CENTRO UNIVERSITÁRIO FADERGS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
MICROCONTROLADORES

MONITOR DE TEMPERATURA COM SNMP

Gustavo Conforti
Marcelo Jordano C. Menezes
Ronaldo Pinto

Porto Alegre, 2019

1. DEFINIÇÃO DO PROBLEMA

Fornecer uma solução para monitoramento de temperatura de ambientes, como data centers, CPDs, depósitos de matéria prima, etc. Além do desenvolvimento de captura de temperatura através da plataforma Arduino, o projeto contará com o envio de informações para dispositivos externos.

O projeto surgiu de uma oportunidade comercial na empresa de um dos integrantes. É uma companhia de gerenciamento e implementação de redes, segurança e virtualização para terceiros. A ideia é criar um dispositivo que possibilite a monitoria remota da temperatura do data center do cliente 24/7, como serviço; o termômetro irá enviar dados para as ferramentas de monitoramento da empresa. Cada técnico levará um desses consigo e será bonificado com uma porcentagem de cada implementação.

Atualmente a empresa não disponibiliza controle de temperatura para os clientes, logo seria uma adição ao seu portfólio de produtos.

2. OBJETIVOS

Proporcionar uma opção barata de geração de dados temperatura, para alimentar ferramentas externas de monitoramento de recursos. Com isto, será possível a configuração de *thresholds* máximos e mínimos e o envio de alertas mediante a ultrapassagem dos mesmos, uma vez os valores lidos pelo dispositivo poderão ser interpretados por softwares terceiros.

O projeto será baseado em três pilares principais. Para hardware, utilizaremos o módulo NodeMCU e o sensor de temperatura DHT11; eles serão responsáveis pela coleta e conversão dos dados climáticos do ambiente. O protocolo SNMP será o encarregado de transmitir essas informações para outros dispositivos em rede.

Como prova de conceito, usaremos uma VM do Zabbix para visualizar os dados gerados pelo termômetro.

3. ANÁLISE DAS TECNOLOGIAS

3.1 NodeMCU

O NodeMCU é uma plataforma *open source* com o microcontrolador chinês ESP8266. Possui funcionalidade *Wi-Fi* integrada, por isso muitas vezes é chamado comumente de "módulo *Wi-Fi".*

O ESP8266 pode funcionar no modo *standalone*, ou seja, sendo a MCU do projeto. É possível também utilizá-lo como escravo com interface em outra controladora central do projeto.

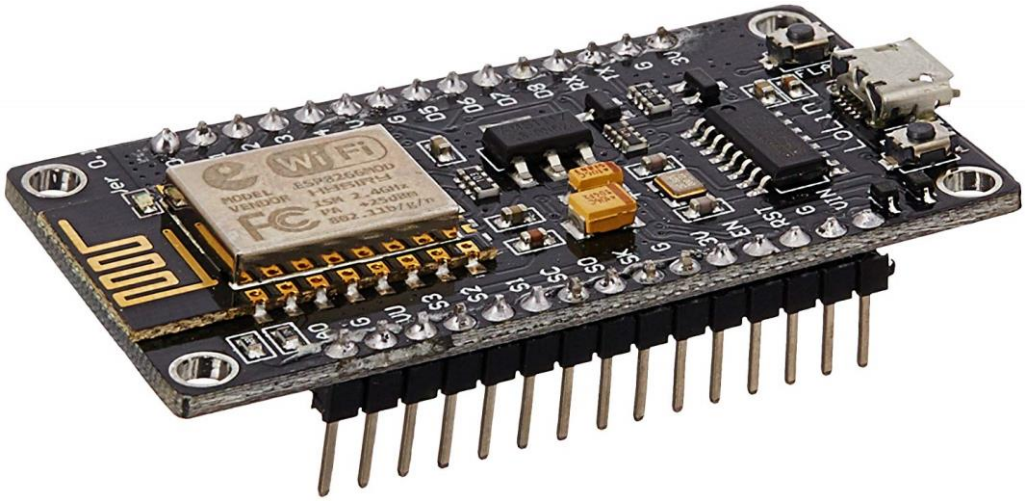

Figura 1 - NodeMCU

Algumas das características do NodeMCU podem ser vistas abaixo:

- Microcontrolador ESP8266-12E;
- Arquitetura RISC de 32 bits;
- Processador pode operar em 80MHz / 160MHz;
- 4Mb de memória flash;
- 64Kb para instruções;
- 96Kb para dados;
- WiFi nativo padrão 802.11b/g/n;
- Opera em modo AP, Station ou AP + Station;
- Pode ser alimentada com 5VDC através do conecto micro USB– Possui 11 pinos digitais;
- Possui 1 pino analógico com resolução de 10 bits

- Pinos digitais, exceto o D0 possuem interrupção, PWM, I2C e one wire;
- Pinos operam em nível lógico de 3.3V
- Pinos não tolerantes a 5V;
- Possui conversor USB Serial integrado;
- Programável via USB ou WiFi (OTA);
- Compatível com a IDE do Arduino ou programável na linguagem de programação desenvolvida por brasileiros, a LUA;
- Compatível com módulos e sensores utilizados no Arduino;

3.2 Sensor DHT11

O DHT11 é um sensor de temperatura e umidade complexo, com saída de sinal digital calibrada. Contém medição de umidade de tipo resistivo e um componente de medição de temperatura NTC, conectado a um microcontrolador interno de 8 bits de alto desempenho. Este sensor é altamente utilizado em projetos de automação e provas de conceito de produtos. Possui uma resposta e precisão satisfatória, que aliada com o bom custo benefício o torna uma das principais opções para projetos simples de automação.

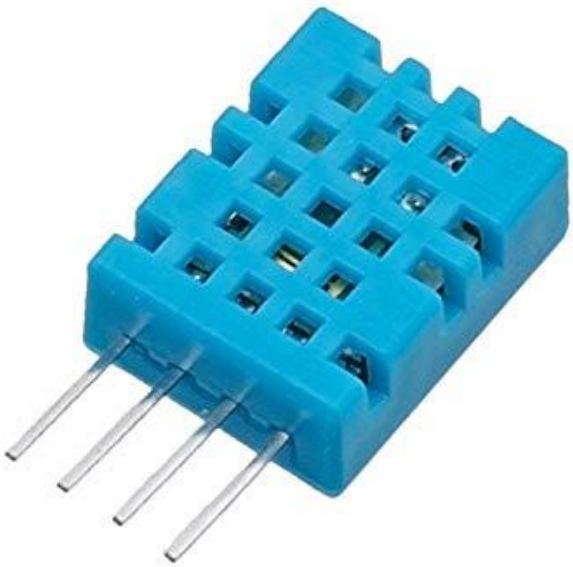

Figura 2 - Sensor de temperatura e umidade DHT11

- Especificações técnicas:
- Faixa de medição de umidade: 20 a 90% UR;
- Faixa de medição de temperatura: 0º a 50ºC;

- Alimentação: 3-5VDC (5,5VDC máximo);
- Corrente: 200uA a 500mA, em stand by de 100uA a 150 uA;
- Precisão de umidade de medição: ± 5,0% UR;
- Precisão de medição de temperatura: ± 2.0 ºC;
- Tempo de resposta: 2s;
- Dimensões: 23 x 12 x 5mm (incluindo terminais);

3.3 Protocolo SNMP

O Simple Network Management Protocol (SNMP) teve sua origem na RFC 1067 em 1988, evoluindo por algumas versões, estando atualmente na versão 3. É um protocolo da camada 7, que utiliza usualmente a porta 161 do protocolo 17.

É baseado no modelo gerente – agente; a estação de gerenciamento se comporta como cliente e o dispositivo de rede a ser monitorado se comporta como servidor, enquanto que na operação TRAP ocorre o oposto, pois é o dispositivo gerenciado que inicia a comunicação. Como as tarefas mais complexas de processamento e armazenamento de dados ficam com o gerente, o protocolo requer pouco processamento e pouco software, tornando sua adoção extremamente barata.

O protocolo, usado na maioria das vezes em sistemas de gerenciamento de dispositivos ligados em rede, é composto por dois objetos fundamentais: MIB e OID

3.3.1 MIB

Estrutura em árvore padronizada que contém os objetos gerenciáveis (OIDs) de um determinado dispositivo de rede. Essa estrutura não tem limites e, de acordo com a necessidade, pode ser atualizada e expandida. Exemplo:

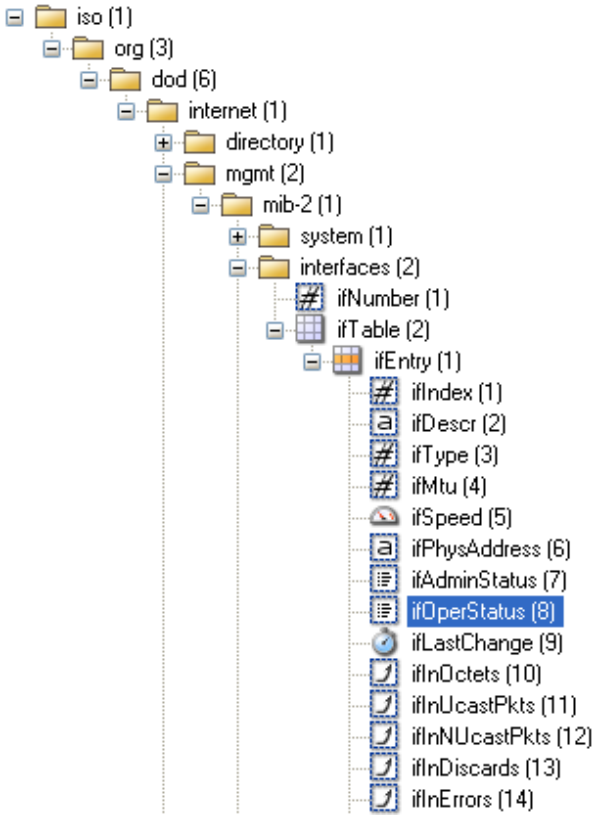

Figura 3 - Exemplo de MIB

3.3.2 OID

Uma variável que de fato guarda uma informação do sistema, com uma identificação única denominada (*Object IDentification*), que é composta por uma seqüência de números que identifica a posição do objeto na árvore da MIB.

Exemplo:

```
iso.3.6.1.2.1.17.4.0 = STRING: Comparitech
iso.3.6.1.2.1.17.4.1 = STRING: APC-3425
iso.3.6.1.2.1.17.4.2 = STRING: 3425EDISON
iso.3.6.1.2.1.17.4.4 = INTEGER: 72
iso.3.6.1.2.1.17.4.5 = STRING: veya
```

Ou, resolvendo seus nomes:

```
SNMPv2-MIB::sysContact.0 = STRING: Comparitech
SNMPv2-MIB::sysName.0 = STRING: APC-3425
SNMPv2-MIB::sysLocation.0 = STRING: 3425EDISON
SNMPv2-MIB::sysServices.0 = INTEGER: 72
IF-MIB:: ifDescr.1 =STRING: veya
```

3.3.3 ZABBIX

Uma ferramenta de monitoramento de dispositivos e aplicações em rede. Mesmo sendo totalmente *open-source*, é altamente adotada no ambiente corporativo, devido a sua facilidade de implementação e vasta documentação; o grande número de usuários culmina em um grande número de casos de uso e *templates*, diminuindo o tempo de configuração e troubleshooting de novos monitores ou dashboards. Além disso, o protocolo aberto possibilita uma fácil integração com outras ferramentas, como o Grafana, para o desenvolvimento dashboards complexos, o ELK Stack, para centralização, armazenamento e correlacionamento de eventos, entre outros. A seguir, pode ser visualizado um exemplo de *dashboard* do sistema Zabbix.

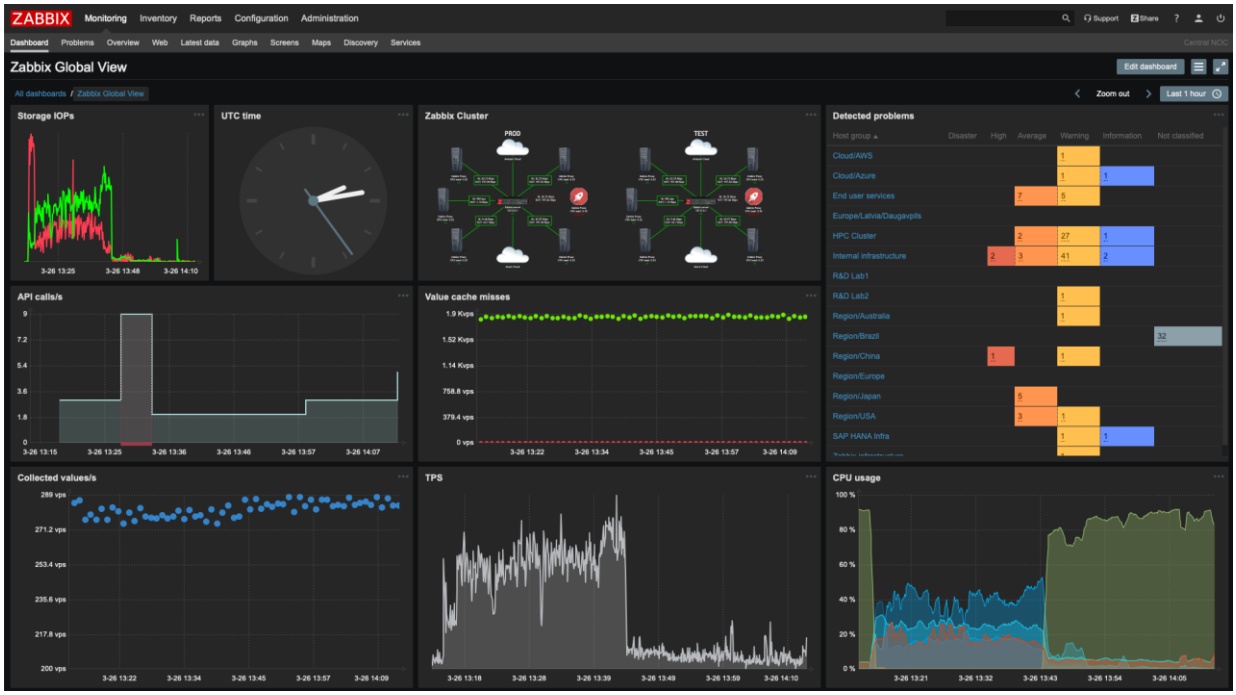

Figura 4 - Exemplo de *dashboard* do Zabbix

4. DESCRIÇÃO DA SOLUÇÃO

O armazenamento das informações se dará através de uma pequena estrutura SNMPv2 estática dentro da memóra do NodeMCU contendo os valores de umidade, temperatura, descrição do sistema, nome do sistema, localidade e tempo online do sistema. As OIDs são as seguintes:

sysDescr	1.3.6.1.2.1.1.1.0
sysUpTime	1.3.6.1.2.1.1.2.0
sysName	1.3.6.1.2.1.1.3.0
sysLocation	1.3.6.1.2.1.1.4.0
sysTemperature	1.3.6.1.2.1.1.5.0
sysHumidity	1.3.6.1.2.1.1.6.0

Esses valores serão inseridos na árvore através das leituras em tempo real realizadas pelo DHT11. A transformação de dado analógico para digital é feita pelo próprio sensor, jogando pulsos interpretáveis pelo pino DATA OUT no pino Digital 3 do NodeMCU. O DHT11 é alimentado nos pinos VCC (+5V) e GND, do NodeMCU.

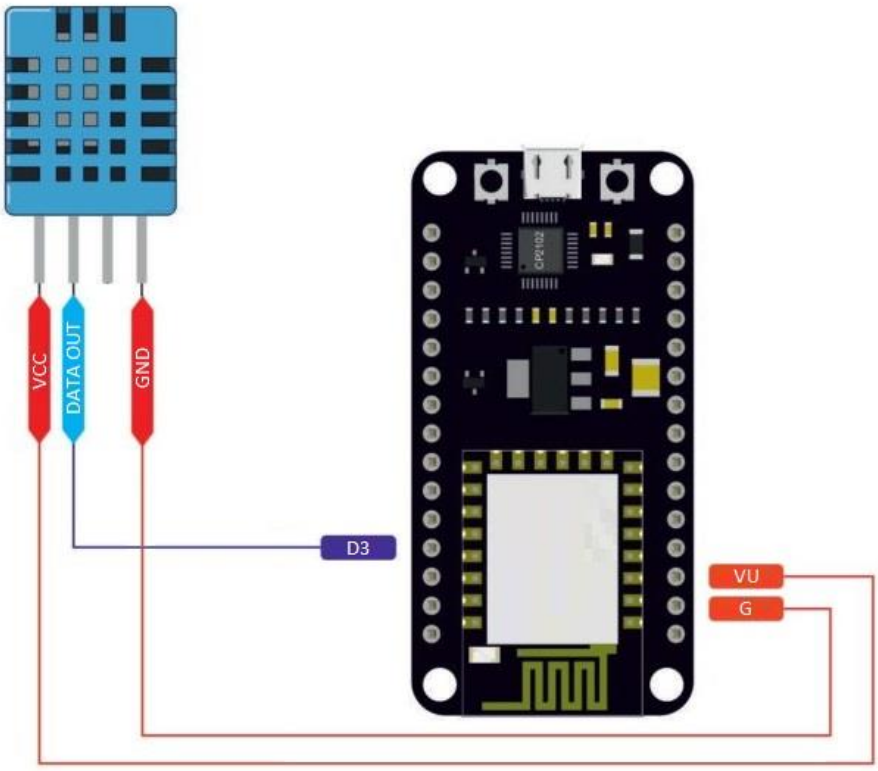

Figura 5 - Diagrama de conexões

A coleta das informações será realizada pelo Zabbix através de *gets* periódicos nas OIDs estabelecidas no código. Os itens *Temperature* e *Humidity* do *Host* NodeMCU irão alimentar o Zabbix. Estes valores serão a base para o estabelecimento de *thresholds*, notificações, dashboards, etc.

5. METODOLOGIA

5.1 Inicialização

Os passos de inicialização são executados na seguinte sequência:

- Alocação da árvore SNMP

```
const char *sysDescr       = "1.3.6.1.2.1.1.1.0";
const char *sysName        = "1.3.6.1.2.1.1.2.0";
const char *sysUpTime      = "1.3.6.1.2.1.1.3.0";
const char *sysLocation    = "1.3.6.1.2.1.1.4.0";
const char *sysTemperature = "1.3.6.1.2.1.1.5.0";
const char *sysHumidity    = "1.3.6.1.2.1.1.6.0";
```

- Inicialização do pino de saída com o LED para identificar que o sistema está rodando.

```
pinMode(D1,OUTPUT);
```

- Inicializa driver do Wi-Fi, conecta e obtém IP.

```
Serial.println("Starting Wi-Fi connection...");
WiFi.begin(ssid, pass);
while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
Serial.println("WiFi connected!");
Serial.println("Current IP Address: ");
Serial.println(WiFi.localIP());
```

- Inicializa driver do sensor de temperatura.

```
Serial.println("Starting DHT11...");
dht.begin();
```

- Inicializa agente SNMP e registra função de *callback* de dados recebidos

```
Serial.println("Starting SNMP agent...");
api_status = Agentuino.begin();
if ( api_status == SNMP_API_STAT_SUCCESS )
{
  Agentuino.onPduReceive(pduReceivedCallback);
  Serial.println("SNMP Agent is runnning!");
  Serial.println("Listening for GET requests...");
  delay(10);
  return;
}
```

5.2 Loop

Após a inicialização, a execução passa para a etapa do *loop*. O que acontece é realizado na seguinte sequência:

- Liga o LED conectado ao pino D1. Isto serve para identificar que o sistema está rodando.

```
digitalWrite(D1,HIGH); /* On */
delay(100);
```

- Coleta os valores de temperatura e humidade fornecidos pelo DHT11. Como pode ser visto, os valores, originalmente ponto flutuante, são multiplicados por 10 e é realizado o *casting* para inteiro. Isto é feito para facilitar o envio do valor via SNMP. O destinatário do pacote deverá estar ciente disto para realizar a divisão do valor por 10.

```
/* Get temperature and humidity and sends the value as
integers */
   temperature = (int)dht.readTemperature() * 10;
   humidity = (int)dht.readHumidity() * 10;
```

- Atualiza o valor que é enviado quando é realizada uma requisição para o OID *sysUpTime*.

```
   if ( millis() - prevMillis > 1000 ) /* Updates sysUpTime
*/
   {
    prevMillis += 1000;
    locUpTime += 100;
   }
```

- Desliga o LED de atividade.

```
digitalWrite(D1,LOW); /* Off */
delay(100);
```

- Chama o método **`listen()`** da biblioteca Agentuino. Este método, internamente, chama o método **`listen()`** da biblioteca Wi-Fi UDP. Caso existam pacotes UDP recebidos, a biblioteca Agentuino realiza o *call-back*, ou seja, executa o método de tratamento **`pduReceivedCallback()`** presente no *firmware* desenvolvido para o trabalho.

```
Agentuino.listen();
```

6. RESULTADOS

6.1 Validação

A validação do *firmware* foi realizada em etapas. Primeiramente, fizemos uma aplicação simples da leitura de temperatura e humidade provenientes do DHT11, enviando os valores para a serial e os observando através do monitor serial nativo da IDE do Arduino. Com isto, conseguimos validar a montagem, apesar de extremamente simples e os pinos utilizados.

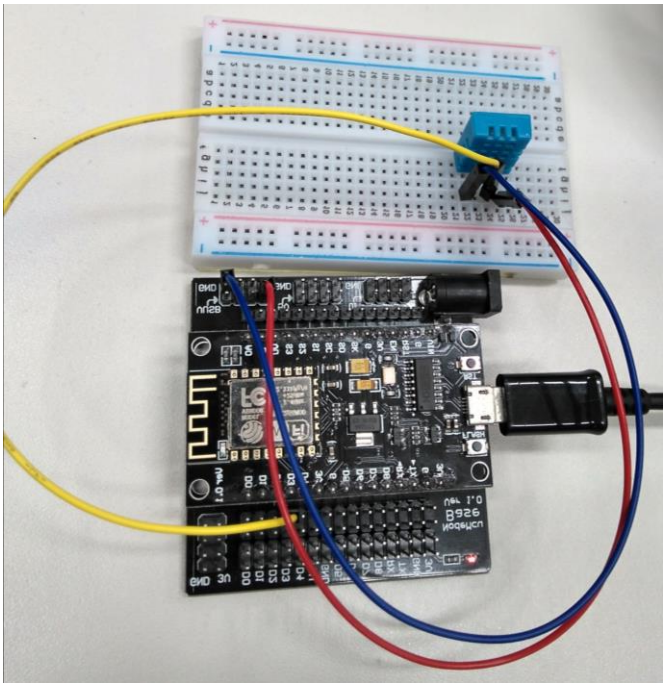

Figura 6 - Conexões físicas do projeto

Após esta etapa, partimos para o teste e validação da conexão Wi-Fi. Colocamos o NodeMCU na rede e foi possível realizar testes de PING entre as máquinas e o dispositivo.

Por último, foi realizada a validação da comunicação SNMP. Para esta etapa, após pesquisas, escolhemos a biblioteca Agentuino. Trata-se de uma biblioteca antiga e bem simples, sendo recomendado para aplicações mais simples. Foi necessário modificar a biblioteca para realizar a comunicação via Wi-Fi. Por padrão, ela é implementada para utilizar um *shield* Ethernet. Utilizamos o software *Paessler SNMP Tester* para realizar as requisições. A seguir podem ser vistos os resultados dos testes efetuados.
A configuração de conexão é realizada no *software* de teste, onde:

- *Local IP*: é o IP do computador, no caso, o host.
- *Device IP/Port:* IP do NodeMCU e a porta 161 (padrão UDP).
- *SNMP Version:* É utilizada a versão 1 do protocolo.
- *Community:* A comunidade é pública.

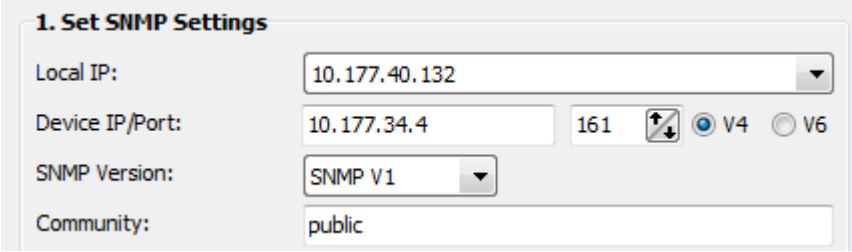

Figura 7 - Configurações do software de teste

O NodeMCU, ao inicializar, realiza as configurações e habilita a comunicação Wi-Fi. Abaixo pode ser visto a saída serial do dispositivo durante a inicialização:

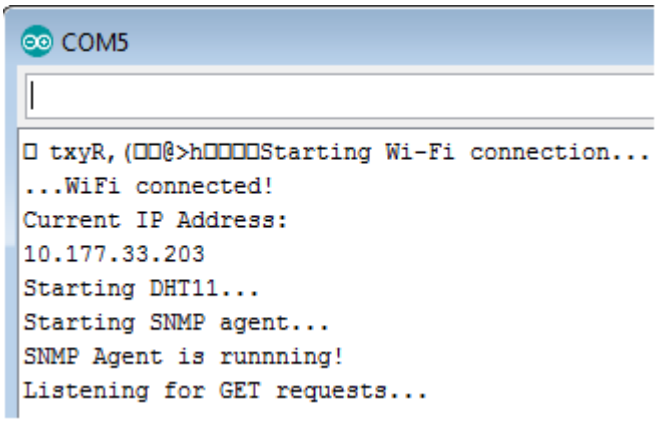

Figura 8 - Saída padrão do dispositivo durante a inicialização

Neste momento, é possível realizar as requisições via *software* de teste. A seguir estão exemplos de requisições e respostas:

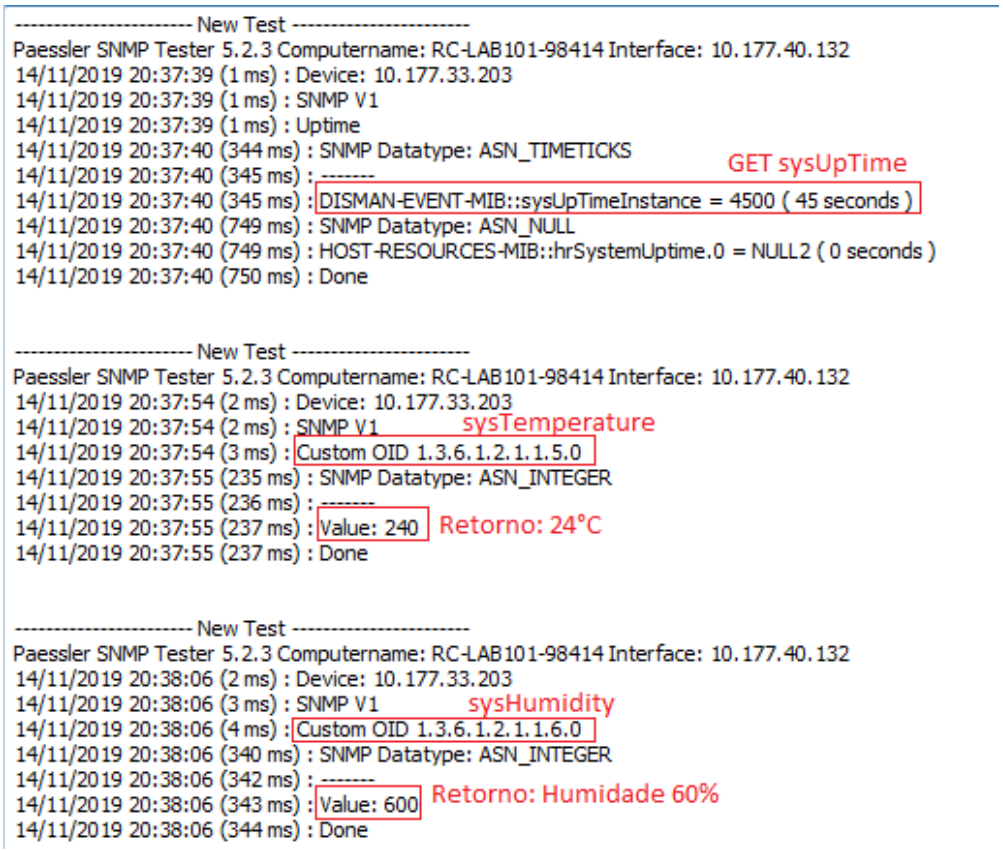

Figura 9 - Tela de teste do software SNMP Tester

No monitor serial é possível acompanhar as requisições recebidas, conforme abaixo:

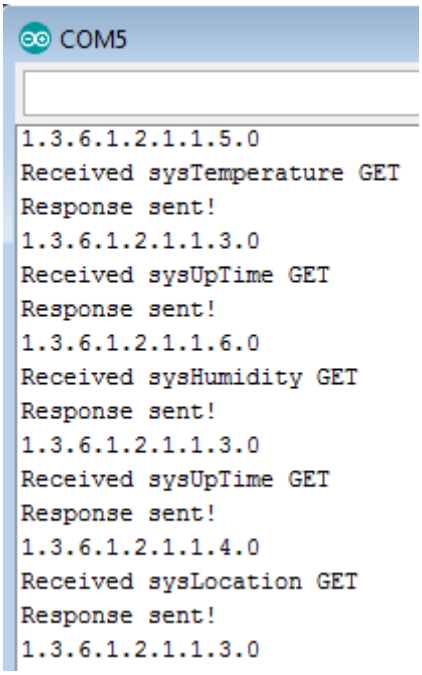

Figura 10 - Mensagens enviadas pela serial de pacotes recebidos

6.2 Conclusão

A escolha deste projeto pelo grupo se mostrou um ótimo desafio. Trata-se de uma aplicação relativamente simples, surgindo de uma necessidade empresarial e com potencial de se tornar produto.

O processo de desenvolvimento se mostrou relativamente simples, principalmente pela vasta quantidade de material sobre Arduino na internet. A comunidade ativa disponibiliza diversas bibliotecas que facilitam muito o desenvolvimento com sensores e outros *shields*. Porém, a falta de recursos da IDE do Arduino torna o tratamento de problemas mais dificultoso quando o processo "sai da curva" dos diversos tutoriais online. Nestes momentos, é essencial o conhecimento teórico sobre microcontroladores e programação.

Encontramos dificuldades, principalmente, com a biblioteca do protocolo SNMP. A biblioteca Agentuino é antiga, sendo portada do antigo repositório Google Code para o Github. A biblioteca é limitada e é necessário diversas strings, uma para cada OID. Quanto mais valores são utilizados, mais cresce o consumo de memória. Com o NodeMCU isto pode se tornar um problema. Após pesquisa e tentativas, conseguimos alterar o código da biblioteca para utilizar a biblioteca **WiFiUdp.h** pois a padrão utiliza *Ethernet*. Isto é possível pois a IDE do Arduino compila as bibliotecas no mesmo momento de compilação do *firmware* principal.

A implementação do *callback* é pouco otimizada. A função de tratamento implementada no firmware do projeto é baseada na do projeto de teste presente na biblioteca Agentuino. Este é mais um ponto negativo da biblioteca, sendo possível implementar algo mais elegante do que foi desenvolvido.

Foi possível completar, praticamente, a proposta do projeto. Isto provavelmente foi possível devido ao *hardware* mínimo para implementar a prova de conceito. Dentro do tempo disponível do projeto é difícil completar um projeto complexo, com *hardware* e *firmware* bem elaborado. Isto também pesou na escolha do grupo, cujo o objetivo era entregar a prova de conceito funcional.

O projeto foi uma excelente oportunidade para aprender, na prática, desenvolver um projeto do zero, com desenvolvimento próprio de *firmware* e comunicação com outros dispositivos, sendo o principal aspecto de IoT (*Internet of Things*).

7. REFERÊNCIAS

- Datasheet ESP8266. Disponível em: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. Acessado em 31/10/2019.

- Datasheet DHT11. Disponível em: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf. Acessado em 31/10/2019.

- Datasheet ESP8266. Disponível em: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. Acessado em 31/10/2019.

- Teleco: Tutoriais de Banda Larga - SNMP. Disponível em: https://www.teleco.com.br/tutoriais/tutorialsnmp/pagina_2.asp. Acessado em 31/10/2019.

- Samples de snmpwalk. Disponível em: https://networkengineering.stackexchange.com/questions/2990/translating-snmpwalk-output-into-human-readable-format. Acessado em 31/10/2019.

- Biblioteca Agentuino. Disponível em: https://github.com/1sw/Agentuino. Acessado em 14/11/2019.

- Biblioteca do sensor de temperatura e humidade DHT11. Disponível em: https://github.com/adafruit/DHT-sensor-library. Acessado em 14/11/2019.