

Thiago de Souza Alves

**Framework para Controle de Sistemas
Industriais via MQTT**

Uberlândia
2022

Thiago de Souza Alves

Framework para Controle de Sistemas Industriais via MQTT

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de Bacharel em Engenharia Mecatrônica.

Área de concentração: Engenharia Mecatrônica

Orientador: José Jean-Paul Zanlucchi Souza Tavares

Uberlândia
2022

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

A474 Alves, Thiago de Souza, 1999-
2022 Framework para Controle de Sistemas Industriais via
MQTT [recurso eletrônico] / Thiago de Souza Alves. -
2022.

Orientador: José Jean-Paul Zanlucchi Souza Tavares.
Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Uberlândia, Graduação em
Engenharia Mecatrônica.

Modo de acesso: Internet.
Inclui bibliografia.

1. Mecatrônica. I. Tavares, José Jean-Paul Zanlucchi
Souza,11 -, (Orient.). II. Universidade Federal de
Uberlândia. Graduação em Engenharia Mecatrônica. III.
Título.

CDU: 621.03

Thiago de Souza Alves

Framework para Controle de Sistemas Industriais via MQTT

Trabalho aprovado. Uberlândia, 29 de agosto de 2022:

**José Jean-Paul Zanoluchi de Souza
Tavares
Orientador**

**Professor
José Reinaldo Silva**

**Professor
Roberto Mendes Finzi Neto**

**Uberlândia
29 de agosto de 2022**

*Este trabalho é dedicado aos meus pais,
os primeiros a me ensinar que as coisas tem um porque de acontecer,
a crença mais louca e inexplicável que nos faz humanos,
que insiste em permanecer viva apesar das constantes provas contrárias.*

Agradecimentos

Agradeço também ao professor Jean-Paul, pela orientação e motivação, mostrando qual o real sentido do trabalho realizado. E também a toda a equipe do projeto FEMEC Maker que tem feito um excelente trabalho em conjunto no sentido de desenvolver o aparato da faculdade de Mecânica da UFU.

“Sobre isto dizem os filósofos que a Natureza não faz nada em vão, e mais algo é vão quando menos serve. Pois a Natureza é simples e não se compraz com causas supérfluas.”
(Isaac Newton)

Resumo

Uma das linhas de desenvolvimento na Indústria 4.0 é a *Internet of Things* (IOT) onde busca-se a inclusão de componentes responsáveis por realizar parte do processamento em *edge*, de forma a enviar para os servidores centrais da aplicação (geralmente alocados em lugares especiais ou até em nuvem) somente os dados cruciais para o sistema. Um dos principais meios para a concretização deste conceito é o protocolo *Message Queuing Telemetry Transport* (MQTT), pois o mesmo permite que cada objeto tenha maior autonomia para definir suas funções na rede. Neste sentido este projeto se dispõe a apresentar um modelo comunicação para inclusão de elementos básicos utilizados em sistemas de manufatura ao modelo IOT; apresentando uma série de tecnologias selecionadas para manter a implementação destes conceitos simples e adaptável a uma grande gama de componentes. Por fim, foram realizados uma série de testes em bancada eletro-hidráulica e eletro-pneumática comprovando fisicamente a funcionalidade do modelo e suas limitações.

Palavras-chave: MQTT, Nuvem, IOT, Comando, microcontroladores, Industria 4.0.

Abstract

One of development lines in Industry 4.0 is IOT which seeks to include development of components responsible for perform part of the processing at the edge, sending to the main application servers (generally allocated in special places or even in the cloud) only the crucial data to the system. One of the main lines for implementation of this concept is MQTT protocol, whitch allows each object to have more autonomy to define its functions on the network. In this sense, this project intends to present a communication model to include basic components used in manufacturing systems in this model; featuring a range of technologies selected to keep the implementation of these concepts simple and adaptable to a wide range of components. Finally, a series of tests were carried out on electro-hydraulic and electro-pneumatic benchs, physically proving the model functionality and its limitations.

Keywords: MQTT, Cloud, IOT, Command, microcontrolers, Industry 4.0.

Lista de ilustrações

Figura 1 – Exemplo de componentes pneumáticos.	17
Figura 2 – Exemplo de diagrama proposto na norma ISO 1219.	18
Figura 3 – Exemplo de código Ladder proposto na norma IEC 61131-3.	20
Figura 4 – Exemplo de código GRAFCET proposto na norma IEC 60848.	21
Figura 5 – Diagrama representativo da arquitetura MQTT.	22
Figura 6 – Distinção entre os componentes baseados no ESP32.	23
Figura 7 – Modelo <i>Espressif® IOT Development Framework</i> (ESP-IDF).	24
Figura 8 – Raspberry PI 4.	25
Figura 9 – Diagrama de herança das classes da biblioteca wifi.	29
Figura 10 – Diagrama esquemático da classe MQTT.	29
Figura 11 – Diagrama esquemático da classe serial.	30
Figura 12 – Diagrama do primeiro experimento realizado.	33
Figura 13 – Diagrama representativo do fluxo de dados do experimento 1.	33
Figura 14 – Estrutura de permissões.	34
Figura 15 – Diagrama do segundo experimento realizado.	35
Figura 16 – Diagrama representativo do fluxo de dados do experimento 2.	35
Figura 17 – Diagrama da comunicação entre o Broker local e em nuvem.	36
Figura 18 – Montagem realizada na bancada hidráulica.	37
Figura 19 – Montagem realizada na bancada pneumática.	37

Lista de siglas

CLP Controlador Lógico Programável

LEM Laboratório de Ensino de Mecatrônica

SO Sistema Operacional

MQTT *Message Queuing Telemetry Transport*

CI Circuito Integrado

SSL *Socket Secure Layer*

TLS *Transfer Layer Security*

MAPL *Manufacturing Automation Planning Lab*

ESP-IDF *Espressif® IOT Development Framework*

UART *Universal Asynchronous Receiver Transmitter*

PEAP-MSCHAPv2 *Protected Extensible Authentication Protocol with Microsoft Challenge Handshake Authentication Protocol version 2*

WPA2-ENT *Wi-Fi Protected Access 2- Enterprise mode*

WPA2-PSK *Wi-Fi Protected Access 2 - Pre-Shared-Key*

IL *Instruction List*

ST *Structured Text*

LD *Ladder Diagram*

FBD *Function Block Diagram*

GRAFCET *GRAphe Fonctionnel de Commande Etape Transition*

IaaS *Infrastructure as a Service*

WAH *Windows Azure Hypervisor*

IOT *Internet of Things*

SoC *System on Chip*

LTS *Long Term Support*

GCC *GNU Compiler Collection*

AES *Advanced Encryption Standard*

RSA *Rivest-Shamir-Adleman*

ECDSA *Elliptic Curve Digital Signature Algorithm*

SHA *Secure Hash Algorithm*

RNG *Random Number Generator*

Sumário

1	INTRODUÇÃO	15
1.1	Motivação	15
1.2	Objetivos e Desafios	16
1.3	Organização do Trabalho	16
2	FUNDAMENTAÇÃO TEÓRICA	17
2.1	Circuitos Eletrohidráulicos e Eletropneumáticos	17
2.1.1	Métodos de Representação e Simulação	18
2.1.2	CLP e linguagem de programação Ladder	19
2.1.3	GRAFCET	20
2.2	Computação em Nuvem	20
2.3	O Protocolo MQTT	21
2.4	Microcontroladores ESP32	22
2.4.1	Características do Componente	22
2.4.2	Ambiente de Desenvolvimento	23
2.5	A plataforma Raspberry PI®	24
2.6	Mosquitto® Broker	25
2.7	SSL e TLS	26
3	PROPOSTA	28
3.1	ESP-Wifi	28
3.2	ESP-MQTT	28
3.3	ESP-Serial	30
4	EXPERIMENTOS E ANÁLISE DOS RESULTADOS	32
4.1	Método para a Avaliação	32
4.2	Experimentos	32
4.2.1	Experimento 1 — Controle de 1 cilindro	32

4.2.2	Experimento 2 — Controle de 2 cilindros	34
4.3	Avaliação dos Resultados	36
5	CONCLUSÃO	38
5.1	Principais Contribuições	38
5.2	Trabalhos Futuros	38
REFERÊNCIAS		40
APÊNDICES		42
APÊNDICE A	— CÓDIGOS DO SERVIDOR LÓGICO	43
APÊNDICE B	— ARQUIVO DE CONFIGURAÇÃO DO BROKER	45
APÊNDICE C	— CÓDIGOS - EXPERIMENTO 1	46
APÊNDICE D	— CÓDIGOS - EXPERIMENTO 2	48
ANEXOS		50
ANEXO A	— DOCUMENTAÇÃO DO MODO BRIDGE	51
ANEXO B	— ESQUEMA DEVKIT	63
ANEXO C	— ESQUEMA MÓDULO ESP	65

CAPÍTULO 1

Introdução

Ao estudar a história das civilizações humanas como um todo, um fato que ganha bastante evidência é a ligação próxima entre o desenvolvimento de novas e disruptivas técnicas e grandes transformações sociais e econômicas fato que pode ser observado muito claramente, nas três primeiras revoluções industriais com a inclusão, respectivamente, da máquina a vapor, energia elétrica e computadores gerando transformações incomensuráveis no *status social*. Atualmente, vivemos também um momento de grande transformação dirigido principalmente pelo desenvolvimento das comunicações, expondo ao mundo um novo nível de informações, o que, como era de se esperar, também está atrelado a uma nova revolução das técnicas, a indústria 4.0. O que está em curso nesta revolução é a redução do custo dos microprocessadores o que propicia o uso dos mesmos de forma distribuída e dispersa, e permite desmontar o paradigma da lógica e processamento centralizado. A influência da descentralização e dispersão dos microprocessadores no mundo é perceptível em várias áreas: criando ambientes virtuais, comunicação entre maquinários e serviços, aumento das plataformas "*as a service*", entre outras ramificações.

Entre os vários caminhos abertos na quarta revolução industrial, um dos mais expressivos é o da descentralização de sistemas de grande complexidade, neste sentido o uso de componentes inteligentes capazes de resolver parte de seus desafios em *Edge* (ou localmente) associados a uma infraestrutura flexível¹ disponibilizada via nuvem tem se mostrado como uma arquitetura extremamente poderosa e promissora. Surge a questão referente ao modo de conectar componentes essenciais da manufatura nesta arquitetura, seus ganhos e observações da aplicação prática.

1.1 Motivação

A motivação para a realização deste estudo se dá por dois grandes vieses, um deles surge pelo aumento do maquinário existente no *Manufacturing Automation Planning*

¹ Entende-se aqui como infraestrutura flexível aquela que pode ser facilmente ajustada as necessidades momentâneas.

Lab (MAPL) que aumenta a necessidade de um sistema que auxilie no controle e gerenciamento destes componentes tanto de forma local como remotamente. Além disso, o grande período de afastamento dos laboratórios por conta da pandemia enfrentada nos últimos anos fez aumentar em muito os questionamentos sobre a necessidade de muitos processos essencialmente presenciais e acelerou o desenvolvimento de várias tecnologias voltadas a realização de atividades via acesso remoto, contexto no qual este estudo se insere.

1.2 Objetivos e Desafios

O principal objetivo deste trabalho é a construção de um modelo de comunicação, ou *Framework* MQTT para aplicações acadêmico-industriais, que seja de simples entendimento e possível adaptação para comunicação com outros tipos de dados, do ponto de vista de código; que seja modular, isto é, que possa ser adaptado a diferentes tipos de maquinários sem precisar passar por grandes alterações; que possa ser acessado remotamente, sem que isso torne o sistema dependente de redes externas; e que, por fim, possa ser alterado facilmente dada a alta mutabilidade dos sistemas atuais.

Entre os objetivos específicos do *framework* pode-se citar:

- ❑ Ser acoplável a diferentes maquinários
- ❑ Deve ser capaz de ser comunicar com a nuvem, mas não pode ser dependente dela
- ❑ Os códigos utilizados devem ser modulares e de fácil implementação em outros projetos.

1.3 Organização do Trabalho

Seguindo os objetivos descritos acima o presente trabalho, passa pela construção de *Framework* a que se impõe grande versatilidade, desta forma durante sua construção passar-se-á por uma grande revisão de documentos normativos, manuais de fabricantes e documentações oficiais; pois somente assim podemos focar em quais são os limites da aplicação de suas funções diferentemente de um trabalho onde não se sabe sobre o funcionamento e faz-se um teste para saber se funciona. Em segundo momento serão apresentados os requisitos que guiaram o desenvolvimento e; por fim; será realizada uma série de testes, como uma prova de conceito, comprovando a aplicabilidade do *Framework* de forma prática.

CAPÍTULO 2

Fundamentação Teórica

Neste capítulo serão apresentados os componentes utilizados durante o desenvolvimento do trabalho, assim como suas principais características e motivos de cada escolha. Componentes relacionados diretamente a montagem física do sistema serão abordados de forma sucinta, pois o escopo do trabalho se dispõe a apresentar um *Framework* de desenvolvimento que não se limita aos circuitos e componentes utilizados, por outro lado é importante pontuar alguns detalhes quanto a parte de implementação dos códigos que serão vitais para o entendimento da abrangência do modelo utilizado.

2.1 Circuitos Eletrohidráulicos e Eletropneumáticos



Figura 1 – Exemplo de componentes pneumáticos.

No contexto da Engenharia Mecatrônica e da automação no geral, uma das tecnologias utilizadas na solução de problemas de diferentes áreas são os circuitos de comando hidráulicos e pneumáticos, seu uso já de longo período e sua capacidade de ser implementada em diferentes campos fazem com que esta ferramenta possua boa documentação e

suporte de uma estabelecida cadeia de suprimentos, com os mais diversos tipos de componentes. Este consolidado *status* garante que estes circuitos sejam um tópico abordado durante a formação do engenheiro, e como em qualquer outra ferramenta neste contexto são desenvolvidos diversos modelos e métodos que auxiliam em sua implementação.

2.1.1 Métodos de Representação e Simulação

Uma das ferramentas mais importantes que garantem a circuitos hidráulicos e pneumáticos a posição mencionada acima é a existência de uma simbologia bastante sólida para os cada vez mais complexos sistemas utilizados na industria. Esta simbologia, instituída primeiramente pela ISO 1219:1997 [1], mas substituída algumas vezes logo depois até atingir a estrutura atual, na qual seu escopo foi dividido em três partes, um exemplo desta simbologia é apresentado na Figura 2 onde está representado um sistema de comando de três atuadores e um motor eletro-hidráulico:

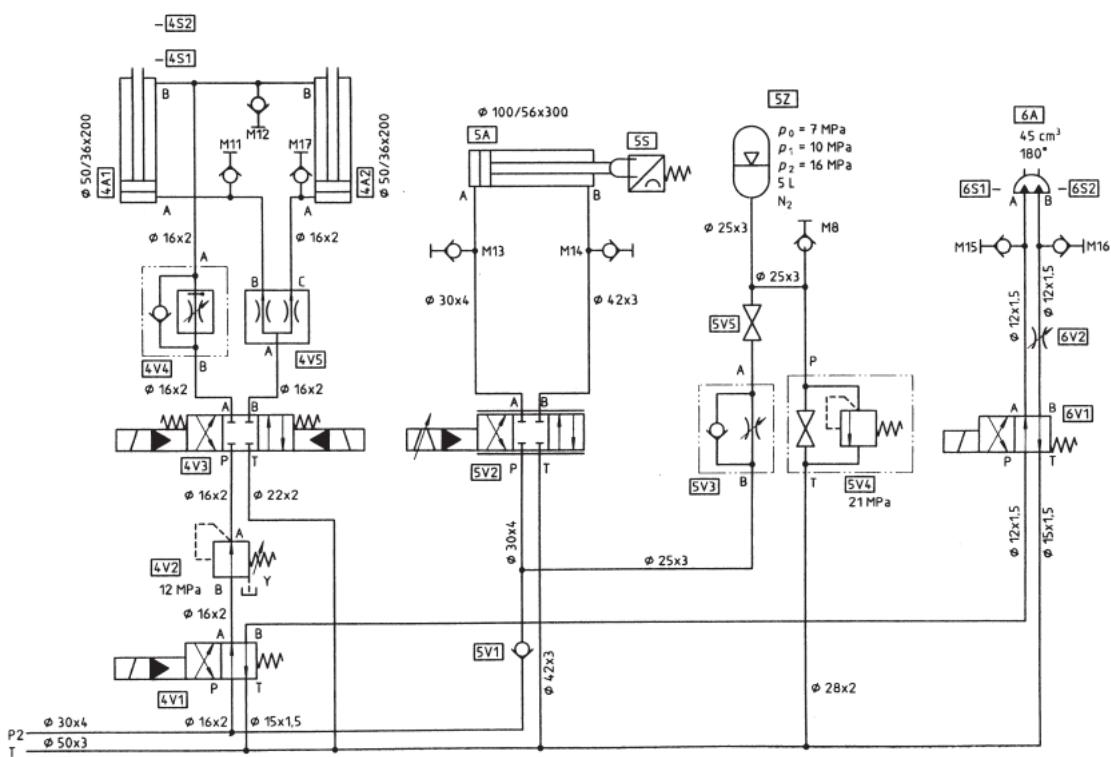


Figura 2 – Exemplo de diagrama proposto na norma ISO 1219.

- ❑ ISO 1219-1:20012 [2]: que versa diretamente sobre os símbolos utilizados nos diagramas de forma isolada (esta parte recebeu em 2016 a inclusão da emenda AMD 1:2016 [3])
- ❑ ISO 1219-2:20012 [4]: que, por sua vez, versa sobre as características construtivas de circuitos conexões, agrupamento e entre outros.

- ❑ ISO 1219-3:20016 [5]: que versa sobre a utilização de módulos, isto é, modelos de representação de pequenos agrupamentos utilizados repetitivamente em circuitos muito grandes.

A partir desta simbologia pode-se criar uma série de *softwares* de simulação capazes de auxiliar na implementação dos mais diversos sistemas, como o FluidSIM® e o Automation Studio, onde podemos executar versões virtuais dos ambientes reais, e observar os resultados.

2.1.2 CLP e linguagem de programação Ladder

Apesar do ambiente altamente robusto descrito acima, o crescimento da complexidade dos problemas abordados no meio da automação industrial ainda tem se mostrado como um grande desafio. Neste contexto um segundo tópico a ser abordado é o desenvolvimento de métodos que pudessem auxiliar no desenvolvimento dí lógica, sendo em um primeiro momento a inclusão dos diagramas de relés, de forma a ficarem estes componentes os principais responsáveis pelo comando; mas depois sendo substituído por Controlador Lógico Programável (CLP) um componente capaz de implementar via código as complicadas sequências de acionamento. Os primeiros esforços no sentido de garantir um modelo padronizado neste sentido vieram da norma IEC 61131 que se dispunha a trazer uma série de conceitos relacionados aos CLPs em geral como apresentado logo no início do documento.

This Part of IEC 61131 applies to programmable controllers (PLC) and their associated peripherals such as programming and debugging tools (PADTs), human-machine interfaces (HMIs), etc., which have as their intended use the control and command of machines and industrial processes. [6]

Entre todo o material apresentado na norma, a parte que mais interessa a este trabalho é a descrita na IEC 61131-3:2013 em que são propostas quatro linguagens de programação: *Instruction List* (IL), *Structured Text* (ST), *Ladder Diagram* (LD) e *Function Block Diagram* (FBD) [7] que tornavam simples a implementação de código nos CLPs e, falando especificadamente dos códigos em LD a linguagem foi construída de forma que estes códigos pudessem ser facilmente interpretados como um diagrama de relés.

A LD program enables the programmable controller to test and modify data by means of standardized graphic symbols. These symbols are laid out in networks in a manner similar to a “rung” of a relay ladder logic diagram. LD networks are bounded on the left and right by power rails. [7]

Na Figura 3 é possível visualizar um exemplo de código Ladder com dois acionadores paralelos e intertravamento.

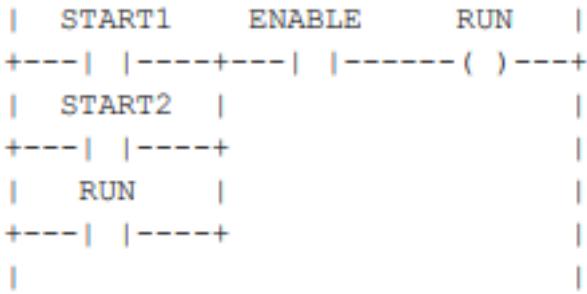


Figura 3 – Exemplo de código Ladder proposto na norma IEC 61131-3.

2.1.3 GRAFCET

Por fim, como último método de grande importância no desenvolvimento de circuitos hidráulicos e pneumáticos em geral, podemos mencionar o *GRAphe Fonctionnel de Commande Etape Transition* (GRAFCET) já que este é um tipo de modelagem onde conseguimos representar de uma forma sequencial modelos altamente complexos. Construído a partir de um caso especial dos Grafos Bipartidos, com elementos subdivididos nos grupos etapas e transições o GRAFCET é visto como um modelo com forte base matemática que remonta a Teoria de Conjuntos. A Figura 4 apresenta um exemplo de representação gráfica de um GRAFCET, em que os retângulos representam etapas e os pequenos traços na horizontal são transições.

Apesar de ser um método para modelagem e dimensionamento de processos, alguns CLPs aceitam o GRAFCET como uma linguagem de programação diretamente assim como previsto na norma IEC 60848:2013 [8], o que traz uma grande praticidade na implementação de processos longos, além de garantir uma maneira mais simples para entendimento dos processos permitindo que grandes times possam trabalhar no seu desenvolvimento.

2.2 Computação em Nuvem

Visto como uma das maiores *buzzwords*¹ da atualidade e com uso ainda crescente. Porém, a definição de computação em nuvem é bastante vaga [10] em alguns casos sendo associada a virtualização de servidores, em outros casos a disponibilização de Infraestrutura como serviço (*Infrastructure as a Service* (IaaS)) ou alguma das variações do "as a service"; ou até em alguns casos uma referência a localização material dos grandes *Data Centers* sua disposição territorial e etc. Fato é que nem entre os próprios provedores destes serviços existe uma padronização, de como eles são disponibilizados; observando os três principais hoje vemos que a AWS® utiliza um modelo de *server virtualization*,

¹ Segundo o dicionário Cambridge: *Buzzword* é uma palavra ou expressão de uma área de assunto particular que se tornou moda, porque tem sido muito usado [9]

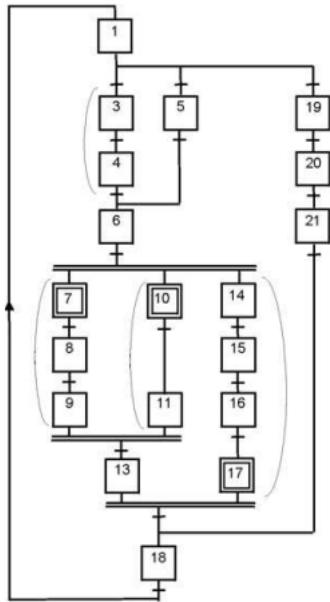


Figura 4 – Exemplo de código GRAFCET proposto na norma IEC 60848.

enquanto a Azure® se baseia no *Windows Azure Hypervisor* (WAH) e o Google®, por sua vez, segue o conceito de *technique-specific sandbox* [10].

No contexto deste trabalho considerar-se-á nuvem como um modelo de virtualização, no qual se delega toda a parte de infraestrutura computacional e podemos acessar os recursos desejados a partir de uma representação abstrata na maioria das vezes disponibilizadas a partir de uma interface via Internet. Conceito que apesar de gerar um grande barulho atualmente não tem nada de novo, ao contrário, era uma dos conceitos já discutidos desde a concepção do *Multics*, sistema operacional precursor do Unix e, portanto, do modelo seguido no GNU atual, como pode ser visto no paper de apresentação do projeto.

It is important to recognize that the average user of the system will see no part of the segmentation and paging complexity described in the paper by Glaser et al. Instead he will see a virtual machine with many system characteristics which are convenient to him for writing either single programs or whole subsystems. [11]

2.3 O Protocolo MQTT

Apresentado pela própria OASIS® como o protocolo padrão para a troca de mensagens IOT, o protocolo MQTT tem ganhado espaço em diversas aplicações por se mostrar como um modelo em que cada componente é capaz de controlar qual tipo de informação ele pode transmitir e receber, o que é visto como um grande ganho do ponto de vista do IOT; além disso, em uma rede MQTT a entrada e saída de novos componentes em rede acontece de forma bastante simples, sem necessitar que o responsável pela rede refaça toda sua configuração.

Para implementar estes conceitos o protocolo utiliza a arquitetura *publisher/subscriber* em que são definidos uma série de tópicos e cada componente pode se cadastrar como publicador de um grupo de tópicos e subscritor de outro grupo de tópicos², na figura Figura 5 é possível ver a representação de um sistema onde o celular e servidor secundário para *Backend* consultam dados publicados pelo sensor de temperatura. Assim, os servidores desta rede (ou *Brokers* seguindo a terminologia descrita no OASIS MQTTv5 Standard [12]) podem deixar a função de orquestradores da rede e se tornarem responsáveis por rotear as mensagens transmitidas por cada tópico. Com o uso do MQTT, temos um grande avanço no sentido de objetos em rede autogerenciáveis, mais informações sobre o uso desta ferramenta em específico neste trabalho serão apresentadas mais a frente.

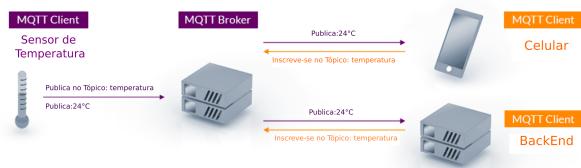


Figura 5 – Diagrama representativo da arquitetura MQTT.

2.4 Microcontroladores ESP32

Um dos principais microcontroladores para desenvolvimento IOT os ESP tem uma grande vantagem em relação a uma gama de outros microcontroladores pois possuem módulos para comunicação *wireless* embutidos no próprio *chip*, neste meio a família ESP32 apresenta-se, entre as linhas de produto ainda em desenvolvimento pela Expressif®, como uma das mais completas apresentando como principais características:

- ❑ 2.4 GHz WiFi + Bluetooth® + Bluetooth LE module
- ❑ microprocessador Xtensa® dualcore 32bit LX6
- ❑ opções de: 4/8/16 MB memória flash
- ❑ 26 GPIOs, com grande gama de periféricos disponíveis
- ❑ Antena PCB na placa ou conector para antena externa

2.4.1 Características do Componente

Apesar de muitas pessoas entenderem os ESPs diretamente como uma placa de desenvolvimento, às vezes até como uma alternativa semelhante ao arduino, é importante

² Estes grupos não são exclusivos

fazer a distinção clara entre os dois componentes. Diferente do Arduino onde temos vários Circuito Integrado (CI) com diferentes funções, no caso do ESP32 estamos falando de um *System on Chip* (SoC) responsável por todas estas capacidades acima. Por outro lado, o fabricante disponibiliza em seu catálogo três modelos de encapsulamento para o mesmo *chip*, assim como demonstrado na Figura 6, sendo eles:

- ❑ **ESP32 DOWD v3**, o próprio SoC, sendo este o modelo mais simples e econômico para inclusão em produtos já de longo ciclo no mercado, porém utilizar este componente diretamente no circuito exige um maior tempo dedicado ao *design* da interface, uma vez que ele traz consigo uma série de *guidelines* a serem seguidos para sua instalação. [13]
- ❑ **ESP32-WROOM-32E**, o módulo, sendo este o grande foco deste trabalho, é um módulo de fácil inclusão no *design* de outras PCBs, já possui proteção térmica e de ruído, circuito para filtro da fonte de tensão, gerador de *clock* e antena PCB inclusos. [14]
- ❑ **ESP32-DevKitC**, a placa de prototipagem, neste componente temos uma pinagem de simples acesso para as saídas do módulo de forma que o uso de *jumpers* fica simples, e também temos incluído botões para mudar o *chip* para o modo de gravação, uma porta USB, e Leds indicadores de funcionamento.



Figura 6 – Distinção entre os componentes baseados no ESP32.

Embora os experimentos deste trabalho tenham sido realizados com o ESP32-DevKitC, não existem restrições quanto ao seu uso também no módulo ESP32-WROOM-32E, as representações esquemáticas destes componentes estão disponíveis respectivamente no Anexo B e no Anexo C.

2.4.2 Ambiente de Desenvolvimento

Outra característica importante dos ESP32 é que eles utilizam o ESP-IDF o ambiente de desenvolvimento oficial disponibilizado pelo fabricante baseado *FREERTOS®*. Um ambiente amplamente documentado com utilização do *Cmake* para organização do processo compilação e inclusão das bibliotecas e, em nível mais alto, de *scripts Python* para

automatização de processos como reconhecimento da placa, *link* e *upload*, assim como demonstrado na figura Figura 7. Além disso, do ponto de vista da compilação o ESP-IDF utiliza versões oficiais do *GNU Compiler Collection* (GCC) com *backend* específico para a família de processadores Xtensa® LX6³; e são, portanto, compatíveis com todas as linguagens suportadas no *frontend* do GCC utilizado (C, C++, Objective-C, FORTRAN, Go e etc).

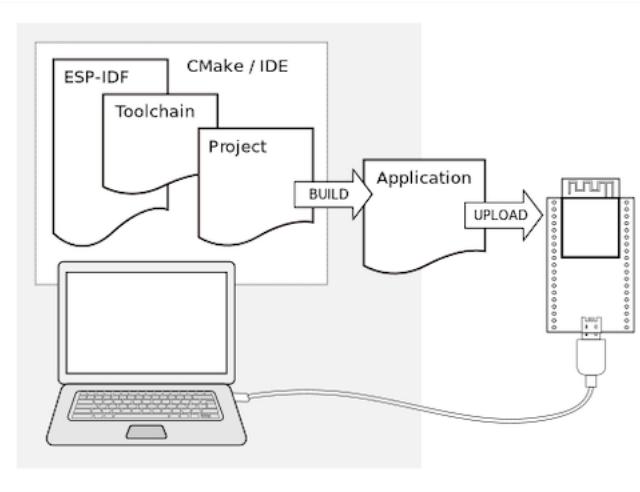


Figura 7 – Modelo ESP-IDF.

No momento da construção deste trabalho, a versão de suporte padrão para o *framework* utilizava o GCC 5.3 com suporte máximo para o Std17 [15], assim como esperado de acordo com as informações de *release* do GCC 5 [16].

Optou-se, portanto, pela utilização em todos os códigos deste trabalho do C++17. O uso do C++ se justifica porque o grande objetivo do trabalho é produzir uma plataforma de fácil entendimento. Por outro lado, o uso do padrão 17 se dá com o objetivo de seguir as recomendações do próprio *release* disponível no site da GNU designando ente como novo padrão para o compilador [16].

2.5 A plataforma Raspberry PI®

Uma plataforma que tem ganhado bastante notoriedade tanto nos assuntos que envolvem engenharia quanto na área da computação são os Raspberry PI. Um pouco diferente do que costuma-se ver em *chips all-in-one* os Raspberry-PI se dispõe a ser um *Desktop* completo com entradas para teclado, mouse, saídas de audio e video, se mostrando até como uma opção para uso pessoal. Estes componentes impressionam bastante por suas

³ Essa informação não abrange os ESP32-S2 ou ESP32-C3, estas séries de processadores possuem versões diferentes do GCC (*backend* LX7, RISC-V e etc.) assim como descrito no Github do MAPL <<https://github.com/MAPL-UFU/IndustrialAutomationMQTT>>

capacidades em um primeiro contato. Apesar de aparecerem associados a projetos *open source* de várias formas, eles não o são; muito embora seja possível encontrar muito sobre seu funcionamento, inclusive os desenhos esquemáticos, na própria documentação do fabricante. [17]

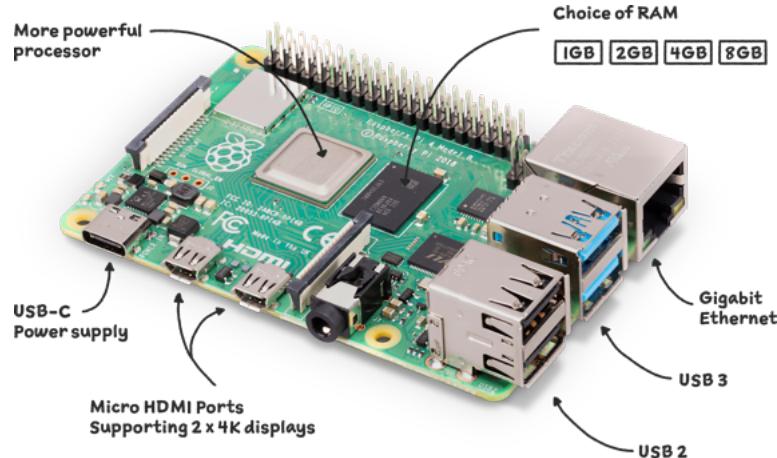


Figura 8 – Raspberry PI 4.

Do ponto de vista de *software*, os Raspberry PI possuem compatibilidade com uma série de Sistema Operacional (SO) Linux o que faz destes componentes também uma opção interessante para a prototipagem de programas mais completos e que vão rodar sobre o ambiente de um SO.

Apesar da Raspberry disponibilizar em seu site um SO específico para uso nesta placa, capaz de extrair melhor desempenho da mesma; no contexto do corrente trabalho o SO utilizado foi a atual versão *Long Term Support* (LTS) do Ubuntu Server (22.04), que por ser uma das versões mais utilizadas mundialmente em servidores garante que todos os códigos utilizados aqui podem ser portabilizados para servidores padrão, estejam eles em nuvem ou não. O passo a passo desta instalação pode ser encontrado em na página dos desenvolvedores do Ubuntu. [18]

2.6 Mosquitto® Broker

Desenvolvido pela Eclipse Foundation® o *Broker* Eclipse Mosquitto, ou somente Mosquitto, é uma implementação de um *Broker* MQTT com interface de simples aprendizagem, tanto para instalações locais, quanto para aquelas feitas de forma remota. Além disso, o Mosquitto é conhecido por ser um programa que requer para seu funcionamento poucos recursos seja no âmbito do processamento ou no âmbito do espaço em memória.

Embora muito simples, o *Broker* Mosquitto possui uma série de funções importantes que vão desde segurança via certificados x509 até a implementação do modo *Bridge*: Um modo de funcionamento onde as mensagens publicadas em um *Broker* são repetidas em um

ou mais espelhos, sendo o contrário também possível, todas estas possíveis configurações podem ser consultadas na documentação dos arquivos ".conf". [19]

2.7 SSL e TLS

Quando se fala em comunicação via Internet uma das primeiras preocupações que surgem em qualquer projeto é quanto a segurança e integridade dos dados trafegados pela rede; e quando se fala de utilização de servidores em nuvem, esta preocupação fica ainda maior, visto que os dados serão tratados e armazenados seguindo as políticas de terceiros que muitas das vezes podem não ser compatíveis com os requisitos desejados. Atualmente, técnica mais utilizada para garantir a segurança na transmissão de dados é a utilização do *Transfer Layer Security* (TLS), uma série de protocolos definidos no RFC 5246 [20] para o processo de encriptação utilizando, em partes criptografia de chave simétrica e em outros momentos criptografia de chaves assimétricas, podendo ser dividido em dois grandes protocolos:

- **TLS Record Protocol** utilizado diretamente para a transferência de dados, que serão criptografados geralmente seguindo o modelo definido em pela *Advanced Encryption Standard* (AES) tendo como chave simétrica o valor definido ao final do processo de negociação.
- **TLS Handshake Protocol** onde são definidas as regras para a negociação da forma como será feita a transferência dos dados (algoritmo de criptografia, algoritmo de *hash*, tamanho de bloco, *padding*, entre outros). Vale ressaltar que nesta etapa deve ser utilizado algum algoritmo de chaves assimétricas como *Rivest-Shamir-Adleman* (RSA) ou *Elliptic Curve Digital Signature Algorithm* (ECDSA) para garantir a identidade dos negociantes.

O *Socket Secure Layer* (SSL), por sua vez, possui grande semelhança com o protocolo TLS inclusive na maneira como é redigido o documento emitido pela Netscape® em 1996 revisado em 2011 "The SSL Protocol - Version 3.0" [21], possuindo praticamente a mesma estrutura. Porém, o desenvolvimento dos métodos utilizados na Internet e principalmente da segurança da informação mostraram que algumas alterações eram necessária para a manutenção da segurança nas transmissões, assim limitando a utilização de algumas funções de *Hash*, métodos de geração de números pseudo-aleatórios, incluindo passos intermediários e entre outros. Assim como pode ser confirmado pelo trecho da própria RFC 5246:

The differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough that the various versions of TLS and SSL 3.0 do not interoperate (although each protocol incorporates a mechanism by which an implementation can back down to prior versions). [20]

Por fim, no contexto deste trabalho, a utilização tanto do TLS quanto do SSL aconteceu de forma bem simples, os ESP-32 possuem aceleradores em *hardware* para: AES, *Secure Hash Algorithm* (SHA), RSA e *Random Number Generator* (RNG); todos compatíveis com ambos os protocolos, esses aceleradores são disponibilizados pelo componente `embedtls` que pode ser acionado nas opções do ESP-IDF.

CAPÍTULO 3

Proposta

O principal objetivo deste trabalho é a construção de um framework modular para acesso a diferentes tipos de máquinário utilizando um componente de baixo custo, no caso o ESP32, via comunicação MQTT, para isso foram desenvolvidas algumas bibliotecas modulares que podem ser utilizadas para controlar três funções vitais do microcontrolador, de forma a transformá-lo em um componente capaz de funcionar como um intermediário na comunicação do maquinário com a rede em nuvem; sendo, neste trabalho, a proposta dividir todo o código em três bibliotecas principais:

3.1 ESP-Wifi

Responsável por gerenciar o acesso dos ESP32 na rede Wifi, importante pois facilita o acesso a rede wifi utilizada gerenciando internamente o modelo de acesso que a depender do local e do tipo do experimento mudava de *Wi-Fi Protected Access 2 - Pre-Shared-Key* (WPA2-PSK) para *Wi-Fi Protected Access 2- Enterprise mode* (WPA2-ENT), vale ressaltar que entre os vários protocolos utilizados no WPA2-ENT preferia-se sempre o uso do *Protected Extensible Authentication Protocol with Microsoft Challenge Handshake Authentication Protocol version 2* (PEAP-MSCHAPv2). A Figura 9 apresenta uma re-presentação do sistema de heranças utilizado na biblioteca onde temos, a partir da classe *Wifi Event Handler* que implementa os métodos para o envio e recebimento de dados, duas classes derivadas (*WPA2 PSK* e *WPA2 ENT*) com os métodos responsáveis pelo processo de autenticação, variando para cada protocolo.

3.2 ESP-MQTT

Responsável por controlar a comunicação MQTT como um todo. A maneira utilizada para implementar o protocolo se baseia no sistema de eventos, onde cada evento está relacionado com uma etapa do estabelecimento da comunicação, porém as funções chamadas em cada um das etapas são métodos de uma classe que representa a conexão, de forma

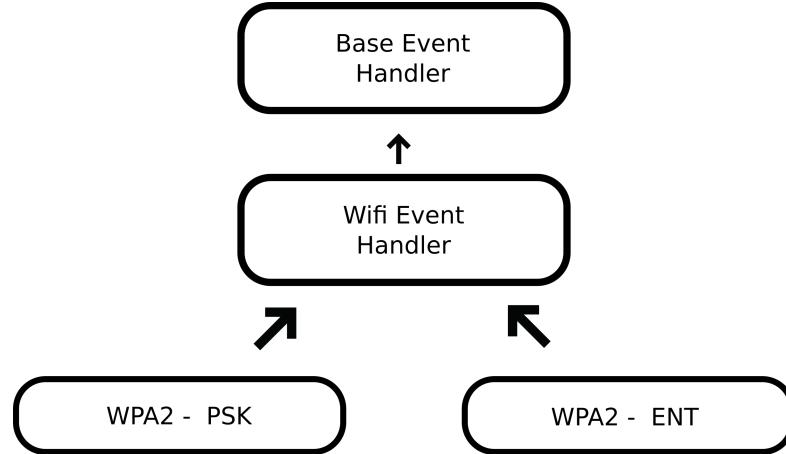


Figura 9 – Diagrama de herança das classes da biblioteca wifi.

que a implementação da biblioteca em projetos futuros fique bem simples, se limitando a pouco mais do que instanciar o objeto e chamar o método *PostData*. A Figura 10 apresenta o modelo utilizado, apresentando os métodos padrão para o estabelecimento da conexão MQTT (destacados em *MQTT Standard Events*) e os *Hi-Level Events* que simplificam o processo de envio e serão os únicos utilizados doravante.



Figura 10 – Diagrama esquemático da classe MQTT.

3.3 ESP-Serial

Responsável por controlar a comunicação serial via *Universal Asynchronous Receiver Transmitter* (UART) com diversos componentes. Do ponto de vista da arquitetura ela segue os mesmos conceitos utilizados pela biblioteca responsável pelo MQTT com as diferenças sendo basicamente internas nas partes responsáveis pela instalação dos drivers para utilização dos pinos dos componentes. A Figura 11 apresenta a estrutura da classe desenvolvida para comunicação serial.



Figura 11 – Diagrama esquemático da classe serial.

Uma colocação importante é a de que estas bibliotecas seguem o modelo de desenvolvimento dos componentes do ESP-IDF¹ de forma que eles devem ser independentes e podem ser facilmente incorporados a códigos de terceiros², mas que quando utilizados em conjunto apresentam ganhos, pois o fato de utilizarem a mesma classe mãe (*Base Event Handler*) garante que eles utilizem somente um *loop* de eventos e, portanto, menos recursos em memória já que na implementação do ESP-IDF todos os *loops* de eventos possuem no mínimo uma *task* em memória associada.

Por fim, um segundo objetivo deste trabalho é propor um modelo de comunicação que possua meios para garantir a comunicação local, diretamente via MQTT, mesmo no caso de falhas dos servidores em nuvem, para isso foi utilizado localmente o *Broker Mosquitto®* em modo *Bridge*, em que, como foi mencionado na fundamentação teórica, é possível replicar mensagens de tópicos específicos em *Brokers* distintos, os códigos de configuração utilizados estão no Apêndice B.

¹ O Framework de desenvolvimento dos ESP32 possui uma documentação bastante robusta e pode ser encontrada em <<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html>> [22]

² Na documentação das bibliotecas no repositório existe um passo a passo de como essa importação pode ser feita utilizando somente um comando dos *submodules* do GIT: <<https://github.com/MAPL-UFU/IndustrialAutomationMQTT>>

CAPÍTULO 4

Experimentos e Análise dos Resultados

Como já especificado em capítulos anteriores, este trabalho de dispõe a apresentar um Framework de controle para sistemas discretos de forma a utilizar os principais ganhos do protocolo MQTT para propor soluções mais simples do que as amplamente utilizadas em processos industriais para problemas como gerenciamento da rede, custo dos equipamentos utilizados e entre outros. Neste sentido, o principal teste realizado foram a realização de duas provas de conceito simulando sistemas amplamente presentes no âmbito industrial para se observar o comportamento do modelo proposto, além disso, a partir das simulações foram encontradas algumas questões para as quais foram propostos testes especiais assim como descrito a seguir.

4.1 Método para a Avaliação

O método utilizado para avaliar a funcionalidade do sistema proposto foi a implementação direta de duas provas de conceitos utilizando o material do Laboratório de Ensino de Mecatrônica (LEM), isto é, a montagem de um circuito eletro-pneumático (primeiramente, mas posteriormente foram feitos testes também com sistemas eletro-hidráulicos) para controle de um ou mais cilindros utilizando o sistema MQTT para promover a comunicação em Edge e em nuvem.

4.2 Experimentos

4.2.1 Experimento 1 — Controle de 1 cilindro

O primeiro experimento realizado se dispunha a realizar o controle de apenas um cilindro pneumático de forma que dois ESP32 foram utilizados no sistema MQTT para promover a comunicação, um microcontrolador era responsável por monitorar os sensores e disponibilizar seu estado em rede, assim sendo caracterizado como um publicador; enquanto o outro microcontrolador era responsável por controlar o estado do cilindro uti-

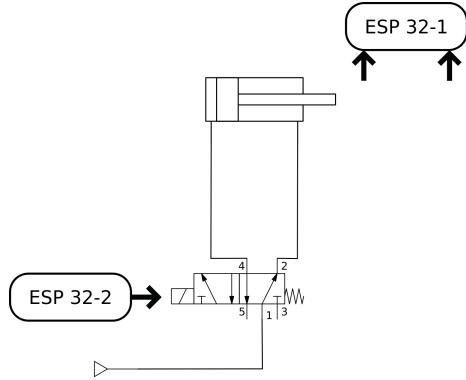


Figura 12 – Diagrama do primeiro experimento realizado.

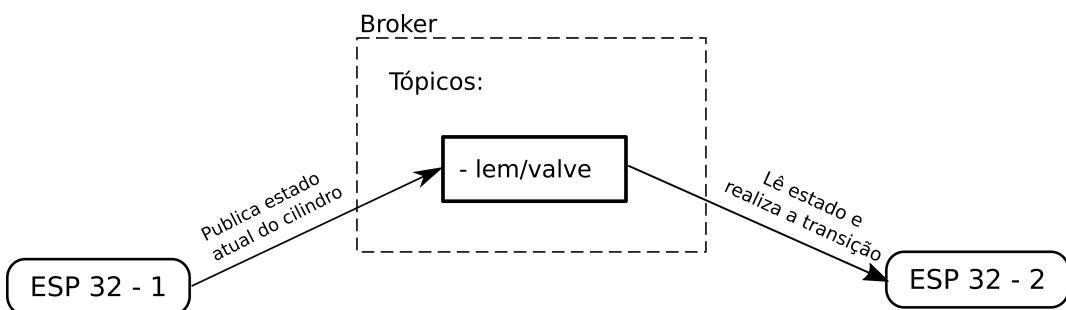


Figura 13 – Diagrama representativo do fluxo de dados do experimento 1.

lizando as informações disponibilizadas pelos sensores assim, assim sendo caracterizado como um subscritor. Na Figura 12 e Figura 13 estão os diagramas da montagem realizada. Neste Exemplo, foram utilizados tanto a implementação local do *Broker Eclipse Mosquitto®* quanto o disponibilizado pela AWS®, AWS IoT Core. Vale ressaltar que no caso do *Broker* via AWS foi necessário a implementação de Segurança TLS e dos Certificados x.509 gerados diretamente na plataforma para a utilização do *Broker*¹, desta forma, seguindo o passo a passo descrito na própria plataforma foram geradas a Política e o Certificado de cada ESP sendo de forma que cada um possui um *Endpoint* dedicado para conexão, ambos capazes de ler e publicar no tópico "lem/valve" a Figura 14 mostra a estrutura característica utilizada para cadastro dos itens na AWS. No apêndice Apêndice C estão disponibilizados os trechos do código responsáveis especificadamente pela transmissão dos dados via MQTT, porém caso queira consultar os códigos completos de cada aplicação (experimento um e dois, ou ainda das bibliotecas que contituem o *Framework*) eles estão disponíveis e documentados repositório no Github® do Laboratório de Planejamento Automático de Manufatura².

¹ O caminho para conexão de novos dispositivos pode ser facilmente encontrado no caminho: AWS IoT > Connect > Connect one device

² Disponível em: <<https://github.com/MAPL-UFU/IndustrialAutomationMQTT>>

ESP32-2

Thing details

- Name: ESP32-2
- ARN: arn:aws:iot:us-west-2:731707703350:thing/ESP32-2
- Type: -
- Billing group: -

Details

Certificate ID: 0fbfb9a75f8a49cdab706bb18637f4b98c37524a0dbf2d3a2a1b72a3f0649042

Certificate ARN: arn:aws:iot:us-west-2:731707703350:cert/0fbfb9a75f8a49cdab706bb18637f4b98c37524a0dbf2d3a2a1b72a3f0649042

Subject: CN=AWS IoT Certificate

Issuer:

SimpleValveControl

Details

Policy ARN: arn:aws:iot:us-west-2:731707703350:policy/SimpleValveControl

Active version: 18

Created: May 01, 2022, 18:33:29 (UTC-0)

Versions | Targets | Noncompliance | Tags

Active version: 18

Policy effect	Policy action	Policy resource
Allow	iot:Publish	arn:aws:iot:us-west-2:731707703350:topic/lem/valve
Allow	iot:Subscribe	arn:aws:iot:us-west-2:731707703350:topicfilter/lem/valve
Allow	iot:Receive	arn:aws:iot:us-west-2:731707703350:topic/lem/valve
Allow	iot:Publish	arn:aws:iot:us-west-2:731707703350:topic/lem/sensor
Allow	iot:Receive	arn:aws:iot:us-west-2:731707703350:topic/lem/sensor
Allow	iot:Subscribe	arn:aws:iot:us-west-2:731707703350:topicfilter/lem/sensor
Allow	iot:Connect	arn:aws:iot:us-west-2:731707703350:client/AActuator
Allow	iot:Connect	arn:aws:iot:us-west-2:731707703350:client/BActuator

Figura 14 – Estrutura de permissões .

4.2.2 Experimento 2 — Controle de 2 cilindros

Já no segundo Experimento, foi utilizada uma montagem onde cada Esp-32 era responsável por um cilindro, isto é, controlando tanto suas entradas como suas saídas, assim cada cilindro é, ao mesmo tempo, um publicador e um subscriptor. Na Figura 15 e na Figura 16 estão os diagramas da montagem realizada. Neste experimento não foram utilizados os recursos da AWS, em vez disso, agora que já tínhamos testado a comunicação pela rede deles, neste segundo momento optou-se pelos serviços gratuitos disponibilizados pelo site shiftr.io já que o foco não era os elementos de segurança da rede em si, implementação do SSL ou geração das permissões. A essa altura o objetivo era a inclusão de um terceiro elemento na rede responsável por duas coisas: funcionar como um servidor de lógica, ou seja, um item externo que relaciona as saídas dos sensores às entradas dos

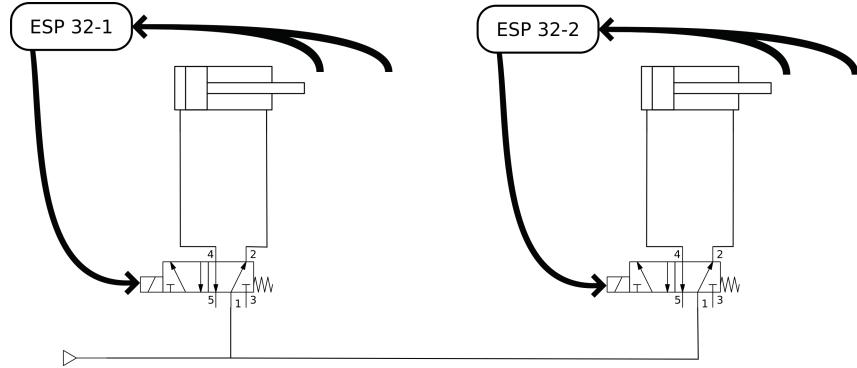


Figura 15 – Diagrama do segundo experimento realizado.

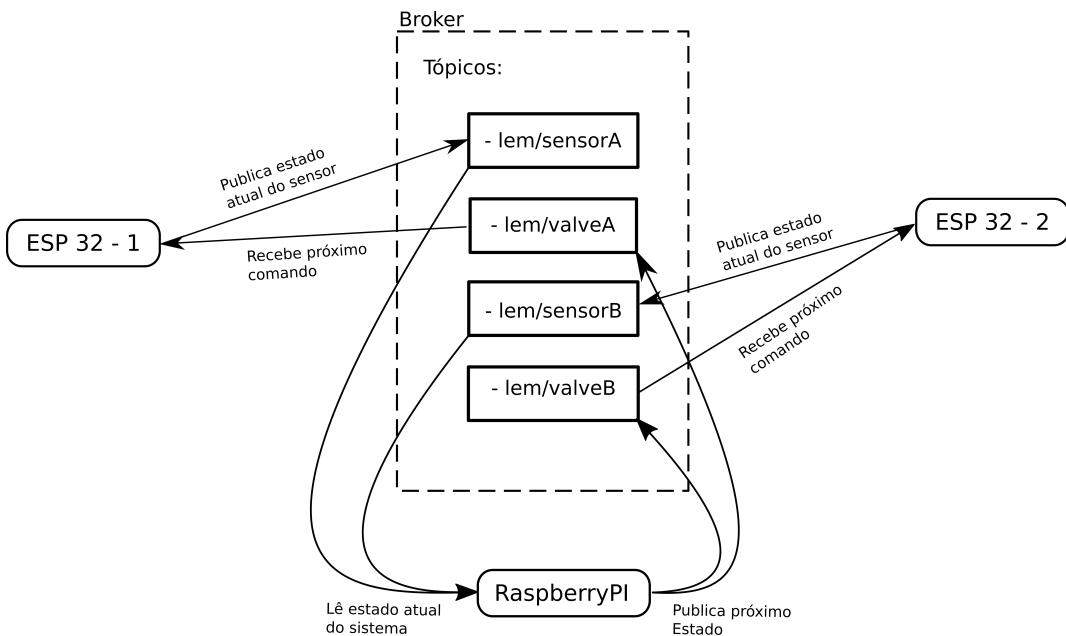


Figura 16 – Diagrama representativo do fluxo de dados do experimento 2.

atuadores e também de um servidor MQTT local que fosse capaz de se comunicar de forma *full-duplex* com o servidor em nuvem, como apresentado na Figura 17, permitindo que o sistema fosse acessado remotamente, mas sem deixá-lo totalmente dependente da nuvem, de forma que o mesmo continua funcionando em casos em que a comunicação entre a rede local e a Internet é perdida. O arquivo de configuração do *Broker* local pode ser consultado no Apêndice B.

Como elemento extra foi utilizado a título de exemplo um Raspberry Pi 4; no qual, beneficiando-se do isolamento de processos característico de sistemas operacionais, era possível a implementação simultânea do broker MQTT Mosquitto e de um *script Python* para controle do sistema. Assim, para a parte de lógica foi utilizada a biblioteca própria para comunicação com servers MQTT Paho, pois dada a praticidade dos *scripts Python* ficava bem simples a aplicação da lógica e a implementação de possíveis modificações já

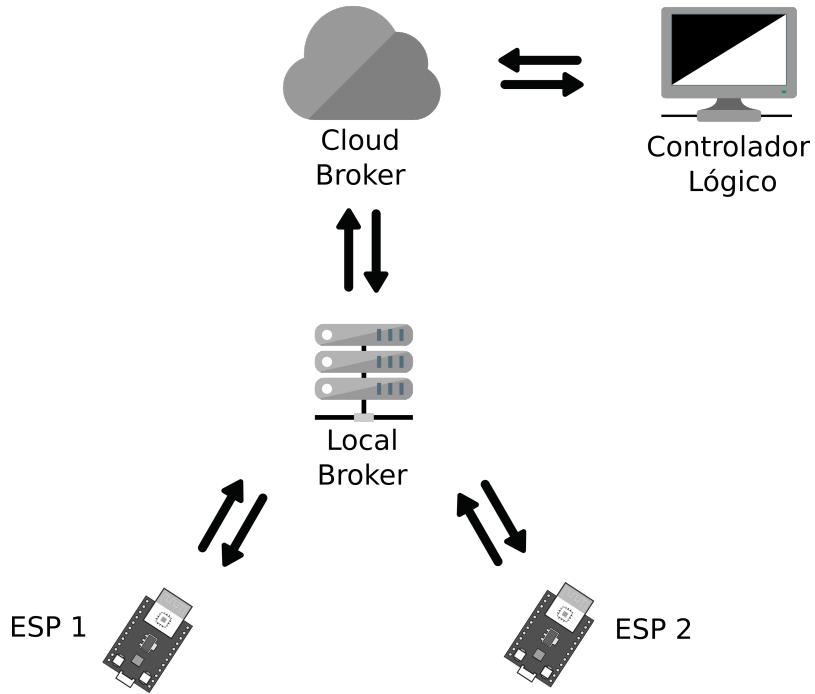


Figura 17 – Diagrama da comunicação entre o Broker local e em nuvem.

com o sistema funcionando, o código *Python* utilizado se encontra no Apêndice A. Por fim, na implementação do *Broker Mosquitto* foi utilizada uma função chamada MQTT Bridge [19] configurada de forma que todos os tópicos utilizados no sistema estão sendo replicados nos dois Brokers assim garantindo que uma queda na Internet não quebrará todo o sistema, além de possibilitar que sejam colocados lógicas de segurança em Edge para casos onde uma parada abrupta geraria sérios problemas.

4.3 Avaliação dos Resultados

Após a implementação em bancada do sistema, o que pode ser visto na Figura 18 e na Figura 18, obteve-se sucesso com a comunicação e com o funcionamento do sistema como um todo, em ambas as montagens. Porém, foram encontradas alguns problemas na transmissão das informações que, embora não sejam capazes de inviabilizar o sistema conforme os objetivos preestabelecidos para o estudo atual, são possíveis objetos de estudo para trabalhos posteriores, sendo eles principalmente o aumento da lentidão na troca de informações dependendo da Nuvem utilizada, o que não era perceptível no primeiro teste, na bancada hidráulica; mas acabou ficando evidente nos testes utilizando o circuito pneumático e a inclusão de ruído proveniente dos sensores utilizados na montagem também a depender do atuador e dos sensores de contato utilizados. Vale ressaltar, ainda, que a velocidade na transmissão de mensagens sofria grandes variações de acordo com o *Broker* utilizado, sendo praticamente instantânea em aplicações com *Broker* local, tendo pequeno



Figura 18 – Montagem realizada na bancada hidráulica.

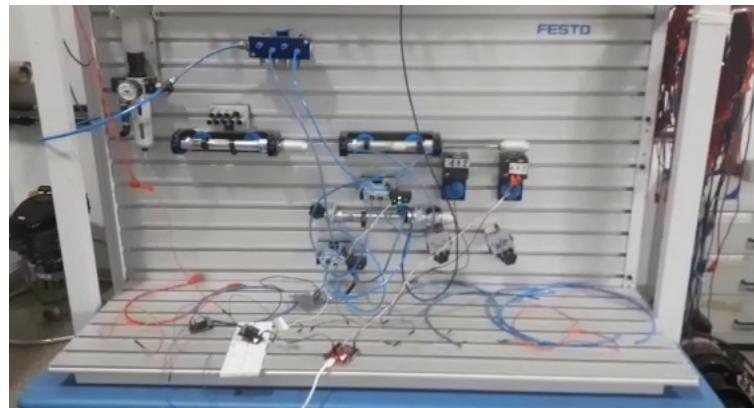


Figura 19 – Montagem realizada na bancada pneumática.

atraso quando instanciado utilizando a nuvem AWS, e maior atraso quando utilizada a nuvem disponibilizada pelo *shift.io* ®.

CAPÍTULO 5

Conclusão

A realização do trabalho se mostrou de grande valia para o desenvolvimento, tanto pessoal do aluno, quanto para as necessidades do laboratório, obtendo pleno sucesso no que tangia ao escopo proposto, sendo possível a realização do controle discreto dos circuitos propostos utilizando os circuitos em *Edge* e em Nuvem utilizando o protocolo MQTT. Mostrando-se diretamente como uma poderosa alternativa aos modelos amplamente utilizados no contexto de sistemas elétronico-hidráulicos e elétronico-pneumáticos atualmente, sendo eles tanto a lógica de relés aplicada diretamente, quanto utilizando CLP, ou até métodos mais complexos; ao passo que é também uma solução de baixo custo e que simplifica em muito a implementação de circuitos lógicos de forma facilmente escalável.

5.1 Principais Contribuições

O principal produto do trabalho realizado é a apresentação de um *framework* para realização do controle de uma sistema via MQTT de forma que os códigos utilizados no controle de cada ESP32 são modulares e podem ser facilmente substituídos para diversos outros tipos de dados a serem repassados a depender do maquinário a ser utilizado. Além disso, o sistema apresenta um segundo grande ganho que é a implementação de um sistema com habilidade de lidar com falhas no sistema de Internet sem perder a capacidade de conectar com a Nuvem.

5.2 Trabalhos Futuros

Apesar do sucesso da comunicação em relação ao funcionamento do sistema, foram encontrados comportamentos que não eram diretamente esperados durante o planejamento da aplicação, desta forma podem ser elencados como estudos ainda a serem realizados no futuro: o estudo sobre quais os métodos para reduzir a latência da comunicação, embora para este trabalho cujo escopo se resume somente a controle discreto de sistemas isso não seja caracterizado ainda como um problema; outro grande problema a ser resolvido no

futuro é a inclusão de sistemas capazes de perceber erros nos sensores em tempo real, para reduzir erros como mal contato e por fim fica também como uma possibilidade a ser estudada o uso deste tipo de comunicação para controle não discreto. Por fim, existe também um grande interesse do MAPL no estudo de quais as melhores maneiras de dimensionar o componente responsável pela lógica do sistema, um aplicativo de controle já está sendo estudado; neste trabalho, este componente foi implementado de forma bastante simples em *Python*, como apresentado no experimento 2.

Referências

- 1 INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *ISO 1219:1997*: Fluid power systems and components — graphic symbols. [S.l.], 1997. 23 p.
- 2 INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *ISO 1219-1:2012*: Fluid power systems and components — graphical symbols and circuit diagrams — part 1: Graphical symbols for conventional use and data-processing applications. [S.l.], 2012. 178 p.
- 3 INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *ISO 1219-1:2012/AMD 1:2016*: Fluid power systems and components — graphical symbols and circuit diagrams — part 3: Symbol modules and connected symbols in circuit diagrams. [S.l.], 2016. 2 p.
- 4 INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *ISO 1219-2:2012*: Fluid power systems and components — graphical symbols and circuit diagrams — part 2: Circuit diagrams. [S.l.], 2012. 42 p.
- 5 INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *ISO 1219-3:2016*: Fluid power systems and components — graphical symbols and circuit diagrams — part 1: Graphical symbols for conventional use and data-processing applications. [S.l.], 2016. 23 p.
- 6 INTERNATIONAL ELECTROTECHNICAL COMMISSION. *IEC 61131-1:2003*: Programmable controllers - part 1: General information. [S.l.], 2003. 36 p.
- 7 INTERNATIONAL ELECTROTECHNICAL COMMISSION. *IEC 61131-3:2013*: Programmable controllers - part 3: Programming languages. [S.l.], 2003. 464 p.
- 8 INTERNATIONAL ELECTROTECHNICAL COMMISSION. *IEC 60848:2013*: Grafcet specification language for sequential function charts. [S.l.], 2013. 109 p.
- 9 buzzword. 2022. [Online; accessed 28. Aug. 2022]. Disponível em: <<https://dictionary.cambridge.org/dictionary/english/buzzword>>.
- 10 QIAN, L. et al. Cloud computing: An overview. In: JAATUN, M. G.; ZHAO, G.; RONG, C. (Ed.). *Cloud Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 626–631. ISBN 978-3-642-10665-1.

- 11 INTRODUCTION and Overview of the Multics System. 1965. [Online; accessed 28. Aug. 2022]. Disponível em: <<https://www.multicians.org/fjcc1.html>>.
- 12 MQTT Version 5.0. 2019. [Online; accessed 28. Aug. 2022]. Disponível em: <<https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>>.
- 13 ESP32 Series Datasheet. 2022. [Online; accessed 29. Aug. 2022]. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>.
- 14 ESP32WROOM32E ESP32WROOM32UE Datasheet. 2022. [Online; accessed 29. Aug. 2022]. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf>.
- 15 ESPRESSIF. *Suporte para C++17*. 2022. [Online; accessed 29. Aug. 2022]. Disponível em: <<https://github.com/espressif/esp-idf/issues/2449>>.
- 16 GCC 5 Release Series Changes, New Features, and Fixes - GNU Project. 2022. [Online; accessed 29. Aug. 2022]. Disponível em: <<https://gcc.gnu.org/gcc-5/changes.html>>.
- 17 RASPBERRY Pi Datasheets. 2021. [Online; accessed 29. Aug. 2022]. Disponível em: <<https://datasheets.raspberrypi.com>>.
- 18 HOW to install Ubuntu Server on your Raspberry Pi Ubuntu. 2022. [Online; accessed 3. Sep. 2022]. Disponível em: <<https://ubuntu.com/tutorials/how-to-install-ubuntu-on-your-raspberry-pi#1-overview>>.
- 19 MOSQUITTO, E. *mosquitto conf man page*. 2022. [Online; accessed 26. Aug. 2022]. Disponível em: <<https://mosquitto.org/man/mosquitto-conf-5.html>>.
- 20 DIERKS, T.; RESCORLA, E. *The Transport Layer Security (TLS) Protocol Version 1.2*. 2008. [Online; accessed 7. Sep. 2022].
- 21 FREIER, A.; KARLTON, P.; KOCHER, P. *Netscape Communications*. 2011.
- 22 ESP-IDF Programming Guide - ESP32 ESP-IDF Programming Guide latest documentation. 2022. [Online; accessed 28. Aug. 2022]. Disponível em: <<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html>>.

Apêndices

APÊNDICE A

Código Fonte Utilizado no Servidor de Lógica

```

1 import random
2
3 from paho.mqtt import client as mqtt_client
4
5
6 broker = 'mapl-iot.cloud.shiftr.io'
7 port = 1883
8 topicValveA = "lem/#"
9 client_id = f'python-mqtt-{random.randint(0, 100)}'
10
11
12 def connect_mqtt() -> mqtt_client:
13     def on_connect(client, userdata, flags, rc):
14         if rc == 0:
15             print("Connected to MQTT Broker!")
16         else:
17             print("Failed to connect, return code %d\n", rc)
18
19     client = mqtt_client.Client(client_id)
20     client.username_pw_set('mapl-iot', 'NgJlVBYOkeNK2GIw')
21     client.on_connect = on_connect
22     client.connect(broker, port)
23     return client
24
25
26 def subscribe(client: mqtt_client):
27     def on_message(client, userdata, msg):

```

```
28     print(f"Received '{msg.payload.decode()}' from '{msg.topic}' topic")
29     if (msg.topic == 'lem/sensorB'):
30         if (msg.payload.decode() == 'Desacionado'):
31             client.publish('lem/valveA', "Acionado")
32         if (msg.payload.decode() == 'Acionado'):
33             client.publish('lem/valveA', "Desacionado")
34     if (msg.topic == 'lem/sensorA'):
35         if (msg.payload.decode() == 'Acionado'):
36             client.publish('lem/valveB', "Acionado")
37         if (msg.payload.decode() == 'Desacionado'):
38             client.publish('lem/valveB', "Desacionado")
39
40
41
42     client.subscribe(topicValveA)
43     client.on_message = on_message
44
45
46 def run():
47     client = connect_mqtt()
48     subscribe(client)
49     client.loop_forever()
50
51
52 if __name__ == '__main__':
53     run()
```

APÊNDICE B

Arquivo de configuração para utilização do Mosquitto® *Broker* em modo *Bridge*

```
1 # Place your local configuration in /etc/mosquitto/conf.d/
2 #
3 # A full description of the configuration file is at
4 # /usr/share/doc/mosquitto/examples/mosquitto.conf.example
5
6 persistence true
7 persistence_location /var/lib/mosquitto/
8
9 log_dest file /var/log/mosquitto/mosquitto.log
10
11 include_dir /etc/mosquitto/conf.d
12
13 listener 6000
14 connection test
15 address mapl-iot.cloud.shiftr.io:1883
16 remote_username mapl-iot
17 remote_password NgJlVBYOkeNK2GIw
18 topic base/# both 0
19
20 allow_anonymous true
```

APÊNDICE C

Códigos Utilizados no Experimento 1

Neste experimento temos códigos diferentes para cada ESP 32 em rede, sendo para o Publicador ligado aos sensores:

```

1 void vTaskLocalControl( void *pvParameters){
2     int i=0;
3     CustomEventHandler* context = (CustomEventHandler*)pvParameters;
4     cout<<("Task Local Control");
5     while(1){
6         if (context->isMQTTConnected()){
7             string message("");
8             if (gpio_get_level(GPIO_INPUT_IO_0) == 1){
9                 message += "{\
10                     \"message\": \"Acionado\",\
11                     \"sequence\": 1\
12                 }";
13             }
14             else{
15                 i++;
16                 if (i == 5){
17                     i=0;
18                     message += "{\
19                         \"message\": \"Desligado\",\
20                         \"sequence\": "+to_string(i)+"\
21                     }";
22                 }
23             }
24             context->postData("lem/valve", message.c_str());
25         }
26         vTaskDelay(1000 / portTICK_PERIOD_MS);
27     }

```

 28 } ;

E para o Subscritor ligado ao atuador:

```

1 void vTaskLocalControl( void *pvParameters){
2     CustomEventHandler* context = (CustomEventHandler*)pvParameters;
3     cout<<("Task Local Control");
4     context->subscribeTo( "lem/valve" ,[]( string pay){
5         cout<<" Received: "<<pay<<endl;
6         cout<<(" Valvula Acionada no local control")<<endl;
7     });
8     while(1){
9         if (context->isMQTTConnected()){
10             cout<<"MQTT Connected "<<endl;
11             context->subscribeTo( "lem/valve" ,[]( string pay){
12                 cout<<" Received: "<<pay<<endl;
13                 cout<<(" Valvula Acionada no local control")<<endl;
14                 if (pay == "Acionado"){
15                     gpio_set_level(GPIO_OUTPUT_IO_0, 1);
16                 }
17                 if (pay == "Desacionado"){
18                     gpio_set_level(GPIO_OUTPUT_IO_0, 0);
19                 }
20             });
21             break;
22         }
23         vTaskDelay(5000 / portTICK_PERIOD_MS);
24     }
25     while(1){
26         cout<<" Waiting Data... "<<endl;
27         vTaskDelay(1000 / portTICK_PERIOD_MS);
28     }
29 };

```

APÊNDICE D

Código Utilizado no Experimento 2

```

1 void vTaskLocalControl( void *pvParameters){
2     CustomEventHandler* context = (CustomEventHandler*)pvParameters;
3     cout<<(" Task Local Control ");
4     context->subscribeTo( "lem/valveA ",[]( string pay){
5         cout<<" Received: "<<pay<<endl ;
6         cout<<(" Valvula Acionada no local control ")<<endl ;
7     });
8     while(1){
9         if ( context->isMQTTConnected() ){
10             cout<<" MQTT Connected "<<endl ;
11             context->subscribeTo( "lem/valveA ",[]( string pay){
12                 cout<<" Received: "<<pay<<endl ;
13                 cout<<(" Valvula Acionada no local control ")<<endl ;
14                 if ( pay == "Acionado" ){
15                     gpio_set_level(GPIO_OUTPUT_IO_0, 1);
16                 }
17                 if ( pay == "Desacionado" ){
18                     gpio_set_level(GPIO_OUTPUT_IO_0, 0);
19                 }
20             });
21             break;
22         }
23         vTaskDelay(5000 / portTICK_PERIOD_MS);
24     }
25     while(1){
26         if ( gpio_get_level(GPIO_INPUT_IO_0) == 1){
27             cout<<" Auto "<<endl ;
28             context->postData( "lem/sensorA ", " Acionado " );
29         }
30         if ( gpio_get_level(GPIO_INPUT_IO_1) == 1){

```

```
31         cout<<"Auto"<<endl;
32         context->postData( "lem/sensorA" , "Desacionado" );
33     }
34     vTaskDelay(1000 / portTICK_PERIOD_MS);
35 }
36 };
```

Anexos

ANEXO **A**

**Documentação official do modo *Bridge*
disponibilizado pelo Mosquitto®**

Cookie settings



mosquitto.conf man page

Configuring Bridges

Multiple bridges (connections to other brokers) can be configured using the following variables.

Bridges cannot currently be reloaded on reload signal.

```
address address[:port] [address[:port]] ,  
addresses address[:port] [address[:port]]
```

Specify the address and optionally the port of the bridge to connect to. This must be given for each bridge connection. If the port is not specified, the default of 1883 is used.

If you use an IPv6 address, then the port is not optional.

Multiple host addresses can be specified on the address config. See the `round_robin` option for more details on the behaviour of bridges with multiple addresses.

```
bridge_attempt_unsubscribe [true|false]
```

If a bridge has topics that have "out" direction, the default behaviour is to send an unsubscribe request to the remote broker on that topic. This means that changing a topic direction from "in" to "out" will not keep receiving incoming messages. Sending these unsubscribe requests is not always desirable, setting `bridge_attempt_unsubscribe` to `false` will disable sending the unsubscribe request. Defaults to `true`.

```
bridge_bind_address ip address
```

If you need to have the bridge connect over a particular network interface, use `bridge_bind_address` to tell the bridge which local IP address the socket should bind to, e.g. `bridge_bind_address 192.168.1.10`.

`bridge_max_packet_size value`

If you wish to restrict the size of messages sent to a remote bridge, use this option. This sets the maximum number of bytes for the total message, including headers and payload. Note that MQTT v5 brokers may provide their own `maximum-packet-size` property. In this case, the smaller of the two limits will be used. Set to 0 for "unlimited".

`bridge_outgoing_retain [true|false]`

Some MQTT brokers do not allow retained messages. MQTT v5 gives a mechanism for brokers to tell clients that they do not support retained messages, but this is not possible for MQTT v3.1.1 or v3.1. If you need to bridge to a v3.1.1 or v3.1 broker that does not support retained messages, set the `bridge_outgoing_retain` option to `false`. This will remove the retain bit on all outgoing messages to that bridge, regardless of any other setting. Defaults to `true`.

`bridge_protocol_version version`

Set the version of the MQTT protocol to use with for this bridge. Can be one of `mqttv50`, `mqttv311` or `mqttv31`. Defaults to `mqttv311`.

`cleansession [true|false]`

Set the clean session option for this bridge. Setting to `false` (the default), means that all subscriptions on the remote broker are kept in case of the network connection dropping. If set to `true`, all subscriptions and messages on the remote broker will be cleaned up if the connection drops. Note that setting to `true` may cause a large amount of retained messages to be sent each time the bridge reconnects.

If you are using bridges with `cleansession` set to `false` (the default), then you may get unexpected behaviour from incoming topics if you change what topics you are subscribing to. This is because the remote broker keeps the subscription for the old topic. If you have this problem, connect your bridge with `cleansession` set to `true`, then reconnect with `cleansession` set to `false` as normal.

`local_cleansession [true|false]`

The regular `cleansession` covers both the local subscriptions and the remote subscriptions. `local_cleansession` allows splitting this. Setting `false` will mean that the local connection will preserve subscription, independent of the remote connection.

Defaults to the value of `bridge.cleansession` unless explicitly specified.

`connection name`

This variable marks the start of a new bridge connection. It is also used to give the bridge a name which is used as the client id on the remote broker.

`keepalive_interval seconds`

Set the number of seconds after which the bridge should send a ping if no other traffic has occurred. Defaults to 60. A minimum value of 5 seconds is allowed.

`idle_timeout seconds`

Set the amount of time a bridge using the lazy start type must be idle before it will be stopped. Defaults to 60 seconds.

`local_clientid id`

Set the clientid to use on the local broker. If not defined, this defaults to `local <remote_clientid>`. If you are bridging a broker to itself, it is important that `local_clientid` and `remote_clientid` do not match.

`local_password password`

Configure the password to be used when connecting this bridge to the local broker. This may be important when authentication and ACLs are being used.

`local_username username`

Configure the username to be used when connecting this bridge to the local broker. This may be important when authentication and ACLs are being used.

`notifications [true | false]`

If set to `true`, publish notification messages to the local and remote brokers giving information about the state of the bridge connection. Retained messages are published to the topic `$SYS/broker/connection/<remote_clientid>/state` unless otherwise set with `notification_topic`s. If the message is 1 then the connection is active, or 0 if the connection has failed. Defaults to `true`.

This uses the Last Will and Testament (LWT) feature.

`notifications_local_only [true | false]`

If set to `true`, only publish notification messages to the local broker giving information about the state of the bridge connection. Defaults to `false`.

`notification_topic topic`

Choose the topic on which notifications will be published for this bridge. If not set the messages will be sent on the topic
`$SYS/broker/connection/<remote_clientid>/state`.

`remote_clientid id`

Set the client id for this bridge connection. If not defined, this defaults to 'name.hostname', where name is the connection name and hostname is the hostname of this computer.

This replaces the old "clientid" option to avoid confusion with local/remote sides of the bridge. "clientid" remains valid for the time being.

```
remote_password  value
```

Configure a password for the bridge. This is used for authentication purposes when connecting to a broker that supports MQTT v3.1 and up and requires a username and/or password to connect. This option is only valid if a `remote_username` is also supplied.

This replaces the old "password" option to avoid confusion with local/remote sides of the bridge. "password" remains valid for the time being.

```
remote_username  name
```

Configure a username for the bridge. This is used for authentication purposes when connecting to a broker that supports MQTT v3.1 and up and requires a username and/or password to connect. See also the `remote_password` option.

This replaces the old "username" option to avoid confusion with local/remote sides of the bridge. "username" remains valid for the time being.

```
restart_timeout  base cap ,  
restart_timeout  constant
```

Set the amount of time a bridge using the automatic start type will wait until attempting to reconnect.

This option can be configured to use a constant delay time in seconds, or to use a backoff mechanism based on "Decorrelated Jitter", which adds a degree of randomness to when the restart occurs, starting at the base and increasing up to the cap. Set a constant timeout of 20 seconds:

```
restart_timeout 20
```

Set backoff with a base (start value) of 10 seconds and a cap (upper limit) of 60 seconds:

```
restart_timeout 10 30
```

Defaults to jitter with a base of 5 seconds and cap of 30 seconds.

```
round_robin [true|false]
```

If the bridge has more than one address given in the address/addresses configuration, the `round_robin` option defines the behaviour of the bridge on a failure of the bridge connection. If `round_robin` is `false`, the default value, then the first address is treated as the main bridge connection. If the connection fails, the other secondary addresses will be attempted in turn. Whilst connected to a secondary bridge, the bridge will periodically attempt to reconnect to the main bridge until successful.

If `round_robin` is `true`, then all addresses are treated as equals. If a connection fails, the next address will be tried and if successful will remain connected until it fails.

```
start_type [automatic|lazy|once]
```

Set the start type of the bridge. This controls how the bridge starts and can be one of three types: `automatic`, `lazy` and `once`. Note that RSMB provides a fourth start type "manual" which isn't currently supported by mosquitto.

`automatic` is the default start type and means that the bridge connection will be started automatically when the broker starts and also restarted after a short delay (30 seconds) if the connection fails.

Bridges using the `lazy` start type will be started automatically when the number of queued messages exceeds the number set with the `threshold` option. It will be stopped automatically after the time set by the `idle_timeout` parameter. Use this start type if you wish the connection to only be active when it is needed.

A bridge using the `once` start type will be started automatically when the broker starts but will not be restarted if the connection fails.

threshold count

Set the number of messages that need to be queued for a bridge with lazy start type to be restarted. Defaults to 10 messages.

topic pattern [[out | in | both] qos-level] local-prefix remote-prefix]

Define a topic pattern to be shared between the two brokers. Any topics matching the pattern (which may include wildcards) are shared. The second parameter defines the direction that the messages will be shared in, so it is possible to import messages from a remote broker using *in*, export messages to a remote broker using *out* or share messages in both directions. If this parameter is not defined, the default of *out* is used. The QoS level defines the publish/subscribe QoS level used for this topic and defaults to 0.

The *local-prefix* and *remote-prefix* options allow topics to be remapped when publishing to and receiving from remote brokers. This allows a topic tree from the local broker to be inserted into the topic tree of the remote broker at an appropriate place.

For incoming topics, the bridge will prepend the pattern with the remote prefix and subscribe to the resulting topic on the remote broker. When a matching incoming message is received, the remote prefix will be removed from the topic and then the local prefix added.

For outgoing topics, the bridge will prepend the pattern with the local prefix and subscribe to the resulting topic on the local broker. When an outgoing message is processed, the local prefix will be removed from the topic then the remote prefix added.

When using topic mapping, an empty prefix can be defined using the place marker `""`. Using the empty marker for the topic itself is also valid. The table below defines what combination of empty or value is valid. The `Full Local Topic` and `Full Remote Topic` show the resulting topics that would be used on the local and remote ends of the bridge. For example, for the first table row if you

publish to `L/topic` on the local broker, then the remote broker will receive a message on the topic `R/topic`.

Pattern	Local Prefix	Remote Prefix	Validity	Full Local Topic	Full Remote Topic
pattern	L/	R/	valid	L/pattern	R/pattern
pattern	L/	""	valid	L/pattern	pattern
pattern	""	R/	valid	pattern	R/pattern
pattern	""	""	valid (no remapping)	pattern	pattern
""	local	remote	valid (remap single local topic to remote)	local	remote
""	local	""	invalid		
""	""	remote	invalid		
""	""	""	invalid		

To remap an entire topic tree, use e.g.:

```
topic # both 2 local/topic/ remote/topic/
```

This option can be specified multiple times per bridge.

Care must be taken to ensure that loops are not created with this option. If you are experiencing high CPU load from a broker, it is possible that you have a loop where each broker is forever forwarding each other the same messages.

See also the `cleansession` option if you have messages arriving on unexpected topics when using incoming topics.

Example Bridge Topic Remapping.

The configuration below connects a bridge to the broker at `test.mosquitto.org`. It subscribes to the remote topic `$SYS/broker/clients/total` and republishes the messages received to the local topic `test/mosquitto/org/clients/total`

```
connection test-mosquitto-org
address test.mosquitto.org
cleansession true
topic clients/total in 0 test/mosquitto/org/ $SYS/broker/
try_private [true|false]
```

If `try_private` is set to `true`, the bridge will attempt to indicate to the remote broker that it is a bridge not an ordinary client. If successful, this means that loop detection will be more effective and that retained messages will be propagated correctly. Not all brokers support this feature so it may be necessary to set `try_private` to `false` if your bridge does not connect properly.

Defaults to `true`.

SSL/TLS Support

The following options are available for all bridges to configure SSL/TLS support.

`bridge_alpn alpn`

Configure the application layer protocol negotiation option for the TLS session. Useful for brokers that support both websockets and MQTT on the same port.

`bridge_cafile file path`

One of `bridge_cafile` or `bridge_capath` must be provided to allow SSL/TLS support.

`bridge_cafile` is used to define the path to a file containing the PEM encoded CA certificates that have signed the certificate for the remote broker.

`bridge_capath file path`

One of `bridge_capath` or `bridge_cafile` must be provided to allow SSL/TLS support.

`bridge_capath` is used to define the path to a directory containing the PEM encoded CA certificates that have signed the certificate for the remote broker. For `bridge_capath` to work correctly, the certificate files must have ".crt" as the file ending and you must run "openssl rehash <path to bridge_capath>" each time you add/remove a certificate.

`bridge_certfile file path`

Path to the PEM encoded client certificate for this bridge, if required by the remote broker.

`bridge_identity identity`

Pre-shared-key encryption provides an alternative to certificate based encryption. A bridge can be configured to use PSK with the `bridge_identity` and `bridge_psk` options. This is the client identity used with PSK encryption. Only one of certificate and PSK based encryption can be used on one bridge at once.

`bridge_insecure [true | false]`

When using certificate based TLS, the bridge will attempt to verify the hostname provided in the remote certificate matches the host/address being connected to. This may cause problems in testing scenarios, so `bridge_insecure` may be set to `true` to disable the hostname verification.

Setting this option to `true` means that a malicious third party could potentially impersonate your server, so it should always be set to `false` in production environments.

bridge_keyfile *file path*

Path to the PEM encoded private key for this bridge, if required by the remote broker.

bridge_psk *key*

Pre-shared-key encryption provides an alternative to certificate based encryption. A bridge can be configured to use PSK with the `bridge_identity` and `bridge_psk` options. This is the pre-shared-key in hexadecimal format with no "0x". Only one of certificate and PSK based encryption can be used on one bridge at once.

bridge_require_ocsp [true | false]

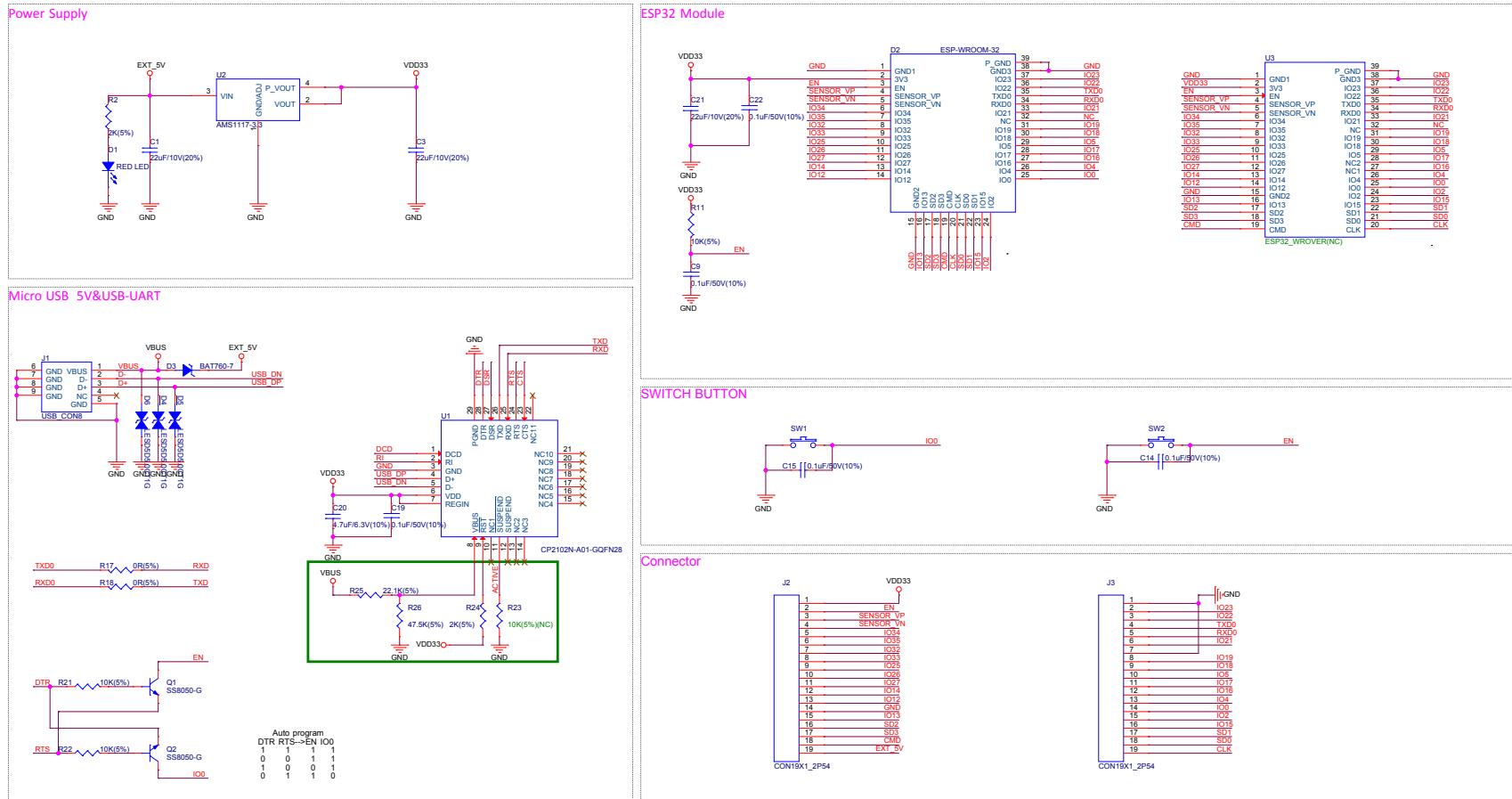
When set to true, the bridge requires OCSP on the TLS connection it opens as client.

bridge_tls_version *version*

Configure the version of the TLS protocol to be used for this bridge. Possible values are `tlsv1.3`, `tlsv1.2` and `tlsv1.1`. Defaults to `tlsv1.2`. The remote broker must support the same version of TLS for the connection to succeed.

ANEXO **B**

**Desenho Esquemático Módulo
ESP32-DevKitC-v4**



ESPRESSIF

乐鑫信息科技(上海)有限公司

Title: ESP32_Device_V4

Size: C Document Number: <Doc>

Date: Wednesday, December 06, 2017

Rev: V4

Sheet 1 of 1

ANEXO C

**Desenho Esquemático Módulo
ESP32-WROOM-32E**

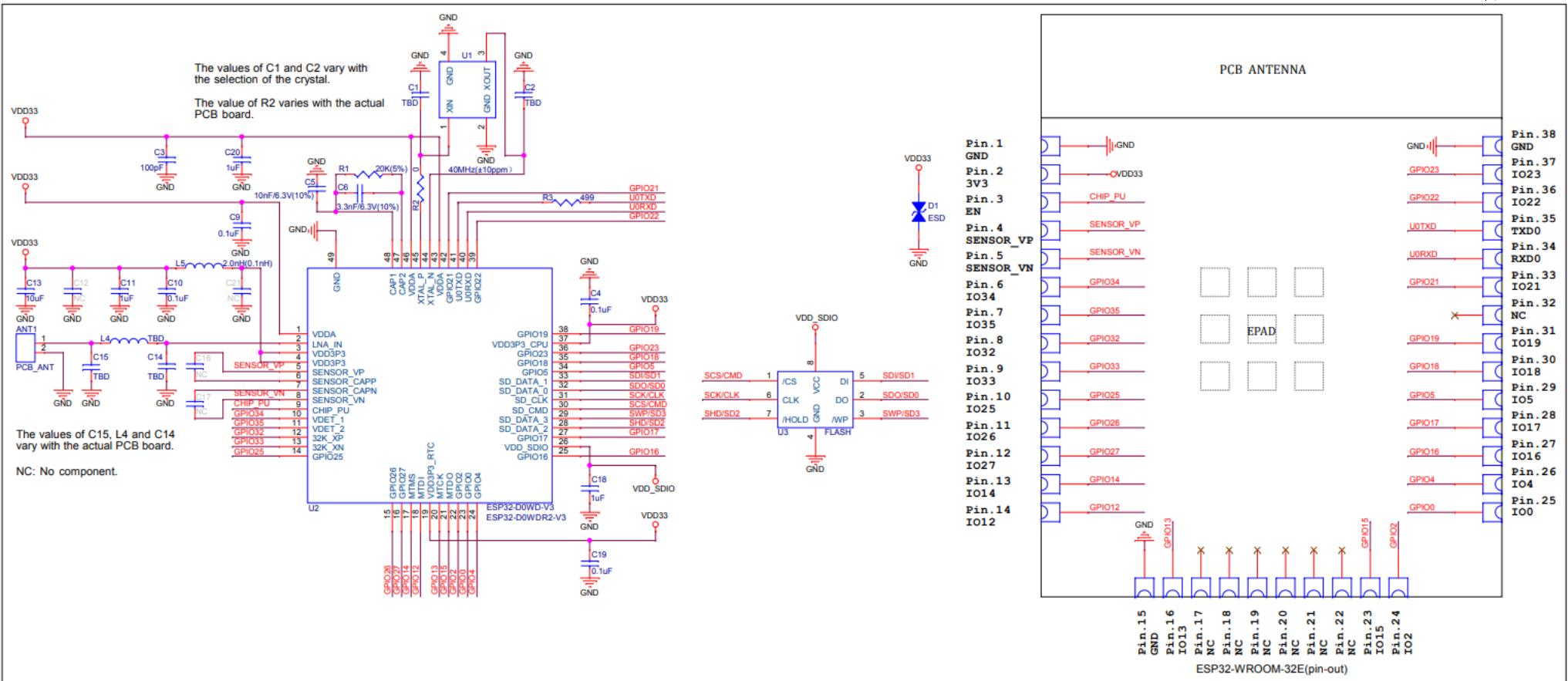


Figure 5: ESP32-WROOM-32E Schematics