.

"Os que se encantam com a prática sem a ciência são como os timoneiros que entram no navio sem timão nem bússola, nunca tendo certeza do seu destino."

Leonardo da Vinci

RESUMO

Neste trabalho é desenvolvida uma plataforma de monitoramento de condições de operação de equipamentos em ambiente de *datacenter*. Será utilizada a plataforma Arduino para desenvolvimento do protótipo e facilitação dos testes de projeto. Serão feitas breves introduções das tecnologias e arquiteturas utilizadas no desenvolvimento. Assim como, será apresentada a plataforma Arduino, o sistema Zabbix e os sensores utilizados. E, serão discutidas também, as metodologias para o desenvolvimento deste projeto.

Palavras-chave: *Monitoramento, Arduino, Condições ambientais, Datacenter*.

ABSTRACT

In this work, a platform is developed to monitor the operating conditions of equipment in a data center environment. The Arduino platform will be used to develop the prototype and facilitate design testing. Brief introductions of the technologies and architectures used in development will be made. As well as, will be presented the Arduino platform, the Zabbix system and the sensors used. And, will also be discussed, the methodologies for the development of this project.

Keywords: *Monitoring, Arduino, Environmental conditions, Datacenter*.

LISTA DE FIGURAS

LISTA DE SIGLAS

DHCP 32

Dynamic Host Configuration Protocol: Protocolo responsável pela configuração automática de interfaces de rede. Consiste em um equipamento enviando para todos da rede informações sobre como se configurar automaticamente.

EEPROM 21

Electrically-Erasable Programmable Read-Only Memory: Consiste em uma memória não-volátil apagável eletronicamente, possui um número alta grandeza para a quantidade de possíveis gravações.

PWM 21

Pulse Width Modulation: Modulação de largura de pulsos. Necessária para controlar velocidade e outros parâmetros em algumas aplicações. Consiste em alterar a largura dos pulsos de uma determinada onda quadrada conforme for necessário.

SPI 20, 23

Serial Peripheral Interface Bus: Barramento de comunicação serial na qual diversos dispositivos periféricos podem se interconectar respeitando a ordem de comunicação estabelecida pelo protocolo.

SRAM 21

Static Random-Access Memory: Memória do tipo volátil que ao ser desenergizada é apagada. Sua característica principal é a baixa latência e rapidez na escrita.

SUMÁRIO

Capítulo 1: Introdução

Datacenter, ou Centro de Processamento de Dados, é um ambiente projetado para abrigar equipamentos de processamento, armazenamento e sistemas ativos de redes como comutadores de tráfego e roteadores. Estes equipamentos são fundamentais para prover os serviços e atividades de diversos setores da economia.

Nestes ambientes, percebe-se que o aparato utilizado no controle dos parâmetros climáticos tem suma importância para o bom funcionamento e conservação dos equipamentos, possuindo, porém, preço elevado. Para fomentar a necessidade de crescimento dos datacenters brasileiros, é necessário reduzir os custos destes aparatos e melhorar a acessibilidade a equipamentos de uso essencial. Com isso, o desenvolvimento de plataformas que reduzam os custos, mantendo a qualidade necessária nestes ambientes, é importante.

No mercado atual encontramos diversos fabricantes que oferecem soluções para supervisionar a qualidade ambiental de datacenters. O problema encontrado é que além do preço alto, têm-se alto custo em instalação e em alguns casos, na operação.

Figura 1: Sensor de umidade e temperatura da fabricante APC

O sensor da Figura 1 do fabricante APC, por exemplo, faz parte de um pacote de sensoriamento ambiental e seu custo é agregado ao de uma central de monitoramento dos sensores e licenciamento do seu software, tornando assim o custo proibitivo para datacenters menores.

A importância do controle ambiental em data center se dá principalmente, pelas falhas ocasionadas por altas temperaturas e humidade. A falha de hardware, por exemplo, ocorre devido a superaquecimento e é um dos principais motivos de tempos de indisponibilidade em datacenters. E, como datacenters menores tendem a não usar sistemas de refrigeração de precisão devido ao custo elevado, os sistemas de monitoramento ambiental são vitais.

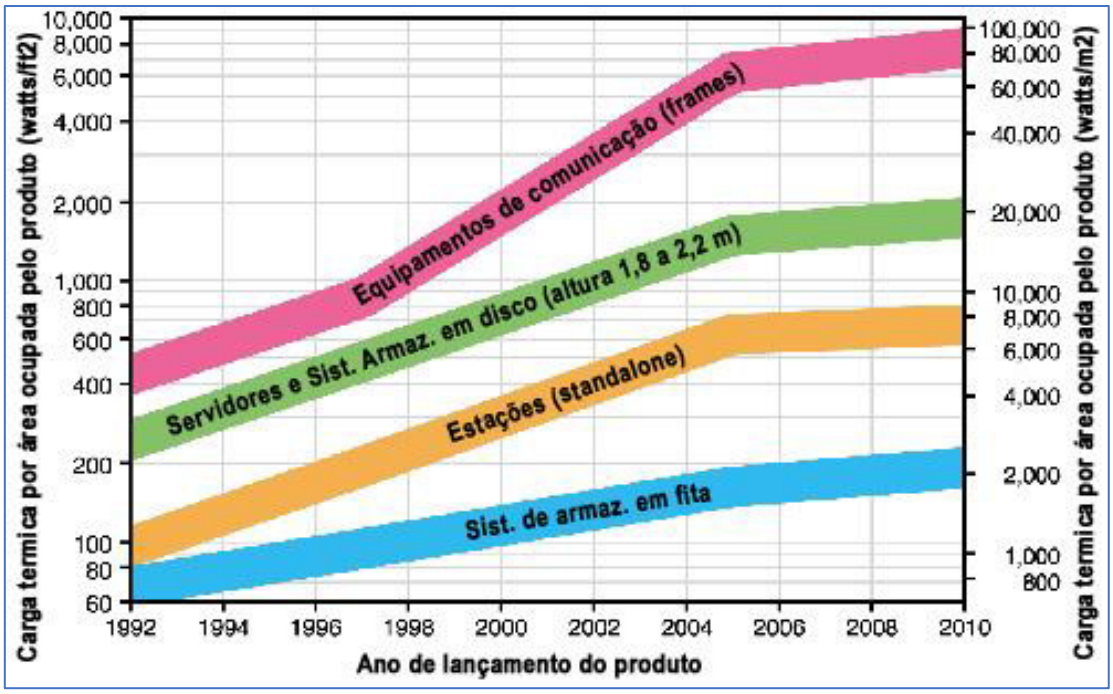

Figura 2: Crescimento da carga térmica em datacenters

Conforme ilustrado na Figura 2, pode-se notar o crescimento anual da carga térmica por área ocupada dos *datacenters*, de acordo com estudo da empresa APC [1]. Com este crescimento, a necessidade de monitoramento das condições ambientais torna-se ainda mais importante para manutenção do funcionamento dos equipamentos abrigados. As cores na figura representam o crescimento da carga térmica por categoria de equipamento, rosa para equipamentos de rede, verde para equipamentos de armazenamento de dados e servidores, laranja para estações de trabalho e azul para sistemas de armazenamento de dados de cópias de segurança em fitas.

No presente trabalho, será apresentado o desenvolvimento de um projeto que é uma plataforma alternativa de custo baixo para comtemplar a necessidade de monitoramento desses datacenters. O desenvolvimento do projeto será com base em componentes com custo acessível e fácil programação, como o Arduino [2].

Neste documento serão apresentados os detalhes referentes ao desenvolvimento e configuração necessários para o funcionamento do projeto. O capítulo 2 descreve a metodologia e os materiais utilizados. O capítulo 3 descreve a implementação dos códigos e a configuração necessária no ambiente de visualização de eventos Zabbix. O capítulo 4 descreve os resultados obtidos com o monitoramento. O capítulo 5 finaliza com as conclusões obtidas durante o desenvolvimento. Os anexos I e II contém os códigos utilizados no capítulo 3 para o Arduino e o programa de controle.

Capítulo 2: Metodologia

A metodologia deste trabalho foi baseada em uma pesquisa descritiva sobre o tema onde objetiva-se descrever as características e funções de cada um componente utilizado para o projeto proposto. Dessa forma, procurou-se demonstrar como essas partes funcionam e como podem se integrar para implementar o produto final.

A plataforma desenvolvida utilizará como base o Arduino que é formado por um circuito integrado Atmega328P [3]. O sensor utilizado é um DHT22 [4]. Para comunicação será utilizado o circuito integrado ENC28J60 [5]. E, para armazenamento e produção dos gráficos e alertas o Zabbix.

Um elemento importante para o desenvolvimento do projeto foi o desenvolvimento de um sistema computacional cujo objetivo seria receber os dados transmitidos pelo Arduino e enviá-los para o sistema de armazenamento. Inicialmente este sistema foi pensado e desenvolvido na linguagem *Python,* porém, a dificuldade em encontrar bibliotecas descomplicadas foi o gatilho para que o desenvolvimento tendesse para a linguagem C.

Para permitir a comunicação entre os elementos do projeto através da rede utilizou-se o JSON. O JSON é um formato de troca de dados entre sistemas independente de linguagem de programação derivado do JavaScript, mas a partir de 2017 muitas linguagens de programação incluíram código para gerar, analisar sintaticamente dados em formato JSON e também converter para objetos da linguagem [6] [7].

Referente ao software do Arduino, o desenvolvimento ocorreu através das funções da biblioteca *EtherCard.* O programa Wireshark [8] foi utilizado para monitorar o envio de pacotes durante o desenvolvimento. Com esta ferramenta é possível observar o conteúdo de todos os pacotes que trafegam na rede, sendo necessário apenas filtrar as buscas de pacotes de modo a se observar facilmente o que se deseja. Através deste, foi formatado o modelo de pacote a ser transmitido via *multicast.*

O próximo passo foi o desenvolvimento do esquema do circuito para implementação na placa de fenolite. Foi utilizado o software de prototipagem Proteus para esta tarefa. Este, foi utilizado para desenho e simulação do circuito, além do desenho da placa.

A seguir serão apresentados a arquitetura destes componentes, o desenvolvimento da plataforma e o sistema que receberá os dados.

2.1. Zabbix

Zabbix é um sistema de código aberto utilizado para monitorar e registrar os estados de serviços de rede, servidores e hardware de rede. [9] Para seu funcionamento é necessário que uma base de dados seja configurada para que as variáveis recebidas possam ser armazenadas. Seu código é desenvolvido em C e o sistema web é desenvolvido em PHP. Diversos protocolos de monitoramento são suportados. Ele possui um sistema de notificações robusto e extensível com utilização de programas externos.

Por ser uma plataforma fácil de configurar e bem versátil pode ser configurado para monitorar uma vasta gama de parâmetros. Nele configura-se os itens de coleta de dados com diversos protocolos suportados e nestes pode-se configurar alarmes com diferentes níveis. A Figura 3 apresenta o painel de monitoramento do Zabbix, onde é possível visualizar na lista os alarmes ativos em um sistema e a abaixo a tabela com os grupos de alertas configurados.

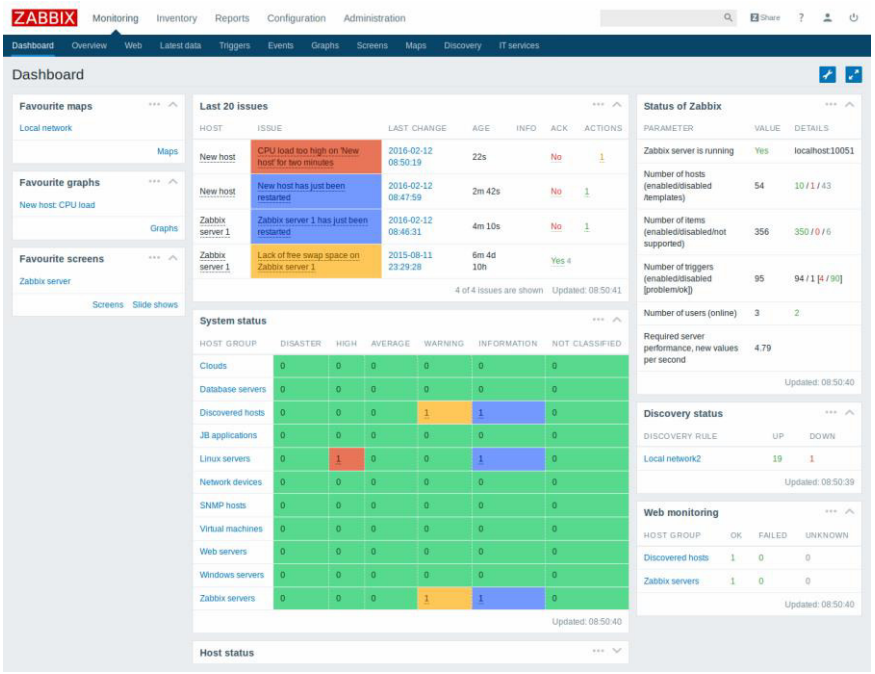

Figura 3: Painel principal de monitoramento no Zabbix

2.2. Arduino

O Arduino [2] é uma plataforma para prototipagem de hardware aberto e flexível que possibilita o uso de bibliotecas que estendem e facilitam suas funcionalidades. A placa utilizada neste projeto é o Arduino UNO [2], exibido na Figura 4 abaixo.

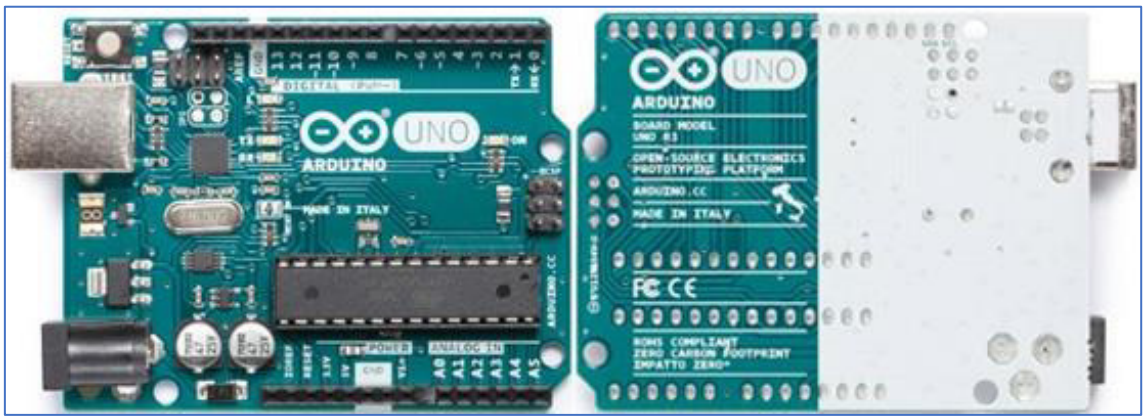

Figura 4: Placa de prototipagem Arduino UNO

Através desta placa, pode-se enviar e receber sinais digitais e analógicos através das entradas enumeradas. Com isso, vários tipos de sensores eletrônicos e atuadores podem ser conectados.

O microcontrolador Atmel ATmega328, presente no Arduino utilizado no projeto, utiliza arquitetura Harvard de 8 bits, possui 32 registradores para uso geral, 3 temporizadores, uma porta serial, uma porta para comunicação SPI. A tensão de operação é entre 1.8 e 5.5 V. [2] Mais informações retiradas da folha de especificações do microcontrolador são expostos na Tabela 1.

Microcontrolador	ATmega328P
Tensão de operação	5 V
Tensão de entrada recomendada	7-12 V
Tensão de entrada limite	6-20 V

Pinos digitais	14 (dos quais 6 provém saída PWM)
Pinos analógicos	6
Corrente contínua por pino	40 mA
Corrente contínua no pino 3.3V	50 mA
Memória *flash*	32 kB (31.5 kB disponíveis)
Memória SRAM	2 kB
Memória EEPROM	1 kB
Frequência	16 MHz

Tabela 1: Dados técnicos do Arduino

O software que controla o Arduino foi desenvolvido utilizando o programa Arduino IDE [10] que utiliza uma linguagem de programação baseada na linguagem C/C++. [2]

A sequência de programação é de inserir as definições de variáveis e bibliotecas inicialmente, em seguida a função *setup()* para configurar os parâmetros iniciais e por fim a função *loop()* que ficará rodando ciclicamente.

2.3. Sensor DHT22

O sensor utilizado para medir os parâmetros climáticos foi o DHT22. Este é um sensor digital de umidade relativa e temperatura de alta precisão, mostrado na Figura 5, do tipo capacitivo com as especificações exibidas na Tabela 2 a seguir.

Modelo	AM2302 / DHT22
Tensão de entrada	3.3-5.5 V
Sinal de saída	Sinal digital em 1 pino
Elemento sensitivo	Capacitor polimérico
Faixa de operação	Umidade relativa entre 0-100% e temperatura entre -40 e 80 °C
Precisão	Umidade: +-2% e temperatura: +-0.5 °C
Sensibilidade	Umidade: 0.1% e temperatura: 0.1 °C

Tabela 2: Especificação técnica do sensor DHT22 [4]

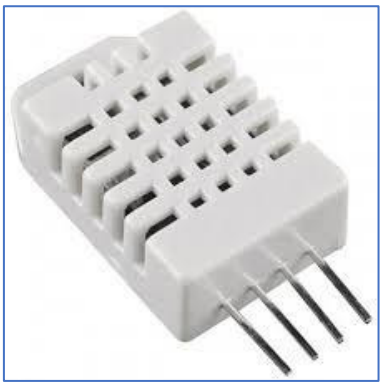

Figura 5: Sensor DHT22 [4]

Este sensor foi escolhido devido a sua simplicidade de operação, ótimo custo benefício relacionando preço x precisão das medições e facilidade de ser encontrado no mercado.

2.4. Circuito integrado ENC28J60

O ENC28J60 foi o controlador de rede escolhido para permitir a comunicação dos componentes na rede. Este controlador de rede *ethernet* comunica-se a 10 Mbps, padrão 10BASE-T. Possui memória de 8 Kb.

Uma vantagem deste controlador sobre os demais é permitir a configuração do seu endereço físico e o uso do protocolo de comunicação é o SPI (Serial Peripheral Interface)SPI é um protocolo que permite a comunicação com diversos outros componentes, formando uma rede. Ele é uma especificação de interface de comunicação série síncrona usada para comunicação em curta distância, em sistemas embarcados e afins. A interface do SPI foi desenvolvida pela empresa Motorola e tornou-se, através do seu uso frequente, um padrão de fato.to.

Os dispositivos SPI comunicam entre si em modo "*full duplex*" usando uma arquitetura "mestre-escravo" com um único mestre. O dispositivo mestre origina a comunicação para a leitura e a escrita. Múltiplos dispositivos escravos são suportados através da utilização de linhas de seleção de escravos individuais (SS).

Do ponto de vista elétrico, este componente utiliza uma tensão de operação entre 3.3-5.5V. Neste caso usaremos o módulo que contém o conector RJ45 e o controlador sendo necessário apenas conectar os pinos de 5V/GND e os quatro referentes ao protocolo SPI. É exibido na Figura 6 o módulo utilizado no projeto.

Figura 6: Módulo do ENC28J60 [5]

2.5. Protocolo *Multicast*

Multicast é uma forma de comunicação de equipamentos em rede onde a transmissão de dados é endereçada a um grupo de computadores simultaneamente, ao contrário da comunicação Unicast que a forma tradicional de comunicação onde uma mensagem é destinada somente a um endereço.

Para enviar ou receber informações no grupo, é necessário enviar um comando ao sistema operacional para este se registrar no grupo. Desta forma, computadores que se registram podem receber os dados direcionados ao grupo.

A utilização deste protocolo, resulta na redução dos parâmetros necessários para a transmissão dos dados. Visto que não é mais necessário saber à qual endereço se destinam os dados.

A escolha deste protocolo ocorreu devido à redução das configurações individuais em cada módulo Arduino, que resulta na simplificação do processo de implantação do sistema. Já que não é necessário configurar o endereço de envio dos dados em cada Arduino.

2.6. Formato JSON

É um formato aberto de arquivo que consiste da formação de pares de atributo-valor e matrizes. Sua principal vantagem é a facilidade de interpretação e leitura dos dados pelos programas que o recebem. É utilizado para transmissão de informações entre sistemas (variáveis).

Exemplo: “`{"id":2,"nome":"Teste","nota":"10"}`”.

Conforme exposto no exemplo acima, as variáveis id, nome e nota tem valores

atribuídos como dois, Teste e dez, respectivamente.

Em um outro exemplo, considerando a comunicação utilizada neste projeto:

```
{“ID”:”00001”,”T”:”31.39”,”U”:”96.30” }
```

No exemplo exposto acima, são definidas as variáveis “ID”, “T” e “U”. E, seus respectivos valores são 00001, 31.39 e 96.30.

2.7. Plataforma de Monitoramento

A plataforma que compõe o sistema de monitoramento desenvolvida ao longo do trabalho são a plataforma de transmissão e plataforma de recebimento. A primeira plataforma será responsável por coletar as informações dos sensores e transmitir para a rede. A segunda será responsável pelo recebimento das informações transmitidas e registro em banco de dados.

2.7.1. Plataforma de transmissão

Construído com a IDE do Arduino, o programa gravado na memória do microcontrolador faz a leitura do sinal do sensor e coordena a conexão à rede e o envio dos dados. O código utilizado no microcontrolador foi desenvolvido com base na biblioteca para o ENC28J60, chamada *EtherCard* [9]. Esta facilita a comunicação com a rede pois providencia funções de alto nível que tornam a configuração do módulo mais ágil. No anexo I apresenta-se a codificação detalhada da implementação deste software.

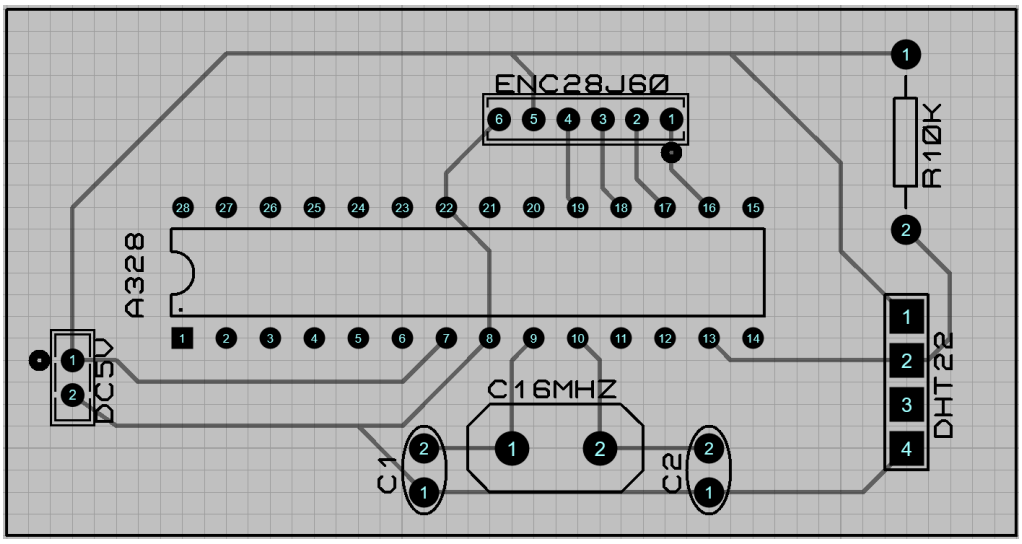

Figura 7: Circuito desenvolvido para a plataforma de transmissão.

Como prova de conceito, foi desenvolvido, em uma placa de fenolite na Figura 8, o circuito da Figura 7 acima. Este serviu para facilitar a visualização do produto e reduzir ainda mais os custos finais devido a substituição do Arduino completo por apenas seu microcontrolador.

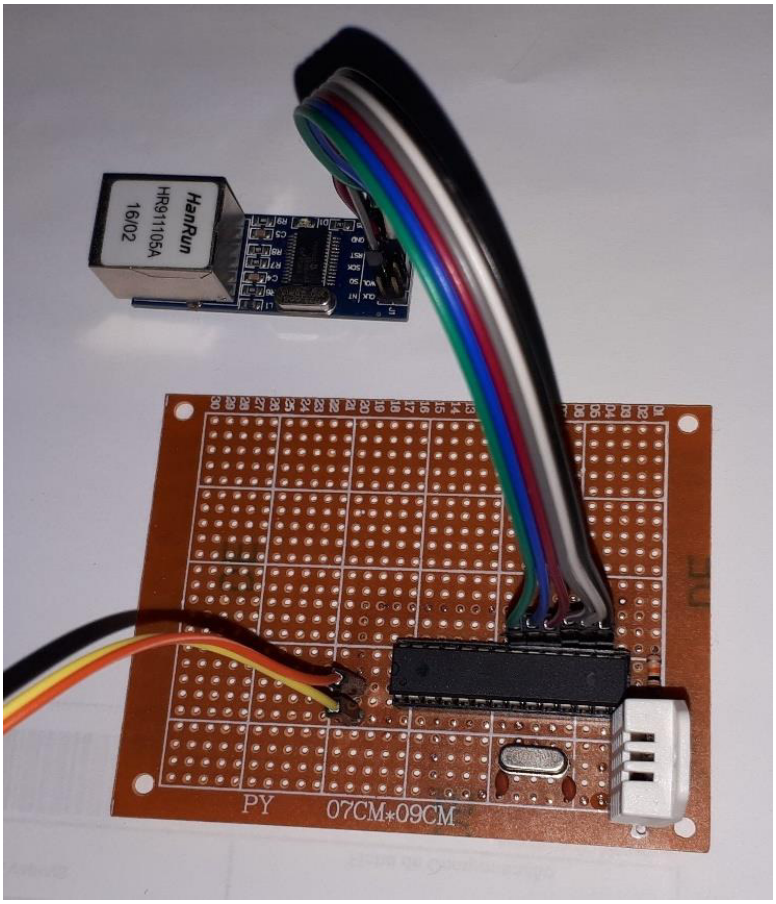

Figura 8: Placa de fenolite com o circuito montado

2.7.2. Plataforma de recebimento

A plataforma de recebimento foi desenvolvida em C e tem como função primordial a captura dos dados enviados via *multicast* e transmissão para o Zabbix. O algoritmo de *multicast* desenvolvido teve como base a função exemplo implementada pelo departamento de redes do Trinity College Dublin [10].

Nesta plataforma, a execução será efetuada na forma que se segue. A função *main()*, função principal que geralmente estrutura um programa desenvolvido na linguagem C, executa em loop e a cada iteração escuta a rede por pacotes de *multicast* na porta 4999. Caso receba um pacote este é repassado a função *json()* que decodifica os dados recebidos em 3 variáveis (id, temperatura e umidade). Estas, são enviadas para o Zabbix pela chamada do programa *Zabbix_Sender*, que é parte da distribuição oficial do Zabbix e é responsável por transmitir dados ativos para o Zabbix. Na Figura 9 podemos observar o fluxograma que define a sequência de funcionamento do programa e na Figura 10, a seguir, a saída com os dados recebidos e processados pelo sistema.

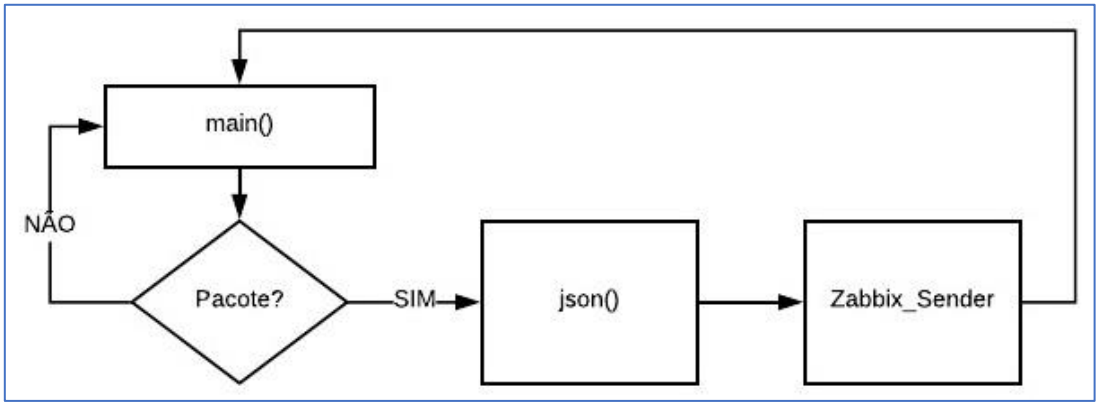

Figura 9: Fluxograma do programa desenvolvido

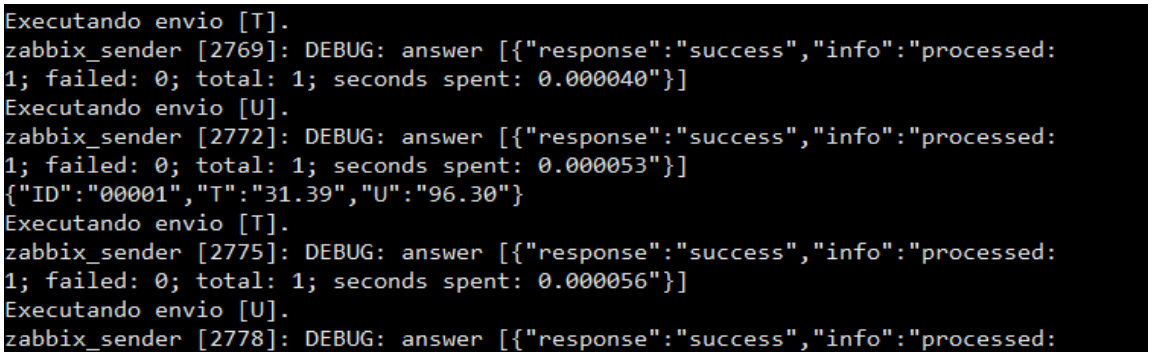

Figura 10: Programa recebendo os dados e enviando para o Zabbix

A variável ID é utilizada para identificar o Arduino que originou a mensagem e representa um número de série a ser cadastrado no Zabbix. As variáveis T e U trazem os valores lidos pelo sensor de temperatura e umidade, respectivamente.

No anexo II apresenta-se a codificação detalhada da implementação deste software.

Capítulo 3: Implementação

Neste capítulo, apresenta-se com mais detalhe a implementação de cada um dos módulos do sistema.

Na figura abaixo é exibido o fluxograma que demonstra o completo funcionamento do projeto, onde a linha vermelha representa a comunicação em rede.

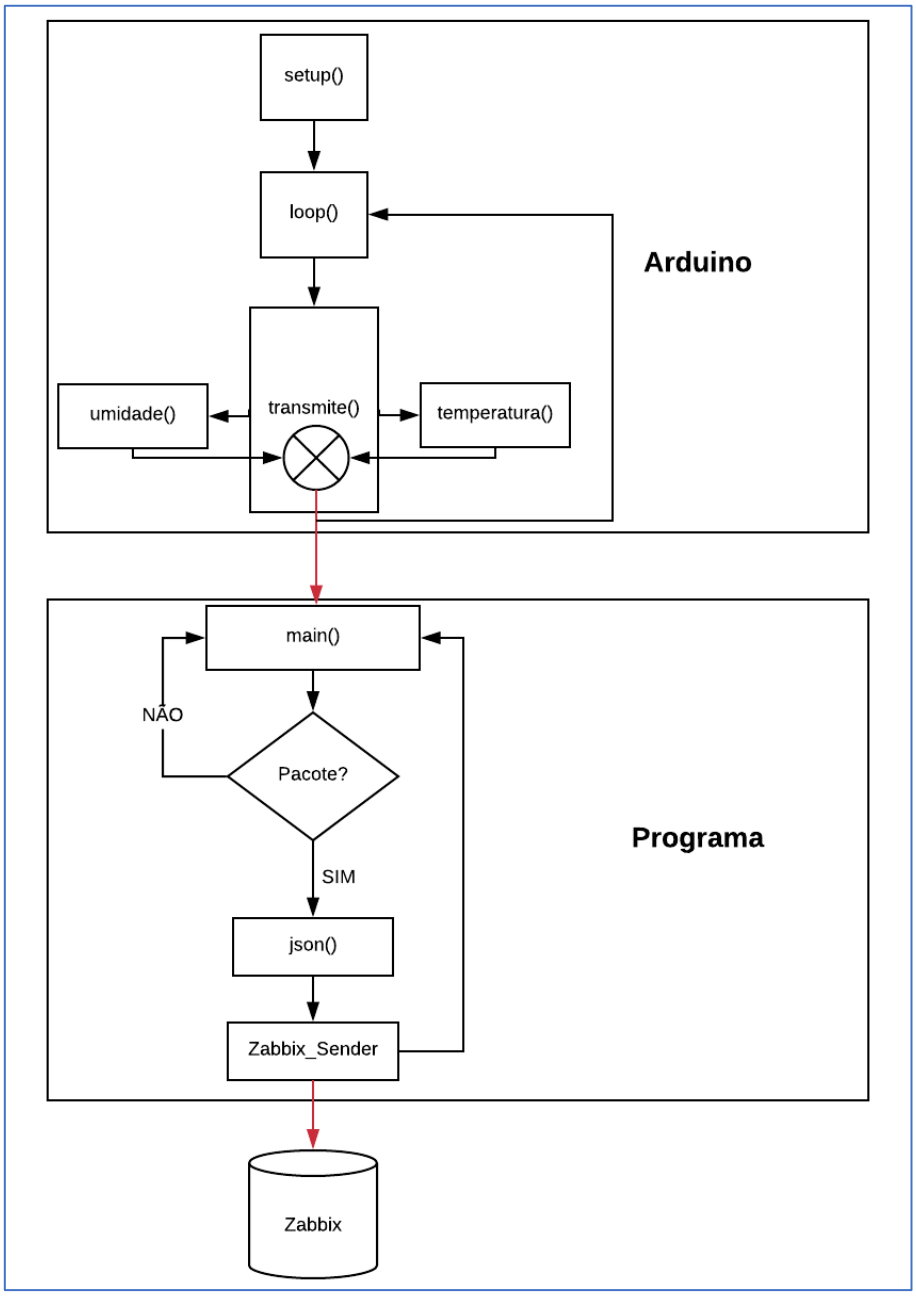

Figura 11: Fluxograma com o funcionamento geral do projeto

3.1. Software do Arduino

Software desenvolvido utilizando como ferramenta a IDE própria do projeto Arduino [2]. Utiliza linguagem C e conta com um arquivo fonte, que inclui as variáveis e funções necessárias para o projeto. Na primeira metade do fluxograma da Figura 11 é mostrada a sequência das funções aqui descritas.

Por padrão, o microcontrolador, ao inicializar executa duas funções. A primeira sendo a função *setup()* e em seguida, a função *loop()* executada em laço infinito.

Desta forma, a sequência lógica do programa é de inicialmente declarar as bibliotecas a serem carregadas em seguida as suas variáveis globais. Na sequência é definida a função *setup()* que contém as configurações dos parâmetros necessários para o funcionamento do sensor e do módulo ethernet. Neste ponto é iniciado o DHCP e executadas a função de preparação do DHT22 contidas em sua biblioteca.

Assim, a função *setup()* é responsável por configurar os parâmetros iniciais, pinos, entradas e saídas de informações e inicializar os módulos de sensor e comunicação com as seguintes etapas:

- Inicializar a porta *serial* e enviar informações sobre as etapas seguintes.
- Inicializar o módulo sensor
- Inicializar o módulo de rede.
- Executar o processo de configuração da rede via DHCP

Em seguida, são declaradas as funções de leitura temperatura() e umidade() para

recuperar do sensor DHT22 os valores respectivos. Além destas, é declarada também a função transmite(), esta executa as funções de leitura de umidade e temperatura e transmite os valores em formato JSON para a rede através do grupo de comunicação configurado via protocolo *multicast.* Neste modo, os dados enviados a rede são transmitidos a todos os participantes de rede que respondam pelo endereço especial atribuído ao grupo. Ou seja, é uma transmissão um-para-todos. Assim não é necessário configurar a comunicação entre o Arduino e o programa de recebimento, este último irá apenas aguardar o recebimento dos dados enviados.

Na função principal *loop()* são executadas as funções declaradas anteriormente, com exceção da *setup()* que só é executada na inicialização do microcontrolador. A função *loop()* possui um contador de segundos e executa a função transmite() a cada ciclo de 15 segundos e informa a execução desta via *serial*.

3.2. Software de recebimento

Programa desenvolvido em C para capturar os pacotes enviados à rede via *multicast* e interpretá-los de modo a decodificar o formato JSON em três variáveis, sendo elas temperatura, umidade e identificação do Arduino. Na segunda metade do fluxograma da Figura 11 é mostrada a sequência das funções aqui descritas.

O programa, ao inicializar executa a função *main()*, esta responsável por coordenar a comunicação do *kernel* do sistema operacional com a rede via *multicast*, e abrir a porta necessária para o recebimento dos pacotes. Quando um pacote é recebido, o conteúdo deste é enviado na integra via passagem de parâmetro para a função *json()* juntamente com o parâmetro IP, necessário para a execução da função *json()*.

A função *json*() é responsável por receber os parâmetros pacote e IP e tratá-los. O pacote é decodificado em três variáveis utilizando as funções providas pela biblioteca Tiny-Json [11]. Em seguida, estas variáveis são enviadas para o IP recebido como parâmetros do programa *Zabbix_Sender* que é utilizado para cada envio. As variáveis decodificadas são temperatura, umidade e ID (número identificador da placa do Arduino).

3.3. Configuração do Zabbix

As informações enviadas pelo Zabbix-Sender para o Zabbix são transmitidas via rede com protocolo próprio na porta 10050. Estas informações são então interpretadas e alocadas aos *hosts* e itens correspondentes conforme configuração descrita a seguir.

Primeiramente foi criado um padrão de configuração de *hosts* no qual os itens de temperatura e umidade foram criados e seus devidos gráficos e alertas associados também foram configurados. Com este padrão, podemos criar um *host* e associar o padrão de modo a vincular as configurações e alterações feitas. Desta forma, podemos transferir facilmente o padrão entre instalações do Zabbix sem ter que reconfigurar do zero. A seguir será exibido o procedimento exposto acima.

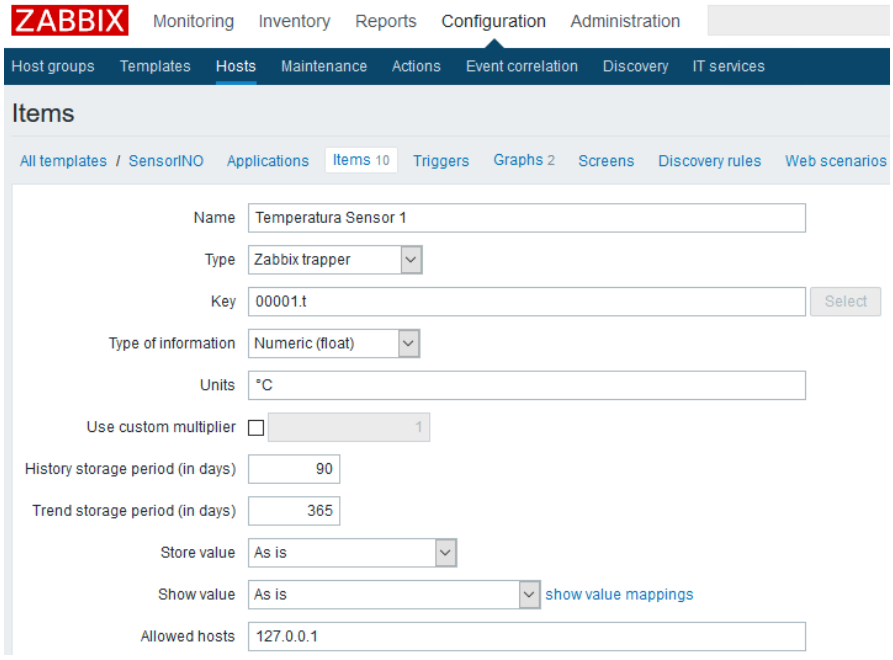

Figura 12: Criação do item temperatura no modelo

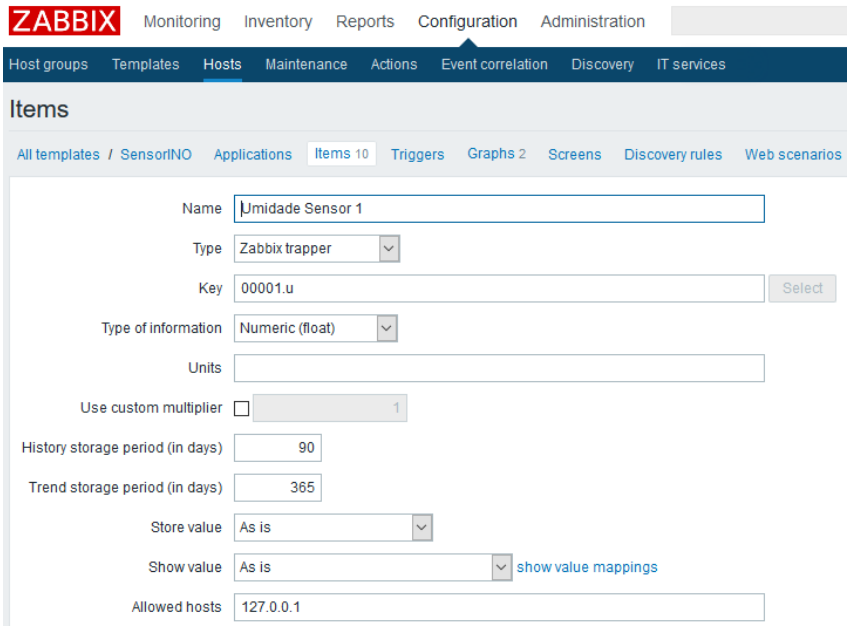

Figura 13: Criação do item umidade no modelo

A criação dos itens é demonstrada na Figura 12 e Figura 13. No campo *Key*, o número de série (variável ID) gravado no Arduino será colocado de modo a identificar os itens com a placa. Desta forma, os dados recebidos são armazenados no *host* certo de acordo com a ID atribuída.

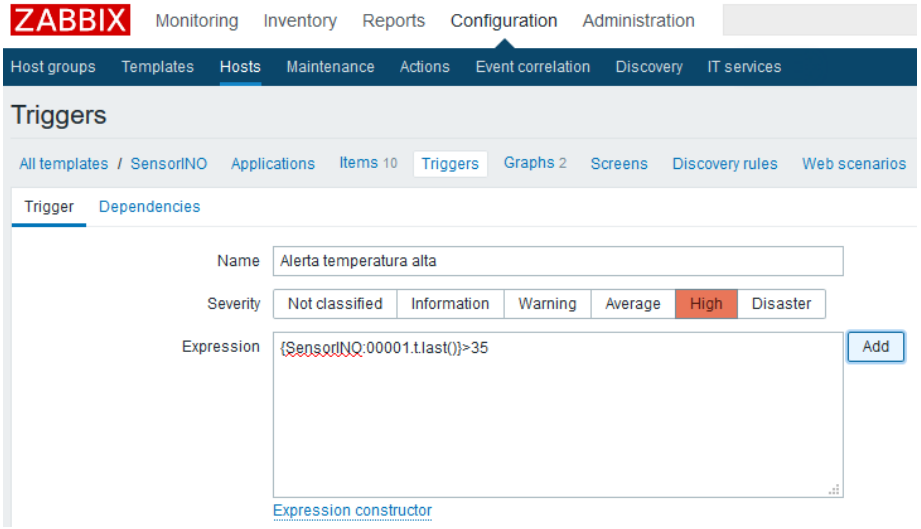

Figura 14: Criação do alarme de temperatura para o item

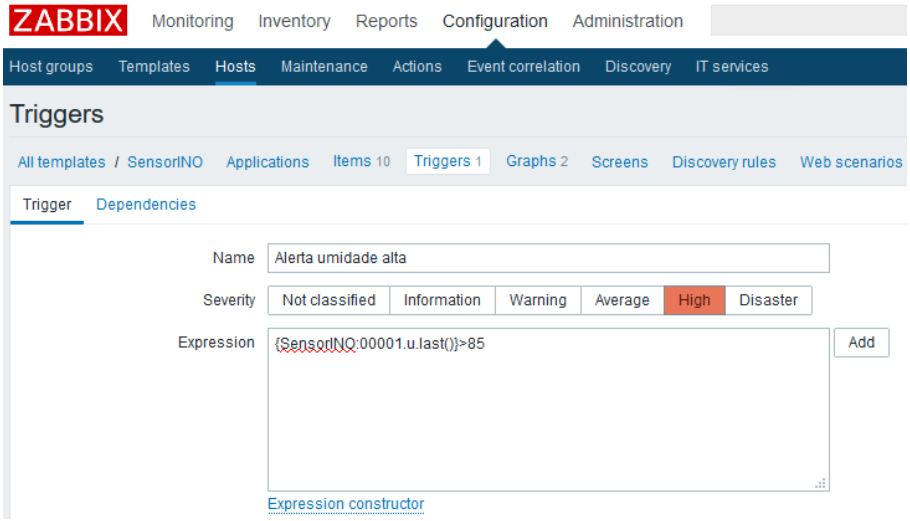

Figura 15: Criação do alarme de umidade para o item

Após a criação dos itens, deve-se criar os alarmes para cada um deles. De acordo com a configuração exposta nas figuras Figura 14 e Figura 15. Deve-se atentar para a expressão digitada, esta será a condição para disparo do alarme, neste caso acima de 35°C para temperatura e acima de 85% para umidade. O nível de severidade é escolhido acima da expressão e nos dois foi escolhido o nível Alto. Mais de um alarme pode ser configurado para o mesmo item de modo que se pode ter vários disparos em várias severidades.

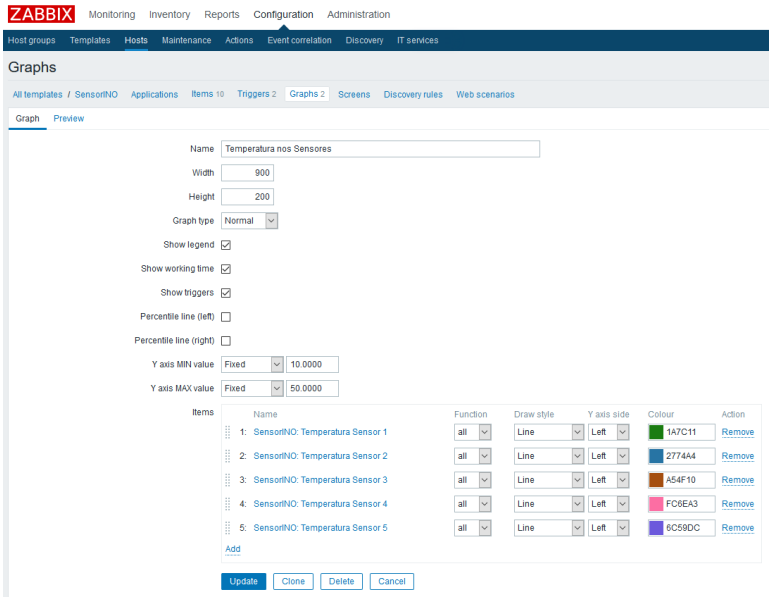

Figura 16: Criação do gráfico de temperatura dos sensores

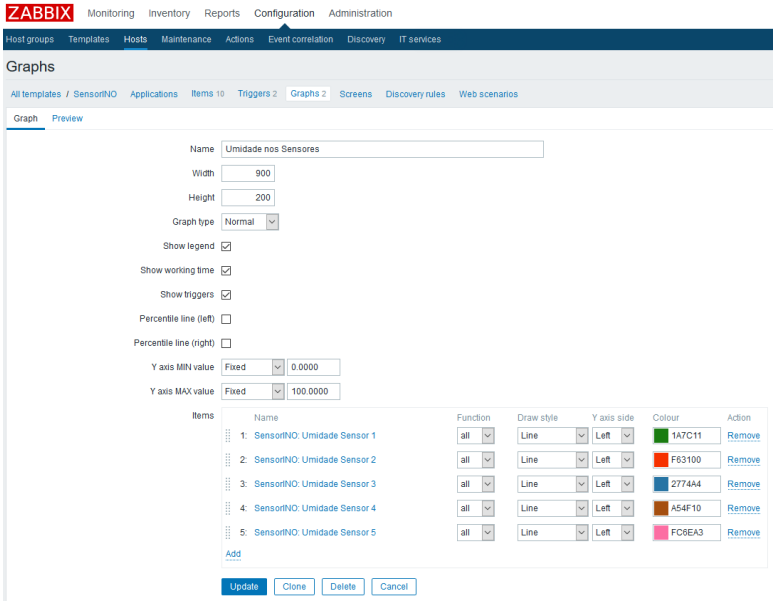

Figura 17: Criação do gráfico de umidade dos sensores

Na Figura 16 e Figura 17 acima foram criados os gráficos para os itens criados anteriormente. Neste passo apenas adiciona-se os itens um a um e configura-se os eixos e as cores do gráfico. Por preferência foram criados apenas dois gráficos agrupando itens semelhantes.

Capítulo 4: Resultados

O sensor foi testado durante cinco horas em ambiente residencial e sem refrigeração e os resultados foram registrados no Zabbix. Pode se observar nos gráficos a seguir os valores registrados e seus devidos horários. O alarme configurado aparece no gráfico como uma linha tracejada em vermelho. Quando esta é cruzada, o alarme dispara.

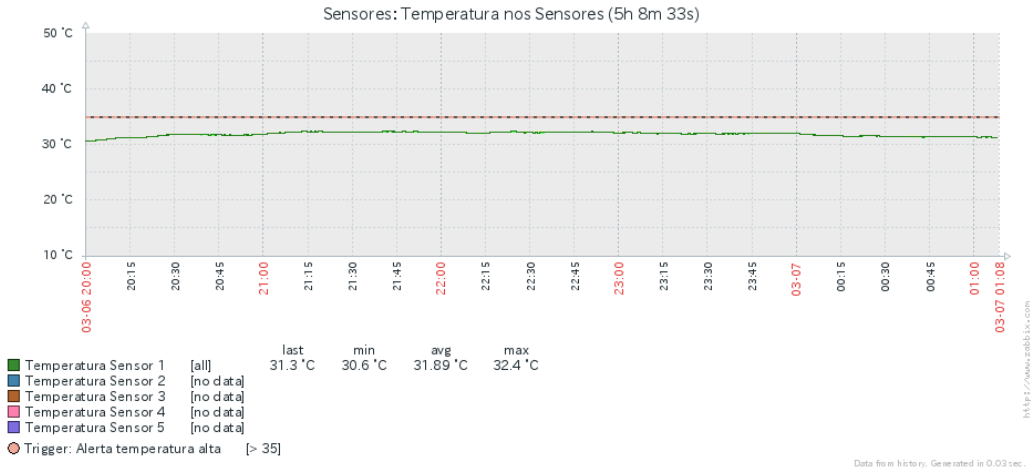

Figura 18: Gráfico de temperatura nos sensores exibindo limiar de alerta

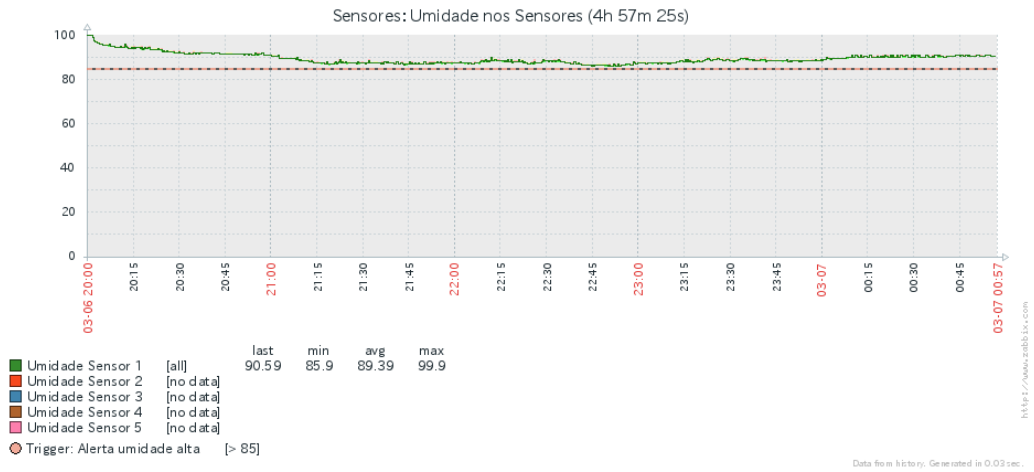

Figura 19: Gráfico de umidade nos sensores exibindo limiar de alerta

Foram capturados os pacotes da rede via o programa Wireshark [8] conforme exposto na figura a seguir. Cada linha da imagem é um pacote recebido e na seção abaixo é exibido o conteúdo do pacote. Pode-se observar na seção *Text* o conteúdo do pacote em formato JSON.

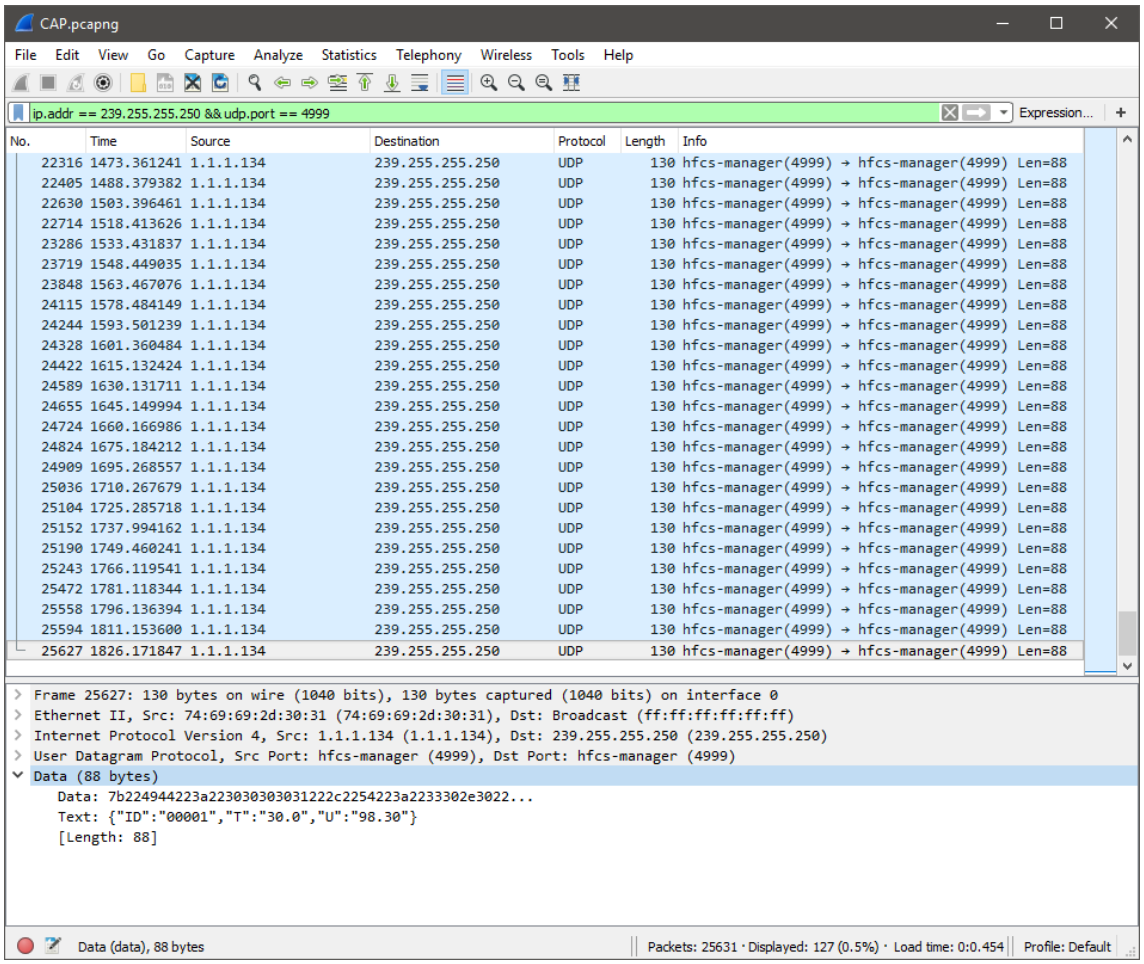

Figura 20: Wireshark listando os pacotes capturados

Durante o teste, o sistema se comportou de acordo com o esperado. O circuito de transmissão operou sem a ocorrência de erros e o software de recebimento entregou todos os resultados recebidos da rede. As medições de temperatura foram comparadas com as leituras de um multímetro da marca Minipa modelo ET-1649 e não foi encontrada diferença maior que 0.01% entre os valores. Com isso, foi possível ao Zabbix gerar os gráficos corretamente de todas as leituras. E, caso estas ultrapassassem os limites de temperatura configurados, os alarmes dispariam.

Capítulo 5: Conclusão

Este trabalho procurou apresentar inicialmente uma solução para o problema de monitoramento ambiental de datacenters, para isso, foi projetada e desenvolvida uma plataforma de monitoramento utilizando microcontrolador Arduino que integra monitoramento remoto de temperatura e umidade e opera com sensores de forma modular.

A construção dessa plataforma foi um grande desafio e com bons resultados, após a sua implementação. A escolha do protocolo *multicast* mostrou-se uma decisão importante já que a transmissão dos dados por meio deste é bastante eficaz por não depender de comunicação bidirecional, desta forma a transmissão e a recepção de informações dos módulos foi bastante simplificada.

O objetivo deste projeto foi atingido já que o produto deste trabalho é uma possibilidade viável para implementação e pode ajudar na fomentação do setor de tecnologia da informação brasileiro. Realizar este projeto e desenvolver novas tecnologias é parte do papel desempenhado pelo engenheiro eletricista, e expõe o conhecimento obtido durante o curso.

5.1 Trabalhos futuros

Com a conclusão do projeto e acredita-se que algumas melhorias futuras podem ser implementadas. Como mais sensores sendo integrados ao projeto, adaptação do código do software de recebimento para o sistema operacional Windows e integração com outros sistemas de monitoramento. Estas melhorias estenderiam as funcionalidades do projeto e aumentariam o alcance de público alvo atendido.

E, futuramente, pode-se pensar inclusive em formas de introduzir no mercado este sistema como um produto a ser comercializado.

Anexo I – Código do Arduino

```
#include <EtherCard.h>
#include <DHT.h>

#define DHTPIN 7     // pino em que o DHT22 está conectado
#define DHTTYPE DHT22   // DHT 22  (AM2302)
DHT dht(DHTPIN, DHTTYPE); // Inicializa o DHT para 16MHz
float temp;
float hum;
const char id[] = "00001"; //”Número de série”
static byte mip[] = { 239,255,255,250 };
static byte mymac[] = { 0x74,0x69,0x69,0x2D,0x30,0x31 };
char mac[] = "7469692d3031";
byte Ethernet::buffer[750];
BufferFiller bfill;
unsigned long timer=9999;

void setup() {
  Serial.begin(9600);
  Serial.println(F("Iniciando..."));
  dht.begin();
  if (ether.begin(sizeof Ethernet::buffer, mymac, SS) == 0)
    Serial.println(F("Falha ao acessar o controlador de rede!"));
  Serial.println(F("Iniciando DHCP..."));
  if (!ether.dhcpSetup())
    Serial.println(F("DHCP falhou..."));
  ether.printIp("IP: ", ether.myip);
  ether.printIp("Mascara: ", ether.netmask);
  ether.printIp("Gateway: ", ether.gwip);
  ether.printIp("DNS: ", ether.dnsip);
  ENC28J60::disableMulticast();
}

char temperatura() {
  temp = dht.readTemperature();
  return temp;
}

char umidade() {
  hum  = dht.readHumidity();
  return hum;
}

void transmite() {
    char pacote[89];
```

```
    int t = temperatura();
    int t2 = (temp - t) * 100;
    int u = umidade();
    int u2 = (hum - u) * 100;
sprintf(pacote,"%s%s%s%i%s%i%s%i%s%i%s","{\"ID\":\"",id,"\",\"T\":\"",t,".",t2
,"\",\"U\":\"",u,".",u2,"\"}");
    Serial.print(pacote);
    int udppos = UDP_DATA_P;
    EtherCard::udpPrepare(4999,mip,4999);
    memcpy(Ethernet::buffer + udppos, pacote, sizeof pacote);
    udppos = udppos  + sizeof pacote-1;
    EtherCard::udpTransmit(udppos-UDP_DATA_P);
    //addip(udppos);
}

void loop() {
    if (((millis()-timer)>15000)||(timer>millis())) {
      timer=millis();
      Serial.print("\r\n\r\nANUNCIANDO.\r\n");
      transmite();
    }
}
```

Anexo II – Código do software de recebimento

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tiny-json-master/tiny-json.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <errno.h>

#define HELLO_PORT 4999
#define HELLO_GROUP "239.255.255.250"
#define MSGBUFSIZE 256

/* Parser a json string. */
int json( char str[], char arg1[] ) {
    //char str[] = "{\"ID\":\"00001\",\"T\":\"30.0\",\"U\":\"98.30\"}";
       char comando1[100] = "";
       char comando2[100] = "";
       FILE *p;
    char ch;
    puts( str );
    json_t mem[32];
    json_t const* json = json_create( str, mem, sizeof mem / sizeof *mem );
    if ( !json ) {
        puts("Erro ao importar o json.");
        return EXIT_FAILURE;
    }
       if ( !arg1 ) {
             puts("IP incorreto.");
             return EXIT_FAILURE;
       }

    json_t const* ID = json_getProperty( json, "ID" );
    if ( !ID || JSON_TEXT != json_getType( ID ) ) {
        puts("Erro, json incorreto. [1]");
        return EXIT_FAILURE;
    }
    char const* IDVal = json_getValue( ID );
       if ( !IDVal ) {
        puts("Erro, ID não encontrado.");
        return EXIT_FAILURE;
    }
```

```
   json_t const* T = json_getProperty( json, "T" );
if ( !T || JSON_TEXT != json_getType( T ) ) {
    puts("Erro, json incorreto. [2]");
    return EXIT_FAILURE;
}
char const* TVal = json_getPropertyValue( json, "T" );
if ( !TVal ) {
    puts("Erro, T não encontrado.");
    return EXIT_FAILURE;
}
   json_t const* U = json_getProperty( json, "U" );
if ( !U || JSON_TEXT != json_getType( U ) ) {
    puts("Erro, json incorreto. [3]");
    return EXIT_FAILURE;
}
char const* UVal = json_getPropertyValue( json, "U" );
      if ( !UVal ) {
    puts("Erro, U não encontrado.");
    return EXIT_FAILURE;
}

strcat(comando1,"zabbix_sender -z ");
strcat(comando1, arg1);
strcat(comando1, " -vv -p 10051 -s \"Sensores\" -k ");
strcat(comando1, IDVal);
strcat(comando1, ".t -o ");
strcat(comando1, TVal);

p=popen(comando1,"r");
if (!p) {
         puts("Erro ao enviar [T].");
         return EXIT_FAILURE;
   }
   else {
         puts("Executando envio [T].");
   }
   while((ch=fgetc(p)) != EOF) {
         //printf("%c", ch);
   }
   pclose(p);

   strcat(comando2,"zabbix_sender -z ");
   strcat(comando2, arg1);
   strcat(comando2, " -vv -p 10051 -s \"Sensores\" -k ");
   strcat(comando2, IDVal);
   strcat(comando2, ".u -o ");
```

```
	strcat(comando2, UVal);

	p=popen(comando2,"r");
	if (!p) {
		puts("Erro ao enviar [U].");
		return EXIT_FAILURE;;
	}
	else {
		puts("Executando envio [U].");
	}
	while((ch=fgetc(p)) != EOF) {
		//printf("%c", ch);
	}
	pclose(p);

	return EXIT_SUCCESS;
}


int main(int argc, char *argv[])
{
	struct sockaddr_in addr;
	int fd, nbytes;
	u_int addrlen;
	struct ip_mreq mreq;
	char msgbuf[MSGBUFSIZE];
	u_int yes=1;

	if (!argv[1]) { puts("Falta o endereço IP do Zabbix!");
	  return EXIT_FAILURE;
	}

	if ((fd=socket(AF_INET,SOCK_DGRAM,0)) < 0) {
	return EXIT_FAILURE;
	}

	if (setsockopt(fd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(yes))<0){
		return EXIT_FAILURE;
	}

	memset(&addr,0,sizeof(addr));
	addr.sin_family=AF_INET;
	addr.sin_addr.s_addr=htonl(INADDR_ANY);
	addr.sin_port=htons(HELLO_PORT);

	if (bind(fd,(struct sockaddr *) &addr,sizeof(addr)) < 0) {
```

```
            return EXIT_FAILURE;
        }

        mreq.imr_multiaddr.s_addr=inet_addr(HELLO_GROUP);
        mreq.imr_interface.s_addr=htonl(INADDR_ANY);
        if(setsockopt(fd,IPPROTO_IP,IP_ADD_MEMBERSHIP,&mreq,sizeof(mreq))<0){
            return EXIT_FAILURE;
        }

        while (1) {
            addrlen=sizeof(addr);
            if((nbytes=recvfrom(fd,msgbuf,MSGBUFSIZE,0,(structsockaddr*)&addr
        ,&addrlen))<0){
            return EXIT_FAILURE;
            }
        json(msgbuf, argv[1]);
        }
}
```

Referências

[1] APC, “A Temperatura De Sua Sala de Servidores é Adequada e Controlada?,” APC, [Online].
Disponível em: http://www.apc.com/solutions/display.cfm?id=9B0D4608-3B9B-4C4B-BB198BE92BB28D96&ISOCountryCode=BR.
[Acesso em 03 fev 2018].

[2] “Arduino,” [Online].
Disponível em: https://www.arduino.cc.
[Acesso em 25 out 2017].

[3] “Atmega328P,” [Online].
Disponível em: http://www.microchip.com/wwwproducts/en/ATmega328p.
[Acesso em 22 dez 2017].

[4] “DHT22,” [Online].
Disponível em: https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf.
[Acesso em 27 jun 2017].

[5] “ENC28J60,” [Online].
Disponível em: https://www.microchip.com/wwwproducts/en/en022889.
[Acesso em 22 dez 2017].

[6] “ECMA-404 The JSON Data Interchange Standard.,” [Online].
Disponível em: http://www.json.org.
[Acesso em 01 05 2018].

[7] “Network Working Group,” [Online].
Disponível em: http://www.ietf.org.
[Acesso em 01 maio 2018].

[8] “WIRESHARK,” [Online].
Disponível em: https://www.wireshark.org.
[Acesso em 25 out 2017].

[9] “Zabbix,” [Online].
Disponível em: https://www.zabbix.com/.
[Acesso em 21 dez 2017].

[10] “Arduino IDE,” [Online]. Disponível em: https://www.arduino.cc/en/Main/Software.
[Acesso em 25 out 2017].

[11] “Biblioteca de rede EtherCard,” [Online].

Disponível em: https://github.com/jcw/ethercard.
[Acesso em 20 jul 2017].

[12] “NTRG, Trinity College Dublin,” [Online].
Disponível em: http://ntrg.cs.tcd.ie/.
[Acesso em 02 ago 2017].

[13] rafagafe, “Tiny-Json,” [Online].
Disponível em: https://github.com/rafagafe/tiny-json.
[Acesso em 21 ago 2017].

[14] J. F. KUROSE e K. W. ROSS, Redes de computadores e a internet: uma abordagem top-down., 3 ed., São Paulo, SP: Pearson Addison-Wesley, 2006.