CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA


Universidade Federal
de Campina Grande

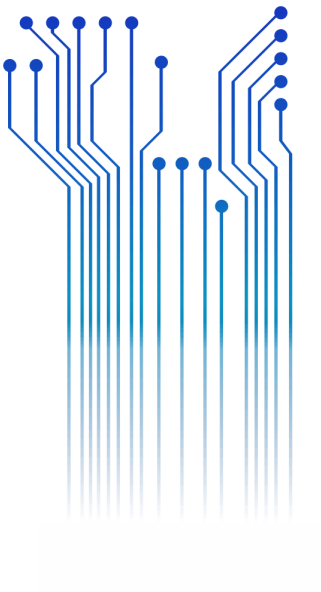
ARTUR SOUSA FREITAS

Centro de Engenharia
Elétrica e Informática

TRABALHO DE CONCLUSÃO DE CURSO

# ESTUDO E IMPLEMENTAÇÃO DO PROTOCOLO DE COMUNICAÇÃO MQTT APLICADO A UM SISTEMA DE AUTOMAÇÃO PREDIAL


Departamento de
Engenharia Elétrica



Campina Grande
2017

ARTUR SOUSA FREITAS

# ESTUDO E IMPLEMENTAÇÃO DO PROTOCOLO DE COMUNICAÇÃO MQTT APLICADO A UM SISTEMA DE AUTOMAÇÃO PREDIAL

*Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.*

Área de Concentração: Internet das Coisas

Orientador:
Professor Péricles Rezende Barros, Ph.D.

Campina Grande
2017

ARTUR SOUSA FREITAS

# ESTUDO E IMPLEMENTAÇÃO DO PROTOCOLO DE COMUNICAÇÃO MQTT APLICADO A UM SISTEMA DE AUTOMAÇÃO PREDIAL

*Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.*

Área de Concentração: Internet das Coisas

Aprovado em / /

**Professor Avaliador**
Universidade Federal de Campina Grande
Avaliador

**Professor Péricles Rezende Barros, Ph. D.**
Universidade Federal de Campina Grande
Orientador, UFCG

Dedico este trabalho aos meus pais e irmãos, por todo o apoio e incentivo durante minha vida acadêmica.

# AGRADECIMENTOS

Agradeço a Deus, em primeiro lugar, pela minha vida, pela minha saúde e pelas oportunidades cujo me foram concedidas.

Agradeço também à meus pais João e  Arleide Freitas por terem se esforçado tanto para me proporcionar uma boa educação, por terem me alimentado com saúde, força e coragem, as quais foram essenciais para superação de todas as adversidades ao longo desta caminhada.

Agradeço também aos meus irmãos Raul e Rafael,por todo o incentivo e suporte durante minha vida acadêmica.

Agradeço ao Professor Péricles Barros, por ter aceitado me orientar neste trabalho, ao Prof. Rafael Lima  ao técnico Simões e aos amigos José Igor e Felipe Pontes pelo apoio durante a execução deste trabalho.

Enfim, agradeço a todos que de alguma forma, passaram pela minha vida e contribuíram para a construção de quem sou hoje.

*"Remembering that you are going to die is the best way I know to avoid the trap of thinking you have something to lose. You are already naked. There is no reason not to follow your heart"*

Steve Jobs

# RESUMO

O número de objetos inteligentes com capacidade de monitoramento, processamento e comunicação tem crescido significativamente nos últimos anos. Neste cenário, a Internet das Coisas (Internet of Things (IoT)) surge ao habilitar que estes objetos estejam conectados à Internet e promovam comunicação entre usuários e dispositivos. Por outro lado, emergem diversos desafios no âmbito social, teórico e prático. Por exemplo, é necessário um modo eficiente para a troca de dados entre estes bilhões de objetos conectados uma vez que estes dispositivos, em sua maioria, possuem recursos como mémoria e processamento bastante limitados.

Este trabalho de conclusão de curso se propôs , em um primeiro momento, realizar um estudo do protocolo de comunicação MQTT, protocolo este cujo representa uma ótima alternativa para projetos de IoT quando comparado com outros protocolos como o http. Em seguida, como prova de conceito do protocolo, foi desenvolvido um sistema de automação predial cujo teve como objetivo realizar o monitoramento em tempo real da temperatura, umidade e luminosidade de uma sala e efetuar o controle do ar condicionado e de algumas lâmpadas presentes no recinto com o auxílio do protocolo de comunicação MQTT.

**Palavras-chave:** MQTT, Internet das Coisas, Automação Predial.

# ABSTRACT

The number of smart devices with capacity of monitoring, processing and communication has increased significantly during the past years. In this scenario the Internet of Things (IoT) comes to enable these objects to be connected to the Internet and promote communication between users and devices. However, several practical and social challenges emerges. For instance, it is necessary an efficient way for exchanging data between these billions of devices once these objects, in its majority, have resources such as memory and processing very limited.

The present work proposed, in a first moment, make an study of the MQTT communication protocol. Due to its simplicity and efficiency, this protocol represents a great alternative for IoT projects. Following, as a proof of concept of the protocol, it was developed an automation system for monitoring the temperature, humidity and luminosity of a room and control the air conditioner and some lamps inside the room by using the MQTT protocol.

**Keywords:** MQTT, Internet of Things, Building automation.

# LISTA DE ILUSTRAÇÕES

Figura 1: Modelo *Publish/Subscribe* ..... 16
Figura 2: ORGANIZAÇÃO DOS PROTOCOLOS EM CAMADAS ..... 18
Figura 3: conexão mqtt ..... 19
Figura 4: fluxo de pacotes - assinatura de um tópico ..... 24
Figura 5: FLUXO DE PACOTES - CANCELAMENTO DE ASSINATURA ..... 25
Figura 6: Programa desenvolvido no ambiente node-red ..... 34
Figura 7: Arduino IDE ..... 35
Figura 8: System Workbench for stm32 ..... 36
Figura 9: ESP8266 - ESP 12 ..... 36
Figura 10: módulo 1 - ESP8266, LDR e Led Emissor Infravermelho ..... 37
Figura 11: Módulo 2 - Arduino MKR1000 e sensor DHT22 ..... 38
Figura 12: Plataforma experimental ..... 39
Figura 13: Raspberry pi modelo B ..... 39
Figura 14: STM32F7 – DISCOVERY ..... 41
Figura 15: Interface Web desenvolvida ..... 42
Figura 16: Interface gráfica desenvolvida ..... 43
Figura 17: AR CONDICIONADO INSTALADO NA SALA ..... 43

# LISTA DE TABELAS

Tabela 1: PACOTE DE CONTROLE CONNECT ..... 20
Tabela 2: PACOTE DE CONTROLE CONNACK ..... 21
Tabela 3: Códigos de retorno - pacote connack ..... 21
Tabela 4: PACOTE DE CONTROLE PUBLISH ..... 22
Tabela 5: PACOTE DE CONTROLE SUBSCRIBE ..... 23
Tabela 6: PACOTE DE CONTROLE SUBACK ..... 23
Tabela 7: Códigos de retorno pacote suback ..... 24

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| TCP | *Transport Control Protocol* |
| IP | *Internet Protocol* |
| LWT | *Last Will and Testament* |
| M2M | *Machine to Machine* |
| IoT | *Internet of Things* |
| HTTP | *Hypertext Transfer Protocol* |

# SUMÁRIO

1 Introdução ..... 13
1.1 Objetivos ..... 14
1.2 Visão Geral do Texto ..... 14
2 Introdução ao Protocolo MQTT ..... 14
2.1 Padrão *Publish/Subscribe* ..... 15
2.2 Cliente, *Broker* e Conexão ..... 17
2.3 Publicação, Assinatura, Cancelamento de Assinatura ..... 22
2.4 Estrutura de Tópicos ..... 25
3 Especificações do Protocolo ..... 27
3.1 Níveis de Qualidade de Serviço ..... 27
3.2 Sessão Persistente ..... 28
3.3 Mensagens Retidas ..... 29
3.4 *Last Will and Testament* (LWT) ..... 29
3.5 *Keep Alive* e *Client Take-Over* ..... 30
4 Desenvolvimento do Sistema ..... 32
4.1 Objetivo do sistema ..... 32
4.2 *Software* ..... 32
4.3 Descrição dos *hardwares* utilizados ..... 36
4.4 Funcionamento do Sistema ..... 41
5 Conclusão ..... 44
5.1 Considerações Finais ..... 44
5.2 Perspectivas Futuras ..... 44
Referências ..... 45
APÊNDICE A – Código Fonte Esp8266 módulo 1 ..... 46
APÊNDICE B – Código Fonte Arduino MKR1000 (módulo 2) ..... 52

# 1 INTRODUÇÃO

A Internet é um sistema global de redes de computadores que usam um conjunto de protocolos (TCP/IP) para conectar dispositivos ao redor do mundo. A internet surgiu a partir de pesquisas militares no auge da Guerra Fria (1969) e inicialmente foi utilizada para a troca de mensagens entre universidades e centros de pesquisas com intuito de compartihar informações valiosas e descentralizar as mesmas. Com o passar do tempo, a Internet se tornou bastante poupular, deixando assim de ser usada apenas para conectar computadores entre si, mas também para conectar pessoas por meio de redes sociais. Hoje em dia, milhões de pessoas criam perfis pessoais e profissionais e se conectam umas as outras por meio da rede de computadores.

No cenário atual, uma nova fase da internet se encontra em ascensão. A esta nova etapa da Internet dá-se o nome de Internet das Coisas. Nesta nova tendência, a internet está sendo usada para conectar dispositivos eletrônicos utilizados no dia a dia(coisas) como aparelhos eletrodomésticos, máquinas industriais, meios de transporte, entre muitos outros. A internet das coisas (IoT) permite que objetos de diferentes naturezas sejam monitorados, controlados e que troquem informações entre si. Isto possiblita uma maior integração entre o mundo físico e o mundo dos sistemas computacionais, o que resulta em uma melhora da eficiência, precisão e benefícios econômicos por parte dos usuários.

Diante disso, é necessário que haja um padrão de comunicação entre os dispositivos conectados de forma que eles consigam conversar entre si de forma interoperável. Uma vez que, estes dispositivos, em sua maioria, possuem recursos de memória, processamento e energia bastante limitados,protocolos já consagrados como o http não são a opção mais adequada para projetos de IoT. Em consequência disso, estão surgindo alguns protocolos desenhados exclusivamente para estas aplicações e cujo levam em conta as limitações acimas citadas.

Dentre estes protocolos, está o protocolo de comunicação MQTT. Este protocolo foi desenvolvido pela IBM  e por ser simples e eficiente está sendo bastante utilizado em aplicações de IoT e possui potencial para se tornar o protocolo padrão da Internet das Coisas.

## 1.1 OBJETIVOS

O objetivo deste trabalho de conclusão de curso foi realizar um estudo do funcionamento e das especificações do protocolo de comunicação MQTT e aplicar este protocolo no desenvolvimento de um sistema de automação predial capaz de monitorar a temperatura, umidade e luminosidade de uma sala e efetuar o controle do ar condicionado e de lâmpadas presentes no recinto em tempo real e de forma remota.

## 1.2 VISÃO GERAL DO TEXTO

Este relatório está dividido em 7 capítulos. O capítulo 1 compreende a presente introdução em que são apresentados os objetivos deste trabalho e a visão geral do texto.

Nos capítulos 2 e 3 são apresentados os conceitos básicos do protocolo , seguidos das especificações do protocolo.

Nos capítulo 4 é descrito o processo de desenvolvimento do sistema e o seu funcionamento.

Por fim, no capítulo 5 são apresentadas as considerações finais acerca do assunto tratado no trabalho e as perspectivas futuras do projeto desenvolvido.

# 2 INTRODUÇÃO AO PROTOCOLO MQTT

MQTT é um protocolo Cliente/Servidor de troca de mensagens e que utiliza o padrão *publish/subscribe*. É leve, aberto, simples e projetado para ser fácil de implementar. Estas características o tornam ideal para ser usado em muitas situações, incluindo ambientes restritos como por exemplo nos contextos de comunicação Máquina para Máquina (*Machine to Machine(*M2M*)*) e IoT onde códigos que ocupam pouca memória são necessários e/ou a largura de banda da rede é limitada.[2]

O protocolo funciona acima da pilha de protocolos TCP/IP, ou acima de outros protocolos de rede que fornecem conexões sem perdas de pacotes e bidirecionais. Dentre as características do protocolo MQTT estão:

- ✓ Uso do padrão de mensagens *publish/subscribe* que fornence distribuição de mensagens de um-para-muitos e desacoplamento entre as aplicações.
- ✓ Um transporte de mensagens que é agnóstico em relação ao conteúdo do payload, ou seja dados de qualquer tipo podem sem transmitidos.
- ✓ Três níveis de qualidade de serviço.
- ✓ Um pequeno *overhead* de transporte e trocas de mensagens minimizadas para reduzir o tráfego de dados na rede.
- ✓ Um mecanismo para notificar os dispositivos interessados quando desconexões anormais ocorrerem.

MQTT foi inventado pelo Dr. Andy Stanford-Clark da IBM , e por Arlen Nipper of Arcom, em 1999. Neste capítulo serão apresentados os conceitos básicos por trás do protocolo e como ele pode pode ser utilizado para conectar diversos dispositivos através da Internet.[1]

## 2.1 PADRÃO *PUBLISH/SUBSCRIBE*

O protocolo MQTT é baseado no princípio de publicação de mensagens e assinatura de estruturas denominadas tópicos, o que é tipicamente referido como modelo *publish/subscribe.* Neste modelo, clientes podem assinar tópicos que sejam pertinentes a eles e portanto receber quaisquer mensagens que sejam publicadas nestes tópicos. Alternativamente, clientes também podem publicar mensagens em tópicos, deixando-as disponíveis para todos os assinantes destes tópicos. [1]

O padrão *publish/subscribe* (pub/sub) é uma alternativa ao tradicional modelo cliente-servidor, onde um cliente se comunica diretamente com um servidor. Neste padrão o cliente, que está enviando uma determinada mensagem (chamado publicador) é desacoplado de outro cliente (ou mais clientes), que estão recebendo a mensagem (chamado assinante). Isto signfica que o publicador e assinante não sabem a respeito um do outro. Existe um terceiro componente, chamado *broker*, que é conhecido por ambos o publicador e o assinante, que é responsável por filtrar todas as mensagens que chegam até ele e as distribui adequadamente.

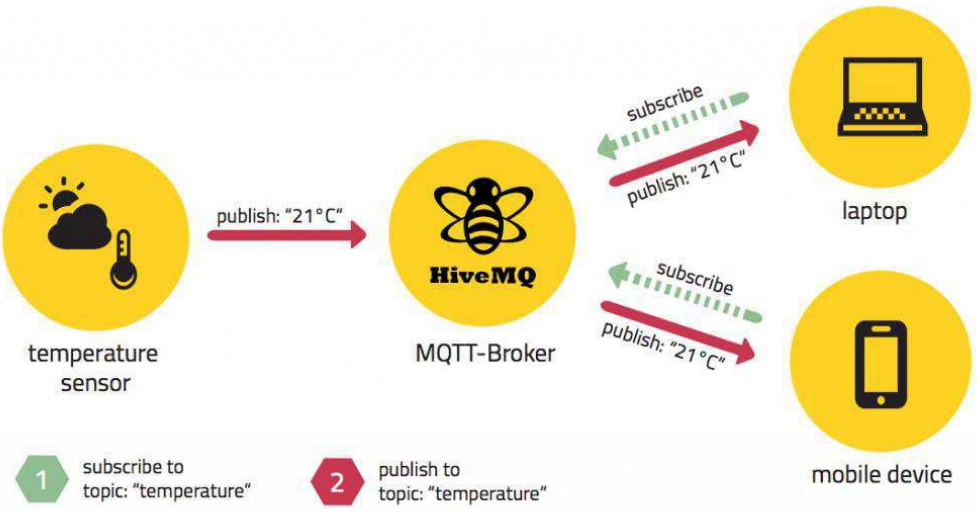
Como já foi mencionado, o principal aspecto no padrão *pub/sub* é o desacoplamento entre o publicador e o assinante, que pode ser caracterizado em três tipos:

- **Desacoplamento de espaço:** Publicador e assinante não precisam ter conhecimento um do outro (pelo endereço de ip e porta por exemplo).
- **Desacoplamento de tempo:** Publicador e assinante não precisam ser executados ao mesmo tempo.
- **Desacoplamento de sincronização:** As operações em ambos os componentes não são interrompidas durante a publicação ou recebimento de mensagens.

Em resumo, o padrão *publish/subscribe* desacopla o remetente e o receptor da mensagem através da filtragem destas mensagens. Desta forma, é possível que apenas determinados clientes recebam certas mensagens.

Na figura abaixo, é apresentado um exemplo do uso do padrão *pub/sub* para a troca de mensagens entre um dispositivo medidor de temperatura e outros clientes interessados em receber estes valores de temperatura.

FIGURA 1: MODELO *PUBLISH/SUBSCRIBE*




Fonte: HiveMQ Enterprise MQTT Broker

Como pode ser visto na imagem, o sensor de temperatura envia (*publish*) mensagens para um servidor (*broker*) contendo o valor de temperatura medido e este reenvia estas mensagens para todos os clientes que assinaram (*subscribe*) o tópico

"*temperature"*. Desta forma, os clientes assinantes receberão quaisquer mensagens que sejam publicadas neste tópico de forma imediata.

## 2.2 CLIENTE, *BROKER* E CONEXÃO

Nesta subseção serão apresentadas as definições de cliente e *broker* MQTT e uma explicação de como se dá a conexão entre um cliente e um *broker*.

- **Cliente**

Um cliente MQTT (também chamado de aplicação cliente) coleta informação de um determinado ambiente, conecta-se a um servidor de mensagens, e publica esta informação em um modo que permita que outros clientes ou aplicações recuperem esta mensagem. Um cliente MQTT também pode assinar (*subscribe*) um tópico,receber as mensagens associadas a estes tópicos, e então tomar decisões para controlar este dispositivo. [2]

Em suma, um cliente MQTT é um programa ou dispositivo que usa MQTT. Um cliente sempre estabelece a conexão de rede com um servidor. Ele pode:

- Publicar mensagens cujo outros clientes possam estar interessados.
- Solicitar mensagens cujo eles estejam interessados em receber
- Cancelar a assinatura para parar de receber mensagens
- Desconectar de um servidor.

O uso de bibliotecas cliente pode simplificar o processo de projeto de aplicações que utilizam clientes MQTT. Por exemplo, existem algumas bibliotecas escritas em diversas linguagens que podem ser utilizadas para habilitar o uso do protocolo MQTT em diversas plataformas. Quando estas bibliotecas são incorporadas em aplicações MQTT, um cliente MQTT funcional pode ser criado com apenas algumas linhas de código. Neste trabalho foi utilizada a biblioteca *PubSubClient* cujo é uma implementação compatível com diversas plataformas dentre elas a plataforma Arduino e será abordada em mais detalhes no decorrer deste texto.

- ***Broker***

Um *Broker* MQTT é um servidor que implementa o protocolo MQTT. Ele é o núcleo de qualquer protocolo *publish/subscribe.* Dependendo da implementação, o *broker* pode lidar com até milhares de clientes MQTT conectados. O principal papel deste servidor é receber todas as mensagens, filtrá-las, decidir quais clientes estão interessados nestas mensagens e encaminhar estas mensagens a todos os clientes assinantes [2]. O *broker* também é responsável por amarzenar as sessões de todos os clientes persistentes incluindo as assinaturas e as mensagens perdidas (Ver subseção 3.2). Outra responsabilidade de um *broker* é a autenticação e autorização dos clientes.

Em resumo, é um programa ou dispositivo que age como um intermediário entre o clientes que publicam mensagens e clientes que realizaram assinaturas. Um *broker* é responsável por:

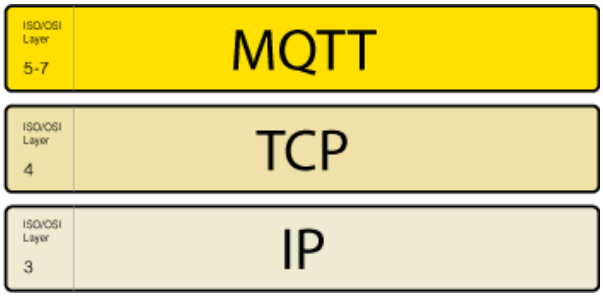
- Aceitar conexões de clients
- Aceitar mensagens publicadas por clientes
- Processar solicitações de assinaturas e cancelamento de assinaturas de clientes
- Encaminhar mensagens relacionadas às assinaturas dos clientes

Existem diversas implementações de servidores MQTT disponíveis. O uso destas implementações agiliza o desenvolvimento de projetos envolvendo MQTT. No projeto desenvolvido durante este trabalho de conclusão de curso foi utilizada a implementação *open source* Mosquitto cujo implementa a versão 3.1 do protocolo MQTT.

- **Conexão**

Como já mencionado o protocolo MQTT é baseado em cima da pilha de protocolos TCP/IP (Ver figura abaixo). Logo, ambos os clientes e o *broker* precisam possuir a pilha TCP/IP.

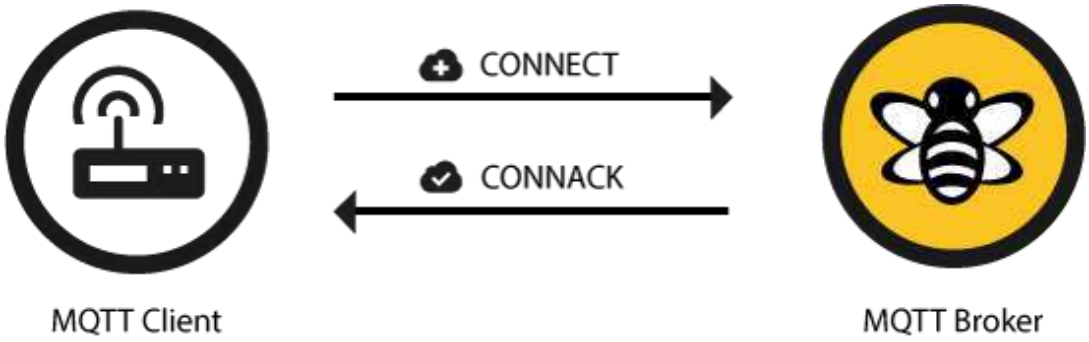
FIGURA 2: ORGANIZAÇÃO DOS PROTOCOLOS EM CAMADAS




Fonte: HiveMQ Enterprise MQTT Broker

Uma conexão MQTT ocorre sempre entre um cliente e o *broker*, ou seja, não existe conexão direta entre dois clientes. No protocolo MQTT a comunicação se dá por meio da troca de pacotes de controle do tipo TCP entre os dispositivos envolvidos na comunicação. A conexão é sempre iniciada pelo cliente através do envio de um pacote de controle do tipo CONNECT do cliente para o *broker*. O *broker,* então, responde com um pacote de controle do tipo CONNACK e um código de status (Ver figura abaixo). Uma vez que a conexão é estabelecida, o servidor manterá a conexão aberta contanto que o cliente não envie um comando de desconexão ou que a conexão seja perdida.

FIGURA 3: CONEXÃO MQTT




Fonte: HiveMQ Enterprise MQTT Broker

TABELA 1: PACOTE DE CONTROLE CONNECT

| **Atributo** | **Descrição** | **Exemplo** |
|---|---|---|
| clientId | Identificador do cliente | “cliente-1” |
| cleanSession | *Flag* que indica ao *broker* se a seção é persistente ou não (ver item 3.2) | “*true*” |
| username (opcional) | Nome de usuário | “admin” |
| password (opcional) | Senha | “senha” |
| lastWillTopic (opcional) | Tópico de *Last Will*(Ver item 3.4) | “admin/will” |
| lasWillQos (opcional) | QoS de *Last Will*(Ver itens 3.1 e 3.4) | “2” |
| lastWillMessage (opcional) | Mensagem de *Last Will* (ver item 3.2) | “desconexão inesperada” |
| lastWillRetain (opcional) | *Flag* de mensagem retida (Ver item 3.3) | *false* |
| keepAlive | Maior intervalo de tempo em segundos que o *broker* e o cliente podem ficar sem se comunicar (Ver item 3.5) | 60 |

Fonte: Elaborada pelo autor

Quando um *broker* recebe uma mensagem do tipo CONNECT, o mesmo responde com uma mensagem do tipo CONNACK. Este pacote possui dois parâmetros: o *flag* de sessão presente e o código de retorno de conexão. A seguir é apresentada uma tabela descrevendo este pacote de controle.

TABELA 2: PACOTE DE CONTROLE CONNACK

| **Atributo** | Descrição | Exemplo |
|---|---|---|
| **sessionPresent** | *Flag* que indica a presença/ausência de uma sessão antiga. | *true* |
| **returnCode** | Código de retorno referente à solicitaçã de conexão. | 0 |

Fonte: Elaborada pelo autor

Em seguida, é apresentada uma tabela com os possíveis códigos de retorno e seus respectivos significados.

TABELA 3: CÓDIGOS DE RETORNO - PACOTE CONNACK

| **Código de Retorno** | **Significado** |
|---|---|
| 0 | Conexão aceita |
| 1 | Conexão recusada, versão do protocolo inaceita |
| 2 | Conexão recusada, identificador rejeitado |
| 3 | Conexão recusada, servidor indisponível |
| 4 | Conexão recusada, nome de usuário ou senha inválidos |
| 5 | Conexão recusada, não autorizada |

Fonte: Elaborada pelo autor

## 2.3 PUBLICAÇÃO, ASSINATURA, CANCELAMENTO DE ASSINATURA

- **Publicação**

Uma vez que um cliente está conectado a um *broker*, ele pode publicar mensagens.No protocolo MQTT a filtragem de mensagens é baseada em estruturas denomidas tópicos. Desta forma, cada mensagem deve conter um tópico, que será usado pelo *broker* para encaminhar a mensagem aos clientes interessados. De forma geral, cada mensagem possui um *payload* que contém o dado a ser transmistido. A publicação de mensagens tanto de um cliente para o servidor quanto do servidor para um cliente é feita por meio do envio de um pacote de controle do tipo PUBLISH. De forma análoga ao pacote de controle CONNECT, este pacote também possui alguns atributos. Na tabela a seguir é apresentada a estrutura de um pacote PUBLISH.[5]

TABELA 4: PACOTE DE CONTROLE PUBLISH

| **Atributo** | Descrição | Exemplo |
|---|---|---|
| packetId | Identificador do pacote. | "4392" |
| topicName | Nome do tópico. | "*casa/temperatura*" |
| qos (Ver item 3.1) | Nível de qualidade de serviço. | 1 |
| payload | Mensagem a ser transmitida. | "25 C" |
| dupFlag | *Flag* que indica se a mensagem é duplicada ou não. | *false* |

Fonte: Elaborada pelo autor

- **Assinatura**

Para receber mensagens, um cliente precisa enviar um pacote do tipo SUBSCRIBE para o *broker*. Uma mensagem de assinatura é bastante simples e contém um único identificador de pacote e uma lista de assinaturas. Na tabela a seguir, é mostrada a estrutura deste pacote.[6]

TABELA 5: PACOTE DE CONTROLE SUBSCRIBE

| **Atributo** | Descrição | Exemplo |
|---|---|---|
| *packetId* | Identificador do pacote | 4312 |
| *qos1* | Nível de qualidade de serviço com o qual se deseja assinar o tópico 1 | 1 |
| *topic1* | Nome do tópico 1 | “topico/1” |
| *qos2* | Nível de qualidade de serviço com o qual se deseja assinar o tópico 2 | 0 |
| *Topic2* | Nome do tópico 2 | “topico/2” |
| . . . | Pares QoS/Tópico seguintes | . . . |

Fonte: Elaborada pelo autor

Cada assinatura será confirmada pelo *broker* através de o envio de uma mensagem do tipo SUBACK. Esta mensagem contém o mesmo identificador de pacote que a mensagem SUBSCRIBE enviada anteriormente e uma lista de códigos de retorno. A seguir é apresentada uma tabela descrevendo a estrutura do pacote de controle SUBACK.

TABELA 6: PACOTE DE CONTROLE SUBACK

| **Atributo** | Descrição | Exemplo |
|---|---|---|
| *packetId* | Identificador de pacote | 4312 |
| *returnCode 1* | Código de retorno referente a solicitação de assinatura do tópico 1 | 2 |
| *returnCode 2* | Código de retorno referente a solicitação de assinatura do tópico 2 | 0 |
| . . . | Códigos de retorno seguintes | . . . |

Fonte: Elaborada pelo autor

O *broker* envia um código de retorno para cada par tópico/QoS recebido na mensagem SUBSCRIBE. Portanto, se o pacote SUBSCRIBE possui 5 assinaturas, o *broker* responderá com 5 códigos de retorno para confirmar cada tópico com o nível de QoS garantido pelo *broker*. Os códigos de retorno podem variar conforme mostrado na tabela a seguir:

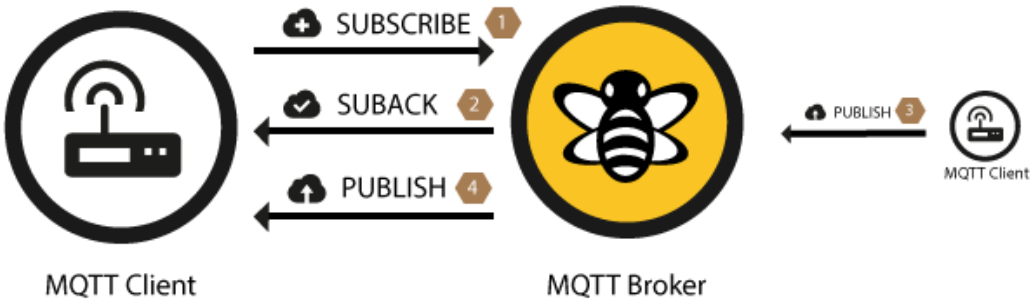
TABELA 7: CÓDIGOS DE RETORNO PACOTE SUBACK

| **Código de Retorno** | **Significado** |
|---|---|
| 0 | Sucesso – Máximo QoS 0 |
| 1 | Sucesso – Máximo QoS 1 |
| 2 | Sucesso – Máximo QoS 2 |
| 128 | Falha |

Fonte: Elaborada pelo autor

Depois que um cliente enviar uma mensagem de SUBSCRIBE e receber um SUBACK de forma bem sucedida, ele receberá cada mensagem publicada no tópico assinado. A figura abaixo ilustra este processo:

FIGURA 4: FLUXO DE PACOTES - ASSINATURA DE UM TÓPICO



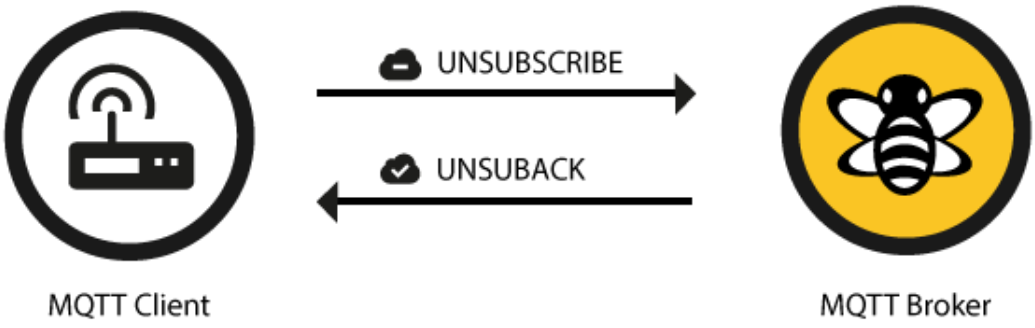

Fonte: HiveMQ Enterprise MQTT Broker

- **Cancelamento de assinatura**

O cancelamento de uma assinatura é feito por meio do envio de um pacote denominado UNSUBSCRIBE e que serve para deletar assinaturas existentes de um cliente em um *broker*. A estrutura de uma mensagem do tipo UNSUBSCRIBE é similar ao de um pacote SUBSCRIBE e também possui um identificador de pacote e uma lista de tópicos. O *broker* irá reconhecer a solicitação de cancelamento de assinatura com uma mensagem do tipo UNSUBACK. Esta mensagem contém apenas um identificador de pacote.[6]

Após receber uma mensagem de UNSUBACK proveniente do *broker*, o cliente pode assumir que todas as assinaturas presentes na mensagem de UNSUBSCRIBE foram deletadas. Na figura abaixo é ilustrado o processo de cancelamento de assinatura descrito:

FIGURA 5: FLUXO DE PACOTES - CANCELAMENTO DE ASSINATURA




Fonte: HiveMQ Enterprise MQTT Broker

## 2.4 ESTRUTURA DE TÓPICOS

Mensagens em MQTT são publicadas em tópicos, que podem ser pensadas como áreas de interesse. Clientes, por sua vez, se inscrevem para receber mensagens particulares por meio da assinatura de um tópico. Assinaturas podem ser explícitas,o que limita as mensagens que estão sendo recebidas a um tópico em específico ou podem usar designadores *wildcards,* como por exemplo o sinal (#) para receber mensagens de uma variedade de tópicos. [1]

Em outras palavras, um tópico é uma *string* que é utilizada pelo *broker* para filtrar as mensagens para cada cliente conectado. Cada nível de um tópico é separado por uma barra. A seguir alguns exemplos de tópicos:

- ✓ minhaCasa/salaDeEstar/temperatura
- ✓ Brasil/Paraiba/CampinaGrande/UFCG
- ✓ 5ff4a2ce-e485-40f4-826c-b1a5d81be9b6/status

- ➢ ***Wildcards***

*Wildcards* são artifícios que podem ser utilizados por um cliente que deseja assinar mais de um tópico ao mesmo tempo. *Wildcards* podem ser utilizados apenas no ato da inscrição em tópicos e não são permitidos na publicação de mensagens. Existem dois tipos de *wildcards*: único nível e múltiplos níveis.

- **Único nível: +**

O caractere '+' pode ser usado para substituir um nível da hierarquia do tópico. Por exemplo, o tópico "esporte/tenis/+" abrange os tópicos "esporte/tenis/jogador1" e esporte/tenis/jogador2" porém não inclui o tópico "esporte/tenis/jogador1/ranking".

- **Múltiplos níveis: #**

Em contrapartida ao *wildcard* de único nível, o *wildcard* de múltiplos níveis é um caractere que substitui vários níveis hierárquicos em um tópico. Por exemplo, se um cliente assinar o tópico "esporte/tenis/jogador1/#", ele receberá mensagens publicadas em todos os outros tópicos que possuem os níveis subsequentes, por exemplo:

- ✓ esporte/tenis/jogador1
- ✓ esporte/tenis/jogador1/ranking
- ✓ esporte/tenis/jogador1/placar

# 3 ESPECIFICAÇÕES DO PROTOCOLO

## 3.1 NÍVEIS DE QUALIDADE DE SERVIÇO

O nível de qualidade de serviço (*Quality of Service(QoS)*) é um acordo entre o remetente e o receptor da mensagem no que concerne a garantia de entrega da mensagem.No protocolo em estudo, existem 3 níveis de QoS:

- **QoS 0 -** ***at most once***

Este é o menor nível de qualidade de serviço e o mais rápido. Ao usar este nível de qualidade de serviço o cliente/*broker* tentará entregar a mensagem uma vez, porém sem nenhuma comfirmação de entrega.[7]

- **QoS 1 -** ***at least once***

Ao usar o nível de qualidade de serviço 1, existe a garantia de que a mensagem chegará ao seu destino pelo menos uma vez. Desta forma, é possível que a mensagem seja entregue mais de uma vez ao destinatário.[7]

- **QoS 2 -** ***exactly once***

Este é o mais alto nível de QoS e também o mais lento. Ao usar o nível 2 de qualidade de serviço é garantido que a mensagem será entregue apenas uma vez no destinatário.[7]

Em suma, o nível de QoS define o quanto o cliente/*broker* irá se esforçar para garantir que a mensagem seja entregue com sucesso ao destinatário. A escolha do nível de QoS a ser utilizado depende bastante da aplicação em questão e dos recursos de rede disponíveis. Por exemplo, se o dado transmitido não é tão importante ou se é enviando em pequenos intervalos de tempo e  para aplicação em questão é aceitável que algumas mensagens não cheguem ao destinatário,o nível de QoS 0 é suficiente. Por outro lado, se é crítico para a aplicação receber todas as mensagens e apenas uma vez deve-se optar pelo nível QoS 2.

## 3.2 SESSÃO PERSISTENTE

Quando um cliente se conecta a um *broker*, ele precisa realizar assinaturas nos tópicos de interesse para passar a receber mensagens. A princípio, caso haja um problema na rede e o cliente perca a conexão com o servidor e tenha que se reconectar ele deverá realizar novamente as assinaturas nos tópicos de interesse para receber mensagens. Uma vez que este processo de reassinatura pode ser bastante custoso e indesejável, o protocolo MQTT possui suporte à criação de sessões persistentes que resolvem este problema para o cliente.

Caso o cliente opte por iniciar uma sessão persistente com o *broker*, todas as informações relevantes da conexão serão salvas pelo *broker* de forma que caso haja uma reconexão o cliente não deverá se preocupar em assinar os tópicos novamente. Durante uma sessão persistente são armazenados os seguintes dados:

- ✓ Existência de uma sessão, mesmo que não haja assinaturas
- ✓ Todas as assinaturas
- ✓ Todas as mensagens com QoS maior que 0 , que ainda não foram confirmadas pelo cliente
- ✓ Todas as mensagens com QoS maior que 0 que o cliente perdeu enquanto estava desconectado

Desta forma, todos as informações enumeradas acima serão salvas pelo *broker* e estarão disponíveis no ato da reconexão do cliente.

Uma sessão persistente pode ser solicitada pelo cliente no ato da conexão com o *broker* por meio do uso do *flag clean session*. Caso este *flag* seja setado como falso o servidor não armazenará nenhum dado do cliente e toda a informação. Porém, se este *flag* for setado como verdadeiro as informações da conexão serão armazenadas e estarão disponíveis ao cliente caso ele precise se reconectar por alguma razão.

## 3.3 MENSAGENS RETIDAS

Por padrão, depois que uma mensagem é publicada e recebida pelos assinantes, a mensagem é descartada. Porém, um publicador pode especificar que a cópia de uma determinada mensagem seja retida e que ela seja enviada para os futuros assinantes do tópico no ato na inscrição.

Deletar mensagens depois de elas serem enviadas aos assinantes é uma opção adequada para informação de eventos, porém não é a melhor opção para informação de estados. Através do uso de mensagens retidas, novos assinantes não precisam esperar até que uma nova mensagem seja publicada de novo para saberem uma informação do estado atual. Por exemplo, um assinante inscrito em um tópico que informa o preço de um determinado estoque pode receber imediatamente uma mensagem retida que contém o valor atual do preço. Sem o uso de mensagens retidas, o assinante deveria esperar até que o preço mude para ter alguma informação a respeito do preço do estoque.[4]

Enviar uma mensagem retida da perspectiva do desenvolvedor é bastante simples e direto. É necessário apenas setar o *flag retain* de um mensagem do tipo PUBLISH para *true.* Em geral, cada biblioteca cliente MQTT fornece um modo fácil de realizar esta tarefa.

## 3.4 *LAST WILL AND TESTAMENT* (LWT)

O protocolo MQTT é normalmente utilizado em cenários onde redes não confiáveis são bastantes comuns. Portanto, é esperado que alguns clientes se desconectem abruptamente devido à um problema na rede ou até mesmo devido a algum problema na alimentação elétrica do sistema. Portanto, é necessário que o restante dos dispositivos estejam cientes desta desconexão para poderem tomar alguma ação apropriada. [1]

A ferramenta *Last Will* e Testament (LWT) é utilizada no protocolo MQTT para notificar os outros clientes a respeito de uma desconexão abrubta por parte de um dos clientes. Por meio desta ferramenta, cada cliente pode especificar uma mensagem a ser enviada pelo *broker* caso haja alguma desconexão inesperada. Desta forma, caso o *broker* detecte uma desconexão abrupta por parte do cliente, ele enviará a mensagem setada pelo cliente no tópico também determinado pelo cliente.

A mensagem, tópico e QoS de LWT são especificados pelo cliente no ato da conexão por meio do preenchimento de alguns atributos do pacote de controle CONNECT. De acordo com a especificação da versão 3.1.1 do protocolo MQTT, o *broker* enviará a mensagem de LWT de um determinado cliente nas seguintes situações:

- ✓ Uma falha na rede foi detectada pelo servidor.
- ✓ Um cliente falha em se comunicar-se dentro do tempo de *Keep Alive* (ver seção 3.5).
- ✓ Um cliente fecha a conexão sem enviar um pacote de DISCONNECT.
- ✓ O servidor fecha a conexão devido a um erro no protocolo.

## 3.5 *KEEP ALIVE* E *CLIENT TAKE-OVER*

Nesta subseção serão apresetadas outras duas ferramentas presentes na especificação do protocolo MQTT versão 3.1.1 e que têm como objetivo previnir possíveis erros na comunicação MQTT causados devidos a possíveis incidentes nas conexões TCP entre os dispositivos.

Conforme já foi mencionado, o MQTT é um protocolo projetado sobre a pilha de protocolos TCP/IP. Por padrão, o protocolo TCP poussui uma certa garantia de entrega de pacotes pela internet. Porém, é possível que um dos dispositivos envolvidos na comunicação perca a conexão com outro dispositivo e que o nó da rede que está funcionando normalmente não tome conhecimento desta desconexão e continue a enviar pacotes e esperar por confirmações. Este estado é chamado de conexão meio-aberta e pode comprometer a comunicação entre os dispositivos. A seguir são descritas duas funcionalidades do protocolo em estudo que buscam contornar esta situação.

- ➢ ***Keep Alive***

Segundo a especificação oficial do protocolo MQTT versão 3.1.1, O *keep alive* é um intervalo de tempo medido em segundos e serve para garantir que a conexão ainda se encontra aberta e que ambos o *broker* e o cliente encontram-se conectados um ao outro. Este intervalo de tempo é especificado pelo cliente e comunicado ao *broker* durante o estabelecimento da conexão. Em suma, este intervalo de tempo representa o maior período de tempo possível que o *broker* e o cliente podem ficar sem se comunicar.

Isto significa que contanto que mensagens sejam trocadas frequentemente entre o *broker* e o cliente e o tempo de *keep alive* não for excedido, não há necessidade de enviar uma mensagem extra para garantir que a conexão ainda se encontra aberta. Porém, caso o cliente não publique nenhuma mensagem durante o tempo de *keep alive* ele deve enviar um pacote de controle denominado PINGREQ para o *broker* com o objetivo de confirmar sua disponibilidade. Por padrão, o *broker* desconectará quaisquer clientes que não enviem um PINGREQ ou qualquer outra mensagem dentro do intervalo de *keep alive*. Ao receber um PINGREQ o *broker* deve responder com um pacote de controle do tipo PINGRESP. Ambos os pacotes PINGREQ e PINGRESP não possuem *payload* e são usados apenas com o propósito de verificar se a conexão está ativa.[2]

- ***Client Take-Over***

Caso um cliente se desconecte de forma inesperada de um *broker,*é muito provável que este mesmo cliente tente se reconectar novamente assim que possível. Existe a possibilidade de que o *broker* ainda possua uma conexão meio-aberta com o cliente e que isto atrapalhe a reconexão do mesmo cliente.Neste cenário, o *broker* irá fechar a conexão antiga e estabelecer a conexão com o novo cliente conectado. Este comportamento é denominado *client take-over* e garante que a conexão meio aberta não impossibilitará a reconexão do cliente.

# 4 DESENVOLVIMENTO DO SISTEMA

Neste capítulo será descrito o projeto do sistema desenvolvido durante este trabalho. Nas subseções seguintes serão abordados os objetivos do sistema, os principais recursos de *hardware* e *software* utilizados e uma descrição do funcionamento do sistema.

## 4.1 OBJETIVO DO SISTEMA

Em suma, o objetivo do sistema de automação desenvolvido neste trabalho de conclusão de curso é monitorar a temperatura, sensação térmica, luminosidade, umidade de uma sala ,efetuar o controle do ar condicionado presente na sala e de algumas lâmpadas presentes em uma plataforma experimental de forma remota por meio de uma interface *web*.

Além disso, este projeto também teve como intuito utilizar o protocolo MQTT para efetuar a comunicação entre os diversos nós da rede criada de forma a entrar em contato prático com o protocolo e validar o seu uso em aplicações de monitoramento remoto e automação predial/residencial.

## 4.2 *SOFTWARE*

- **Biblioteca Pub/Sub *Client***

A biblioteca *Pub/Sub Cliente* é uma biblitoeca *open source* criada por Nick O'Leary e que implementa a parte cliente do protocolo MQTT. Por meio do uso desta biblioteca é possível efetuar publicações e assinaturas de mensagens em um servidor que suporta MQTT[1]. A biblioteca utiliza a Arduino *Client* api para interagir com a camada de rede do *hardware* utilizado e é compatível com diversas plataformas, incluindo:

- ✓ Arduino Ethernet
- ✓ Arduino YUN
- ✓ Arduino WiFi Shield
- ✓ Intel Galileo/Edison

[1] Disponível em: < http://pubsubclient.knolleary.net/index.html>, Acessado em 21/04/2017

- ✓ Arduino MKR1000
- ✓ ESP8266

Neste projeto, esta biblioteca foi utilizada nas plataformas ESP8266 e Arduino MKR1000 e teve um papel bastante importante na comunicação entre os nós da rede.

- **Eclipse Mosquitto – MQTT *broker***

Eclipse Mosquitto é um servidor de mensagens *open source* que implementa o protocolo MQTT nas versões 3.1 e 3.1.1[2]. Neste trabalho, o mosquitto foi utilizado juntamente com a plataforma Raspberry Pi para servir de *broker* na rede de comunicação MQTT criada.

- **Biblioteca STemWin**

A STemWin é uma biblioteca gráfica profissional que habilita a construção de Interfaces gráficas com qualquer microcontrolador STM32 , qualquer *display* LCD/TFT e qualquer controlador LCD/TFT. A biblioteca STemWin possui suporte a vários formatos de imagens como JPG, GIF e PNG, diversos *widgets* (botões, checkbox...) e um servidor VNC que permite que um *display* local seja mostrado de forma remota.[3]

Neste trabalho a biblioteca STemWin foi utilizada juntamente com platafroma STM32F7- DISCO para construir a interface de usuário que possibilitou o controle do ar condicionado pelo usuário do sistema.

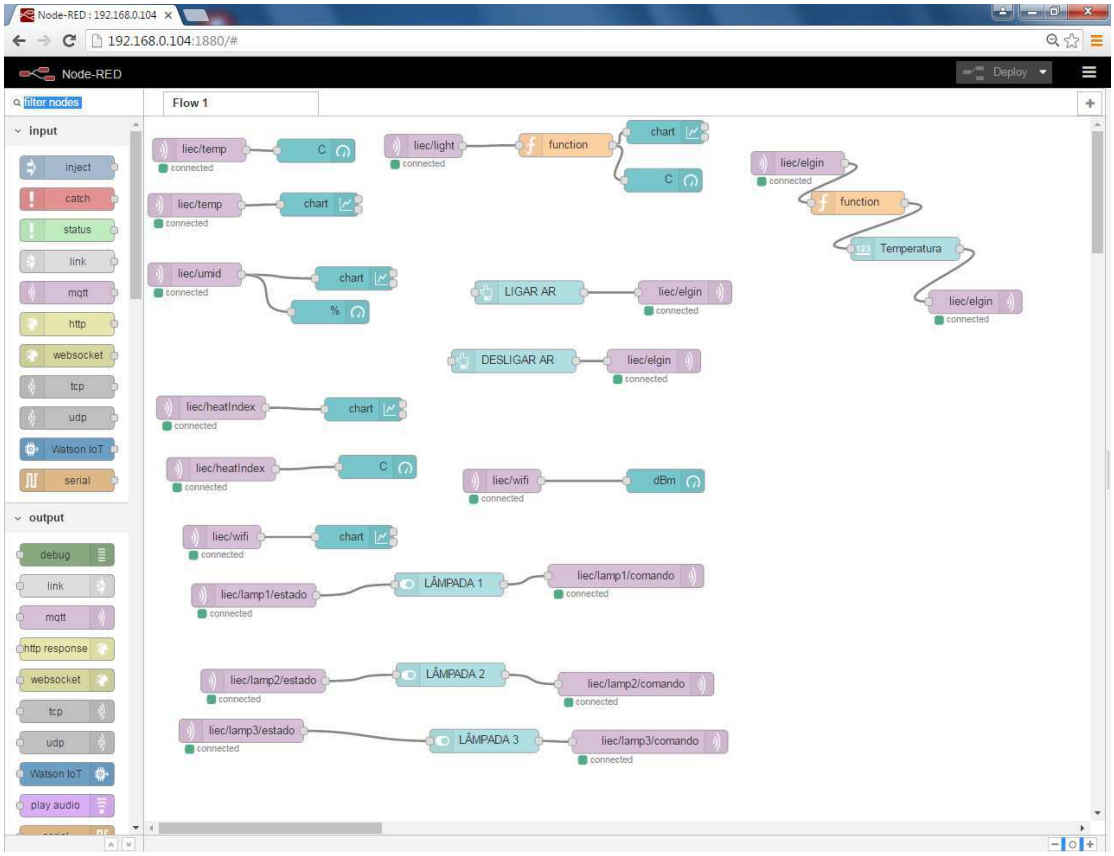
- **Node-RED**

Node-RED é uma ferramenta de *software* desenvolvida pela IBM com o intuito de conectar dispositivos de *hardware* de forma simples e facilitar o desenvolvimento de projetos envolvendo IoT. Neste trabalho a ferramenta Node-RED foi utilizada para a criação de uma interface *web* responsável por mostrar dados recebidos por outros nós da rede de comunicação e possibilitar ao usuário interagir com o sistema por meio de botões e outros *widgets* presentes na página.

A seguir, uma imagem do programa desenvolvido no ambiente de trabalho Node-RED:

[2] Disponível em: < https://mosquitto.org/documentation/>, Acessado em 21/04/2017
[3] Disponível em: < http://www.st.com/en/embedded-software/stemwin.html>, Acessado em 21/04/2017

FIGURA 6: PROGRAMA DESENVOLVIDO NO AMBIENTE NODE-RED

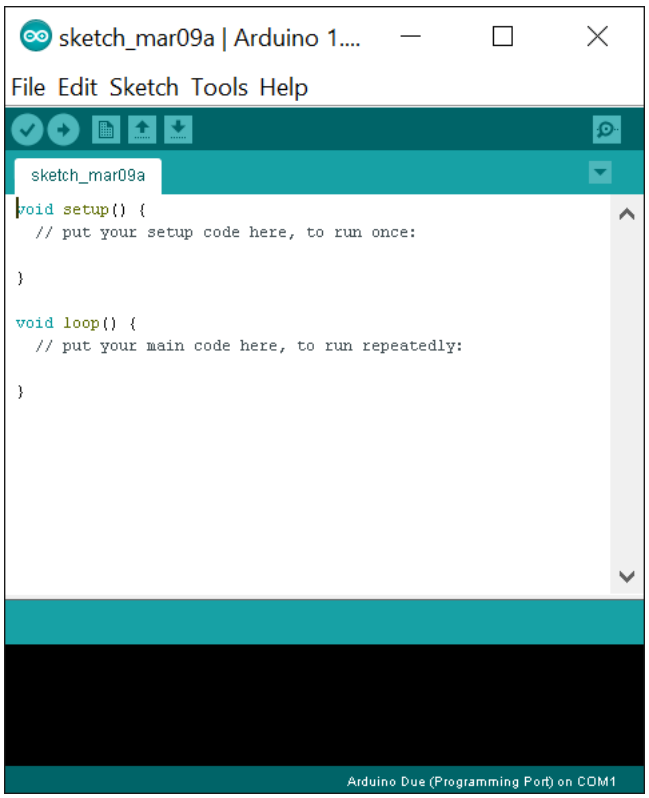



Fonte: Elaborada pelo autor

- **Arduino IDE**

A Arduino IDE é um ambiente de programação *open source* desenvolvido pela organização Arduino que possibilita o desenvolvimento e *upload* de software em placas arduino e outras plataformas compatíveis. Neste projeto, a IDE arduino foi utilizada para efetuar a escrita e *upload* de *software* nas plataformas ESP8266 e Arduino MKR1000. A seguir, uma ilustração da IDE utilizada.

FIGURA 7: ARDUINO IDE

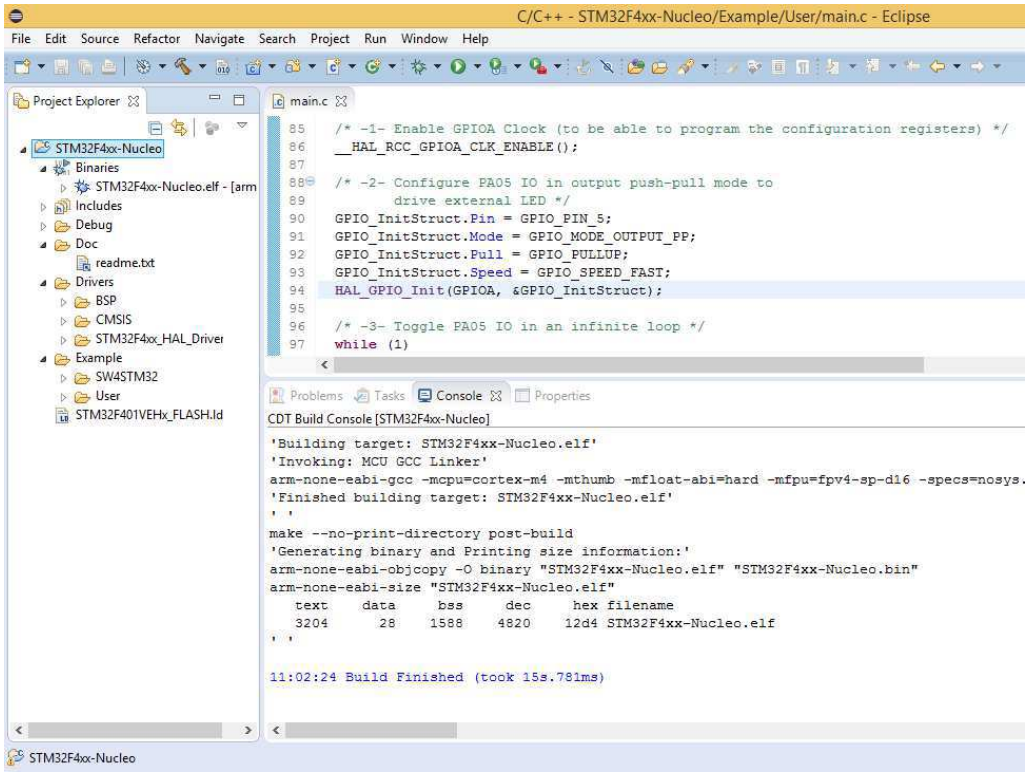



Fonte: Organização Arduino

- ***System Workbench for stm32***

*System Workbench* é uma IDE baseada em Eclipse desevolvida pela organização STMicroelectronics. Por meio deste ambiente é possível desenvolver e fazer *upload* de *firmware* para diversos microcontroladores da plataforma STM32. A seguir uma imagem da IDE utilizada.

FIGURA 8: SYSTEM WORKBENCH FOR STM32



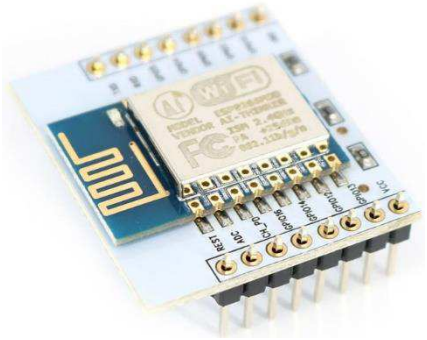

Fonte: Site STM32

## 4.3 DESCRIÇÃO DOS *HARDWARES* UTILIZADOS

- **Microcontrolador ESP8266**

O ESP8266 é um microcontrolador de baixo custo com suporte WiFi e possui as seguintes especificações:

FIGURA 9: ESP8266 - ESP 12



Fonte: Espressif *Systems*

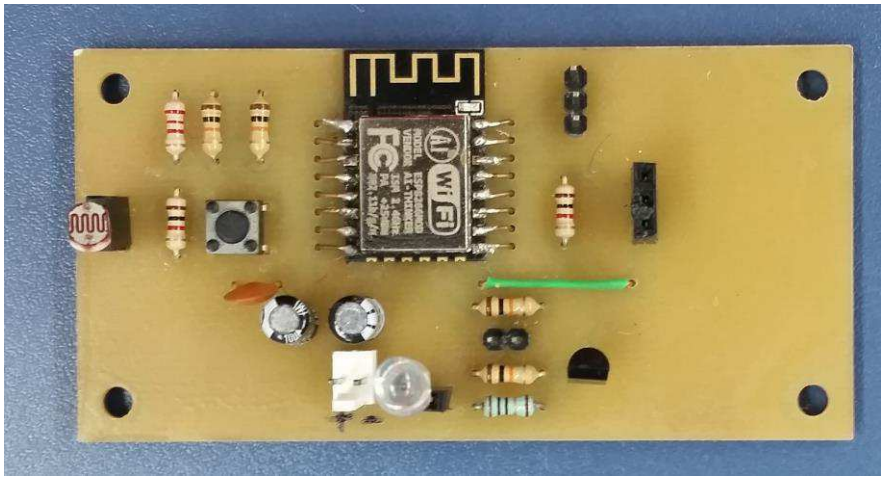
- ✓ Wireless padrão 802.11 b/g/n;
- ✓ Antena embutida;
- ✓ Modos de operação: STA/AP/STA+AP;
- ✓ Segurança WEP, WPA, TKIP, AES;
- ✓ Protocolo TCP/IP embutido;
- ✓ Portas GPIO: 11;
- ✓ Tensão de operação: 3,3V;
- ✓ Taxa de transferência: 110-460800bps;
- ✓ Conversor analógico digital (ADC);
- ✓ Distância entre pinos: 2mm;
- ✓ Dimensões: 24 x 16 x 3,2 mm ;

- ➢ **Módulo 1 – ESP8266, LDR e Led Emissor Infravermelho**

Este módulo consiste de uma placa de circuito impresso contendo um microcontrolador ESP8266 (ESP-12) , um *light dependent resistor* (LDR) e um LED infravermelho.

Este dispositivo foi programado e utilizado para realizar medições da lumisonidade no ambiente e enviá-las para um servidor MQTT periodicamente. Além disso, este módulo também é responsável por enviar os sinais de controle para o ar condicionado por meio do uso de eum emisso infravermelho apontado para o equipamento. O envio destes sinais é controlado por meio de mensagens recebidas remotamente utilizando o protocolo MQTT. A seguir uma imagem do módulo utilizado.

FIGURA 10: MÓDULO 1 - ESP8266, LDR E LED EMISSOR INFRAVERMELHO



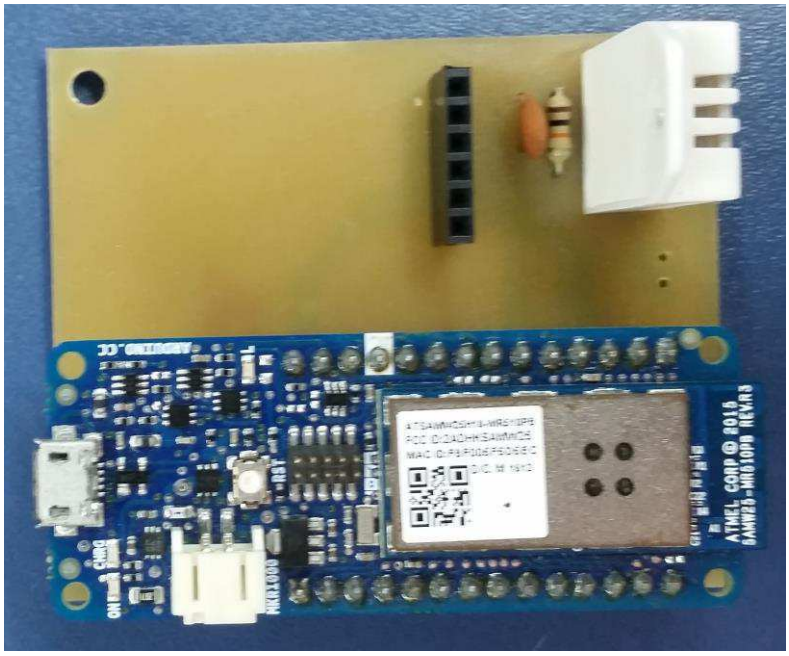
Fonte: Elaborada pelo autor

- **Módulo 2 – Arduino MKR1000 e Sensor DHT22**

O módulo 2 utilizado neste projeto consiste de uma placa de circuito impresso contendo uma placa de prototipagem Arduino MKR1000 e um sensor de umidade/temperatura DHT22.

Este dispositivo foi programado para se conectar a um *broker* MQTT , realizar medições de umidade e temperatura periodicamente e publicar estas mensagens no servidor MQTT. A seguir é mostrada uma imagem do módulo utilizado.

FIGURA 11: MÓDULO 2 - ARDUINO MKR1000 E SENSOR DHT22



Fonte: Elaborada pelo autor

- **Plataforma experimental**

Esta plataforma experimental de testes foi criada pelo aluno Luiz Grilo para simular uma instalação monofásica de uma residência com seus circuitos de instalação e algumas cargas (lâmpadas). A plataforma também possui um arduino nano e um ESP8266 responsáveis por controlarem a plataforma.

Neste trabalho, o arduino nano e o microcontrolador ESP8266 foram programados para possiblitar o monitoramento do estado destas lâmpadas e o acionamento das mesmas pela internet utilizando o protocolo MQTT.

FIGURA 12: PLATAFORMA EXPERIMENTAL

Fonte: Elaborada pelo autor

- **Raspberry Pi**

O Raspberry Pi é um computador do tamanho de um cartão de crédito, que pode ser conectado a um monitor ou TV e a um teclado e mouse padrão, desenvolvido no Reino Unido pela Fundação Raspberry Pi. Todo o hardware é integrado em uma única placa. O Raspberry Pi foi criado inicialmente com o objetivo de promover o ensino em Ciência da Computação básica em escolas. A seguir é apresentada uma imagem do Raspberry Pi juntamente com algumas especificações do Raspberry Pi modelo B:

FIGURA 13: RASPBERRY PI MODELO B

Fonte: Fundação Raspberry PI

- ✓ Processador Broadcom BCM2837 64bit ARMv8 Cortex-A53 Quad-Core
- ✓ Clock 1.2 GHz
- ✓ Memória RAM: 1GB
- ✓ Adaptador Wifi 802.11n integrado

- ✓ Bluetooth 4.1 BLE integrado
- ✓ Conector de vídeo HDMI
- ✓ 4 portas USB 2.0
- ✓ Conector Ethernet
- ✓ Interface para câmera (CSI)
- ✓ Interface para display (DSI)
- ✓ Slot para cartão microSD
- ✓ Conector de áudio e vídeo
- ✓ GPIO de 40 pinos
- ✓ Dimensões: 85 x 56 x 17mm

Neste trabalho, o servidor *open source* Mosquitto foi instalado no Raspberry Pi e funcionou como *broker* da rede MQTT criada.

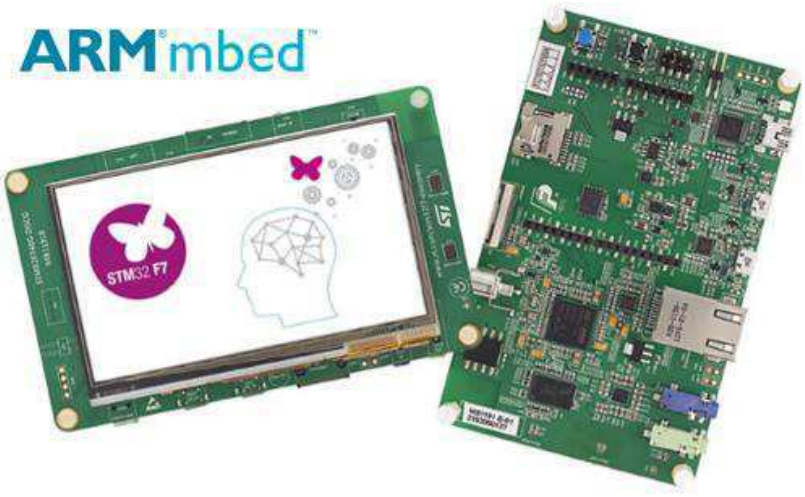
- **STM32F7 – DISCOVERY**

A STM32F7 discovery é uma placa de desenvolvimento que permite aos usuários desenvolver aplicações com a série de microcontroladores STM32F7 baseados no processador ARM Cortex – M7.

Este kit de desenvolvimento possibilita uma larga diversidade de aplicações utilizando áudio, suporte a vários sensores, recursos gráficos, segurança, vídeo e ferramentas de conexão de alta velocidade.

A seguir é apresentada uma imagem desta plataforma de desenvolvimento juntamente com algumas de suas especificações:

FIGURA 14: STM32F7 – DISCOVERY




Fonte: StMicroelectronics

- ✓ Microcontrolador STM32F746NGH6 com 1Mbyte de memória *flash* e 340 Kbytes de memória RAM
- ✓ Tela *touchscreen* 480x272
- ✓ Conector de câmera
- ✓ Saída de tensão para aplicações externas: 3.3V ou 5V
- ✓ Conector para cartão de memória microSD
- ✓ Compatível com Arduino Uno V3
- ✓ 128 –Mbit de memória flash Quad-SPI
- ✓ 128-Mbit de memória SDRAM

## 4.4 FUNCIONAMENTO DO SISTEMA

De forma geral, o sistema de automação desenvolvido neste trabalho de conclusão de curso funciona a partir da troca de mensagens entre nós presentes na rede. Por meio do uso do protocolo de comunicação MQTT cada nó da rede descrito na subseção anterior envia/recebe mensagens relacionadas a sua tarefa específica no sistema.

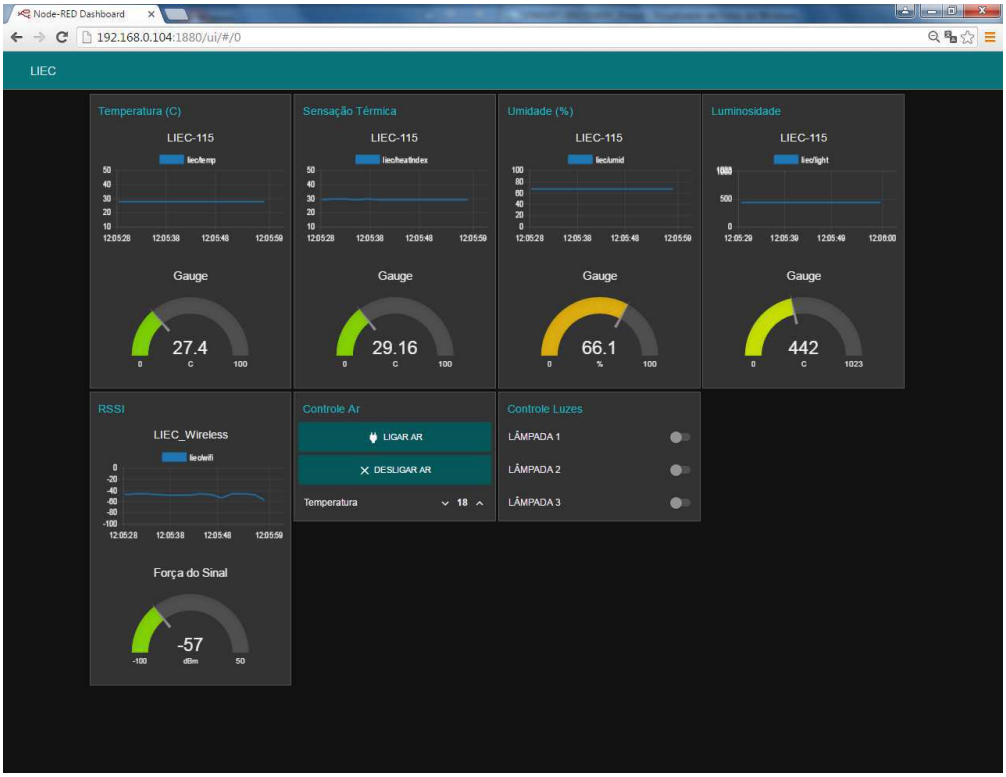
A interação com o usuário do sistema é feita por meio de duas interfaces desenvolvidas. A primeira , e mais completa delas, consiste de uma página *web* acessível localmente. Por meio desta página é possível monitorar em tempo real por meio de

gráficos o valor de diversas grandezas relacionadas ao ambiente em que o ambiente se encontra, são elas:

- ✓ Temperatura
- ✓ Sensação térmica
- ✓ Luminosidade
- ✓ Umidade
- ✓ Intensidade do sinal WiFi

Por meio desta interface também é possível efetuar o controle do ar condicionado instalado no ambiente (Ver figura) e modificar o estado das lâmpadas presentes na plataforma experimental descrita na subseção 4.3. A seguir, uma imagem da interface *web* criada.

FIGURA 15: INTERFACE WEB DESENVOLVIDA




Fonte: Elaborada pelo autor

A segunda interface desenvolvida funciona na tela *touchscreen* da placa de desenvolvimento STM32F7-DISCO. Devido a placa de desenvolvimento não possuir WiFi embutido, optou-se por conectar um microcontrolador ESP8266 à placa para que a mesma pudesse interagir com o a rede de comunicação MQTT criada. A seguir é mostrada uma imagem da placa juntamente com o ESP8266 e a interface gráfica criada.

FIGURA 16: INTERFACE GRÁFICA DESENVOLVIDA


Fonte: Elaborada pelo autor

Por meio dos botões presentes na interface é possível Ligar/Desligar o ar condicionado e setar a temperatura desejada. Abaixo, uma imagem do ar condicionado instalado na sala.

FIGURA 17: AR CONDICIONADO INSTALADO NA SALA

Fonte: Elaborada pelo auto

# 5 CONCLUSÃO

## 5.1 CONSIDERAÇÕES FINAIS

Neste trabalho de conclusão de curso foi feito um estudo das especificações do protocolo de comunicação juntamente com o projeto de um sistema de monitoramento e controle de equipamentos (ar condicionado e lâmpada) que se utilizou deste protocolo para efetuar a troca de mensagens entre os diversos dispositivos envolvidos no sitema.

Diante do exposto neste documento e do projeto desenvolvido, foi possível concluir que o protocolo de comunicação MQTT representa uma ótima alternativa para projetos IoT mais especificamente para automação predial. Uma vez que a troca de mensagens por meio do uso do protocolo se deu de forma rápida e eficiente, o que possibilitou a criação de um sistema funcional e aplicável a diversas situações cotidianas.

## 5.2 PERSPECTIVAS FUTURAS

Dentre as perspectivas futuras pode-se citar a substiuição dos sensores de temperatura e luminosidade utilizados neste trabalho (DHT22 e LDR) por sensores mais precisos como o sensor de temperatura LM35 e  o sensor de luminosidade TSL2561.

Além disso, outra perspectiva futura deste trabalho é a integração do protocolo mqtt utilizado neste projeto com o protocolo industrial OPC-UA.

# REFERÊNCIAS

[1] LAMPKIN, Valerve; LEONG, Weng Tat, *et al*. **Building Smart Planet Solutions with MQTT and IBM WebSphere MQ Telemetry.** United States: IBM Redbooks, 2012.

[2] OASIS Standart. **MQTT Version 3.1.1.** Disponível em: < http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf > Acesso em 13 abr. 2017

[3] HIVEMQ (Germany). **MQTT Essentials Part 5:** MQTT Topics & Best Practices. 2017. Disponível em: < http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices >. Acesso em: 13 abr. 2017.

[4] HIVEMQ (Germany). **MQTT Essentials Part 8:** Retained Messages. 2017. Disponível em: <http://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages >. Acesso em: 13 abr. 2017

[5] HIVEMQ (Germany). **MQTT Essentials Part 2:** *Publish &Subscribe Basics*. 2017. Disponível em: <http://www.hivemq.com/blog/mqtt-essentials-part-2-publish-subscribe >. Acesso em: 13 abr. 2017

[6] HIVEMQ (Germany). **MQTT Essentials Part 3:** *Client, Broker and Connection Estabilishment*. 2017. Disponível em: <http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-estabilishment>. Acesso em: 13 abr. 2017

[7] HIVEMQ (Germany). **MQTT Essentials Part 6:** *Quality of Service 0,1, &2* . 2017. Disponível em: <http://www.hivemq.com/blog/mqtt-essentials-part-6-quality-of-service-levels>. Acesso em: 13 abr. 2017

# APÊNDICE A – CÓDIGO FONTE ESP8266 (MÓDULO 1 )

```
#include <IRDaikinESP.h>
#include <IRremoteESP8266.h>
#include <IRremoteInt.h>
#include <string.h>

#include "ControleAr.h"

#include <ESP8266WiFi.h>
#include <PubSubClient.h>

#define SENSOR_PRESENCA 5



#define IR_EMITER       4
#define FIVE_SECONDS    5000




const char* ssid = "LIEC_Wireless2";
const char* password = "0987ABCDEF";
int keyIndex = 0;

IRsend irsend(4);

unsigned long data = 0xB24D7B84;
int nbits = 32;
int khz=38; //NB Change this default value as neccessary to the correct modulation
frequency
```

```
// Create an instance of the server
// specify the port to listen on as an argument
IPAddress broker(192, 168, 0, 104);  //raspbe pi private IP

void callback(char* topic, byte* payload, unsigned int length) {
 /*Serial.print("Message arrived [");
 Serial.print(topic);
 Serial.print("] ");*/
 String inputString = "";
 for (int i=0;i<length;i++) {
   inputString += (char)payload[i];
 }


 if(inputString == "1"){

   irsend.sendRaw(ON,197,38);
   delay(10);
 }
 else if(inputString == "0"){
 irsend.sendRaw(OFF,197,38);
 delay(10);
 }

 else if(inputString == "25"){
 irsend.sendRaw(temp24to25,197,38);
 delay(10);
 }
 else if(inputString == "26"){
 irsend.sendRaw(temp25to26,197,38);
 delay(10);
 }
```

```
 else if(inputString == "27"){
irsend.sendRaw(temp26to27,197,38);
delay(10);
}

else if(inputString == "28"){
irsend.sendRaw(temp27to28,197,38);
delay(10);
}

else if(inputString == "24"){
irsend.sendRaw(temp23to24,197,38);
delay(10);
}


else if(inputString == "23"){
irsend.sendRaw(temp22to23,197,38);
delay(10);
}
else if(inputString == "22"){
irsend.sendRaw(temp21to22,197,38);
delay(10);
}


else if(inputString == "21"){
irsend.sendRaw(temp20to21,197,38);
delay(10);
}


else if(inputString == "20"){
```

```
  irsend.sendRaw(temp19to20,197,38);
  delay(10);
  }

   else if(inputString == "19"){
  irsend.sendRaw(temp18to19,197,38);
  delay(10);
  }

  else if(inputString == "18"){
  irsend.sendRaw(temp19to18,197,38);
  delay(10);
  }



}

WiFiClient  wifiClient;

PubSubClient client(wifiClient);

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    //Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("espClient")) {

        client.subscribe("liec/elgin");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
```

```
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
void setup() {
  Serial.begin(9600);
  delay(10);

  int flag_presenca = 0;

  irsend.begin();
  delay(30);

  client.setServer(broker, 1883);
  client.setCallback(callback);
  // prepare GPIO2
  pinMode(2, OUTPUT);
  pinMode(SENSOR_PRESENCA,INPUT);
  //digitalWrite(2, 0);

  // Connect to WiFi network
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
```

```
  Serial.println("WiFi connected");

  // Start the server


  Serial.println(WiFi.localIP());
}

char lumi[15];
char presenca[15];

void loop() {



  delay(1);

int light = analogRead(A0);
int flag_presenca = digitalRead(SENSOR_PRESENCA);


   sprintf(lumi,"%d",light);
   sprintf(presenca,"%d",flag_presenca);



    client.publish("liec/light", lumi);
    delay(300);

    //client.publish("liec/presenca", presenca);
    //Serial.println(digitalRead(5));

    client.loop();
  //Serial.println()
```

```
}
```

# APÊNDICE B – CÓDIGO FONTE ARDUINO MKR1000 (MÓDULO 2)

```
#include <DHT.h>
#include <DHT_U.h>
#include "DHT.h"


#include <PubSubClient.h>
#include <SPI.h>
#include <WiFi101.h>


char ssid[] = "LIEC_Wireless2";      //  your network SSID (name)
char pass[] = "0987ABCDEF";   // your network password
int keyIndex = 0;                 // your network key Index number (needed only for WEP)


#define DHTPIN 5

#define DHTTYPE           DHT22     // DHT 22 (AM2302)

DHT dht(DHTPIN, DHTTYPE);
```

```
int status = WL_IDLE_STATUS;
//WiFiServer server(80);
IPAddress broker(192, 168,0, 104);  //raspbe pi private IP

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i=0;i<length;i++) {
    Serial.print((char)payload[i]);
  }

}

WiFiClient  wifiClient;
PubSubClient client(wifiClient);

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("arduinoClient")) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("liec/teste","hello, from arduino MKR1000/2107");
      // ... and resubscribe
      client.subscribe("liec/arduino");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
```

```
    }
   }
  }

  void setup() {
   Serial.begin(9600);      // initialize serial communication
   pinMode(6, OUTPUT);      // set the LED pin mode

   client.setServer(broker, 1883);
   client.setCallback(callback);

   // check for the presence of the shield:
   if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    while (true);       // don't continue
   }

   // attempt to connect to Wifi network:
   while ( status != WL_CONNECTED) {
    Serial.print("Attempting to connect to Network named: ");
    Serial.println(ssid);                   // print the network name (SSID);

    // Connect to WPA/WPA2 network. Change this line if using open or WEP
network:
    status = WiFi.begin(ssid,pass);
    // wait 10 seconds for connection:
    //delay(10000);
   }


   printWifiStatus();                      // you're connected now, so print out the status

    dht.begin();
  }
```

```
 char umid[10];
 char temp[10];
 char heatIndex[10];
 char wifi[10];
void loop() {

 if (!client.connected()) {
   reconnect();
  }

  //delay(1000);



 float h = dht.readHumidity();
 float t = dht.readTemperature();
 float hic = dht.computeHeatIndex(t, h, false);
 int signalStrength = WiFi.RSSI();


   sprintf(umid, "%.2f",h);
   sprintf(temp,"%.2f",t);
   sprintf(heatIndex,"%.2f",hic);
   sprintf(wifi,"%d",signalStrength);

   Serial.println(h);
  //printWifiStatus();
  delay(2000);
    client.publish("liec/umid", umid);
    client.publish("liec/temp", temp);
    client.publish("liec/heatIndex", heatIndex);
    client.publish("liec/wifi", wifi);

    client.loop();
```

```
  //Serial.println()
}

void printWifiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
  // print where to go in a browser:
 // Serial.print("To see this page in action, open a browser to http://");
  //Serial.println(ip);
}
```