INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS
CAMPUS BETIM

Vitor Hugo Laia de Oliveira

GATEWAY PARA INTEGRAÇÃO DA COMUNICAÇÃO MÁQUINA-A-MÁQUINA À INTERNET DAS COISAS

Betim
2021

Vitor Hugo Laia de Oliveira

GATEWAY PARA INTEGRAÇÃO DA COMUNICAÇÃO MÁQUINA-A-MÁQUINA À INTERNET DAS COISAS

Trabalho de Conclusão de Curso submetido ao Instituto Federal de Ciência, Educação e Tecnologia de Minas Gerais Campos Betim para obtenção do Grau de Bacharel em Engenharia de Controle e Automação. Sob a orientação do Professor Victor Alves Silva e Melo e co-orientação do Professor Virgil Del Duca Almeida.

Betim
2021

INSTITUTO FEDERAL
Minas Gerais

REDE DE BIBLIOTECAS

FICHA CATALOGRÁFICA PARA TRABALHO DE CONCLUSÃO DE CURSO

FICHA CATALOGRÁFICA

O48g Oliveira, Vitor Hugo Laia de.
Gateway para integração da comunicação máquina-a-máquina à internet das coisas. / Vitor Hugo Laia de Oliveira. - 2021.
41p.:il.

Orientador: Prof. Me. Victor Alves Silva e Melo.
Orientador: Prof. Me. Virgil Del Duca Almeida.
Trabalho de Conclusão de Curso (Graduação em Engenharia de Controle e Automação) - Instituto Federal de Minas Gerais. *Campus* Betim, 2021.

1. Microcontrolador. 2.Modbus. 3. CoAP. 4.ESP32. 5.IoT. I. Instituto Federal de Minas Gerais. II. Título.

CDD 005.1

Catalogação: Rejane Valéria Santos - CRB-6/2907

Vitor Hugo Laia de Oliveira

GATEWAY PARA INTEGRAÇÃO DA COMUNICAÇÃO MÁQUINA-A-MÁQUINA À INTERNET DAS COISAS

Trabalho de Conclusão de Curso submetido ao Instituto Federal de Ciência, Educação e Tecnologia de Minas Gerais Campos Betim para obtenção do Grau de Bacharel em Engenharia de Controle e Automação. Sob a orientação do Professor Victor Alves Silva e Melo e co-orientação do Professor Virgil Del Duca Almeida.

Aprovado em: 13 / 12 / 21 pela banca examinadora:

Victor Alves Silva e Melo

Prof. Me. Victor Alves Silva e Melo - IFMG (Orientador)

Virgil Almeida

Prof. Me. Virgil Del Duca Almeida - IFMG (Co-orientador)

Arthur H. R. Rosa

Prof. Dr. Arthur Hermano Rezende Rosa - IFMG

Prof. Me. Helbert Ribeiro de Sá - IFMG

Dedico este trabalho aos meus pais, que me proporcionaram, além da vida física, o auxílio necessário para que pudesse me tornar um Engenheiro.

AGRADECIMENTOS

Agradeço, dessa forma minha

Àqueles que, de simples sorriso

Não me deixaram desistir

A minha mãe pela ternura nos momentos críticos

Ao meu pai pelo companheirismo

A minha irmã pelo ombro amigo

Sem vocês aqui não estaria

Isadora, meu amor, por me mandar estudar

Aos meus avós por tentarem entender meu mundo

Agradeço ainda, com a mesma importância,

Dalton e MEJA por me acolherem na bifurcação da vida

Aos bons espíritos pelos conselhos ofertados

Pelos companheiros de trabalho caritativo por me apresentarem um mundo diferente

Jonathan e Lucas pelos momentos de diversão

Aos professores que passaram pela minha jornada, que abriram as portas da ciência

Aos meus orientadores, por dedicarem seu tempo para me auxiliar

Vocês fazem parte da minha história

E a isso sou eternamente grato

RESUMO

A necessidade de se conectar equipamentos e máquinas às redes de dados é uma realidade que agrega um número gigantesco de objetos, ultrapassando a marca dos bilhões de dispositivos conectados, que aumentam de forma acelerada a cada ano. Contudo a diversidade de dispositivos gera uma rede heterogênea, dificultando a conexão entre sistemas de forma eficiente e segura. Neste contexto tornam-se necessários dispositivos microcontrolados que possam fazer a interface entre as diversas redes existentes no mercado. Em meio a esse cenário, tem-se, entre os protocolos diversos que atuam na comunicação entre máquinas, o Modbus e o CoAP; o primeiro com forte presença no mercado; o segundo com a promessa de trabalhar em redes restritas com efetividade. Este trabalho tem como objetivo criar uma interface entre os dois protocolos utilizando o microcontrolador ESP32, permitindo que dados que antes seriam lidos apenas por máquinas conectadas por meios físicos possam ser acessados de forma remota através do protocolo CoAP, que possui fácil interação com aplicações em HTTP. Após o desenvolvimento da interface e dos testes realizados foi constatado a possibilidade da interação entre os protocolos através do NodeMCU, um kit de desenvolvimento do ESP32, e do *hardware* desenvolvido.

Palavras-chave: Microcontrolador; Modbus; CoAP; ESP32; IoT.

ABSTRACT

Connect equipments and machines to data networks is a necessity and a reality that assemble a huge number of objects, surpassing the mark of the billions of connected devices, which increase at an accelerated rate each year. However, the diversity of devices generates a heterogeneous network, making it difficult to connect systems efficiently and safely. In this context, it is necessary to have microcontrolled devices that can interface between the various networks on the market. In the midst of this scenario, there are, among the various protocols that operate in the communication between machines, Modbus and CoAP; the first with a strong presence in the market; the second with the promise of working in restricted networks effectively. This work objective to create an interface between the both protocols using the ESP32 microcontroller, allowing data that previously would have been read only by machines connected by physical environment can be accessed remotely through the CoAP protocol, which has easy interaction with HTTP applications. After the development of the interface and the tests carried out, the possibility of interaction between the protocols through NodeMCU, an ESP32 development kit, and the built hardware was verified.

Key Words: Microcontroler; Modbus; CoAP; ESP32; IoT.

LISTA DE ABREVIATURAS E SIGLAS

ASCII	*American Standard Code for Information Interchange*
CoAP	*Constrained Application Protocol*
CoRE	*Constrained RESTful Environments*
CRC	*Cyclic Redundancy Check*
HTTP	*Hypertext Transfer Protocol*
I/O	*Input/Output*
IDE	*Integrated Development Environment*
IETF	*Internet Engineering Task Force*
IoT	*Internet of Things*
LCR	*Longitudinal Redundancy Check*
M2M	*Machine to Machine*
PCB	*Printed Circuit Board*
PLC	*Programmable Logical Controller*
RFC	*Request for Comments*
RTU	*Remote Terminal Unit*
TCP	*Transmission Control Protocol*
TTL	*Transistor-Transistor Logic*
UDP	*User Datagram Protocol*
URIs	*Uniform Resources Identifiers*

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho visa desenvolver um *gateway* para integrar equipamentos que utilizam Modbus à Internet das Coisas através do protocolo CoAP, disponibilizando dados para dispositivos com capacidade de processamento de informações e conectividade a redes de dados, como *smartphones, tablets, notebooks,* computadores*,* etc*.*

O Modbus é um dos protocolos de comunicação mais utilizados no setor industrial, somando milhões de equipamentos com suporte a ele no mundo, atuando nas mais diversas áreas de produção e gerenciamento (MODBUS ORGANIZATION, 2017). Sua ampla presença global e licenciamento livre abriram espaço para que um grande número de fabricantes de *hardware* produzissem seus próprios equipamentos que utilizam Modbus, gerando um ecossistema de centenas de integradoras ligadas ao universo dele (MODBUS ORGANIZATION, 2017).

Novas tecnologias têm surgido, modificando os meios de comunicação e produção, diminuindo distâncias e facilitado a interação entre homens, entre máquinas e entre ambos, corroborando para uma estimativa de 40 bilhões de dispositivos conectados em rede até o ano de 2020 (PRESS, 2014), além da expectativa de 66% do mundo estar utilizando a internet até 2023 (CISCO, 2020). Essa grande quantidade de dispositivos e pessoas conectados representa o cerne da Internet das Coisas, onde “coisa” pode ser qualquer elemento inteligente com capacidade de comunicação remota. Apesar de o protocolo Modbus ser bem estruturado e trabalhar em rede, ele foi desenvolvido com foco na automação industrial, distanciando-se dessa crescente realidade. Tornando-se, diante dos fatos citados, necessário esforços para integrar o conjunto de máquinas legado da indústria à Internet das Coisas.

1.1 Formulação do problema

O crescente avanço dos meios de comunicação descortina um cenário mundial onde bilhões de dispositivos se conectam através de redes heterogêneas, trocando informações e interagindo entre si. Esse novo panorama converge com a Internet das Coisas, onde tudo pode estar conectado. Paralelo a essa crescente onda tecnológica, tem-se, nos parques industriais, dezenas de milhões de máquinas e equipamentos estruturalmente incapazes de interagir em redes inteligentes. Nesse contexto, o

problema motivador é como interligá-los à Internet das Coisas, garantindo interoperabilidade entre os dispositivos envolvidos.

1.2 Justificativa

A necessidade de se conectar dispositivos eletrônicos e máquinas às redes de dados, em uma escala privada, ou à Internet em uma proporção mais ampla, é uma realidade evidente para o segmento industrial. Segundo Cisco (2016), em 2015 existiam aproximadamente 4,9 bilhões de conexões máquina-a-máquina trafegando pela Internet, com estimativa de 12,2 bilhões até o final de 2020 e se estendendo para 14,7 bilhões para o ano de 2023 (Cisco, 2020). São números expressivos que indicam um crescimento ascendente de dispositivos inteligentes disponibilizando e consumindo informações em larga escala nas redes atuais.

Em contrapartida, tem-se um maquinário diversificado, distribuído globalmente, atendendo a inúmeros setores de produção, que não foi projetado inicialmente com a perspectiva desse cenário de interconexões. São equipamentos com protocolos específicos da indústria, otimizados para determinados segmentos, num ciclo que se fecha neles mesmos.

A iniciativa desse trabalho é desenvolver um *gateway* que possa adicionar a esse parque legado a capacidade de se integrar ao universo da Internet das Coisas, disponibilizando recursos que possam ser acessados via CoAP. Desse modo, um supervisor de produção pode, diante de um controlador lógico programável (PLC - *Programmable Logical Controller*) com suspeitas de mau funcionamento, consultar seu estado utilizando um *smartphone* ou *tablet*, sem a necessidade de deslocamento para a central de operações. Dados que pertenciam somente aos sistemas supervisórios podem ser extraídos e adicionados às plataformas de gestão da empresa de uma maneira padronizada e simples, não exigindo das equipes de desenvolvimento conhecimentos prévios de automação.

O Modbus foi escolhido por ser um protocolo utilizado em milhões de equipamentos nos mais diversos segmentos de produção, extração, beneficiamento e manufatura. O protocolo CoAP é uma alternativa enxuta e leve, que opera sobre HTTP, para interoperabilidade entre dispositivos inteligentes, que tem se expandindo e recebe o apoio de grandes nomes do mercado de tecnologia (BORMANN, 2016).

O microcontrolador ESP32, que é um componente de baixo custo, tem suporte nativo a redes WIFI, bibliotecas para trabalhar com CoAP e ModBus e certificações que o habilitam ao pesado ambiente industrial.

1.3 Objetivo geral

Desenvolver um *gateway*, figura 1, para comunicação máquina-a-máquina (M2M - *Machine-to-Machine*) entre os protocolos Modbus-RTU e CoAP utilizando o ESP32 que permita a inserção de equipamentos legados a Internet das Coisas.

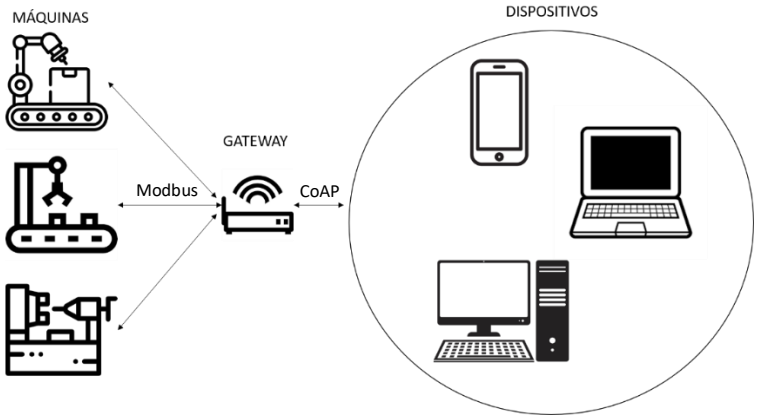


Figura 1 – Esquema de funcionamento

Fonte: Elaborado pelo autor

1.4 Objetivos específicos

- Projetar e construir um circuito utilizando o microcontrolador ESP32;
- Desenvolver um *firmware* para o ESP32 com suporte ao protocolo CoAP no modo servidor;
- Agregar ao *firmware* suporte ao protocolo Modbus;
- Disponibilizar uma interface CoAP para intermediar a comunicação com o Modbus.

1.5 Metodologia

O trabalho será desenvolvido com a utilização do conjunto de desenvolvimento para o ESP32 NodeMCU, que será utilizado para testes, um simulador de mestre e escravo Modbus, um simulador de cliente CoAP e um simulador de circuitos.

O trabalho foi divido nas seguintes etapas:

- Desenvolver o *firmware* base com o protocolo CoAP;

- Realizar testes da implementação do *firmware*;
- Acrescentar ao *firmware* suporte ao protocolo Modbus;
- Simular a comunicação Modbus;
- Desenvolver a integração entre o CoAP e o Modbus;
- Testar o *firmware* final;
- Simular o circuito prevendo porta de comunicação RS-232;
- Projetar a placa do circuito;
- Construir o circuito e carregar o *firmware;*
- Fazer os testes e a revisão final.

2 INTERNET DAS COISAS E M2M

A Internet das Coisas (IoT - *Internet of Things*) é um conceito que está se ampliando e ganhando espaço rapidamente no cenário das telecomunicações, segundo Atzori, Iera e Morabito (2010). Os autores definem IoT como um conceito que engloba a possibilidade, ou capacidade, de integrar uma vasta gama de objetos distintos (como sensores, *smartphones*, computadores, *smartdevices* e mesmo objetos de uso mais comum) que tenham capacidade de se conectar a uma rede de dados, entre si. IoT e M2M se assemelham em alguns aspectos, mas são propostas diferentes, sendo que uma não engloba a outra.

Segundo Borswarthick, Elloumi e Hersent (2012, p. 2):

> O papel de M2M é estabelecer as condições que permitam um dispositivo trocar informações (bidirecionalmente) com uma aplicação através de uma rede de comunicação, de forma que o dispositivo e/ou aplicação possam agir como a base para esta troca de informações.

O autor também esclarece que M2M tem poucos limites definidos, abrangência de escopo muito grande e não prevê suporte para interação humana.

Nas definições de IoT, de acordo com Bandyopadhyay et al. (2011, p. 94), "coisas" são dispositivos, virtuais ou físicos, com identidades e atributos próprios e capazes de interagir com ou através de interfaces inteligentes. Assim, enquanto M2M opera com soluções baseadas em comunicação ponto-a-ponto, interligando dispositivos de um mesmo segmento, as propostas de IoT, segundo Polsonetti (2014), trazem uma visão muito mais ampla, pois ao aceitar objetos diversos em sua rede, ela ultrapassa os limites da busca de melhorias na produção e prestação do serviço,

abarcando e alterando também modelo de negócio das empresas. Gubbi define IoT como:

> Interconexão de dispositivos sensores e atuadores provendo a habilidade de compartilhar informações através de plataformas por meio de um framework unificado, desenvolvendo um cenário operacional comum possibilitando aplicações inovadoras. (GUBBI et al., 2013, p. 1645).

Apesar do termo Internet das Coisas, IoT não se limita ao ecossistema da Internet, ele pode ser implementado em qualquer segmento de rede de dados, desde pequenas redes internas a grandes redes metropolitanas ou mundiais. O foco principal é a capacidade dos dispositivos se conectarem e estabelecerem canais de comunicação, com ou sem interação humana.

Segundo a Cisco (2020), espera-se que em 2023 existam 1,8 conexões M2M para cada ser humano, sendo que os dispositivos residências representarão 48% desse valor, o que confirma o crescimento dessas conexões e a presença mais do que confirmada do IoT no futuro da humanidade.

Figura 2 – Rede IOT
Fonte: DIREITO DA COMUNICAÇÃO, 2018

3 PROTOCOLOS DE COMUNICAÇÃO

Em um cenário composto por máquinas que trabalham em conjunto, trocando ou não informação direta, se faz necessário que haja regras para gerenciar e garantir a comunicação entre elas, a partir dessa necessidade surge os protocolos de comunicação. Segundo Oxford (PROTOCOLO, 2020), protocolo pode ser entendido como normas ou formalidades e, de forma simplificada, Halsall (2020) define protocolo como regras para troca de mensagens. No que tange os processos de comunicação, podemos, então, afirmar que protocolos são as normas que asseguram o

entendimento, seja pela máquina ou pelo Homem, dos processos que ocorrem, internamente ou externamente, entre uma ou mais máquinas.

Um dos modelos recorrentes da forma com que se dá uma comunicação é descrito com os conceitos de emissor, destinatário, código, sinal, informação, codificação e decodificação (SAMPAIO, 2001), que pode ser entendido como demonstra a figura 3.

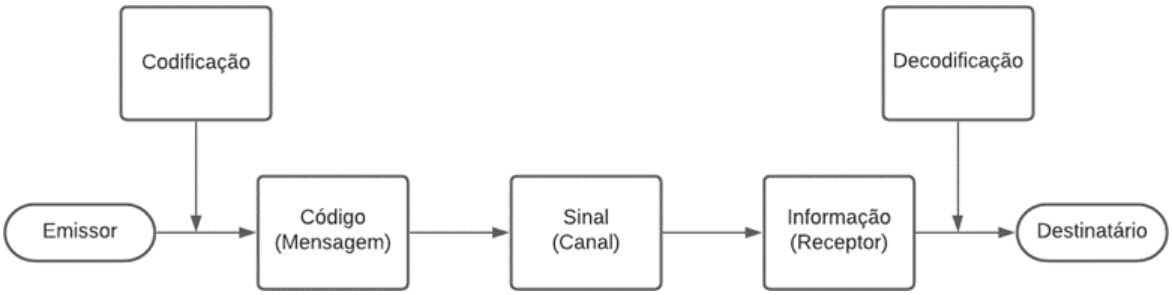


Figura 3 - Modelo comunicação
Fonte: Elaborado pelo autor

O modelo apresentado na figura 3 demonstra, de forma simplificada, o processo de comunicação. No cenário industrial uma máquina (emissor) transmite para outra (destinatário) um conjunto de dados (mensagem), esses dados são codificados e enviados através de um canal (sinal) estabelecido, esses dados são decodificados e entregues ao destinatário.

Dessa forma observa-se a importância da codificação e decodificação da mensagem enviada, que será regrada pelo protocolo estabelecido.

3.1 Análise de mercado dos protocolos industriais

Segundo MHANNA (200?), a rede de campo (Fieldbus) é uma rede utilizada para que os dispositivos de campo se comuniquem com os controladores através de uma única conexão na qual as informações são transmitidas em série de forma multiplexada. Com a necessidade de interligar os dispositivos da planta da fábrica com as redes de Ethernet, foi desenvolvido a Ethernet Industrial (LUGLI, SANTOS, FRANCO, 200?), que é uma adaptação do protocolo Ethernet às necessidades industriais, tais como: interoperabilidade entre os diversos dispositivos, robustez dos equipamentos, aumento da quantidade de dados e diminuição do tempo de ciclo (LUGLI, SANTOS, FRANCO, 2021).

Analisando as informações sobre a divisão do mercado de redes industriais, que pode ser vista na figura 4, obtidas através de HMS (2019), ANYBUS (2018) e HMS (2020), é possível observar o forte crescimento das redes de Ethernet industrial, assim como o surgimento da utilização significativa das redes sem fio, que se mantém constantes desde 2017.

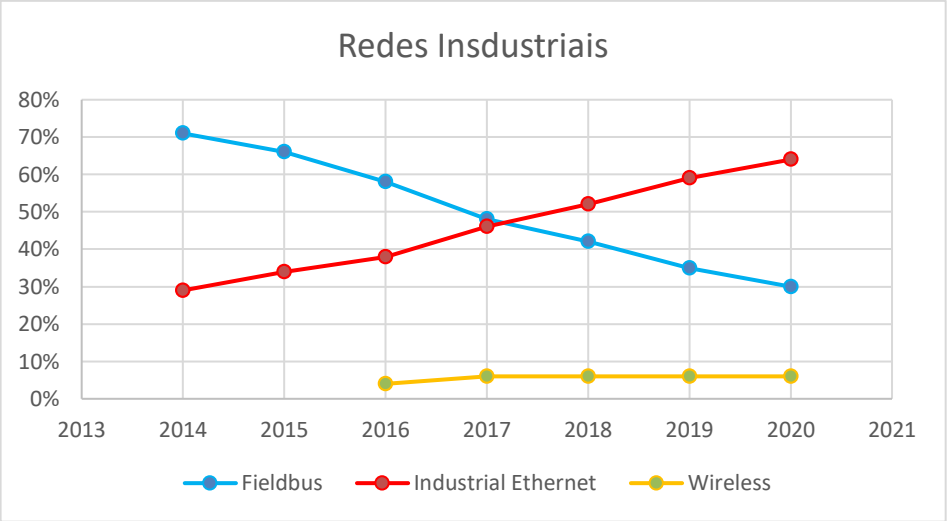


Figura 4 – Mapeamento das redes industriais

Fonte: Elaborado pelo autor

Em relação aos protocolos utilizados, o Modbus-RTU, segundo protocolo mais utilizado nas redes *Fieldbus* em 2020, perde espaço no mercado de forma menos significativa que o protocolo Profibus DP, observando as taxas de utilização no mercado em 2018 e 2020, como pode ser verificado na figura 5. Segundo HMS (2019), as redes de campo apresentam um decaimento anual de 5%, enquanto as redes sem fio crescem 30% e a Ethernet industrial cresce 20%.

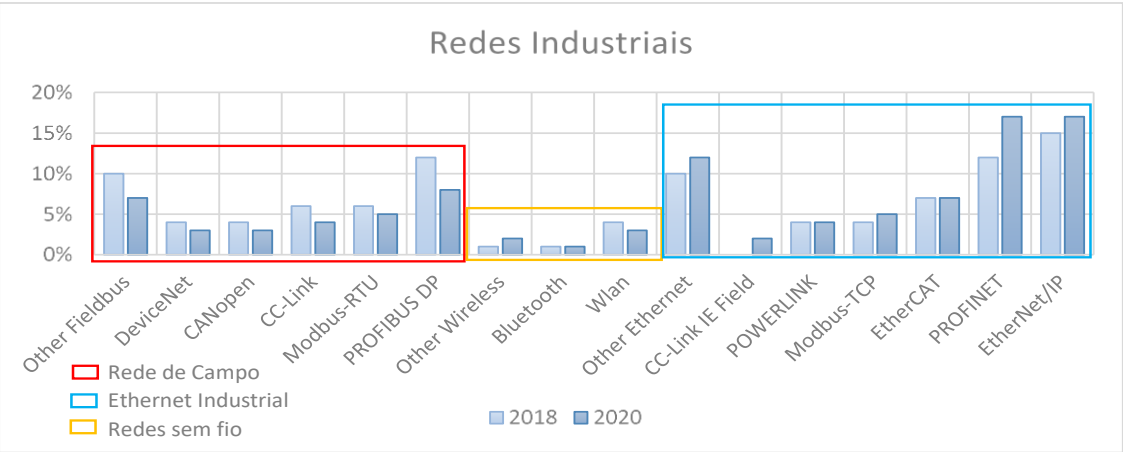


Figura 5 – Protocolos industriais

Fonte: Elaborado pelo autor

Considerando a análise do mercado considerada na figura 5, o Modbus-RTU é um protocolo de utilização estável, diminuindo 1% entre os anos de 2018 e 2020, ou seja, um valor menor que o decaimento da utilização do *Fieldbus*. Dessa forma observa-se a tendência do protocolo Modbus continuar presente no mercado em anos futuros.

Em relação aos equipamentos legados, que continuam presentes no mercado, é vital permitir que o maquinário que utiliza o protocolo Modbus-RTU consiga se integrar tanto ao mercado das redes sem fio, que cresce mais que as redes de Ethernet industrial ao ano, quanto permitir que ele se comunique com os dispositivos que utilizam a Ethernet Industrial.

3.2 Protocolo TCP

De acordo com a RFC 793 (1981), o Protocolo de Controle de Transmissão (TCP-*Transmission Control Protocol*) é um protocolo confiável de ponta a ponta, orientado para conexão, projetado para atuar entre camadas de outros protocolos, com suporte à utilização em redes e aplicações variadas.

O TCP fornece processos confiáveis de comunicação entre computadores interconectados em redes diversas. Por definição do Pedido de Comentários (RFC-*Request for Comments*) do protocolo TCP (RFC 793, 1981), o protocolo deve ser capaz de operar em um amplo espectro de sistemas de comunicação, desde redes com fio a redes comutadas por pacotes ou por circuitos.

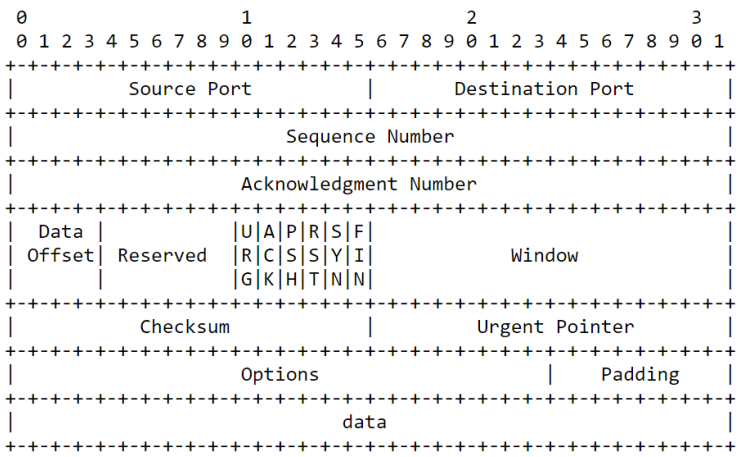


Figura 6 – Cabeçalho TCP

Fonte: RFC 793, 1981

Em uma conexão TCP os dados são fragmentados em porções de bytes, chamadas pacotes. Ele garante que o destinatário receberá os pacotes na ordem em

que foram quebrados e enviados. O destinatário envia mensagens de volta ao remetente dizendo que as recebeu. Se o remetente não obtiver uma resposta correta, ele reenviará os pacotes para garantir que o destinatário os recebeu. Os pacotes também são verificados quanto a erros. Os pacotes enviados são monitorados para que nenhum dado seja perdido ou corrompido durante o tráfego de bytes. Para esse controle, o TCP utiliza uma estrutura de cabeçalho, figura 6, que varia de 20 a 80 bytes, fora o tamanho dos dados transmitidos (RFC 793, 1981).

3.3 Protocolo UDP

O Protocolo de Datagrama de Usuário (UDP-*User Datagram Protocol*) é um protocolo de transmissão de dados que funciona de forma semelhante ao TCP, mas elimina toda a estrutura e custo de verificação de erros. Ao usar UDP, os pacotes são enviados apenas para o destinatário. O remetente não aguarda para ter a confirmação de que o destinatário recebeu o pacote. Os pacotes subsequentes são enviados sem confirmação. Não há garantia de que todos os pacotes chegarão ao destinatário e não há como solicitar um pacote novamente. Este protocolo permite que programas possam trocar mensagens entre computadores ligados em rede com um mínimo de custo de desenvolvimento com os mecanismos e as implementações específicas de protocolo (Postel, J., 1980). Na figura 7 é possível observar o cabeçalho do protocolo.

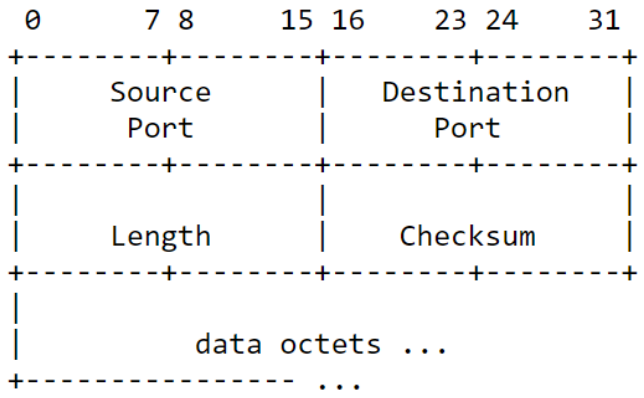


Figura 7 – Cabeçalho UDP

Fonte: Postel, J., 1980

3.4 Protocolo HTTP

O Protocolo de Transferência de Hipertexto (HTTP-*Hypertext Transfer Protocol*) é um protocolo que atua na camada de aplicação e foi projetado para trabalhar com informações distribuídas em sistemas de hipertexto. Ele é um protocolo

genérico, sem estado, que pode ser usado em outras tarefas além do hipertexto, como servidores de nomes e sistemas de gerenciamento de objetos distribuídos, por meio da extensão de seus métodos de requisição, códigos de erro e cabeçalho.

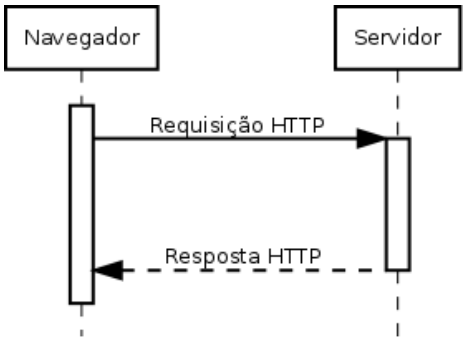


Figura 8 – Fluxo HTTP

Fonte: Fielding, R., 1999

Ele é um protocolo estruturado no modelo solicitação/resposta, figura 8. Um cliente envia uma solicitação ao servidor, figura 9, encapsulando os dados no padrão do protocolo. O servidor responde com uma linha de status, figura 10, incluindo a versão do protocolo da mensagem e um código de sucesso ou erro, seguido por uma mensagem semelhante contendo informações do servidor e os dados do corpo da resposta. O HTTP é o protocolo padrão adotado pelos navegadores para acessar os conteúdos da internet (Fielding, R., 1999).

```
GET /index.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.ifmg.br
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Figura 9 – Requisição HTTP

Fonte: Fielding, R., 1999

```
HTTP/1.0 200 OK
Content-Length: 15
Content-type: text/html
Date: Thu, 24 Dec 2015 01:33:41 GMT
Last-Modified: Thu, 24 Dec 2015 01:33:37 GMT
Server: SimpleHTTP/0.6 Python/3.4.2

<h1>Teste</h1>
```

Figura 10 – Resposta HTTP

Fonte: Fielding, R., 1999

3.5 Protocolo Modbus

O protocolo Modbus foi criado originalmente pela empresa americana Modicon em 1979. Posteriormente a Modicon foi incorporada pela Schneider Electric, a qual participou da iniciativa para criação de uma fundação sem fins lucrativos que passaria a deter os direitos do protocolo Modbus, chamada Modbus Organization. Em 2004 a Schneider cedeu legalmente o protocolo Modbus para a Modbus Organizaton. Desde então, o Modbus se tornou um protocolo aberto, sendo utilizado por centenas de fabricantes de equipamentos programáveis, atingindo, segundo Modbus Organization (2017), a marca de sete milhões de equipamentos nos Estados Unidos e na Europa, pleiteando o título de protocolo padrão de integração de equipamentos industriais.

Por ser um protocolo lógico, ele pode ser implantado sobre diversos meios de transmissão de dados, como cabos seriais, redes cabeadas ou sem fio, entregando interoperabilidade às máquinas instaladas em uma planta local ou ambientes distribuídos, homogêneos ou heterogêneos. Operando através do modelo mestre/escravo, onde um dispositivo central coordena a comunicação e o fluxo operacional dos demais aparelhos envolvidos, o Modbus permite monitorar e operar dispositivos, sensores e instrumentos eletroeletrônicos, atendendo a inúmeras aplicações em áreas como gás e petróleo, subestações de fornecimento de energia, linhas de produção automatizadas e estações de controle climático.

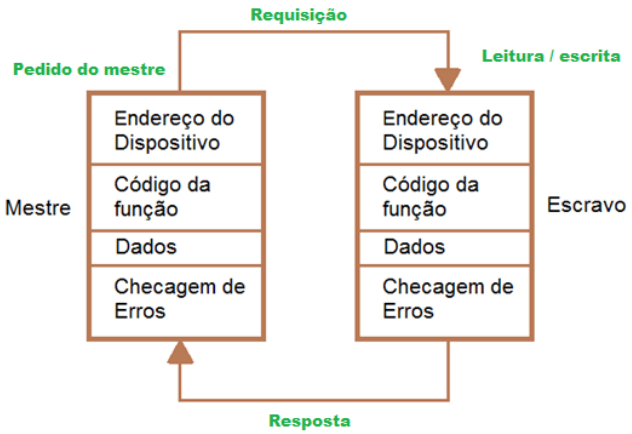


Figura 11 – Fluxo Modbus

Fonte: Elaborado pelo autor

O Modbus permite a atuação nos dispositivos através de leitura e escrita em pontos previamente mapeados. A leitura possibilita a recuperação dos valores desses pontos, trazendo uma imagem do estado atual do terminal operado. As ações de

escrita são utilizadas para operar a máquina remota, alterando seu estado ou iniciando determinados processos ou ações.

Através do fluxo de leitura e escrita, um terminal mestre Modbus pode, por exemplo, coordenar uma linha de produção, recebendo informações de etapas concluídas e sinalizando para dispositivos específicos o momento de pausarem as atividades ou reiniciarem as mesmas. A figura 11 demonstra o fluxo de mensagem do protocolo Modbus.

O Modbus possui algumas versões distintas, dentre elas o Modbus RTU, ASCII e TCP/IP, que apresentam os tipos de dados mostrados na tabela 1.

Tipo de Objeto	Tipo de acesso	Tamanho
Coils	Leitura e escrita	1-bit
Discrete Input	Apenas leitura	1-bit
Input Register	Apenas leitura	16-bits
Holding Register	Leitura e escrita	16-bits

Tabela 1 – Dados Modbus
Fonte: Elaborado pelo autor

3.5.1 Modbus RTU

O Modbus RTU (*Remote Terminal Unit*) define um padrão de comunicação serial que utiliza dados binários. Tanto os bytes de requisição quanto os de resposta possuem código de checagem cíclica redundante (CRC - *Cyclic Redundancy Check*), visando garantir a integridade dos dados trafegados. Este é o modelo com maior adoção do protocolo Modbus (MODBUS ORG, 2006).

Na tabela 2 é possível verificar a estrutura do pacote de dados do modelo RTU.

Endereço do Escravo	***Código da Função***	***Dados***	***CRC***
1 byte	1 byte	0 a 252 bytes	2 bytes (CRC-16)

Tabela 2 – Estrutura de dados RTU
Fonte: Elaborado pelo autor

3.5.2 Modbus ASCII

O ModBus ASCII (American Standard Code for Information Interchange) também trafega os dados através de uma conexão serial, mas utiliza caracteres textuais para representar os bytes da mensagem. Desse modo, cada byte é

representado por dois caracteres ASCII. O Modbus ASCII utiliza a checagem cíclica longitudinal (LCR - *Longitudinal Redundancy Check*). Os frames de comunicação iniciam por “:” e terminam por “CR/LF” (MODBUS ORG, 2006).

Na tabela 3 é possível verificar o pacote de dados do ASCII.

Início	***Endereço***	***Função***	***Dados***	***LRC***	***Final***
":" (ASCII 0x3Ah)	2 caracteres	2 caracteres	0 a 2 x 252 caracteres	2 caracteres	CR+LF (ASCII 0x0Dh + 0x0Ah)

Tabela 3 – Estrutura de dados ASCII

Fonte: Elaborado pelo autor

3.5.3 Modbus TCP/IP

O Modbus TCP é aplicado em cenários que utilizam comunicação através de redes TCP/IP. Ele encapsula o Modbus RTU em um pacote TCP e não requer cálculo de CRC, pois o protocolo TCP garante a integridade dos dados (MODBUS ORGANIZATION, 2006).

Identificador da transação	Identificador do protocolo	Tamanho da mensagem	Endereço do Escravo	Código da Função	Dados
2 bytes	2 bytes	2 bytes	1 byte	1 byte	0 a 252 bytes

Tabela 4 – Estrutura de dados TCP/IP

Fonte: Elaborado pelo autor

3.6 Protocolo CoAP

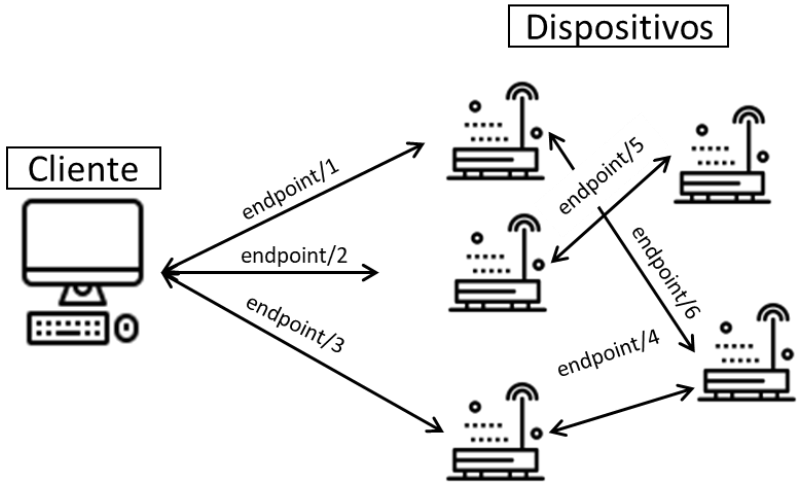
O Protocolo de Aplicação Restrita (CoAP- *Constrained Application Protocol*) foi criado por um grupo de trabalho da Força-Tarefa de Engenharia da Internet (IETF- *Internet Engineering Task Force*) chamado Meios de Repouso Restrito (CoRE- *Constrained RESTful Environments*). O objetivo era desenvolver um *framework* para aplicações que manipulam recursos simples em dispositivos limitados interligados a redes limitadas, incluindo desde aplicações para o monitoramento de sensores até o controle de atuadores, também incluindo o gerenciamento de dispositivos que compõe a rede (SHELBY et al, 2010). O CoAP faz parte deste *framework* e em 2014 tornou-se um RFC.

De acordo com o RFC 7252 (SHELBY et al, 2014), o CoAP é um protocolo de transferência voltado para dispositivos e redes restritas (considerando redes em que

a taxa de perda de pacote é alta e dispositivos com restrições geralmente em memória RAM e ROM), sendo projetado para aplicações M2M e considerado simples, leve e de baixo custo pela *Open Mobile Alliance* (BORMANN, 2016).

O CoAP provê um modelo de interação requisição/resposta entre os *endpoints*[1] das aplicações (vide figura 12), suporta descoberta embutida de serviços e recursos e inclui conceitos chave da *Web* como por exemplo Identificadores Uniformes de Recursos (URIs- *Uniform Resources Identifiers*) e tipos de mídia comuns na Internet. Além disso, o protocolo foi projetado para interagir com o HTTP para integração com a *Web*. Abaixo estão algumas características descritas no RFC 7252 (SHELBY et al, 2014):

- Protocolo *Web* que cumpre com os requerimentos M2M em ambientes restritos;
- Troca de mensagens assíncrona;
- Suporte à URI e *Content-Type*;
- Capacidades simples de *proxy* e *caching* (armazenamento temporário);
- Mapeamento HTTP que permite que proxies possam prover acesso aos recursos do CoAP via HTTP de maneira uniforme;
- Interligação segura para *Datagram Transport Layer Security* (DTLS);
- Possui construção baseada na utilização do *User Datagram Protocol* (UDP) com confiabilidade opcional suportando requisições tanto *unicast* quanto *multicast*;
- Suporte aos métodos *GET, POST, PUT, DELETE*.



[1] Um *endpoint* é um endereço de requisição de um serviço remoto.

Figura 12 – Requisição CoAP
Fonte: Elaborado pelo autor

O CoAP define quatro tipos de mensagens: *Confirmable*, *Non-confirmable*, *Acknowledgment* e *Reset*, além do tipo *Empty*. As quais podem ser descritas como (RFC 7252, 2014, p. 8):

- *Confirmable*: mensagens que exigem confirmação de entrega pelo destino.
  - Desse modo, quando não há perda de pacotes, cada mensagem enviada gera uma confirmação do tipo *Acknowledgment* ou *Reset*. Utilizado em ambientes onde não deve haver perda de pacotes.
- *Non-confirmable*: mensagens que não exigem confirmação de recebimento pelo destino.
  - Esse tipo de mensagem é mais usual em ambientes com grande fluxo de mensagens, onde a perda de algumas não gera prejuízo.
- *Acknowledgment*: mensagens de confirmação do recebimento de outras mensagens.
- *Reset*: mensagens que indicam que uma mensagem (*Confirmable* ou *Non-confirmable*) foi recebida, mas por algum motivo ela não foi ser processada corretamente.
- *Empty*: uma mensagem que contêm somente as informações do cabeçalho.

3.6.1 Formato da mensagem CoAP

A RFC 7252 (SHELBY et al, 2014), define a estrutura de uma mensagem CoAP, figura 12, com os seguintes campos:

- Versão (Ver): inteiro de 2 bits que indica o número da versão do CoAP;
- Tipo (T): inteiro de dois 2 que indica o tipo da mensagem, podendo ser *Confirmable* (0), *Non-confirmable* (1), *Acknowlegment* (2) ou *Reset* (3);
- Largura do *token* (TKL): inteiro de 4 bits que indica o tamanho do *token*;
- Código (*Code*): inteiro de 8 bits que indica o tipo do fluxo mensagem, se ela é um método de requisição ou um código de resposta;
- ID da mensagem (*Message* ID): inteiro de 16 bits usado para verificar a duplicação de mensagens e relacionar mensagens de confirmação com as de origem.

- *Token*: identificador gerado pelo cliente e enviado junto com as requisições. O servidor deve reenviá-lo na resposta da requisição.
- Opções (*Options*): Conjunto opcional de valores de configuração do pacote CoAP;
- Carga útil (*Payload*): campo opcional com dados do corpo requisição ou resposta.

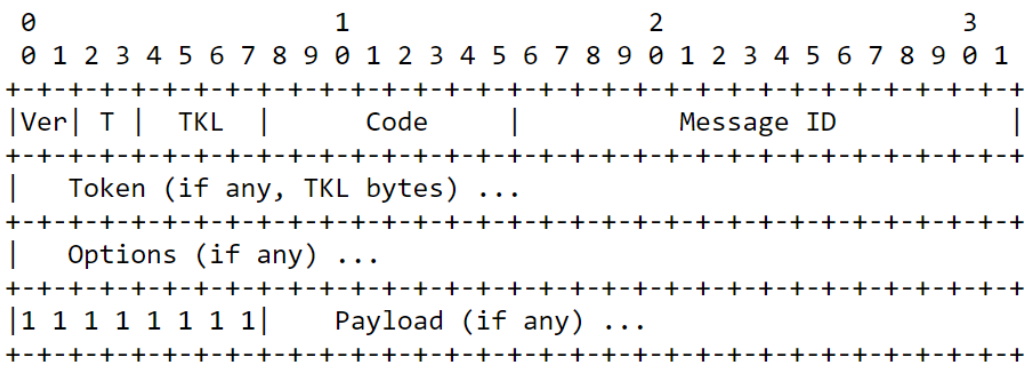


Figura 13 – Formato de mensagem CoAP

Fonte: RFC 7252, 2014

4 MICROCONTROLADOR ESP32

O ESP32, figura 14, é um microcontrolador com suporte a WIFI fabricado pela empresa chinesa Espressif e lançado no mercado internacional em 2014. Segundo Espressif (2017), pode-se citar entre suas principais características:

- *System-On-Chip* com Wi-Fi embutido;
- Interfaces: GPIO, SPI, LCD, UART, I2C, I2S, Camera interface, IR, pulse counter, LED PWM, USB, OTG 1.1, ADC, DAC, sensor de toque, sensor de temperatura;
- CPU operando em 80MHz, com possibilidades de operar em até 240MHz;
- Arquitetura de 32 bits;
- 16 KBytes de SRAM em RTC;
- 320 KBytes de SRAM;
- 128 KBytes de ROM;
- Memória Flash SPI de 4 MBytes;
- Núcleo Xtensa LX7;
- Suporte a redes 802.11 b/g/n;

- Resistência a temperaturas entre -40ºC e +85ºC;
- Suporte WIFI em modo ponto de acesso ou estação.

O ESP32 possui as seguintes certificações do Mercado industrial: RoHS, REACH, FCC, CE-RED e SRRC (ESPRESSIF, 2020).

Figura 14 – ESP32-WROOM
Fonte: DIGIKEY, 2020

4.1 Kit de desenvolvimento

O microcontrolador possui diversos modelos de desenvolvimento, também conhecidos como kits de desenvolvimento. Esses kits permitem que o aprendizado e o trabalho com protótipos ocorram mais facilmente, pois, normalmente, já possuem o que é necessário para funcionamento imediato ou para simples montagem.

Para este trabalho foi escolhido o módulo NodeMCU 32, modelo de desenvolvimento construído a partir do ESP32 WROOM-E. A figura 15 apresenta o NodeMCU com suas respectivas portas de entrada e saída (I/O – Input/Output).

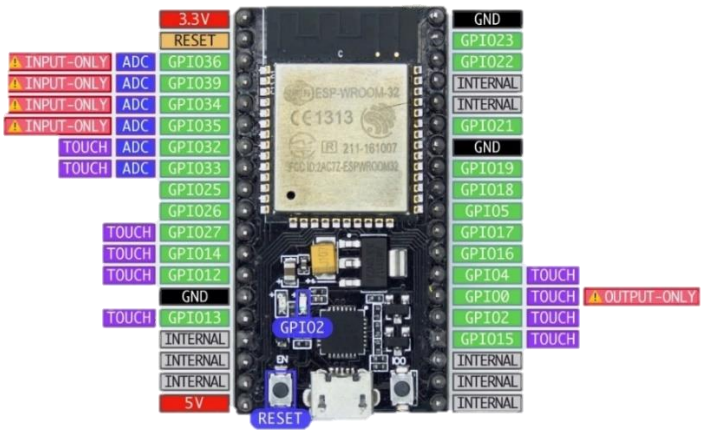


Figura 15 – NodeMCU 32
Fonte: GITHUB, 2019

Esse modelo de desenvolvimento pode ser programado na interface de desenvolvimento (IDE - *Integrated Development Environment*) do Arduino, que integra diversos módulos de programação que facilitam o trabalho de prototipagem e ensaios de funcionamento.

4.2 IDE ARDUINO

A linguagem de programação utilizada na IDE do Arduino é baseada na linguagem de programação C++. Na IDE é possível adicionar módulos de programação para componentes usuais como servos motores e sensores, além de ser possível compilar o código diretamente para o ESP32 fazendo o reconhecimento do modelo.

5 DESENVOLVIMENTO

Neste capitulo será abordado o desenvolvimento prático do trabalho, contemplando a elaboração do *firmware*, o uso do kit de desenvolvimento, o projeto do esquemático do circuito impresso pretendido e a prototipação do mesmo.

5.1 Placa de Circuito Impresso

Para desenvolver a placa de circuito impresso (PCB - *printed circuit board*) pleiteada foi utilizado o software gratuito EasyEDA, disponível em versão *web* e versão aplicativo para Windows. O *software* permite a criação do desenho esquemático e do modelo 3D em formato que pode ser utilizado para construção física do modelo.

Para o presente trabalho, foi estruturada a base de funcionamento do ESP32-Wroom 32E, figura 16, seguindo as recomendações da ESPRESSIF (2020). Desta forma é garantido o funcionamento do mesmo.

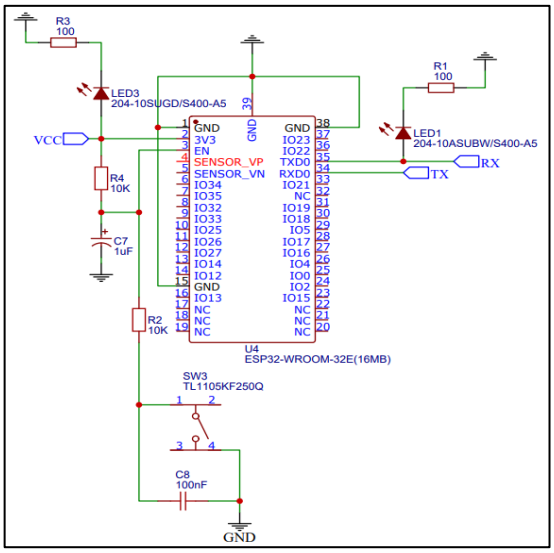


Figura 16 – Base ESP32 Wroom 32E

Fonte: Elaborado pelo autor

Para realizar a comunicação com os dispositivos, foi escolhido o padrão de conexão DB9 e utilizado um componente capaz de converter do padrão RS232 (tipicamente 12 Volts) para o padrão de lógica transistor-transistor (TTL-*Transistor-Transistor Logic*) (tipicamente 3.3 Volts). Como pode ser visto na figura 17, o MAX3232 permite essa conversão e diferente do MAX232, ele aceita alimentação de 3.3 Volts.

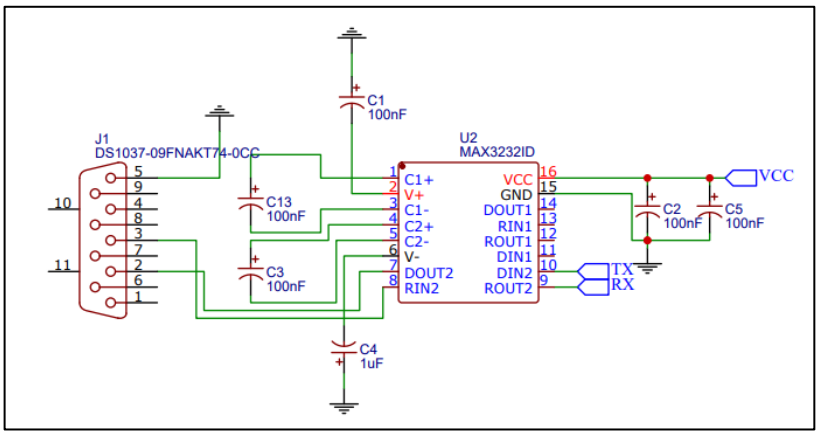


Figura 17 – Circuito MAX3232

Fonte: Elaborado pelo autor

Por último, foi desenvolvida a alimentação do circuito, conforme figura 18, prevendo uma fonte externa de 5 a 8 volts com um conector do tipo P4 de 5,5 x 2,2 mm. A escolha da fonte externa se dá através da possibilidade de adaptação da alimentação em cenários de diferentes tensões de rede, de tal forma é possível utilizar

a mesma placa para tensões de rede de 110V, 220V, entre outras, trocando apenas a fonte externa.

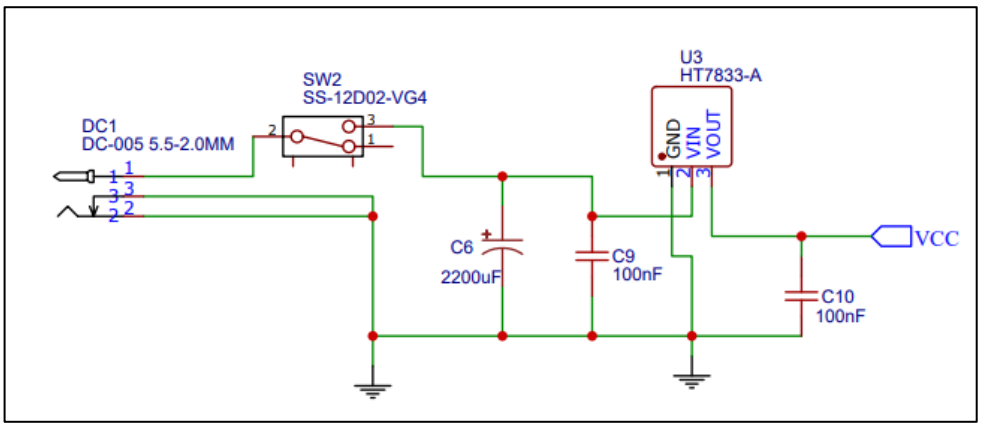


Figura 18 – Circuito Alimentação

Fonte: Elaborado pelo autor

A tabela 5 detalha a função de cada componente utilizado, salvo os resistores que servem para regular a corrente dos Leds e do Reset.

TABELA DE COMPONENTES	
Identificação	**Função**
LED3	Indicar que o circuito está alimentado
LED1	Indicar que há comunicação entre o circuito e o aparelho conectado
SW3	Botão de RESET do controlador
J1	Permitir a Conexão física entre os dispositivos
U2	Realizar a conversão RS232 para TTL
DC1	Permitir a alimentação do circuito
SW2	Permitir ligar/desligar o circuito
U3	Regular a tensão de alimentação
U4	Executar o *firmware*
Nota: os capacitores são requisitos para o funcionamento conforme *datasheet* de cada componente principal (MAX3232, HT7833-A, ESP32-WROOM-32E)	

Tabela 5 – Tabela de componentes

Fonte: Elaborado pelo autor

Em sequência foi desenvolvido o desenho para construção da PCB (figura 19.a) e o modelo 3D da placa (figura 19.b).

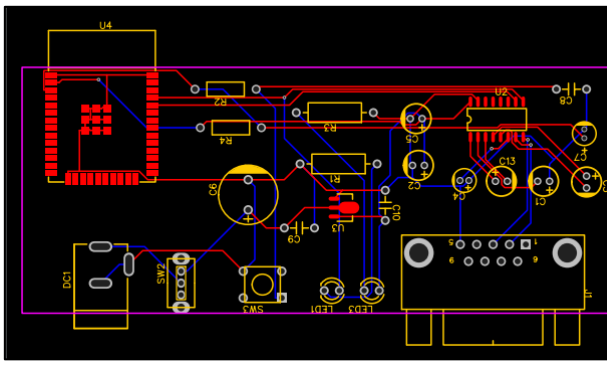


Figura 19.a – Modelo 3D

Fonte: Elaborado pelo autor

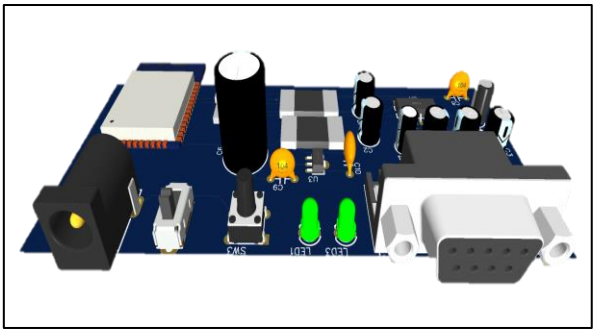


Figura 19.b – Modelo 3D

Fonte: Elaborado pelo autor

Foi desenvolvido um circuito, figura 20 para caso seja mais interessante o uso do RS485, (dispositivos que utilizam este tipo de protocolo no lugar do RS232). Neste modelo foi utilizado o MAX3485 no lugar do MAX3232 com suas devidas peculiaridades.

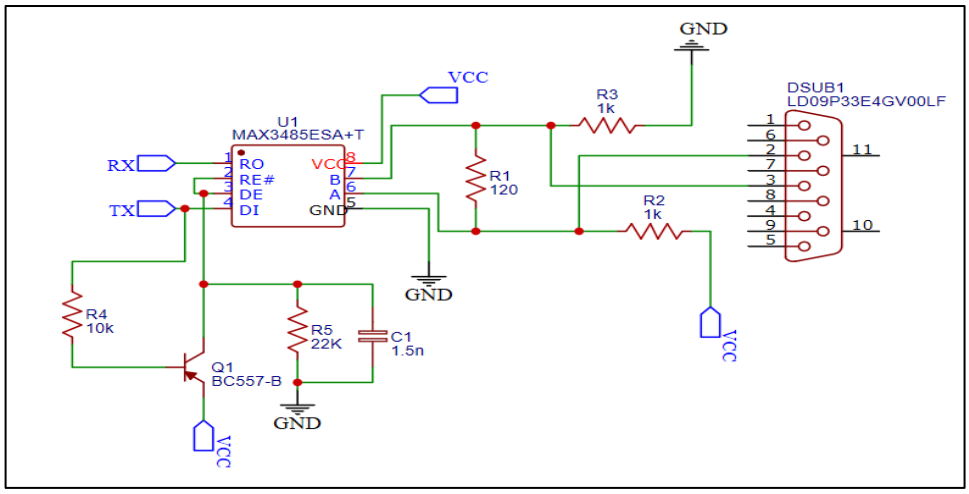


Figura 20 – Circuito MAX3485

Fonte: Elaborado pelo autor

A prototipação da PCB foi realizada conforme figura 21. A diferença entre o modelo apresentado na figura 19.b se dá pela retirada da chave para ligar/desligar a placa e a utilização de capacitores cerâmicos no lugar dos capacitores eletrolíticos do MAX3232, o que pode acorrer conforme a fabricante (TEXAS INSTRUMENTS, 2021).

Figura 21 – Circuito protótipo

Fonte: Elaborado pelo autor

5.1.1 Uso do Kit de Desenvolvimento

Para o desenvolvimento foi utilizado o NodeMCU. O mesmo foi conectado ao computador utilizando a entrada USB-A, que permite a comunicação serial e alimentação do mesmo, figura 22. Não foi, portanto, necessário a utilização de nenhum conversor ou fonte de alimentação externa.

O kit de desenvolvido foi utilizado durante a fase de construção do sistema por permitir que as alterações inerentes à evolução tanto do *hardware* quanto do *firmware* fossem realizadas sem interações diretas, o que poderia conduzir a equívocos e/ou dificuldades de identificação da origem dos erros e, consequentemente, na ação de correção dos mesmos. Nesse contexto, o NodeMCU assume a função do *hardware* pretendido com a garantia do funcionamento esperado, além de facilitar a carga e os testes do *firmware*.

Figura 22 – Kit de desenvolvimento conectado

Fonte: Elaborado pelo autor

5.2 Firmware

No desenvolvimento do programa embarcado no NodeMcu foram utilizadas as bibliotecas ModbusMaster[2] e CoAP-simple-library[3], ambas disponíveis como projetos de código fonte aberto no site do github. A biblioteca ModbusMaster encapsula e disponibiliza, através de uma interface de programação, métodos para leitura e escrita em dispositivos ModBus RTU. CoAP-simple-library é uma biblioteca que auxilia na criação de um servidor CoAP e na interceptação e controle das requisições e respostas do protocolo.

Para iniciar o desenvolvimento do código adicionou-se as bibliotecas utilizadas, como mostra a figura 23, a biblioteca WiFi faz parte da IDE e não precisa de fontes externas.

```
#include <ModbusMaster.h>
#include <WiFi.h>
#include <WiFiUdp.h>
#include <WiFiAP.h>
#include <coap-simple.h>
```

Figura 23 – Chamada das bibliotecas utilizadas

Fonte: Elaborado pelo autor

O processo foi seguido pela inicialização das variáveis utilizadas e das definições das bibliotecas, assim como a nomeação das funções criadas, figura 24.

```
ModbusMaster node;// Definir objeto Modbus

const char* ssid     = "apteste"; //Definição do nome da rede
const char* password = "12345678"; //Definição da senha da rede

//Nomear função para leitura de coils
void callback_readcoils(CoapPacket &packet, IPAddress ip, int port);
//Nomear função para leitura de Holding Registers
void callback_readholding(CoapPacket &packet, IPAddress ip, int port);

WiFiUDP udp; //iniciar WI-FI como UDP
Coap coap(udp); //Definir CoAP UDP

//Inicio definição de variaveis globais
 static uint32_t i;
 uint8_t result;
 uint16_t data;
 char str[10];
//Fim definição de variaveis globais
```

Figura 24 – Chamada das bibliotecas utilizadas

[2] Biblioteca disponível em: https://github.com/4-20ma/ModbusMaster

[3] Biblioteca disponível em: https://github.com/hirotakaster/CoAP-simple-library

Fonte: Elaborado pelo autor

Após a etapa de criação de variáveis e protótipos das funções foi desenvolvido método "*setup()*", função essa que é requisito da IDE e que é chamada apenas quando o sistema é iniciado. Nesta função encontram-se as inicializações dos processos de comunicação serial e *wireless*.

Com a função "coap.*server*" são registradas as interações com o servidor CoAP, dessa maneira quando for enviada a requisição *"CoilStatus"* para o servidor ele executará a função "*callback_readcoils*". Do mesmo modo se for enviado *"HoldingRegisters"* será chamada a função "*callback_readholding*".

Para iniciar o servidor CoAP é necessário usar a função "coap.*start*", ressaltando que a função só deve ser utilizada após a inicialização do ponto de acesso. O objeto ModbusMaster foi atribuído à variável "node". Ao chamar o método "node.*begin*(1, Serial)" é definido que as operações ModBus serão realizadas com o endereço de escravo 1, na porta serial inicializada na chamada "Serial.begin". A figura 25 contém o código utilizado para a função "*setup()*".

```
void setup() {

  Serial.begin(9600, SERIAL_8N1); //Iniciar Serial
  WiFi.softAP(ssid, password); //Iniciar Ponto de Acesso
  IPAddress myIP = WiFi.softAPIP();

  //Chamar função de leitura de Coils
  coap.server(callback_readcoils, "CoilStatus");
  //Chamar função de leitura de Holding Registers
  coap.server(callback_readholding, "HoldingRegisters");

  coap.start(); //Inicair servidor CoAP

  //Comunicação com Modbus escravo com ID1 através da serial
  node.begin(1, Serial);
}
```

Figura 25 – Chamada das bibliotecas utilizadas

Fonte: Elaborado pelo autor

A IDE requer a função "*loop()*", responsável por rodar o programa principal infinitamente. Nesta função foi inserido o "*coap.loop()*", que faz com que o servidor continue esperando o comando de leitura, figura 26.

```
void loop(){
  delay(500);
  coap.loop();
}
```

Figura 26 – Chamada das bibliotecas utilizadas

Fonte: Elaborado pelo autor

A função para leitura de um *coil* é descrita na figura 27. Nela é realizada a leitura do *coil* de endereço 1, valor fixo para os ensaios. Esse valor é transferido para a variável “data” e em seguida é enviado através do CoaP. Também é visível na figura 27 a leitura do *holding register*, que possui estrutura semelhante à leitura do *coil*.

```
void callback_readcoils(CoapPacket &packet, IPAddress ip, int port) {

  char req[packet.payloadlen + 1]; //Define variavel local
  memcpy(req, packet.payload, packet.payloadlen); //Copia registro para variavel
  req[packet.payloadlen] = NULL;//Limpa parte do regsitro

  // Leitura do Coil de endereço 1
  result = node.readCoils(atoi(req), 1);
  // Caso a leitura seja concluida
  if (result == node.ku8MBSuccess)
  {
//inicio da "conversão"
      data = node.getResponseBuffer(0);
      bzero(str, 10);
      sprintf(str, "%d", data);
//Fim da "conversão"
 //envia o coil lido através do CoAP
      coap.sendResponse(ip, port, packet.messageid, str);
  }else{
 //Em caso negativo envia um valor vazio
     coap.sendResponse(ip, port, packet.messageid, "");
  }
}

void callback_readholding(CoapPacket &packet, IPAddress ip, int port) {

  char req[packet.payloadlen + 1];
  memcpy(req, packet.payload, packet.payloadlen);
  req[packet.payloadlen] = NULL;

  String message(req);

  result = node.readHoldingRegisters(atoi(req), 1);
  if (result == node.ku8MBSuccess)
  {
      data = node.getResponseBuffer(0);
      bzero(str, 10);
      sprintf(str, "%d", data);
      coap.sendResponse(ip, port, packet.messageid, str);
  }else{
     coap.sendResponse(ip, port, packet.messageid, "");
  }
}
```

Figura 27 – Comunicação CoAP-Modbus

Fonte: Elaborado pelo autor

Para viabilizar a leitura de endereços diversos é necessário a criação da interface para configuração dos parâmetros, que pode ser via HTTP, e assim enviar para a função “node.*readHoldingRegisters*” ou “node.*readCoils*” o endereço do ponto de leitura desejado, figura 28, onde “numero_ponto” é a variável que receberia o endereço do ponto que se deseja ler.

```
result = node.readHoldingRegisters(atoi(req), numero_ponto);
```

Figura 28 – Função com ponto variável

Fonte: Elaborado pelo autor

6 ENSAIOS E RESULTADOS

Para simular o dispositivo ModBus foi utilizado o programa Dexter, disponível no site sourceforge.net de forma gratuita e com o código fonte aberto. Para configurá-lo deve-se selecionar uma porta serial no computador e montar o mapa de pontos das entradas e saídas que serão utilizadas. Durante os testes foram configurados 8 pontos do tipo *Holding Registers* (16 bits), figura 29, e 8 pontos do tipo *Coils* (1 bit).

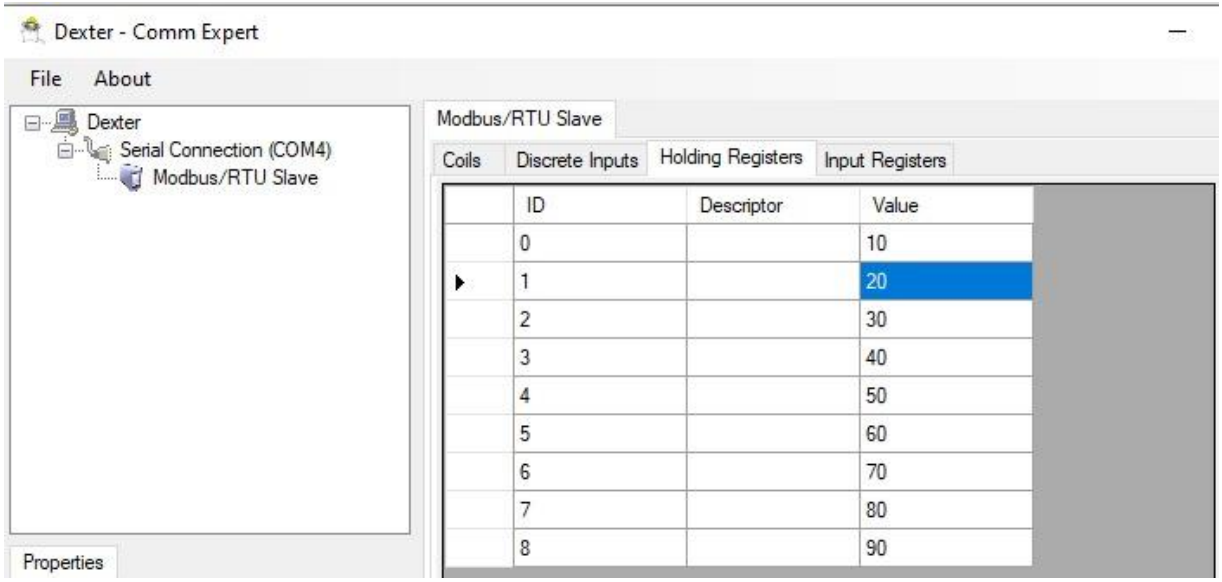


Figura 29 – *Holding Registers* Dexter

Fonte: Elaborado pelo autor

Para realizar as requisições CoAP, figura 30, no servidor iniciado no NodeMCU foi utilizado o comando “coap-client” que faz parte do *software* libcoap[4], compilado em uma máquina virtual com o sistema operacional Linux instalado.

[4] Disponível em: https://libcoap.net/install.html

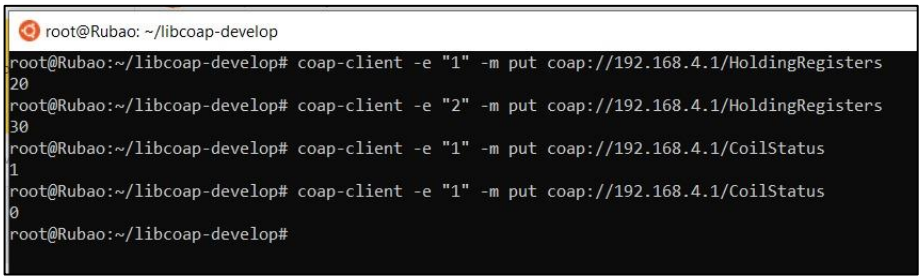


Figura 30 – Requisição e respota CoAP

Fonte: Elaborado pelo autor

O computador foi conectado ao NodeMCU tanto pelo cabo USB (comunicação serial) quanto pelo ponto de acesso criado (comunicação *wireless*), figura 31. Assim foi possível realizar a leitura Modbus pela porta serial e obter a resposta CoAP pelo *wireless*.

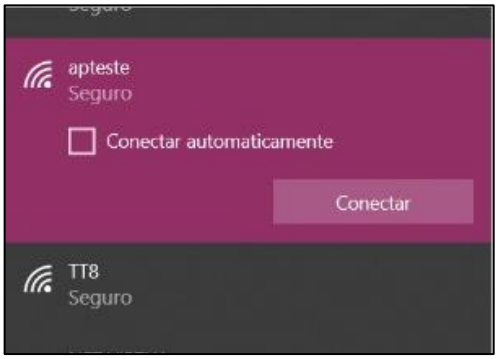


Figura 31 – Ponto de acesso

Fonte: Elaborado pelo autor

A figura 32 mostra o *log* de comunicação entre o NodeMcu e o Dexter através da porta serial COM4, evidenciando os bytes enviados e recebidos durante as sessões Modbus.

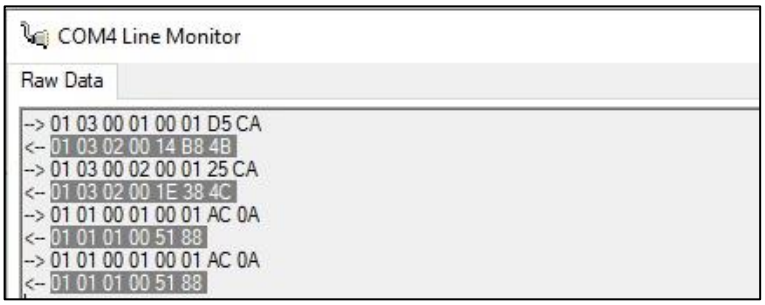


Figura 32 – *Log* de comunicação

Fonte: Elaborado pelo autor

Os ensaios foram realizados novamente após a gravação do *firmware* na PCB desenvolvida, que será detalhada no capitulo 7. Os testes demonstraram resultados positivos quanto a comunicação e funcionamento do hardware. De tal forma, os bytes

enviados pelo simulador foram recebidos pelo dispositivo, traduzidos para CoAP, e transmitidos via wireless sem perda de informação.

7 GRAVAÇÃO DO *FIRMWARE*

Após a realização dos testes, iniciou-se a gravação do ESP-32, antes do mesmo ser soldado na PCB. Conforme figura 33, o mesmo foi conectado no computador utilizando uma placa de gravação.

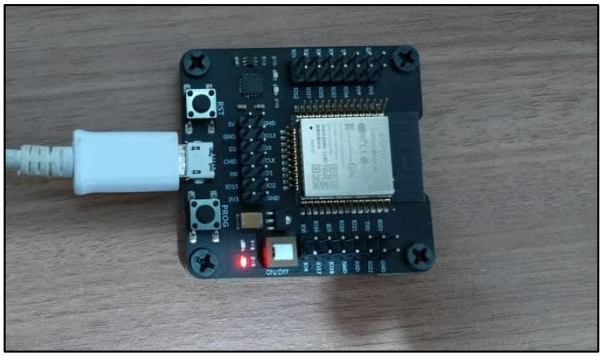


Figura 33 – *Placa de gravação*

Fonte: Elaborado pelo autor

Foi carregado outro *firmware* que permite que o dispositivo seja gravado via conexão remota, verificado o funcionamento do mesmo, ele foi integrado ao *firmware* descrito no capítulo 5.2.

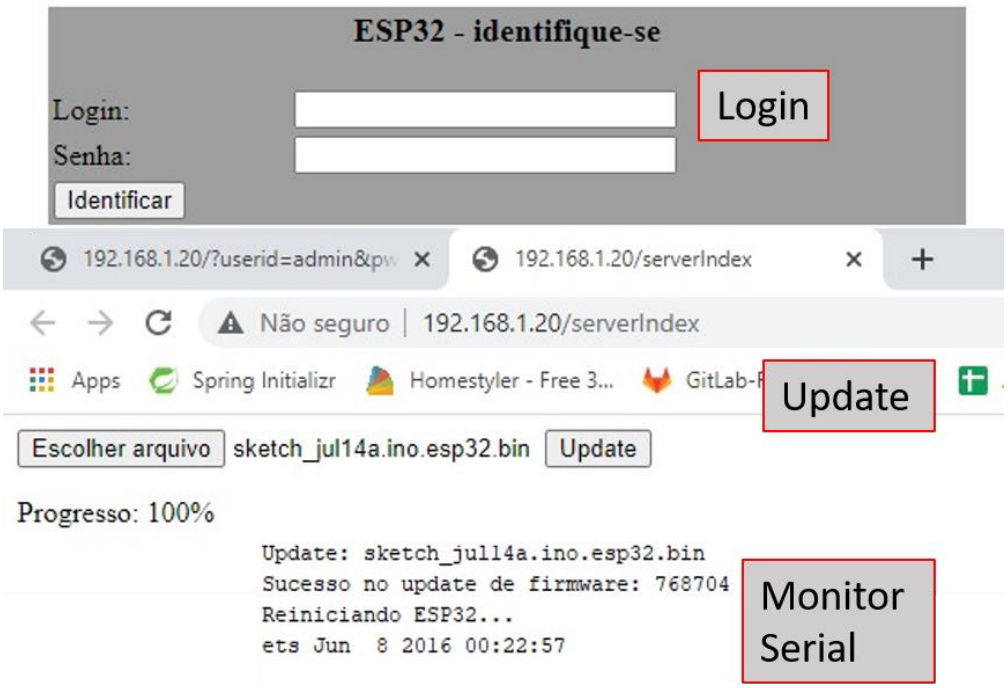


Figura 34 – Gravação remota

Fonte: Elaborado pelo autor

A figura 34 mostra o processo de *upload*, que consiste em:

- Conectar ao endereço de Ip do controlador;
- Escolher o arquivo salvo em formato .bin (binário);
- Carregar o arquivo.

O monitor serial serve para comprovar o carregamento do *firmware*, que também pode ser realizado utilizando o conector DB9 ligado a um conversor Serial-USB conectado ao computador, figura 35.

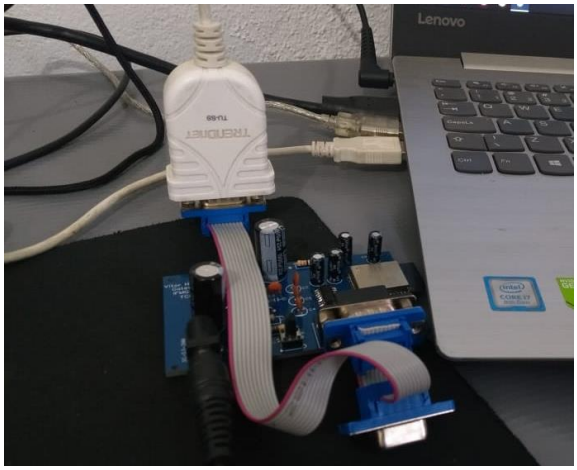


Figura 35 – Gravação com conversor Serial-USB

Fonte: Elaborado pelo autor

8 PROBLEMAS ENCONTRADOS

O desenvolvimento do trabalho apresentou alguns obstáculos, entre eles a falta de documentação das bibliotecas utilizadas, fazendo com que fosse necessário o estudo do código fonte criado em C++ pelos respectivos autores. Foi observado que para entendimento do código fonte é crucial o conhecimento, no mínimo intermediário, dos protocolos CoAP e Modbus RTU, assim como da linguagem de programação C++.

Outro ponto de dificuldade foi encontrar um cliente CoAP para realizar os testes. A princípio foi utilizado um *plugin* para o navegador Firefox 5.6 chamado Cooper, que apresentou falhas a partir do segundo teste realizado. Por esse motivo foi utilizado o pacote de *softwares* da Libcoap, que apresentou estabilidade.

A biblioteca modbus-esp8266[5] apresentou erro de execução, reinicializando o *hardware* durante a comunicação com o escravo Modbus. Para solucionar o problema foi utilizado a biblioteca ModbusMaster.

Em um dos computadores utilizados durante o desenvolvimento do *firmware* apresentou a necessidade de acionar o botão *boot* do NodeMCU para carregamento do código. O problema não ocorreu no outro computador utilizado.

[5] Disponível em: https://github.com/emelianov/modbus-esp8266

9 Conclusão

Para o desenvolvimento deste trabalho foi construído um *gateway* de comunicação entre os protocolos CoAP e ModBus RTU utilizando-se o kit de desenvolvimento NodeMCU do ESP32.

Este trabalho dividiu-se em duas etapas. Na primeira realizou-se uma revisão bibliográfica sobre conceitos de redes industriais e como operam diversos protocolos, com ênfase no Modbus, com forte presença no mercado de automação, e o CoAP, que surge com a promessa de operar em dispositivos com poucos recursos e redes restritas. Na segunda parte utilizou-se o NodeMCU e o simulador de ModBus escravo Dexter para construir e testar o gateway de comunicação

O *firmware* foi desenvolvido em linguagem C++ e utilizando-se a IDE Arduino. As bibliotecas do Arduino para os referidos protocolos facilitaram a programação e deram robustez ao código desenvolvido. Durante os testes o *firmware* se mostrou funcional para a integração Modbus/CoAP e não apresentou nenhuma falha de comunicação. O circuito da PCB foi desenhado no *software* EasyEDA e fabricado utilizando-se as recomendações dos fabricantes dos componentes necessários para os circuitos da placa.

O resultado positivo dos testes leva a concluir que é possível utilizar o circuito do ESP32 para construir um gateway de comunicação entre ModBus e CoAP, permitindo que aparelhos e maquinários sem acesso nativo às redes sem fio, mas com ModBus RTU, possam se integrar à crescente gama de dispositivos IoT.

Cabe a trabalhos futuros validar o *firmware* e o *hardware* com dispositivos ModBus reais. Testes de estresse e implementação de mecanismos de segurança na comunicação CoAP fazem parte de esforços futuros, assim como o desenvolvimento de uma interface *web* para configurar o *wireless,* os dados do escravo e porta serial.

BIBLIOGRAFIA

ANYBUS. **Industrial Ethernet is now bigger than fieldbuses**. Halmstad: [s.n.], 2018. Disponível em: https://www.anybus.com/about-us/news/2018/02/16/industrial-ethernet-is-now-bigger-than-fieldbuses> Acesso em: 10 set 2020

ATZORI, L.; IERA, A.; MORABITO, G. **The Internet Of Things**: a survey. [Cagliari]: [s.n.], 2010. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128610001568>. Acesso em: 20 fev 2020

BANDYOPADHYAY, S. et al. **Role Of Middleware For Internet Of Things**: a study. Kolkata: IJCSES, 2011. Disponível em: < https://pdfs.semanticscholar.org/85e9/6085e989d12b54183f989549af3cbfca1c92.pdf >. Acesso em: 15 fev 2020

BORMANN, C. ***CoAP***. [S.l]: [s.n.], 2016. Disponível em: < http://coap.technology/ > Aceso em: 16 fev 2020

BOSWARTHICK, D.; ELLOUMI, O.; HERSENT, O. **M2M Communications: a systems approach**. New York: John Wiley & Sons, 2012.

CISCO. **Cisco Visual Networking Index Predicts Near-Tripling of IP Traffic by 2020**. San Jose,US-CA: [s.n.], 2016. Disponível em: < https://newsroom.cisco.com/press-release-content?type=press-Release&articleId=1771211 > Acesso em: 15 fev 2020

CISCO. **Cisco Annual Internet Report (2018–2023) White Paper**. San Jose,US-CA: [s.n.], 2020. Disponível em: < https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html#Trends > Acesso em: 15 set 2020

DIGIKEY. **ESP32-WROOM-32**. [S.l]: [s.n.], 2020. Disponível em: < https://www.digikey.co.th/product-detail/en/espressif-systems/ESP32-WROOM-32/1904-1010-1-ND/8544305> Acesso em: 19 jun 2020

DIREITO DA COMUNICAÇÃO. **A internet das coisas (IOT)***, é tendência de mercado em aplicações de internet*. [S.l]: [s.n.], 2018. Disponível em: < https://direitodacomunicacao.com/en/artigos/internet-das-coisas-no-brasil-a-consulta-publica-da-anatel-sobre-a-regulamentacao-das-aplicacoes-de-iot-e-comunicacao-maquina-a- /> Acesso em: 19 jun 2020

ESPRESSIF. **ESP32-S2-WROOM & ESP32-S2-WROOM-I Dataseet***.* Pudong:[s.n.], 2020. Disponível em: < https://www.espressif.com/sites/default/files/documentation/esp32-s2-wroom_esp32-s2-wroom-i_datasheet_en.pdf >. Acesso em: 14 fev 2020

FIELDING, R.; et al. **Hypertext Transfer Protocol**. [S.l]: [s.n.], 1999. Disponivel em: <https://tools.ietf.org/html/rfc2616> Acessada em 20/07/2020

GITHUB. **NodeMCU ESP32**. [S.l]: [s.n.], 2019. Disponível em: < https://github.com/esphome/esphome-docs/blob/current/devices/nodemcu_esp32.rst > Acesso em: 19 jun 2020

GUBBI, J.; et al. **Internet Of Things (IoT)***:* a vision, architectural elements and directions. Elsevier: [s.n], 2013. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X13000241>. Acesso em: 15 mar 2020

HALSALL, Fred. **Data Communications, Computer Networks and Open Systems**. 4 ed. Harlow: Addison-Wesley, 1996

HMS. **Industrial network market shares 2019 according to HMS**. Halmstad: [s.n.], 2019. Disponível em: < https://www.hms-networks.com/news-and-insights/news-from-hms/2019/05/07/industrial-network-market-shares-2019-according-to-hms> Acesso em: 15 set 2020

HMS. **Industrial network market shares 2020 according to HMS Networks**. Halmstad: [s.n.], 2020. Disponível em: https://www.hms-networks.com/news-and-insights/news-from-hms/2020/05/29/industrial-network-market-shares-2020-according-to-hms-networks> Acesso em: 15 set 2020

LUGLI, A.; SANTOS, M.; FRANCO, L. **REDES ETHERNET INDUSTRAIS:** VISÃO GERAL. [S.l]: [s.n.], 200?. Disponível em: < https://docplayer.com.br/931007-Redes-ethernet-industrais-visao-geral.html>. Acesso em: 10 out 2021

LUGLI, A.; SANTOS, M.; FRANCO, L. **UMA VISÃO DO PROTOCOLO INDUSTRIAL PROFINET E SUAS APLICAÇÕES**. Brasil: [s.n.], 2021. Disponível em: < https://www.profibus.org.br/artigo-tecnico-detalhe.php?id=uma-visao-do-protocolo-industrial-profinet-e-suas-aplicacoes>. Acesso em: 10 out 2021

MHANNA. **Introduction to Fieldbus Systems**. Pittsyburgh: [s.n.], 200?. Disponível em: < http://people.cs.pitt.edu/~mhanna/Master/Introduction.pdf> Acesso em: 15 set 2020

MODBUS ORG. **MODBUS over Serial Line**: Specification and Implementation Guide V1.02. [s.n.], 2006. Disponível em: < https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf>.Acesso em: 19 out 2020

MODBUS ORGANIZATION. **MODBUS MESSAGING ON TCP/IP IMPLEMENTATION GUIDE V1.0b**. [s.n.], 2006. Disponível em: https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf>.Acesso em: 19 out 2020

MODBUS ORGANIZATION. **Modbus FAQ. Hopkinton**: [s.n.], 2017. Disponível em: < http://www.modbus.org/faq.php>.Acesso em: 19 mar 2020

POLSONETTI, C. **Understand The Difference Between IoT And M2M**: What you want to accomplish determines which approach is better to use. Schaumburg:[s.n], 2014. Disponivel em:<http://www.chemicalprocessing.com/articles/2014/understand-the-difference-between-iot-and-m2m>. Acesso em: 18 mar 2020

POSTEL, J. **User Datagram Protocol**. [S.l]: [s.n.], 1980. Disponivel em: <https://tools.ietf.org/html/rfc768> Acesso em: 20/07/2020

PRESS,G. **Internet of Things By The Numbers**: Market Estimates And Forecasts. [S.l.]:Forbes,2014. Disponível em: <https://www.forbes.com/sites/gilpress/2014/08/22

/internet-of-things-by-the-numbers-market-estimates-and-forecasts/#614e17db919> Acesso em: 18 mar 2020

PROTOCOLO. In: Oxford Languages and Google, Dicionário Online de Português. Oxônia: Oxford University Press, 2020. Disponível em: < shorturl.at/dhEU5 >. Acesso em: 27/10/2020.

RFC 793. **Transmission Control Protocol**. [S.l]: [s.n.], 1981. Disponivel em: <https://tools.ietf.org/html/rfc793> Acesso em: 20/07/2020

SAMPAIO, I. **Conceitos e Modelos de Comunicação**. Associação Nacional dos Programas de Pós-Graduação em Comunicação, [S.l.]: [s.n], [2001]. Disponível em: < http://www.compos.org.br/data/biblioteca_1275.pdf >. Acesso em: 20 de jul. de 2020.

SHELBY, Z.; et al. **The Constrained Application Protocol (CoAP)**. Bremen:[s.n.], 2014. Disponível em: < https://tools.ietf.org/html/rfc7252 >. Acesso em: 16 mar 2020

SHELBY, Z; et al. **The Constrained Application Protocol (CoAp)**: draft-shelby-core-coap-01. Bremen:[s.n.], 2010. Disponível em: <https://tools.ietf.org/html/draft-shelby-core-coap-01>.Acesso em: 20 jul 2020

TEXAS INSTRUMENTS. **MAX3232 3-V to 5.5-V Multichannel RS-232 Line Driver and Receiver With ±15-kV ESD Protection**. Texas: Dallas, 2021. Disponível em: <https://www.ti.com/lit/ds/symlink/max3232.pdf?ts=1626648531657&ref_url=https%253A%252F%252Fwww.google.ca%252F>. Acesso em: 20 jul 2021

VENERI, G.; CAPASSO, A. **Hands-On Industrial Internet of Things**: Create a powerful Industrial IoT infrastructure using Industry 4.0. 1 ed. Birmingham: Packt Publishing, 2018.