INSTITUTO FEDERAL DO ESPÍRITO SANTO
CURSO DE ENGENHARIA ELÉTRICA

CAROLINE SIQUEIRA LOPES

PROTÓTIPO DE UM SISTEMA DE CONTROLE DE ACESSO BASEADO EM IDENTIFICAÇÃO POR RADIOFREQUÊNCIA

Vitória
2021

CAROLINE SIQUEIRA LOPES

PROTÓTIPO DE UM SISTEMA DE CONTROLE DE ACESSO BASEADO EM IDENTIFICAÇÃO POR RADIOFREQUÊNCIA

Trabalho de Conclusão de Curso apresentado à Coordenadoria de Engenharia Elétrica do *Campus* Vitória do Instituto Federal do Espírito Santo, como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Leandro Bueno

Vitória

2021

Dados Internacionais de Catalogação na Publicação (CIP)
(Biblioteca Nilo Peçanha do Instituto Federal do Espírito Santo)

L864p Lopes, Caroline Siqueira
Protótipo de um sistema de controle de acesso baseado em identificação por radiofrequência / Caroline Siqueira Lopes – 2023.
111 f. : il. ; 30 cm

Orientador: Leandro Bueno.

Monografia (graduação) – Instituto Federal do Espírito Santo, Coordenadoria de Engenharia Elétrica, Curso Superior de Engenharia Elétrica, 2023.

1. Sistemas de controle inteligente. 2. Sistemas de identificação por radiofrequência. 3 Internet das coisas.4. Arduíno (controlador programável). 5. Engenharia Elétrica. I. Bueno, Leandro. II. Instituto Federal do Espírito Santo. III. Título.

CDD 21 – 621.30285

Elaborada por Bruno Giordano Rosa – CRB-6/ES – 699

MINISTÉRIO DA EDUCAÇÃO
INSTITUTO FEDERAL DO ESPÍRITO SANTO
VIT - COORDENADORIA DO CURSO DE ENGENHARIA ELETRICA

FOLHA DE APROVAÇÃO-TCC Nº 1/2023 - VIT-CCEE (11.02.35.01.09.02.11)

Nº do Protocolo: 23148.007390/2023-74

Vitória-ES, 20 de novembro de 2023.

CAROLINE SIQUEIRA LOPES

PROTÓTIPO DE UM SISTEMA DE CONTROLE DE ACESSO BASEADO EM IDENTIFICAÇÃO POR RADIOFREQUÊNCIA

Trabalho de Conclusão de Curso apresentado à Coordenadoria de Engenharia Elétrica do Instituto Federal do Espírito Santo, como requisito parcial para obtenção de título de Engenheiro Eletricista.

Aprovado em 16 de abril de 2021

COMISSÃO EXAMINADORA

Dr. Leandro Bueno
Instituto Federal Do Espírito Santo
Orientador

Dr. Marcelo Brunoro
Instituto Federal do Espírito Santo
Avaliador

Me. Renato Benezath Cabelino Ribeiro
Instituto Federal do Espírito Santo
Avaliador

(Assinado digitalmente em 20/11/2023 14:20)
LEANDRO BUENO
PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO
VIT-CCEE (11.02.35.01.09.02.11)
Matrícula: 1361682

(Assinado digitalmente em 20/11/2023 14:48)
MARCELO BRUNORO
PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO
VIT-CCTE (11.02.35.01.09.02.19)
Matrícula: 1813911

(Assinado digitalmente em 20/11/2023 19:28)
RENATO BENEZATH CABELINO RIBEIRO
PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO
VIT-CCTE (11.02.35.01.09.02.19)
Matrícula: 1341061

AGRADECIMENTOS

Agradeço aos meus pais, Delma e Dalmo, pelo amor e apoio incondicionais. Me orgulho imensamente da família que formamos.

À minha irmã, Luiza, minha maior companheira e confidente. Você me inspira todos os dias a ser uma pessoa melhor.

Ao meu orientador, Leandro, pela paciência e orientação durante a reta final da minha formação. Contar com sua ajuda e seus conselhos foi essencial para o desenvolvimento deste projeto.

Aos professores do Ifes *Campus* Vitória pelos ensinamentos e conselhos valiosos à minha formação acadêmica e profissional.

Aos meus amigos e colegas de curso, especialmente a Dayanne, Paulo, Smith, Georgia e Leticia, por compartilharem comigo alguns dos momentos mais importantes da minha vida.

RESUMO

O gerenciamento eficiente de ambientes em instituições de ensino requer atenção e recursos pois é preciso limitar e monitorar o acesso a esses locais, definindo quem está autorizado a utilizá-los e quando. A automatização do controle de acesso a salas de aula e laboratórios se apresenta como uma solução para promover mais conforto e segurança aos usuários do local. Assim, foi desenvolvido o protótipo de uma rede de controle de acesso, que consiste em uma rede de dispositivos de controle de acesso, empregando a RFID (do inglês, *Radio-Frequency IDentification*) como tecnologia de identificação e o MQTT (do inglês, *Message Queuing Telemetry Transport*) como protocolo de comunicação entre os dispositivos. Este protótipo consiste em um dispositivo servidor que utiliza a placa BeagleBone Black como central de processamento, um dispositivo cliente que utiliza a placa Arduino Mega e uma página *web* para gerenciamento de credenciais e exibição de relatórios. A análise de desempenho do sistema foi realizada segundo parâmetros técnicos tais como tempo médio de inicialização dos dispositivos e tempo médio de resposta do dispositivo a uma solicitação de acesso. Assim, concluídos os testes da solução proposta, verificou-se que é possível desenvolver um sistema de controle de acesso eficiente utilizando materiais facilmente encontrados no mercado e aplicações *open-source*. Adicionalmente, a página *web* provou-se ser uma ferramenta indispensável para o gerenciamento do sistema sem o acesso direto ao sistema operacional do dispositivo servidor, contando com uma interface amigável e intuitiva.

Palavras-chave: Controle de Acesso. RFID. MQTT. BeagleBone Black. Arduino.

ABSTRACT

The efficient management of an educational institution's different environments requires attention and resources because it is necessary to limit and monitor the access to those places, defining who is authorized to use them and when. The automatization of the control access for classrooms and laboratories is presented as a solution that can provide more comfort and safety for the institution and its users. Within this scenario, a prototype for an access control system was designed. It is based on the Radio Frequency Identification (RFID) technology and the Message Queuing Telemetry Transport (MQTT) protocol. This prototype comprises a server device based on the BleagleBone Black board as central processing, a client device based on the Arduino Mega board, and a web page designed to manage credentials and export reports. To validate its functionalities, tests were done to evaluate system performance, following parameters such as the devices' average response time to an access request and the system's average boot time. After the test phase was completed, it became evident that it is possible to create an efficient control access system using open-source applications and materials that are easily found on the market. Also, the web page has proven itself to be an essential tool for managing the system without accessing the server diapositive's OS directly, relying on a user-friendly interface.

Keywords: Access Control. RFID. MQTT. BeagleBone Black. Arduino.

LISTA DE FIGURAS

Figura 1 – Esquema simplificado do sistema de controle de acesso proposto 15
Figura 2 – Representação de um sistema RFID. 19
Figura 3 – Componentes de uma tag RFID 20
Figura 4 – Exemplos de *tags* RFID passivas 21
Figura 5 – Exemplo de *tag* RFID semi-passiva 22
Figura 6 – Exemplos de *tags* RFID ativas 23
Figura 7 – Exemplo de chaves de automóveis com *tags* RFID LF embarcadas 23
Figura 8 – Exemplos de *tags* RFID HF 24
Figura 9 – Exemplos de *tags* RFID UHF 25
Figura 10 – Exemplo de *tag* RFID SHF 25
Figura 11 – Exemplo de leitor RFID 26
Figura 12 – Topologia física em barramento 28
Figura 13 – Topologia física em estrela 28
Figura 14 – Topologia física em anel 29
Figura 15 – Topologia física em malha 30
Figura 16 – Topologia lógica ponto-a-ponto 31
Figura 17 – Topologia lógica cliente/servidor 31
Figura 18 – Esquema da arquitetura de três camadas da IoT 33
Figura 19 – Estrutura de uma mensagem MQTT v3.1.1 36
Figura 20 – Representação da composição de um sistema embarcado 37
Figura 21 – Placa BeagleBone Black 38
Figura 22 – Arduino Mega 2560 40
Figura 23 – Módulo RDM6300 42
Figura 24 – Módulo Ethernet ENC28J60 43
Figura 25 – Módulo RTC DS1307 44
Figura 26 – Esquemático do módulo RDM6300 47
Figura 27 – Tela de configuração (a) e terminal de comandos (b) do PuTTY 48
Figura 28 – Principais componentes do protótipo inicial servidor 55
Figura 29 – Primeiro protótipo do dispositivo servidor 56
Figura 30 – Primeiro protótipo do dispositivo servidor 57
Figura 31 – Tela inicial do programa Arduino IDE 58
Figura 32 – Principais componentes do protótipo inicial servidor 59

Figura 33 – Protótipo inicial do dispositivo cliente 60
Figura 34 – Interface da Cloud9 IDE, embarcada na Beaglebone Black 64
Figura 35 – Portas TCP utilizadas pelos serviços da BBB 65
Figura 36 – Página BeagleBone 101 66
Figura 37 – Página de login do servidor *web* 66
Figura 38 – Página “Acessos” do servidor *web* 67
Figura 39 – Página "Usuários" do servidor *web* 68
Figura 40 – Página "Web" do servidor *web* 68
Figura 41 – Página "Ambientes" do servidor *web* 69
Figura 42 – Segundo protótipo dos dispositivos servidor e cliente 70
Figura 43 – Protótipos conectados ao roteador 70

LISTA DE TABELAS

Quadro 1 – Especificações técnicas da BBB 39
Quadro 2 – Especificações técnicas da placa Arduino Mega 2560 41
Quadro 3 – Especificações técnicas do módulo RDM6300 42
Quadro 4 – Especificações do módulo Ethernet ENC28J60 44
Quadro 5 – Especificações técnicas RTC DS1307 45
Quadro 6 - Campos da tabela “Cadastro” 52
Quadro 7 - Campos da tabela "Acesso" 53
Quadro 8 – Campos da tabela “RegistroLog” 53
Quadro 9 – Campos da tabela "LoginWeb" 54

LISTA DE ABREVIATURAS E SIGLAS

AC/DC	*Alternating Current/Direct Current*
BBB	*BeagleBone Black*
CI	Circuito Integrado
CompTIA	*Computing Technology Industry Association*
DHCP	*Dynamic Host Configuration Protocol*
GUI	*Graphical User Interface*
HDMI	*High-Definition Multimedia Interface*
HF	*High Frequency*
HTTP	*Hypertext Transfer Protocol*
I²C	*Inter-Integrated Circuit*
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronics Engineers*
Ifes	Instituto Federal do Espírito Santo
IoT	*Internet of Things*
IP	*Internet Protocol*
LAN	*Local Area Network*
LCD	*Liquid Crystal Display*
LED	*Light-Emitting Diode*
LF	*Low Frequency*
MAC	*Medium Access Control*
MIT	*Massachussetts Institute of Technology*
MQTT	*Message Queuing Telemetry Transport*
NTP	*Network Time Protocol*

PCI	Placa de Circuito Impresso
PHP	*Hypertext Preprocessor*
PWM	*Pulse-Width Modulation*
QoS	*Quality of Services*
RAM	*Random Access Memory*
RFID	*Radio Frequency Identification*
RTC	*Real-Time Clock*
SHF	*Super High Frequency*
SO	Sistema Operacional
SPI	*Serial Peripheral Interface*
SQL	*Structured Query Language*
SSH	*Secure Shell*
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol*
TSL	*Transport Layer Security*
TWI	*Two-Wire Interface*
UHF	*Ultra-High Frequency*
USB	*Universal Serial Bus*
WAN	*Wide Area Network*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS 14
1.2 ESTRUTURA DA MONOGRAFIA 16
2 FUDAMENTAÇÃO TEÓRICA 18
2.1 CONTROLE DE ACESSO 18
2.2 IDENTIFICAÇÃO POR RADIOFREQUÊNCIA 18
2.2.1 *Tags* 19
2.2.1.1 Classificação por fonte de energia 20
2.2.1.2 Classificação por frequência de operação 23
2.2.2 Leitores 26
2.3 REDE DE COMPUTADORES 27
2.3.1 Topologia física de rede 27
2.3.1.1 Barramento 27
2.3.1.2 Estrela 28
2.3.1.3 Anel 29
2.3.1.4 Malha 29
2.3.2 Topologia lógica de rede 30
2.3.2.1 Ponto-a-ponto (*peer-to-peer*) 30
2.3.2.2 Cliente/servidor 31
2.4 INTERNET DAS COISAS 32
2.5 MQTT 34
2.6 SISTEMAS EMBARCADOS 37
2.6.1 BeagleBone Black 38
2.6.2 Arduino Mega 2560 39
2.6.3 RDM6300 41
2.6.4 Módulo Ethernet ENC28J60 42
2.6.5 Módulo RTC 44
3 DESENVOLVIMENTO DO SISTEMA DE CONTROLE DE ACESSO 46
3.1 PRIMEIROS PROTÓTIPOS E CONFIGURAÇÕES INICIAIS 46
3.1.1 Dispositivo servidor 47
3.1.1.1 Atualização do *kernel* 48
3.1.1.2 Sincronização do módulo RTC 48
3.1.1.3 Instalação do MySQL 49
3.1.1.4 Instalação e configuração do Mosquitto 49
3.1.1.5 Instalação de bibliotecas e atualização do pacote Python 49
3.1.1.6 Criação do banco de dados 51

3.1.1.7 Primeiro protótipo 55
3.1.2 Dispositivo cliente 57
3.1.2.1 Primeiro protótipo 58
3.2 ESTRUTURA DA REDE DE DISPOSITIVOS 60
3.2.1 Configuração MQTT 61
3.3 ELABORAÇÃO DAS APLICAÇÕES 63
3.3.2 Os programas principais 63
3.3.2 O programa como serviço do sistema 64
3.3.2 Página web 64
3.4 SEGUNDO PROTÓTIPO 69
3.5 TESTES 71
4 Conclusão 74
4.1 Trabalhos Futuros 75
REFERÊNCIAS 76
APÊNDICE A – Configuração Inicial do Dispositivo Servidor 80
APÊNDICE B – Programa Principal do Dispositivo Servidor 90
APÊNDICE C – Programa Principal do Dispositivo Cliente 97
APÊNDICE D – Quadros de Conexão 104
ANEXO A – Programa de Configuração RTC NTP do Dispositivo Cliente 107

1 INTRODUÇÃO

Os avanços tecnológicos têm promovido a redução das dimensões de componentes eletrônicos, permitindo que dispositivos com diferentes interfaces e funcionalidades sejam parte do nosso cotidiano; se aliados à programação, esses dispositivos são capazes de analisar eventos relevantes em um ambiente e tomar decisões em tempo real (AUGUSTO; NAKASHIMA; AGHAJAN, 2010). Considerando tais avanços, os ambientes modernos, sejam residenciais ou comerciais, têm sido gradualmente adaptados para se tornarem ambientes inteligentes – espaços com tecnologia instalada que têm como objetivo melhorar a experiência do usuário e reconhecer padrões de comportamento (REIJULA et al., 2011).

O gerenciamento eficiente de salas de aula, laboratórios e escritórios de uma instituição de ensino é usualmente uma fonte recorrente de problemas (PALMA et al., 2014). Limitar e monitorar o acesso a esses locais, definindo quem está autorizado a utilizá-los ou quando, pode se tornar uma tarefa dispendiosa. Assim, inclusa no conceito de ambientes inteligentes, a proposta de um controle de acesso automatizado a salas de aula e laboratórios se apresenta como uma solução que visa promover mais conforto e segurança aos usuários dos ambientes.

Atualmente, o acesso aos laboratórios do *Campus* Vitória do Instituto Federal do Espírito Santo (Ifes) é feito com fechaduras convencionais. A maioria dos laboratórios têm acesso restrito a professores e servidores, podendo ser utilizados pelos alunos quando em horário de aula ou com concessão especial do professor responsável. O Laboratório de Desenvolvimento do Bloco M (sala M204), excepcionalmente, é de acesso livre aos alunos dos cursos de Engenharia Elétrica e Técnico em Eletrotécnica. O espaço foi concebido para que os alunos possam estudar e realizar tarefas propostas em aula. Foi observada pela autora certa dificuldade no acesso dessa sala pelos alunos, pois a chave fica disponível para empréstimo em uma das salas dos professores mediante a assinatura de uma folha de registro. Uma vez feito o empréstimo da chave a um aluno, a devolução deve ser feita quando o aluno deixa o laboratório. Além disso, outro empecilho é a possível perda da chave por um aluno, fato que já ocorreu no passado.

Dessa forma, apesar de a situação atual ser funcional, propõe-se o desenvolvimento de uma rede de dispositivos de controle de acesso baseado na tecnologia RFID e no protocolo MQTT, que possa ser futuramente implementada no instituto. O princípio de utilização é de que o usuário usará seu cartão de identificação da instituição para acessar as salas de aula e laboratórios. Após validação dos dados do usuário de acordo com o banco de dados, o acesso será concedido.

O objetivo é prover mais segurança aos alunos e à instituição, visto que a implantação deste projeto prevê o registro da maioria das pessoas que acessam um ambiente, bem como a redução dos problemas recorrentes com o empréstimo da chave física. Além do desenvolvimento do protótipo da rede de dispositivos, será criado um servidor *web* para gerenciamento de credenciais e exibição de relatórios. A análise de desempenho do sistema será feita segundo parâmetros técnicos tais como tempo médio de inicialização dos dispositivos e tempo médio de resposta do dispositivo a uma solicitação de acesso.

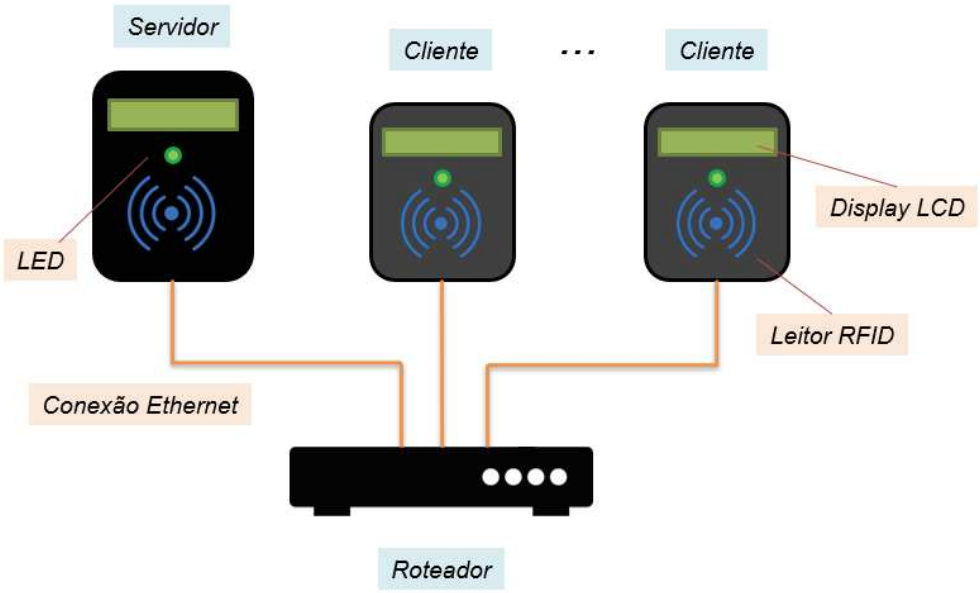
Salienta-se que, embora o controle de acesso à sala M204 do *Campus* Vitória tenha sido a motivação para a realização deste projeto e a situação utilizada para estruturação do banco de dados, não faz parte do escopo do presente trabalho o estudo de viabilidade da implementação do sistema proposto. Estima-se que, em trabalhos futuros, este sistema pode ser adaptado à esta e outras finalidades que usem *tags* RFID como tecnologia de identificação, mediante a adaptação do banco de dados criado.

1.1 OBJETIVOS

O objetivo geral deste trabalho é desenvolver um protótipo de rede de dispositivos de controle de acesso, empregando a RFID como modo de tecnologia de identificação e o MQTT como protocolo de comunicação entre dispositivos.

A estrutura física do sistema proposto, ilustrado na Figura 1, será composto fundamentalmente por um dispositivo denominado **servidor**, um ou mais dispositivos denominados **cliente** e um roteador de rede.

Figura 1 – Esquema simplificado do sistema de controle de acesso proposto



Fonte: Elaborado pela autora, 2020.

O **dispositivo cliente** será composto, essencialmente, por uma placa microcontroladora Arduino Mega, um leitor RFID, um módulo Ethernet, um módulo de relógio de tempo real e um *display* LCD. Ele será responsável por ler *tags* RFID e enviar a informação obtida ao dispositivo servidor.

O **dispositivo servidor**, por sua vez, atuará como o dispositivo central do sistema, no qual serão implementados servidores locais MQTT e MySQL. A princípio, será composto por uma placa BeagleBone Black (BBB), um leitor RFID, um módulo de relógio de tempo real e um *display* LCD. Este dispositivo receberá o código identificador de uma *tag* RFID, seja por uma mensagem enviada por um cliente ou por leitura do leitor RFID próprio, realizará uma busca no banco de dados pela permissão e armazenará um registro da tentativa de acesso.

Os dispositivos se comunicarão via cabos Ethernet RJ45, através da conexão a um mesmo roteador de rede. Os dispositivos serão energizados por fontes de conversão AC/DC adequadas ao funcionamento correto dos componentes.

A liberação do acesso é indicada por uma mensagem exibida no *display* LCD. A abertura da porta, subsequente à liberação, é indicada visualmente por um LED.

Adicionalmente, será criada uma página *web* para gerenciamento de credenciais e exibição de relatórios. A última etapa de desenvolvimento consistirá na realização de testes para comprovação da funcionalidade do sistema.

Dessa forma os objetivos específicos deste trabalho consistem em:

a) construir um dispositivo de controle de acesso servidor, composto por um leitor RFID de 125 kHz e uma placa BeagleBone Black, que atuará como servidor MQTT e MySQL;
b) construir dois dispositivos de controle de acesso cliente, ambos compostos por um leitor RFID de 125 kHz e um Arduino Mega, que atuarm como clientes;
c) configurar a topologia física da rede de dispositivos;
d) configurar a topologia lógica da rede de dispositivos;
e) criar um banco de dados para armazenamento de informações sobre os usuários cadastrados e registros de solicitações de acesso;
f) criar uma página *web* para gerenciamento das credenciais dos usuários e exportação de relatórios;
g) realizar testes de funcionalidades do sistema e avaliar o desempenho da rede de dispositivos através de parâmetros técnicos.

1.2 ESTRUTURA DA MONOGRAFIA

Esta monografia é composta por quatro capítulos, divididos em diversas seções. O Capítulo 1 introduz o tema e a motivação para a realização deste trabalho, assim como os objetivos a serem alcançados.

O Capítulo 2 trata da fundamentação teórica necessária ao desenvolvimento do trabalho, concentrando-se em temas relevantes à construção de um sistema de controle de acessos baseado em RFID. Assim, o segundo capítulo faz menção aos seguintes temas: controle de acesso, tecnologia RFID, redes de computadores, Internet das Coisas, protocolo de comunicação MQTT e sistemas embarcados.

O Capítulo 3 aborda o desenvolvimento do projeto, a criação dos protótipos dos dispositivos físicos e dos programas e aplicações a serem implantados. Os testes realizados para avaliação e validação do funcionamento do sistema também são descritos em uma seção dedicada.

O Capítulo 4 apresenta as conclusões obtidas durante o trabalho, bem como propostas para trabalhos futuros.

Por fim, para não comprometer a fluidez da leitura deste trabalho, os códigos criados e mencionados no Capítulo 3 são apresentados em apêndices. O Apêndice A apresenta os comandos executados na placa BBB para as instalações e configurações necessárias ao desenvolvimento do projeto. Os Apêndice B e C apresentam os códigos fontes dos protótipos dos dispositivos servidor e cliente, respectivamente. O Apêndice D apresenta o mapeamento das conexões físicas entre os pinos das placas BBB e Arduino e os módulos. O Anexo A contém o código para configuração do módulo RTC conectado à placa Arduino.

2 FUDAMENTAÇÃO TEÓRICA

Por se tratar de um projeto multidisciplinar que envolve vasto conhecimento das áreas de telecomunicações e eletrônica digital, limitou-se a apresentar somente conceitos fundamentais à elaboração do projeto, principalmente tecnologias, protocolos e *hardware* utilizados. Dessa forma, este projeto não visa discorrer sobre aspectos conceituais básicos, tais como os princípios de propagação de ondas eletromagnéticas ou funcionamento de microcontroladores.

2.1 CONTROLE DE ACESSO

O controle de portas automatizado, geralmente envolvendo tecnologias de sensores com ou sem fio, tem sido amplamente utilizado em espaços públicos para eliminar a necessidade de abertura manual (YANG et al., 2013). Entretanto, segundo o mesmo autor, apesar de serem eficazes em detectar a presença de pessoas, essas tecnologias não são capazes de identificar se a pessoa é autorizada ou não a entrar no local.

Uma solução à entrada livre de pessoas é o monitoramento e controle de acesso, um sistema que permite que somente usuários com credenciais válidas consigam entrar no ambiente (HWANG; BAEK, 2007). Este tipo de sistema geralmente é operado por senha, chave digital ou códigos, se provando mais confiável e seguro do que sistemas de portas com fechaduras convencionais (HWANG; BAEK, 2007).

2.2 IDENTIFICAÇÃO POR RADIOFREQUÊNCIA

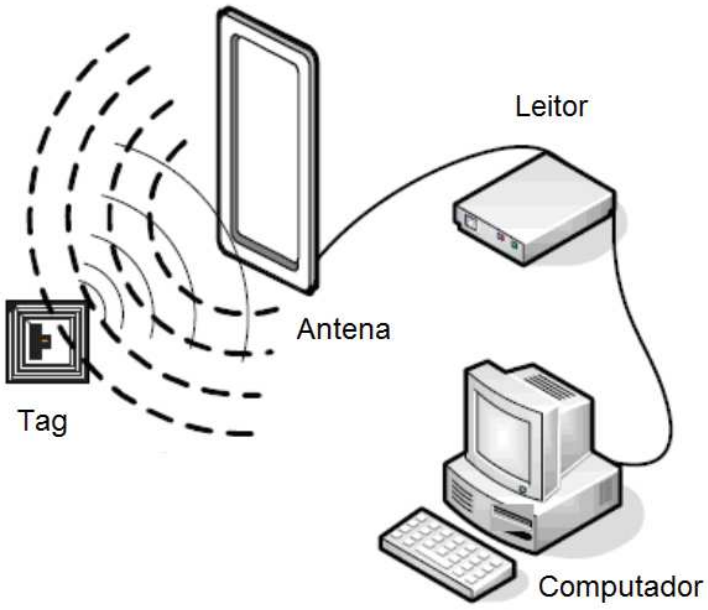
Salienta-se que, exceto onde explicitamente citado, os conceitos apresentados nesta seção são baseados no livro *Mike Meyers' Comptia RFID+ Certification Passport*, dos autores Michael Meyer, Mark Brown, Sam Patadia e Sanjiv Dua. A obra, referência sobre o tema, foi publicada em 2007 pela editora McGraw-Hill Education e foi desenvolvida por membros da Associação das Indústrias de Tecnologia da Computação dos Estados Unidos (conhecida pelo acrônimo CompTIA, do inglês *Computing Technology Industry Association*).

A Identificação por Radiofrequência (referida pela sigla RFID, do inglês *Radio Frequency Identification*) abrange tecnologias que usam ondas de rádio para

identificar individualmente objetos, lugares, animais ou pessoas. As ondas de rádio constituem uma parte do espectro de ondas eletromagnéticas com frequência entre 3 kHz e 300 GHz. A premissa na qual este tipo de tecnologia se baseia é que os objetos e/ou pessoas a serem identificados sejam caracterizados por um código identificador. O código é armazenado em um circuito integrado (CI) conectado a uma antena, formando uma *tag* RFID que deve ser portada pelo indivíduo a ser identificado. Um dispositivo conhecido como leitor se comunica com a *tag* e lê o código identificador armazenado nela. O leitor envia o código a um sistema computacional que o guarda em um banco de dados ou realiza algum tipo de operação com a informação recebida.

Os componentes de um sistema RFID, ilustrados na Figura 2, serão abordados mais detalhadamente nas subseções a seguir.

Figura 2 – Representação de um sistema RFID.



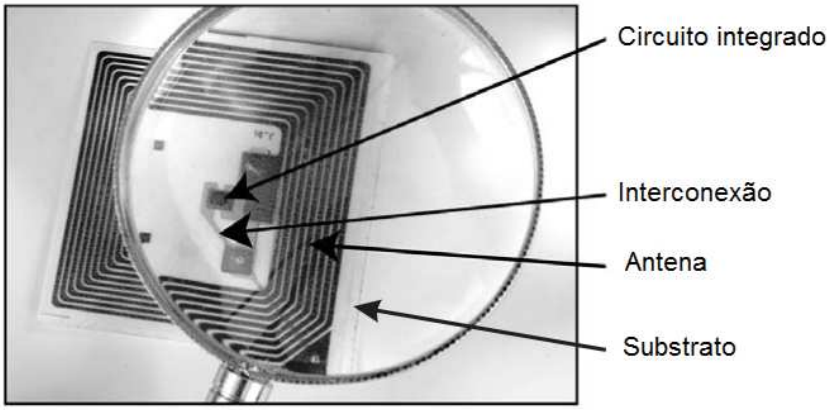
Fonte: Adaptado de Tanim (2016, p. 2).

2.2.1 *Tags*

Tags são *transponders* formados por três tipos de componentes: um CI, uma antena e um substrato, conforme apresentado na Figura 3. O CI é um circuito eletrônico, composto por blocos lógicos e bancos de memória. Os blocos lógicos são responsáveis pelas operações de modulação/demodulação e

codificação/decodificação de dados, enquanto os bancos de memória são responsáveis por armazenar esses dados. A antena pode ser feita de uma bobina de metal ou pode ser impressa em trilhas de CI. Seu formato e tamanho determinam a frequência na qual a *tag* operará. O substrato é o meio de fixação do CI e da antena, geralmente feito de plástico flexível. Por fim, este conjunto é encapsulado em um objeto que melhor sirva o propósito do sistema, sejam cartões plásticos, etiquetas autocolantes ou chaveiros.

Figura 3 – Componentes de uma tag RFID



Fonte: Adaptado de Meyer (2007, p. 63).

Tags podem ser categorizadas essencialmente de acordo com o tipo de fonte de energia utilizado e a frequência de operação, conforme exposto a seguir.

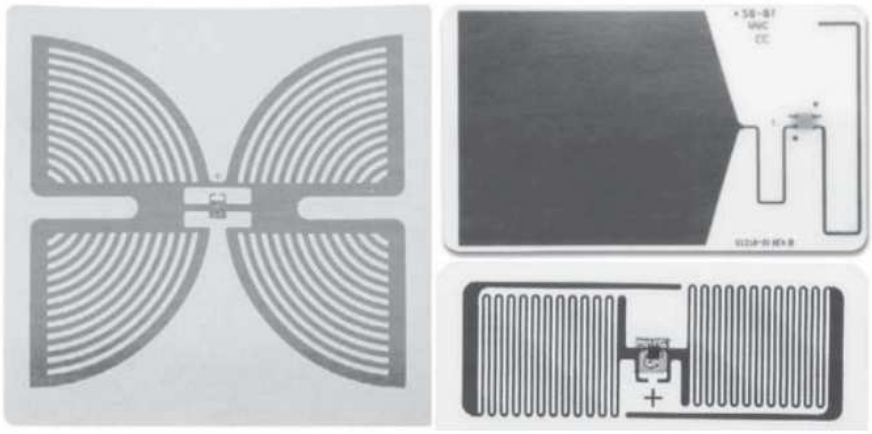
2.2.1.1 Classificação por fonte de energia

Tags necessitam de energia elétrica para alimentar o CI que processará os sinais recebidos do leitor e para mandar o sinal que contém o código identificador. A fonte dessa energia define a classificação das *tags* em passivas, semi-passivas e ativas.

Tags passivas, exemplificadas na Figura 4, não possuem uma fonte de energia própria que possa energizar o CI para a transmissão de dados. A obtenção de energia ocorre através das ondas eletromagnéticas recebidas do leitor, visto que há o acoplamento entre a *tag* e o leitor por meio de indução ou retrodifusão. Acoplamento, neste caso, é a transferência de energia de um meio para outro através de um campo eletromagnético. O tipo de acoplamento usado depende da frequência de operação e da distância entre a *tag* e o leitor. Acoplamento indutivo é usado caso as *tags* sejam lidas no campo próximo (*near-field*, em inglês) da antena

do leitor e acoplamento por retrodifusão é usado no campo distante (*far-field*, em inglês).

Figura 4 – Exemplos de *tags* RFID passivas



Fonte: Adaptado de Meyer (2007, p. 71).

Acoplamento indutivo baseia-se no princípio físico da Lei de Faraday e se refere à transferência de energia de um dispositivo para outro através um campo magnético compartilhado, de forma que a variação no fluxo de corrente de um dispositivo induz fluxo de corrente no outro. Acoplamento por retrodifusão é regido pela reflexão do sinal emitido pelo leitor. O leitor emite um sinal, que quando atinge a *tag*, tem parte de sua energia absorvida e parte refletida.

A quantidade de energia recebida por acoplamento é pequena, suficiente apenas para que o CI gere sinais de resposta ao leitor. Quando fora da área de alcance da antena de um leitor, a *tag* passiva permanece desenergizada e, portanto, inativa. Devido a essas características, este tipo de *tag* tem funcionalidade reduzida e não operam com energia suficiente para suportar um transmissor ativo para se comunicar com o leitor, ou seja, atuam apenas em resposta a um sinal enviado pelo leitor.

Tags semi-passivas (também conhecidas como semi-ativas), exemplificadas na Figura 5, são *tags* que possuem uma bateria integrada destinada à energização do CI mas, assim como *tags* passivas, não possuem um transmissor ativo. Atuam com acoplamento por retrodifusão para se comunicar com o leitor e a bateria é usada somente para aumentar o alcance da antena da *tag* e alimentar o CI e sensores de ambiente, como sensores de temperatura e umidade. Dessa forma, além de informar

o código identificador, este tipo de *tag* pode coletar dados do ambiente em que está presente.

Figura 5 – Exemplo de *tag* RFID semi-passiva

Fonte: Adaptado de Meyer (2007, p. 73).

Devido a adição de uma bateria, *tags* semi-passivas, em comparação a *tags* passivas são maiores, mais pesadas, mais caras, sensíveis à temperatura e têm menor tempo de utilização. Caso a bateria acabe, a *tag* se torna inativa e não poderá ser identificada por leitores.

Tags ativas, por sua vez, exemplificadas na Figura 6, possuem bateria integrada e um transmissor ativo. Isto significa que não necessitam da energia induzida pelos sinais do leitor para transmitir dados. A bateria integrada neste tipo de *tag* fornece mais energia dos que as baterias das *tags* semi-passivas e, portanto, permite a utilização de CIs com maior capacidade de processamento. É possível até o processamento dos dados coletados por sensores de ambiente antes da transmissão. Diferentemente dos outros dois tipos de *tags* citados anteriormente, *tags* ativas utilizam a emissão de ondas de rádio geradas pelo próprio CI e não por acoplamento.

Figura 6 – Exemplos de *tags* RFID ativas

Fonte: Adaptado de Meyer (2007, p. 75).

2.2.1.2 Classificação por frequência de operação

Em relação à frequência de operação, *tags* podem ser categorizadas em quatro faixas: baixa frequência (referida pela sigla LF, do inglês *low frequency*), alta frequência (referida pela sigla HF, do inglês *high frequency*), ultra alta frequência (referida pela sigla UHF, do *inglês ultra high frequency*) e micro-ondas (também conhecida pela sigla SHF, do inglês *super high frequency*).

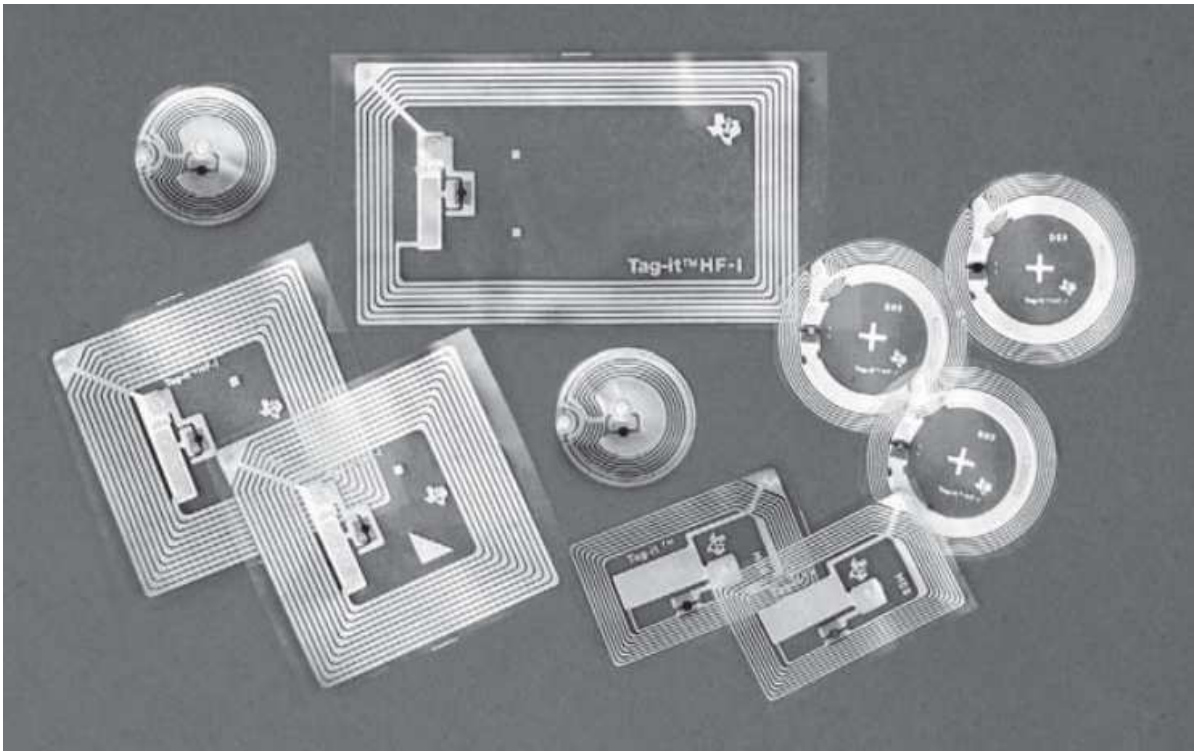
Tags LF, exemplificadas na Figura 7, possuem operam na faixa de frequência de 30 kHz a 300 kHz, porém somente as frequências de 125 kHz e 134 kHz são usadas para aplicações RFID. Essas *tags* são passivas, têm alcance de leitura de poucos centímetros e usam acoplamento indutivo para obter energia e se comunicar com o leitor. Elas têm a menor taxa de transmissão de dados e a menor capacidade de armazenamento de dados entre todas as faixas de operação RFID. Este tipo de *tag* é o mais utilizado atualmente.

Figura 7 – Exemplo de chaves de automóveis com *tags* RFID LF embarcadas

Fonte: Adaptado de Meyer (2007, p. 78).

Tags HF, exemplificadas na Figura 8, operam na faixa de frequência de 3 MHz a 30 MHz, porém somente a frequência de 13,56 MHz é usada para aplicações RFID. Essas *tags* são passivas, têm alcance de leitura de aproximadamente um metro e usam acoplamento indutivo para obter energia e se comunicar com o leitor. A antena HF é menor do que a antena LF devido à sua maior frequência, visto que o tamanho e a frequência de uma antena são propriedades inversamente proporcionais.

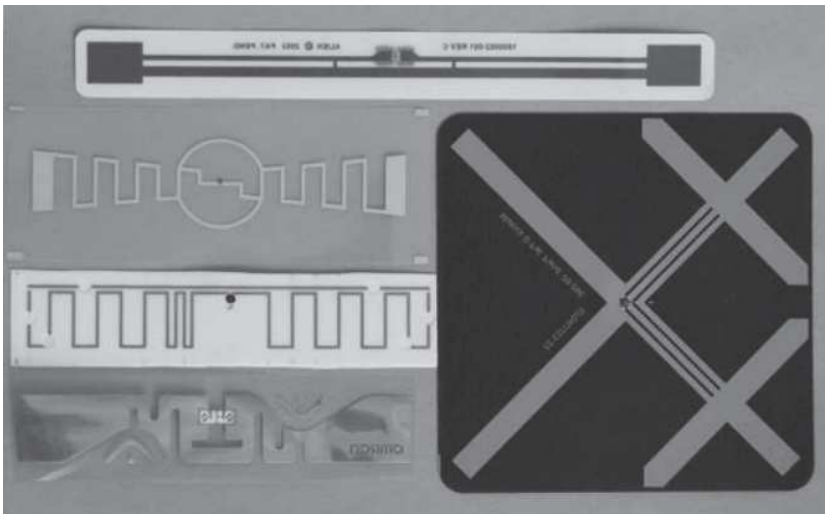
Figura 8 – Exemplos de *tags* RFID HF



Fonte: Adaptado de Meyer (2007, p. 79).

Tags UHF, exemplificadas na Figura 9, operam na faixa de frequência de 300 MHz a 1000 MHz porém apenas as frequências de 433 MHz e 860 - 960 MHz são usadas para aplicações RFID. A frequência de 433 MHz é usada para *tags* ativas enquanto a banda de 860 - 960 MHz é usada principalmente para *tags* passivas e semi-passivas. *Tags* UHF têm alcance de leitura de aproximadamente seis metros e usam acoplamento indutivo para obter energia e se comunicar com o leitor.

Figura 9 – Exemplos de *tags* RFID UHF



Fonte: Adaptado de Meyer (2007, p. 80).

Por fim, existe um último tipo de *tag* pela classificação por frequência de operação, as *tags* SHF, que operam na faixa de frequência de 1 GHz a 10 GHz, posto que apenas as frequências de 2,45 GHz e 5,8 GHz são usadas para aplicações RFID. Essas *tags*, exemplificadas na Figura 10, podem ser passivas, semi-passivas ou ativas. Os alcances de leitura para esses tipos de *tag* de aproximadamente 4 m, 30 m e 106 m, respectivamente. Este tipo de *tag* não é comumente utilizada e existem apenas alguns fabricantes no mundo todo.

Figura 10 – Exemplo de *tag* RFID SHF

Fonte: Adaptado de Meyer (2007, p. 83).

A seguir, serão discutidas a definição e a constituição dos leitores RFID.

2.2.2 Leitores

Leitores, exemplificados Figura 11, são transceptores que leem e gravam dados em *tags* RFID. Em resumo, são responsáveis por fornecer energia às *tags* (para os tipos passiva e semi-passiva), estabelecer fluxo bidirecional (leitura e gravação) de dados e realizar a conversão de sinais analógicos em digitais e vice-versa.

Figura 11 – Exemplo de leitor RFID

Fonte: Intelbras (2019).

A construção desses dispositivos pode variar em complexidade e podem incluir desde estruturas mais simples, compostas de apenas uma antena e um módulo microcontrolador, a estruturas mais complexas, com central de processamento mais robusta e interface gráfica para usuário (usualmente referida pela sigla GUI, do inglês *Graphical User Interface*). Usualmente, um leitor é composto de um transmissor, um receptor e um processador.

O transmissor compreende um modulador que define a frequência de transmissão da onda de radiofrequência, um oscilador que produz a alternação de corrente que gera a onda, um amplificador para amplificar a onda gerada pelo modulador e um transmissor de banda base que transmite a onda portadora que energizará a *tag* passiva. O receptor, por sua vez, compreende um amplificador que amplifica o sinal recebido da *tag* e um demodulador que compara o sinal recebido com o sinal

enviado, extraindo a informação enviada pela *tag*. O processador pode ser tão simples quanto um módulo de CI ou pode ser um computador, a depender da aplicação prevista para o sistema e as operações a serem feitas com os dados recebidos. Esses componentes são, obviamente, conectados a uma antena que é responsável por transmitir e captar sinais.

2.3 REDE DE COMPUTADORES

Uma rede de dispositivos é formada dois ou mais computadores conectados que podem compartilhar recursos como dados, aplicativos ou conexão à Internet. Uma rede pode ser categorizada de acordo com suas topologias física e lógica, como brevemente explicado nas subseções a seguir.

Os conceitos apresentados nesta seção e suas subseções foram, em sua totalidade, baseados na obra *CompTIA Network+ Deluxe Study Guide*, de Todd Lammle (2009).

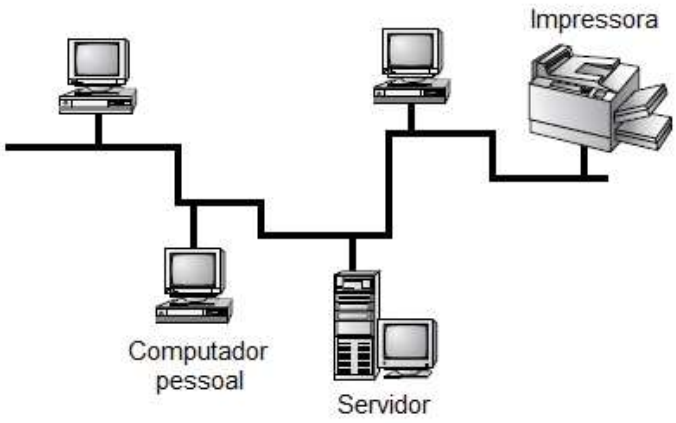
2.3.1 Topologia física de rede

A topologia física de uma rede, como o nome indica, define como seus dispositivos serão conectados fisicamente. Os tipos mais comuns de topologia usados atualmente são: barramento, estrela, anel e malha.

2.3.1.1 Barramento

Este tipo de topologia, ilustrada na Figura 12, é a mais básica do grupo e consiste em duas extremidades distintas, com cada um de seus computadores se conectando a um cabo ininterrupto que percorre todo o seu comprimento. Mesmo que todos os computadores deste tipo de rede vejam todos os dados fluindo através do cabo, somente o computador ao qual os dados são especificamente endereçados realmente os obtém. Alguns dos benefícios do uso de uma topologia de barramento são a facilidade de instalação e o baixo custo, em parte porque não requer tantos cabos quanto os outros tipos de topologias físicas. Entre suas desvantagens, está o impacto à solução de problemas e falhas, visto que todos os dispositivos estão conectados a um único cabo.

Figura 12 – Topologia física em barramento



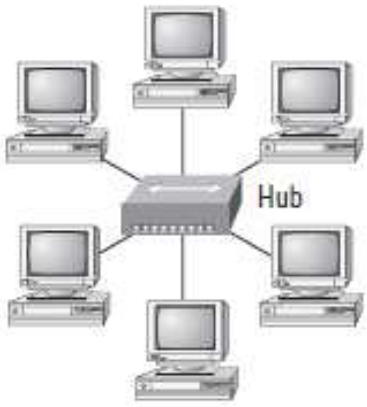
Fonte: Adaptado de Lammle (2009, p. 14).

Este tipo de topologia é usado com cabos coaxiais que não são facilmente encontrados atualmente, porém ainda é amplamente aplicado em redes de dispositivos como CAN, I^2C e RS485.

2.3.1.2 Estrela

Na topologia em estrela, ilustrada na Figura 13, os computadores são conectados a um ponto central com seus próprios cabos individuais ou conexões sem fio. Este ponto central frequentemente é um *hub*, um *switch* ou um ponto de acesso.

Figura 13 – Topologia física em estrela



Fonte: Retirado de Lammle (2009, p. 15).

A topologia em estrela oferece muitas vantagens sobre a topologia de barramento, tornando-a mais amplamente usada, embora obviamente exija mais recursos físicos.

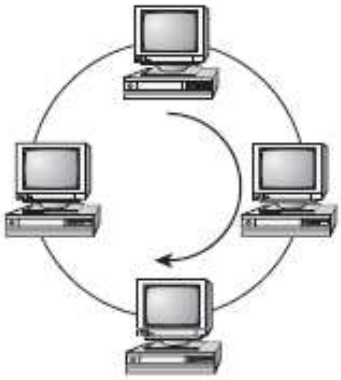
Uma de suas melhores características é que, devido ao fato de que cada dispositivo de rede é conectado ao dispositivo central individualmente, se um cabo falhar, apenas uma máquina será afetada. Isso torna aumenta a tolerância a falhas da rede e torna a solução de problemas mais simples. Outro benefício desta topologia é a escalabilidade – basta conectar um novo dispositivo ao ponto central da rede.

As desvantagens de uma topologia em estrela incluem o custo mais alto de instalação devido ao maior número de cabos e a existência de um ponto crítico de falha (o dispositivo central).

2.3.1.3 Anel

Na topologia em anel, cada computador está diretamente conectado a outros computadores dentro da mesma rede, conforme esquematizado na Figura 14. Similarmente à topologia em barramento, para adicionar um dispositivo à rede, é necessário interromper a conexão em anel de cabo - algo que provavelmente derrubará toda a rede.

Figura 14 – Topologia física em anel

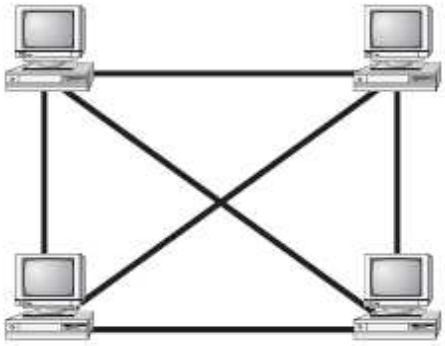


Fonte: Retirado de Lammle (2009, p. 16).

2.3.1.4 Malha

Neste tipo de topologia, encontra-se uma conexão direta entre cada um dos dispositivos da rede, conforme ilustrado na Figura 15.

Figura 15 – Topologia física em malha



Fonte: Retirado de Lammle (2009, p. 17).

Esta topologia é raramente utilizada devido ao alto número de conexões. Versões híbridas, combinando a topologia em malha a outro tipo, têm sido utilizadas recentemente como o objetivo de instalar conexões entre determinados dispositivos para criar redundância (*backup*). Entretanto, não é uma topologia de malha completa, se não houver uma conexão entre todos os dispositivos na rede.

Para cada número *n* de dispositivos, são necessárias $n(n - 1) / 2$ conexões. Isso significa que em uma rede composta por 10 computadores, haverá 45 conexões a serem feitas. Logo, esta topologia é indicada somente para redes pequenas.

Em contrapartida, as principais vantagens são o alto nível de tolerância a falhas e a facilidade de manutenção aos dispositivos.

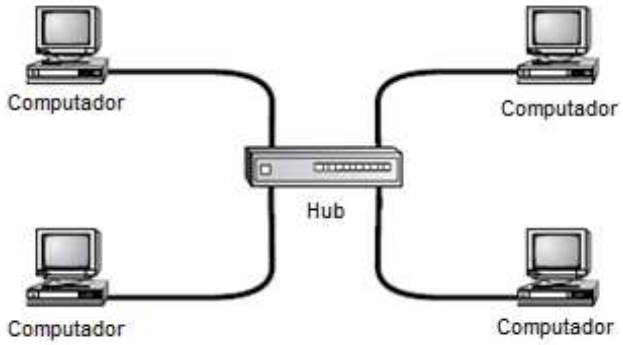
2.3.2 Topologia lógica de rede

As topologias lógicas descrevem como os dados são movidos pela rede.

2.3.2.1 Ponto-a-ponto (*peer-to-peer*)

Os dispositivos conectados em redes ponto-a-ponto (*peer-to-peer*), exemplificada na Figura 16, não têm nenhuma autoridade central ou especial, ou seja, operam em mesmo nível hierárquico. Dessa forma, a verificação de segurança dos direitos de acesso a seus recursos é atribuída ao computador que possui o recurso solicitado.

Figura 16 – Topologia lógica ponto-a-ponto



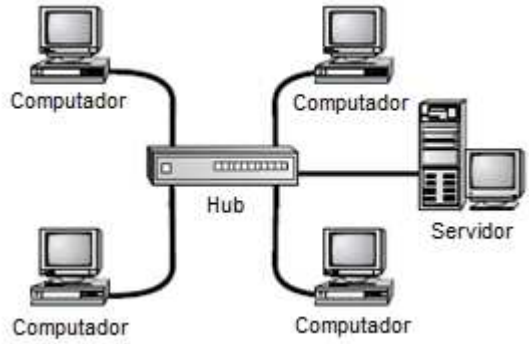
Fonte: Adaptado de Lammle (2009, p. 12).

Neste tipo de topologia lógica, os dispositivos podem ser máquinas clientes que recebem informações e máquinas servidores que as fornecem. Esta estrutura não é otimizada para uma rede com uma quantidade elevada de dispositivos, visto que cada computador realiza o backup localmente e a rede não exige muita segurança.

2.3.2.2 Cliente/servidor

As redes cliente/servidor têm, contrariamente às redes ponto-a-ponto, um único servidor que usa um sistema operacional de rede para gerenciar toda a rede. Portanto, a solicitação de uma informação de um dispositivo cliente vai para o servidor principal, que responde enviando a informação ao dispositivo de destino e respeitando as medidas de segurança necessárias.

Figura 17 – Topologia lógica cliente/servidor



Fonte: Adaptado de Lammle (2009, p. 12).

Este tipo de topologia, ilustrada na Figura 17, apresenta diversos benefícios. O endereçamento de clientes é facilitado visto que os endereços de IP (do inglês

Internet Protocol) são armazenados no servidor. A segurança da rede também se torna mais rígida pois todos os nomes de usuário e senhas estão neste servidor. Em adição, ganha-se em escalabilidade, visto que as redes cliente/servidor podem ter inúmeras estações de trabalho. E mesmo com todas essas demandas, seu desempenho é realmente otimizado.

2.4 INTERNET DAS COISAS

O termo Internet das Coisas (IoT, do inglês *Internet of Things*) surgiu há aproximadamente vinte anos, fruto do trabalho em redes baseadas em RFID desenvolvido pelo Auto-ID Labs no Instituto de Tecnologia de Massachusetts (MIT, do inglês *Massachussetts Institute of Technology*) (WORTMANN; FLÜCHTER, 2015).

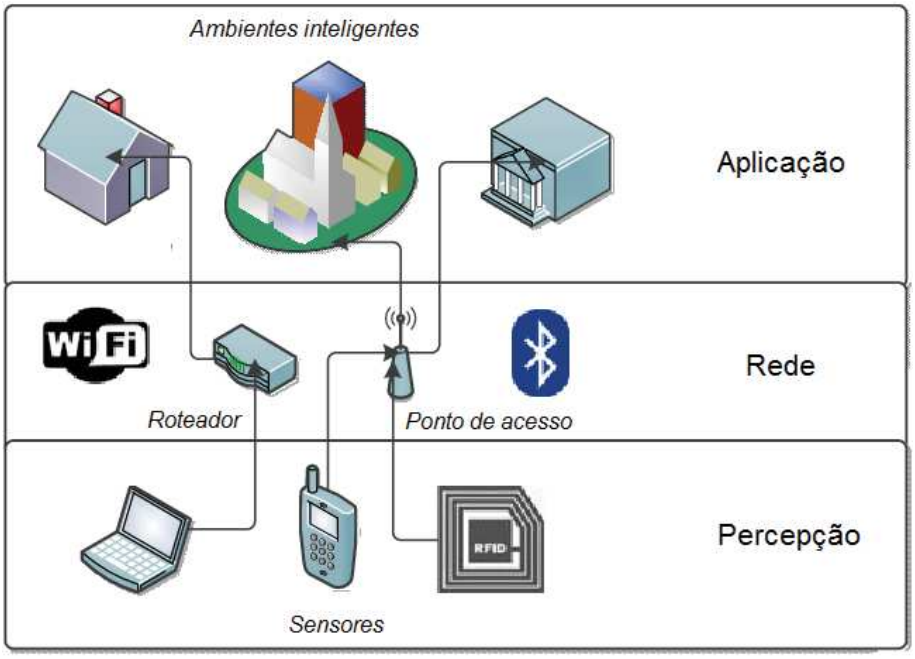
Embora o conceito tem sido cada vez mais recorrente no meio tecnológico nos últimos anos, ainda não existe uma definição absoluta sobre o que é a IoT ou sobre os protocolos que a constituem (WORTMANN; FLÜCHTER, 2015). Algumas das definições propostas têm ênfase nos objetos conectados pela IoT ao passo que outras focam nos aspectos da comunicação por Internet, como protocolos e estruturas de rede (ATZORI; IERA; MORABITO, 2010). Como parte do escopo do presente trabalho, optou-se por adotar a segunda abordagem.

Um amplo conceito frequentemente usado define IoT como o conjunto de tecnologias para comunicação entre dispositivos inteligentes conectados por rede sem ou com fio (AL-FUQAHA et al., 2015). Dispositivos inteligentes são projetados para coletar informações do ambiente no qual estão localizados, analisar as informações coletadas, interagir com o ambiente e se comunicar em rede com outros dispositivos (CISCO, 2016). Essencialmente, dispositivos inteligentes eram dispositivos eletrônicos que operavam isoladamente com funcionalidades restritas e foram aperfeiçoados através da adição de novos componentes eletrônicos para comunicação em rede e interação com o ambiente (AL-FUQAHA et al., 2015).

A IoT deve ser capaz de interconectar bilhões ou trilhões de objetos heterogêneos pela Internet, portanto há a necessidade crítica de uma arquitetura de camadas

flexível. O número cada vez maior de arquiteturas propostas ainda não convergiu para um modelo de referência, porém o modelo básico é uma arquitetura de três camadas que consiste nas camadas de Aplicação, Rede e Percepção (AL-FUQAHA et al., 2015). A Figura 18 apresenta um esquemático deste modelo, expondo algumas das tecnologias e dispositivos usados em cada camada.

Figura 18 – Esquema da arquitetura de três camadas da IoT



Fonte: Adaptado de Mahmoud (2015, p. 337).

A primeira camada, a Camada de Percepção, representa os sensores físicos da IoT que visam coletar informações. Esta camada inclui sensores e atuadores para executar diferentes funcionalidades, como consultar localização, temperatura, peso, movimento, vibração, aceleração ou umidade. A Camada de Percepção coleta, digitaliza e transfere dados para a camada de rede através de canais seguros (AL-FUQAHA et al., 2015).

A segunda camada, a Camada de Rede, transfere os dados produzidos pela Camada de Percepção à Camada de Aplicação. Os dados podem ser transferidos através de várias tecnologias, como RFID, 3G, WiFi, Bluetooth, infravermelho, etc. Além disso, outras funções como computação em nuvem e processos de gerenciamento de dados são tratadas nesta camada (AL-FUQAHA et al., 2015).

A terceira e última camada, a Camada de Aplicação, fornece os serviços solicitados pelos clientes. Por exemplo, a Camada de Aplicação pode fornecer medições de temperatura e umidade do ar ao cliente que solicita esses dados. A importância desta camada para a IoT é que ela é capaz de fornecer serviços inteligentes de alta qualidade para atender às necessidades dos clientes. A Camada de Aplicação abrange diversas aplicações, como ambientes inteligentes, transporte, automação industrial e serviços de saúde inteligentes (AL-FUQAHA et al., 2015).

Para a implementação das camadas descritas acima, é necessária a adoção de protocolos de comunicação. Um protocolo descreve um conjunto de regras que são aplicadas quando dois equipamentos trocam informações. Na IoT, os protocolos são limitados pelas restrições de recursos dos dispositivos e um dos protocolos usados na Camada de Rede é o TCP/IP. Este protocolo é altamente usado e outros protocolos como o HTTP (sigla advinda do termo em inglês *HyperText Markup Language*) são construídos sobre ele. Cada participante da rede é identificado com um endereço IP. Além de computadores, roteadores, servidores de impressão, telefones IP e rádios IP são conectados à Internet via TCP/IP (KOREN; KLAMMA, 2018).

Os protocolos da Camada de Aplicação são criados sobre protocolos de nível inferior como o HTTP. Eles possibilitam a troca de dados entre *softwares* instalados em diferentes dispositivos. Um exemplo é o protocolo a ser utilizado neste trabalho, o MQTT. Este é um representante dos protocolos que estão reutilizando conceitos de HTTP e será explicado em detalhe na seção a seguir (KOREN; KLAMMA, 2018).

2.5 MQTT

O MQTT (advindo do termo *Message Queuing Telemetry Transport*, em inglês) é um protocolo de mensagens que foi introduzido por Andy Stanford-Clark e Arlen Nipper em 1999 e padronizado em 2013 pela organização OASIS (AL-FUQAHA et al., 2015). É conhecido por ser um protocolo de leve desenvolvimento para comunicação máquina a máquina em ambientes IoT (LUOTO; SYSTÄ, 2018). Segundo Hillar (2017), o protocolo foi desenvolvido para suportar os seguintes desafios em sistemas IoT: (i) ser um protocolo de leve processamento para tornar

possível a transmissão de grandes volumes de informação, sem a necessidade de grande quantidade de metadados; (ii) publicar informação em redes instáveis e possivelmente inseguras e garantir uma transmissão confiável em conexões frágeis; (iii) funcionar bem com dispositivos alimentados por bateria ou requerer baixo consumo de energia; (iv) oferecer segurança e privacidade de informação; e (v) ser escalável de modo a distribuir informação a milhares de dispositivos se preciso.

Ainda segundo Hillar (2017), essas configurações permitem que o MQTT se torne uma opção eficiente para implementação em aplicações que requerem troca de dados entre dispositivos periféricos, tais como automação residencial e rastreamento de bens.

O MQTT utiliza o modelo Publicador-Subscritor para a troca de mensagens, no qual as mensagens são publicadas para todos os possíveis receptores ao invés de serem enviadas exclusivamente de um dispositivo para outro (LUOTO; SYSTÄ, 2018). Este modelo estrutura-se com base em um servidor, também conhecido como *broker*, um dispositivo centralizado que gerencia toda a troca de dados. Todos os dispositivos periféricos conectados à rede, chamados de clientes, estão conectados ao servidor. O cliente que transmite uma informação pela rede é conhecido como publicador. O servidor filtra as mensagens transmitidas e as distribui para os clientes que estão interessados neste tipo de informação. Esses clientes, por sua vez, são conhecidos como subscritores. Dessa forma, os dispositivos publicadores e subscritores estão conectados ao servidor, porém não estão conectados entre si (HILLAR, 2017).

A filtragem de mensagens feita pelo servidor mencionada acima é estruturada em tópicos. Cada mensagem transmitida pertence a um tópico, que pode ser livremente definido pelo administrador da rede. Um tópico é um canal, caracterizado por metadados presentes nas mensagens. Em outras palavras, cada mensagem leva os dados originais a serem transmitidos e informações inerentes ao protocolo, tal como o tópico específico (HILLAR, 2017).

A estrutura de uma mensagem do protocolo MQTT na versão v3.1.1, esquematizada na Figura 19, possui três elementos principais: o cabeçalho fixo, o cabeçalho variável e o corpo de dados. O cabeçalho fixo pode compreender de dois a cinco bytes. O primeiro byte contém o tipo da mensagem e os indicadores (*flags*) DUP, QOS e RETAIN. O segundo byte informa o comprimento restante da mensagem, ou

seja, a quantidade de bits do cabeçalho variável e do corpo de dados (OASIS OPEN, 2014). Neste formato, o tipo de mensagem compreende quatro bits e indica uma variedade de mensagens, incluindo “CONNECT”, “PUBLISH”, “SUBSCRIBE” e assim por diante. O *flag* DUP, composto por um bit, indica se a massagem está duplicada e que o receptor pode ter a recebido antes. O *flag* QOS, composto por dois bits, indica um dos três níveis de QoS (do inglês, *Quality of Service*). O *flag* RETAIN, composto por um bit, informa se o servidor deve reter a última mensagem do tipo “PUBLISH” recebida e enviá-la aos novos assinantes como primeira mensagem (AL-FUQAHA et al., 2015). O cabeçalho variável é opcional e carrega informações adicionais necessárias a alguns tipos de mensagens. O corpo de dados, comumente referido como *payload*, também é opcional e pode conter a identificação do cliente e informações coletadas por sensores, por exemplo (OASIS OPEN, 2014).

Figura 19 – Estrutura de uma mensagem MQTT v3.1.1

<table>
<tr><th>Bit
Byte</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th></th></tr>
<tr><td>1</td><td colspan="3">TIPO DA MENSAGEM</td><td>DUP</td><td colspan="2">QOS</td><td>RETAIN</td><td rowspan="2">CABEÇALHO FIXO</td></tr>
<tr><td>1-4</td><td colspan="8">COMPRIMENTO RESTANTE</td></tr>
<tr><td>0 – N</td><td colspan="8">CABEÇALHO VARIÁVEL (OPCIONAL)</td><td></td></tr>
<tr><td>0 – M</td><td colspan="8">CORPO DE DADOS (OPCIONAL)</td><td></td></tr>
</table>

Fonte: Baseado em Oasis Open (2014).

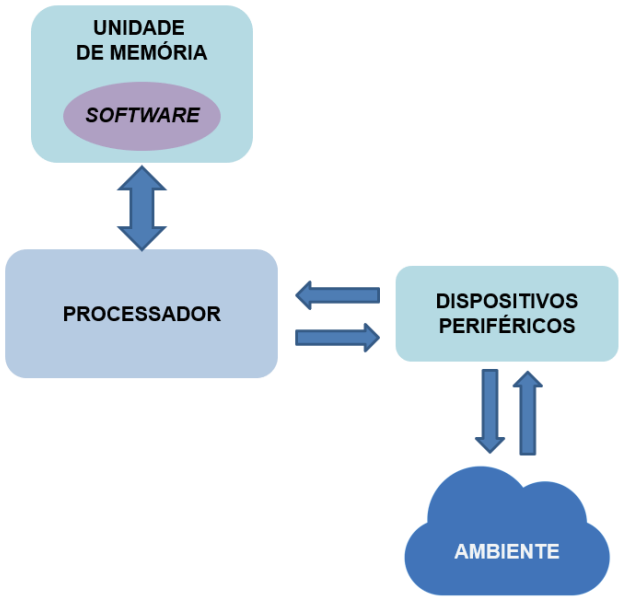
Dos protocolos existentes para a Camada de Aplicação IoT, o MQTT se mostrou o mais indicado para a aplicação proposta neste trabalho. Ele foi projetado para redes com baixa largura de banda e tem baixo consumo de energia, enquanto garante confiabilidade na transmissão de dados. Além disso, ele é ideal para redes com limitações em volume de dados e conexão frágil. Outras características decisivas na escolha do protocolo foram a facilidade de implementação – visto que muitas aplicações MQTT podem ser executadas usando uma média de quatro comandos base (*connect, publish, subscribe* e *disconnect*) – e o suporte existente para perda de conexão entre cliente e servidor (LAMPKIN et al., 2012).

2.6 SISTEMAS EMBARCADOS

A maioria dos dispositivos que compõe as redes IoT são sistemas embarcados. Um sistema embarcado é um sistema baseado em um microprocessador incorporado a um dispositivo e sua função é monitorar e/ou controlar uma tarefa (REDDY, 2002). Diferentemente de um computador pessoal que pode executar diversas aplicações, como um jogo ou um editor de texto, um sistema embarcado executa somente a rotina para a qual foi criado (HEATH, 2002). Por exemplo, ainda segundo o mesmo autor, o sistema embarcado criado para controlar uma máquina de lavar não pode ser usado para controlar uma geladeira.

De modo geral, a estrutura de um sistema embarcado, representada na Figura 20, é composta por processador, unidade de memória, *software* instalado e dispositivos periféricos (GODSE; MULANI, 2009). O processador é responsável por interpretar e executar códigos armazenados na unidade de memória, que é um chip que armazena toda a informação contidas no dispositivo (REDDY, 2002). O *software* instalado pode ser entendido como um conjunto de instruções que controlam o funcionamento do sistema embarcado, enquanto os dispositivos periféricos podem ser definidos como os componentes eletrônicos externos que funcionam como entrada e saída de dados, sensores e atuares, respectivamente (HEATH, 2002).

Figura 20 – Representação da composição de um sistema embarcado



Fonte: Elaborado pela autora, 2020.

2.6.1 BeagleBone Black

A placa BeagleBone Black, apresentada na Figura 21, é uma plataforma de prototipagem eletrônica de *hardware* livre, ou seja, o projeto completo da placa é divulgado gratuitamente na internet, desde a lista de materiais até a conexão entre os componentes (BEAGLEBOARD.ORG, 2019a).

Figura 21 – Placa BeagleBone Black

Fonte: BeagleBoard (2019a).

Considerando a estrutura de um sistema embarcado, esta placa incorpora o processador e a unidade de memória em uma única estrutura. O modelo também apresenta portas de entrada e saída de dados para conexão de dispositivos periféricos (BEAGLEBOARD.ORG, 2019a).

Assim como o *design* da placa, os diversos *softwares* que podem ser usados para programação são livres e disponibilizados online gratuitamente. Alguns dos Sistemas Operacionais compatíveis com a BBB são *Debian*, *Android* e *Ubuntu* (BEAGLEBOARD.ORG, 2019b). O Quadro 1 traz um breve resumo das principais especificações técnicas da placa.

Quadro 1 – Especificações técnicas da BBB

Processador	Sitara AM3358 (Arm Cortex-A8), 1 GHz
Memória SDRAM	512 MB, 800 MHz
Memória Flash	4 GB
Conectores	Ethernet (RJ45), micros, mini HDMI, USB tipo mini B, USB tipo A
Vídeo	HDMI 16 b, resolução 1280x1024
Áudio	Stereo, via HDMI
Sistemas Operacionais compatíveis	Debian, Angstrom, Android e outros
Botões	Reset, Boot e Power
Tensão de alimentação	DC 5 V
Dimensões	101,52 x 53,3 mm
Peso	39,68 g

Fonte: Adaptado de BeagleBoard.org (2019a).

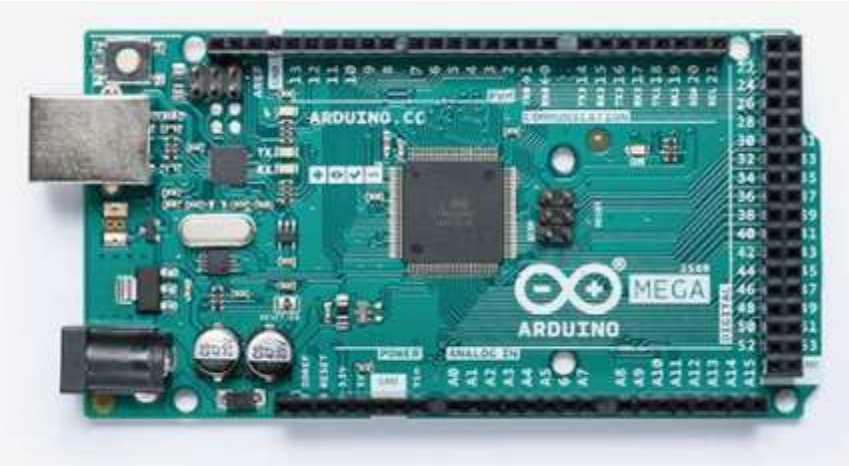
Esta placa foi escolhida devido à sua alta capacidade de processamento em comparação à placa Arduino Mega, que será apresentada na sessão a seguir. O dispositivo servidor possui múltiplas funções que não seriam suportadas por uma placa mais simples como a Arduino. Ele realiza o controle de acesso ao ambiente em que está instalado e atua como servidor do protocolo de mensagens com os dispositivos clientes. Além disso, também atua como banco de dados, armazenando as tentativas de acesso de todos os dispositivos clientes conectados a ele.

2.6.2 Arduino Mega 2560

A placa Arduino, apresentada na Figura 22, é a principal plataforma de prototipagem de código aberto do mundo e foi originalmente iniciada como um projeto de pesquisa

de Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis no *Interaction Design Institute* de Ivrea, Itália, no início dos anos 2000. A primeira placa foi introduzida em 2005 para ajudar os alunos, que não tinham experiência anterior em eletrônica ou programação de microcontroladores, a criar protótipos funcionais que conectam o mundo físico ao mundo digital. Desde então, tornou-se a ferramenta de prototipagem eletrônica mais popular usada por engenheiros e até grandes corporações (ARDUINO.CC, 2019a).

Figura 22 – Arduino Mega 2560



Fonte: Arduino.cc (2019b).

Desde a fundação do projeto Arduino, novas placas de desenvolvimento e bibliotecas de *software* foram criadas, expandindo o leque de possibilidades disponíveis para a comunidade. Entre elas, a placa Arduino Mega 2560 foi desenvolvida para projetos que requerem mais pinos de E/S (entrada e saída), mais memória de código e mais memória RAM (*Random Access Memory*) do que uma placa mais simples como a Arduino Uno. Suas especificações técnicas são listadas no Quadro 2.

Quadro 2 – Especificações técnicas da placa Arduino Mega 2560

Microcontrolador	ATmega2560			
Tensão de operação	5 V			
Tensão de alimentação (recomendado)	7 – 12 V			
Tensão de alimentação limite	6 – 20 V			
Pinos digitais I/O	54, das quais:			
	PWM	Serial	SPI	TWI
	15	8	4	2
Pinos analógicos	16			
Corrente contínua por pino I/O	20 mA			
Corrente contínua por pino analógico	50 mA			
Memória Flash	256 KB (8 KB para bootloader)			
SRAM	8 KB			
EEPROM	4 KB			
Frequência de relógio	16 MHz			
Dimensões	101,52 x 53,3 mm			
Peso	37 g			

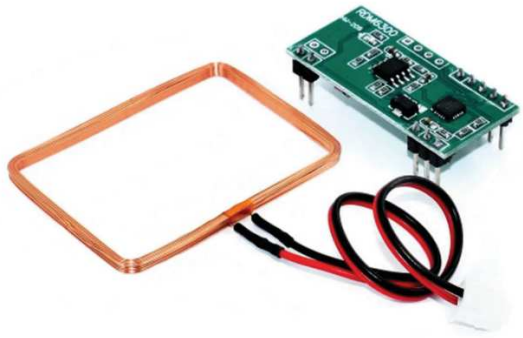
Fonte: Adaptado de Arduino.cc (2019b).

Outra placa Arduino amplamente utilizada é o modelo Uno. Tem 32 KB de memória *flash* e somente 14 pinos digitais I/O no total (ARDUINO.CC, 2021).

2.6.3 RDM6300

O módulo RDM6300, apresentado na Figura 23, é um módulo leitor RFID com frequência de operação de 125 kHz, compatível com placas de prototipagem como a BBB e a placa *Arduino*. O módulo vem acompanhado de uma antena de fio de cobre (ELTY ELETRONIKA, 2019).

Figura 23 – Módulo RDM6300



Fonte: Phipps Electronics (2019).

As condições de funcionamento do módulo são apresentadas a seguir no Quadro 3.

Quadro 3 – Especificações técnicas do módulo RDM6300

Frequência de operação	125 kHz
Taxa de transferência de dados	9600 bps
Interface	Weigang26 Or TTL Electricity Level RS232 format
Tensão de alimentação	DC 5V（±5%）
Corrente	< 50 mA
Temperatura de funcionamento	-10 ℃ ~ +70 ℃
Temperatura de armazenamento	-20 ℃ ~ +80 ℃
Humidade máxima	Relative humidity 0 ~ 95%
Dimensões	38,5 × 19 × 9 mm

Fonte: Adaptado de Elty Eletronika (2019).

2.6.4 Módulo Ethernet ENC28J60

O módulo de Ethernet ENC28J60, representado na Figura 24, é composto essencialmente pelo CI controlador Ethernet ENC28J60 e pelo conector RJ45. As informações a seguir são baseadas na folha de dados (usualmente referida como *datasheet*) disponibilizada pelo fabricante do CI (MICROCHIP TECHNOLOGY INC., 2008).

O módulo, que atende a todas as especificações IEEE 802.3, foi projetado para atuar como interface de rede Ethernet para qualquer controlador equipado com SPI (do inglês *Serial Peripheral Interface*). Ele deve ser conectado a um microcontrolador, como a placa Arduino.

Figura 24 – Módulo Ethernet ENC28J60

Fonte: Eletrogate (2019).

A comunicação com o controlador é implementada através de um pino de interrupção e do SPI, com taxas de *clock* de até 20 MHz. O chip ENC28J60 consiste em sete principais blocos funcionais:

a) interface SPI que serve como um canal de comunicação entre o microcontrolador e o ENC28J60;
b) registradores de controle usados para controlar e monitorar o ENC28J60;
c) buffer de RAM de porta dupla para pacotes de dados recebidos e transmitidos;
d) um árbitro para controlar o acesso ao buffer de RAM quando solicitações são feitas do DMA, transmitir e receber blocos;
e) interface de barramento que interpreta os dados e comandos recebidos pela interface SPI;
f) módulo MAC (*Medium Access Control*) que implementa a lógica MAC compatível com IEEE 802.3;
g) módulo PHY de camada física (*Physical Layer*) que codifica e decodifica os dados analógicos presentes na interface de par trançado do cabo.

O CI também contém outros blocos de suporte, como o oscilador, regulador de tensão e lógica de controle do sistema.

Um resumo das especificações do módulo é apresentado a seguir no Quadro 4.

Quadro 4 – Especificações do módulo Ethernet ENC28J60

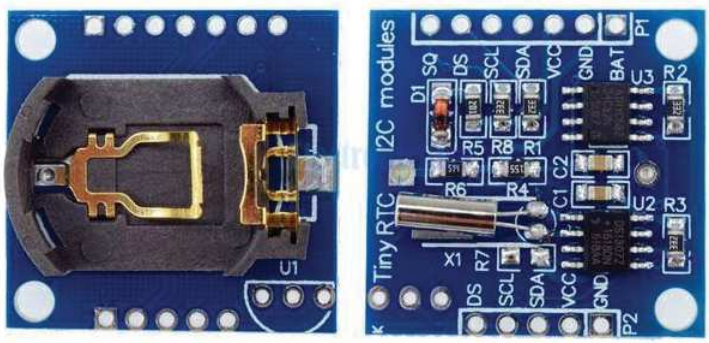
Interface Ethernet	IEEE 802.3 10/100/1000 Full e half duplex
Interface SPI	20 MHz de clock Modo 0,0
Tensão de alimentação	3,3 V (3,1 – 3,6)
Tensão de entrada dos pinos	5 V
Temperatura de funcionamento	0℃ ~ +70℃
Dimensões	38,5 × 19 ×9 mm

Fonte: Adaptado de Microchip Technology Inc. (2008).

2.6.5 Módulo RTC

O módulo de relógio de tempo real (usualmente referido pela sigla RTC, do inglês *Real-Time Clock*), exibido na Figura 25, é baseado no CI DS1307, que suporta o protocolo I²C para barramentos seriais. Ele utiliza uma bateria de célula de lítio como fonte de energia. O relógio fornece informações sobre segundos, minutos, horas, dias, mês e ano. A data do final do mês é ajustada automaticamente para meses com menos de 31 dias, incluindo correções para o ano bissexto. O relógio opera no formato de 24 ou 12 horas com o indicador AM/PM (ELECROW, 2014).

Figura 25 – Módulo RTC DS1307



Fonte: Easytronics (2019).

Assim como o módulo Ethernet ENC28J60, este dispositivo periférico deve ser conectado a um controlador ou placa de prototipagem. Caso o controlador seja desligado ou a alimentação do módulo seja interrompida, é feita a comutação para alimentação via bateria. Desse modo, a data armazenada na memória do CI permanece atualizada. As condições de operação do módulo são apresentadas no Quadro 5.

Quadro 5 – Especificações técnicas RTC DS1307

Interface Serial	I²C
Interface SPI	20 MHz de clock Modo 0,0
Tensão de alimentação	5 V (4,5 – 5,5)
Tensão da bateria	3 V (2 – 5,3)
Temperatura de funcionamento	0℃ ~ +70℃
Memória nvRAM	56 B
Dimensões	29mm x 26mm

Fonte: Adaptado de Maxim Integrated (2015).

3 DESENVOLVIMENTO DO SISTEMA DE CONTROLE DE ACESSO

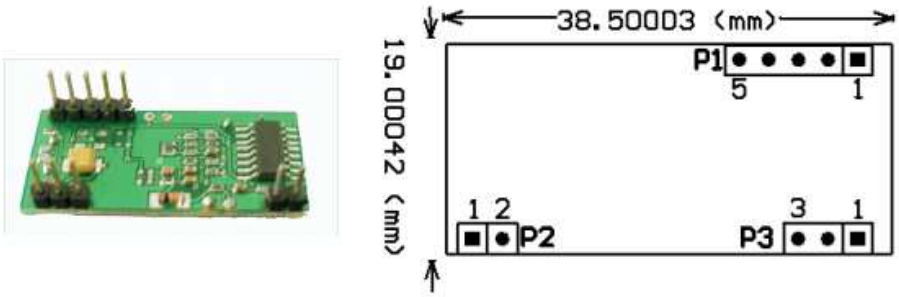
Esta seção descreve o desenvolvimento do sistema de controle de acesso, descrito na Seção 1.1. A princípio, realizou-se a montagem dos primeiros protótipos para o dispositivo servidor e para o dispositivo cliente. Em seguida, planejou-se as configurações física e lógica da rede de comunicação dos dispositivos, incluindo a conexão física entre os dispositivos e as estruturas de tópicos e mensagens utilizadas pelo protocolo MQTT. Prosseguiu-se com a elaboração das aplicações a serem executadas pelos dispositivos, incluindo a página *web* para gerenciamento de credenciais. Por fim, desenvolveu-se uma segunda versão dos protótipos, seguida pela realização de testes do funcionamento do sistema. Estas etapas são detalhadas a seguir.

Para não comprometer a fluidez de leitura do trabalho, optou-se por reunir os comandos e linhas de código utilizados em apêndices e anexos ao final do trabalho. Esse material foi idealizado para se assemelhar a um manual e pode ser utilizado de maneira prática para a reprodução futura do sistema desenvolvido.

3.1 PRIMEIROS PROTÓTIPOS E CONFIGURAÇÕES INICIAIS

Inicialmente, para a montagem dos protótipos dos dispositivos servidor e cliente, é fundamental definir quais serão seus componentes eletrônicos através do estudo de folhas de especificações técnicas e manuais de utilização. As folhas de especificações técnicas de um produto, comumente conhecidas como *datasheets*, são documentos disponibilizados pelos fabricantes de dispositivos eletrônicos. De modo geral, incluem uma breve descrição da finalidade do produto, seguida por condições de operação tais como tensão de alimentação, corrente e temperatura. A título de exemplo, a Figura 26, retirada do *datasheet* do módulo RDM6300, reproduz o esquemático dos pinos de conexão, bem como uma imagem ilustrativa do módulo.

Figura 26 – Esquemático do módulo RDM6300



Fonte: Elty Eletronika (2019).

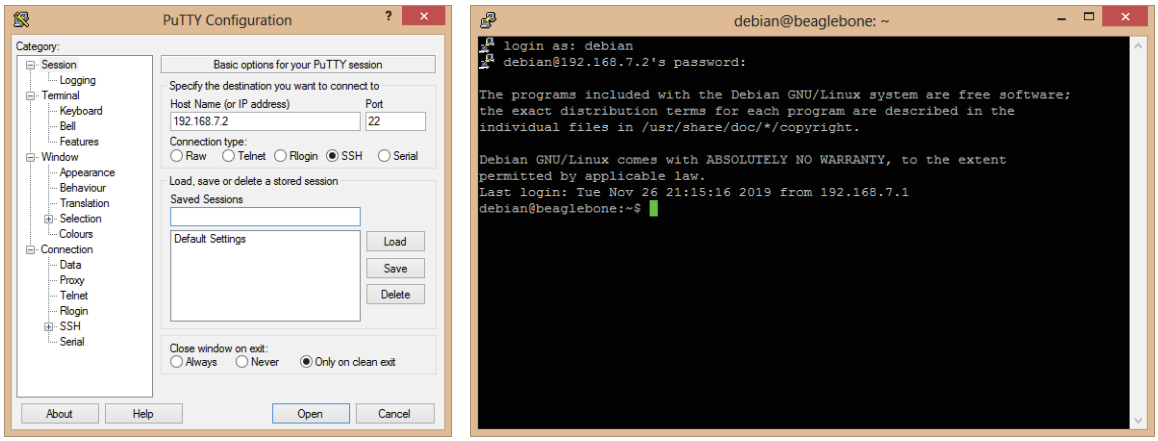
Após o estudo da funcionalidade de cada componente escolhido, foi iniciada a configuração inicial das placas de prototipagem e a montagem dos protótipos iniciais.

3.1.1 Dispositivo servidor

A primeira etapa para a utilização da placa BBB neste projeto foi a instalação do sistema operacional (usualmente referido pela sigla SO) e o *download* de *softwares* e bibliotecas necessários à integração dos demais módulos periféricos. A Beagleboard.org, fabricante da BBB, disponibiliza em sua página na Internet versões de sistemas operacionais para as mais diversas finalidades. Foi instalado o sistema Debian, na versão 9.5.

A versão do Debian escolhida não possui interface gráfica, portanto não é possível utilizar a BBB como um computador pessoal, por exemplo. Portanto, o acesso ao SO foi feito via SSH (do inglês, *Secure Shell*). Optou-se por conectar a BBB a um computador por um cabo USB e a conexão SSH é estabelecida através de um *software* chamado PuTTY. O PuTTY abre um terminal de comandos, apresentado na Figura 27, no qual é possível acessar a BBB pelo computador.

Figura 27 – Tela de configuração (a) e terminal de comandos (b) do PuTTY



(a) (b)

Fonte: A autora, 2020.

A seguir, é exposta a sequência das configurações e instalações feitas no sistema da BBB via terminal de comando. Os comandos executados e as versões instaladas são descritos no Apêndice A.

3.1.1.1 Atualização do *kernel*

O núcleo do sistema Debian, também conhecido como *kernel*, contém códigos de suporte à execução de *softwares* e *drivers*, portanto, visando o bom funcionamento do sistema, sua atualização é recomendada.

3.1.1.2 Sincronização do módulo RTC

Nesta etapa, objetivou-se sincronizar a data e o horário armazenados no módulo RTC DS1307 a um servidor NTP (do inglês, *Network Time Protocol*), de acordo com o fuso horário escolhido. O procedimento seguido foi baseado no tutorial *Adding a Real Time Clock to BeagleBone Black*, disponibilizado no site da empresa Adafruit e desenvolvido por Justin Cooper (2019).

Inicialmente, o módulo RTC foi conectado ao barramento I²C da placa BBB e sincronizado ao servidor “pool.ntp.org”. Posteriormente, foi criado um arquivo de serviço do sistema que será executado toda vez que o Debian for inicializado. Dessa forma, o horário utilizado pelo sistema será sempre obtido através do módulo RTC e permanecerá correto mesmo diante de falhas na rede.

Um serviço é um programa executado pelo SO de forma independente, em segundo plano. Segundo Red Hat Inc. (2020), um arquivo de serviço tem uma estrutura básica que conta com três seções principais: [Unit], [Service] e [Install]. A seção [Unit] contém a descrição do arquivo de serviço, especifica o seu comportamento e define dependência a outros serviços. A seção [Service] define a ação a ser executada pelo SO e o usuário a executá-la. Por fim, a seção [Install] contém informações sobre a instalação do serviço, usadas pelos comandos “systemctl enable” e “systemctl disable”.

3.1.1.3 Instalação do MySQL

Esta etapa consiste na criação de um banco de dados. Segundo a Oracle Corporation (2020), o *software* MySQL oferece um servidor de banco de dados SQL (*Structured Query Language)* rápido, robusto e com suporte para acesso de múltiplos usuários. Ele foi escolhido pela utilização intuitiva e facilidade de integração através de pacotes existentes para linguagens como o Python e o PHP.

3.1.1.4 Instalação e configuração do Mosquitto

Para estabelecer a comunicação via protocolo MQTT entre os dispositivos, é possível utilizar servidores online ou instalar um servidor local na BBB. Optou-se pela segunda opção pois é possível realizar uma configuração extensa dos parâmetros de funcionamento do servidor. Além disso, o uso de um servidor local pode prover mais segurança ao sistema pois só pode ser acessado por dispositivos conectados à mesma rede.

Portanto, foi feita a instalação do servidor MQTT escolhido, o Eclipse Mosquitto, que suporta a versão 3.1 do protocolo. Além do servidor, também foi feito o *download* do cliente Mosquitto, utilizado para os testes iniciais da ferramenta. O Mosquitto possui um arquivo de configuração padrão do servidor. Este arquivo foi alterado para que a conexão dos clientes seja feita somente mediante autenticação por usuário e senha.

3.1.1.5 Instalação de bibliotecas e atualização do pacote Python

Em continuidade, foi instalada a atualização do pacote da linguagem Python, escolhida como a principal linguagem deste protótipo por se tratar de uma linguagem

desenvolvida sob a licença livre de código aberto (usualmente referido pelo termo *open-source,* em inglês). Isso significa que ela pode ser utilizada gratuitamente em qualquer tipo de aplicação, pessoal ou comercial. Linguagens e programas de acesso livre tendem a oferecer bibliotecas diversificadas e frequentemente atualizadas, visto que uma vasta comunidade de programadores pode contribuir com códigos para novas funcionalidades. Existem bibliotecas, algumas disponibilizadas no site oficial da linguagem, para todas as funcionalidades implantadas neste protótipo como a leitura de dados RFID e o acesso a um banco de dados. O princípio *open-source* se aplica a todos os *softwares* e bibliotecas utilizados como base neste trabalho.

Os compiladores das versões 2 e 3 do Python são nativos do SO Debian. Optou-se por utilizar a versão 3 da linguagem para otimizar a compatibilidade com as bibliotecas escolhidas para o projeto.

Como os códigos das aplicações a serem executadas por este protótipo serão escritos, em sua maioria, em Python, optou-se por instalar bibliotecas para cada funcionalidade implementada. As bibliotecas possuem funções pré-definidas que facilitam a criação de códigos e, geralmente, diminuem seu número de linhas. Inicialmente, instalou-se a biblioteca Paho MQTT, desenvolvida pela mesma organização que mantém o Mosquitto, a Eclipse. A Paho é a biblioteca Python para a implementação de um cliente MQTT com suporte para as versões 3.1 e 3.1.1 do protocolo.

Posteriormente, foram instaladas as bibliotecas responsáveis pela comunicação com o módulo RDM6300 e o display LCD. Essas bibliotecas são, em resumo, responsáveis pelo acesso às portas digitais e analógicas, da BBB. A biblioteca pySerial é usada, especificamente, para a comunicação entre as portas digitais seriais da BBB e um módulo periférico, neste caso, o RDM6300. A biblioteca BBIO da empresa Adafruit foi desenvolvida para o controle de portas digitais da BBB em geral.

Adicionalmente, foram instalados os pacotes de bibliotecas CircuitPython e Blinka, também desenvolvidos pela Adafruit. Esses pacotes são uma extensão do Python pensada para a integração de diversos componentes eletrônicos como um display LCD.

De forma análoga, foi instalada a biblioteca MySQL Connector/Python, a partir da qual é possível realizar a integração com o servidor MySQL. Ou seja, é possível adicionar trechos de códigos em SQL dentro de um código em Python, sem a necessidade de um compilador diferente. Esta biblioteca é essencial para a busca e inserção de registros no banco de dados.

3.1.1.6 Criação do banco de dados

Complementarmente, foi preciso criar o banco de dados que armazenará as informações coletadas pelo sistema de controle de acesso. A escolha dos tipos de dados a serem coletados deve ser pertinente à finalidade do sistema. Um banco de dados destinado ao controle de acesso implantado em uma instituição de ensino contempla perfis de usuários diferentes de um banco destinado a um ambiente industrial, por exemplo. A seguir, será descrita a criação do banco de dados dentro do servidor MySQL previamente instalado.

Primeiramente, criou-se o banco de dados “Registro”, assim como o usuário “debian” e sua senha. Em seguida, criaram-se as tabelas que registram as informações necessárias ao funcionamento do sistema de controle de acesso. Os campos e os formatos dos dados das tabelas variaram consideravelmente ao decorrer do desenvolvimento deste projeto. Conforme as aplicações eram criadas e testadas, observavam-se possíveis melhorias às tabelas. Por tanto, as tabelas descritas a seguir são o resultado dessas modificações.

A primeira tabela criada foi a tabela “Cadastro”, que contém as informações pessoais dos usuários. Nos campos, descritos no Quadro 6, é possível adicionar dados como nome completo, matrícula e e-mail.

Quadro 6 - Campos da tabela “Cadastro”

Campo	**Descrição**	**Exemplo**
tag_id	Código de identificação da tag RFID (cartão de identificação do usuário)	01000BCAE6
matricula	Matrícula do usuário	20201EN20201
nome	Nome completo do usuário	Caroline Siqueira Lopes
coordenadoria	Coordenadoria a qual o usuário está ligado	Engenharia Elétrica
perfil	Pode assumir os valores: • ALUNO • PROFESSOR • COORDENADOR	Aluno
nivel	Nível administrativo do sistema de controle de acesso. Pode assumir os valores: 0 – superusuário; 1 – usuário administrador, 2 – usuário comum	2
email	Endereço de e-mail do usuário	caroline@email.com
data_adicao	Data de adição do cadastro	19-04-2021 12:00:00
resp_adicao	Responsável pelo cadastro	Leandro

Fonte: Elaborado pela autora, 2020.

Em seguida, foi criada a tabela “Acesso”, que contém as permissões de acesso dos usuários e o período de vigência, conforme descrito no Quadro 7.

Quadro 7 - Campos da tabela "Acesso"

Campo	Descrição	Exemplo
tag_id	Código de identificação da tag RFID	01000BCAE6
area	Área da instituição em que o ambiente está localizado	BlocoM
sala	Nome do ambiente	M205
inicio_vigencia	Início do período da permissão de acesso	19-04-2021 12:00:00
fim_vigencia	Final do período da permissão de acesso	19-04-2022 12:00:00
data_adicao	Data de adição do cadastro	19-04-2021 12:00:00
resp_adicao	Responsável pelo cadastro	Leandro

Fonte: Elaborado pela autora, 2020.

Foi criada a tabela “RegistroLog”, que contém todas as solicitações de acesso feitas nos ambientes monitorados. As solicitações são armazenadas conforme descrito no Quadro 8.

Quadro 8 – Campos da tabela “RegistroLog”

Campo	Descrição	Exemplo
horario	Data e horário da solicitação de acesso	07-05-2021 13:10:37
tag_id	Código de identificação da tag RFID	01000BCAE6
area	Área da instituição em que o	BlocoM

	ambiente está localizado	
sala	Nome do ambiente	M205
acesso	Indica se o acesso foi liberado ou negado. Pode assumir os estados: • LIBERADO • NEGADO	LIBERADO
entrada	Indica se a porta está aberta ou fechada no momento da solicitação. Pode assumir os estados: 0 – porta fechada 1 – porta aberta	0

Fonte: Elaborado pela autora, 2020.

Por fim, foi criada a tabela “LoginWeb”, que contém as autorizações de acesso ao servidor *web*, armazenadas conforme descrito no Quadro 9.

Quadro 9 – Campos da tabela "LoginWeb"

Campo	**Descrição**	**Exemplo**
tag_id	Código de identificação da tag RFID	01000BCAE6
usuario	Login de usuário *web*	caroline_lopes
senha	Senha do usuário *web*	12345#%!
inicio_vigencia	Início do período da permissão de acesso ao servidor *web*	19-04-2021 12:00:00
fim_vigencia	Final do período da permissão de acesso ao servidor *web*	05-05-2022 12:00:00

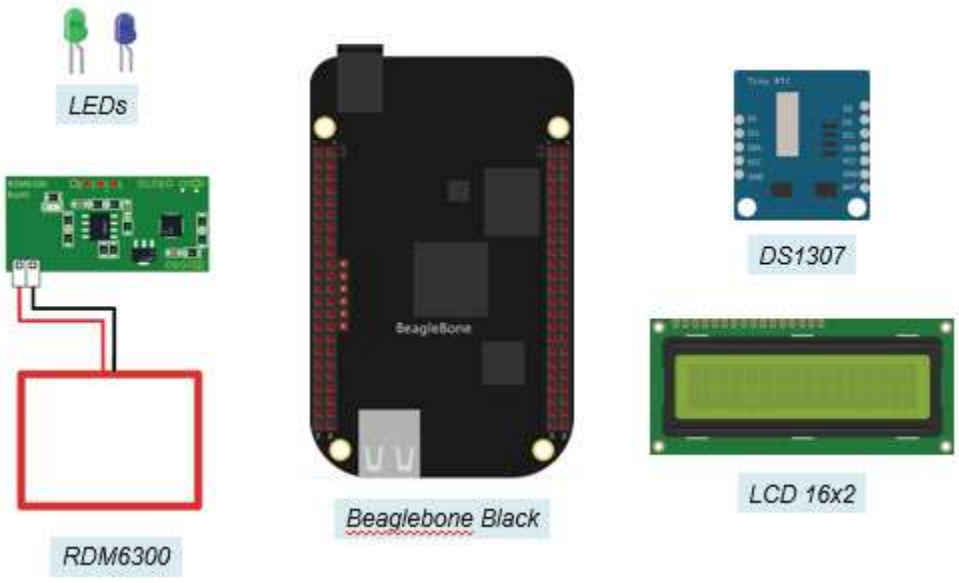
data_adicao	Data de adição do usuário *web*	19-04-2021 12:00:00

Fonte: Elaborado pela autora, 2020.

3.1.1.7 Primeiro protótipo

Depois de realizadas as configurações iniciais e atualização do SO da BBB, iniciou-se a montagem de um protótipo inicial do dispositivo servidor. Idealizou-se a construção de um circuito composto pelos componentes esquematizados na Figura 28: a placa BBB como central de processamento, o módulo DS1307 como relógio de tempo real, o módulo RDM6300 como leitor RFID, o *display* LCD 16x2 para informar ao usuário o status da solicitação de acesso e LEDs para representar a resposta da solicitação e a abertura da porta. Adicionalmente, utilizou-se componentes auxiliares como um *protoboard*, fios conhecidos popularmente como *jumpers*, resistores e um botão.

Figura 28 – Principais componentes do protótipo inicial servidor

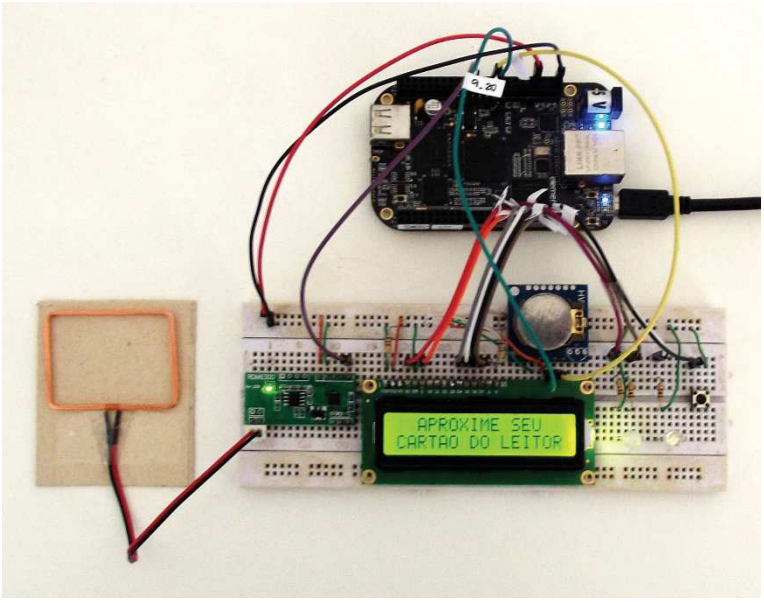


Fonte: Elaborado pela autora, 2020.

Os módulos foram fixados ao *protoboard* e conectados à BBB com o auxílio de *jumpers*. Para esta etapa, a BBB permaneceu conectada ao computador via USB.

A Figura 29 apresenta o protótipo exibindo a mensagem “Aproxime seu cartão do leitor”, apropriada para o estado em que o sistema aguarda a aproximação de uma *tag*.

Figura 29 – Primeiro protótipo do dispositivo servidor

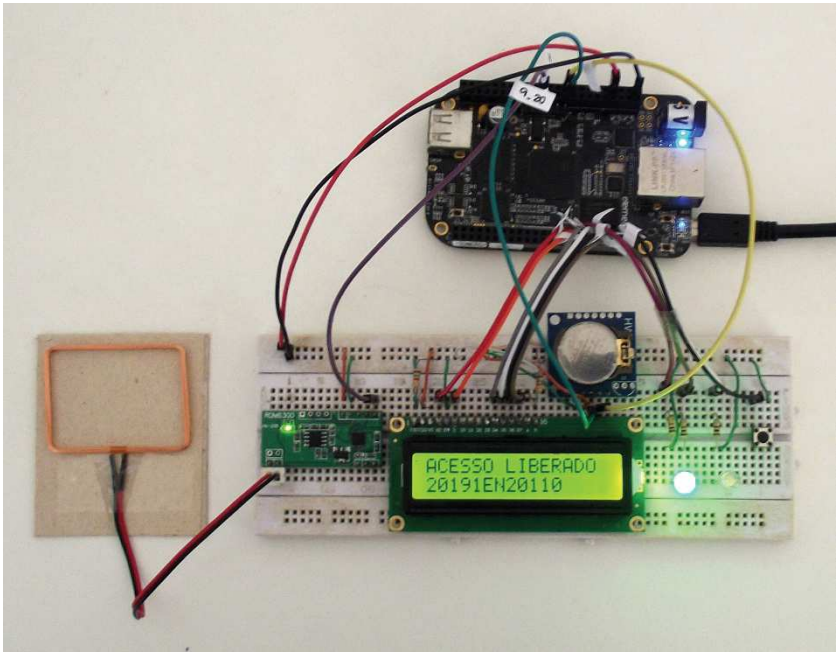


Fonte: Elaborado pela autora, 2020.

A

Figura 30 apresenta a resposta visual do sistema mediante à uma solicitação de acesso bem-sucedida.

Figura 30 – Primeiro protótipo do dispositivo servidor



Fonte: Elaborado pela autora, 2020.

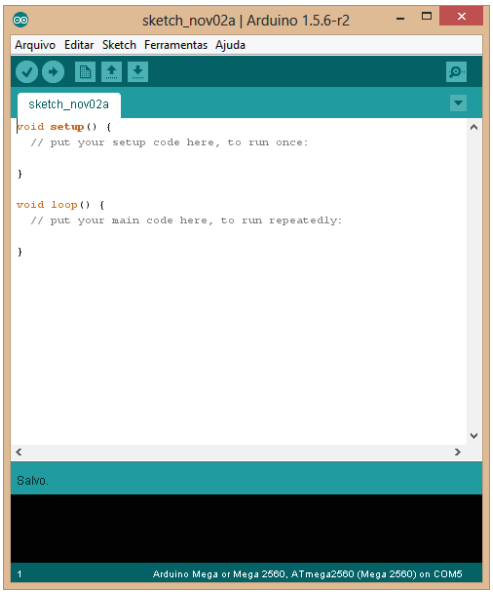
Em seguida, prosseguiu-se com a montagem do protótipo inicial do dispositivo cliente.

3.1.2 Dispositivo cliente

Diferentemente da BBB, a placa Arduino Mega não possui um SO embarcado como o Debian em sua configuração de fábrica e permite a execução de um único código. Este código, que é executado ininterruptamente enquanto a placa estiver energizada, pode ser escrito e compilado através da Arduino IDE. Esta, por sua vez, utiliza uma linguagem de programação derivada do C++, incluindo algumas mudanças para o controle do *hardware* da placa.

Desse modo, instalou-se a Arduino IDE em um computador e sua interface é apresentada na Figura 31. A placa Arduino deve ser conectada via USB para o armazenamento do código.

Figura 31 – Tela inicial do programa Arduino IDE



Fonte: Elaborado pela autora, 2020.

Assim como com o protótipo servidor, se faz necessário o uso de bibliotecas, dessa vez específicas para uso em placas Arduino. A maioria das bibliotecas utilizadas a seguir é disponibilizada online gratuitamente sob a licença *open-source*. O *download* foi feito e os arquivos foram adicionados à pasta “libraries” da IDE.

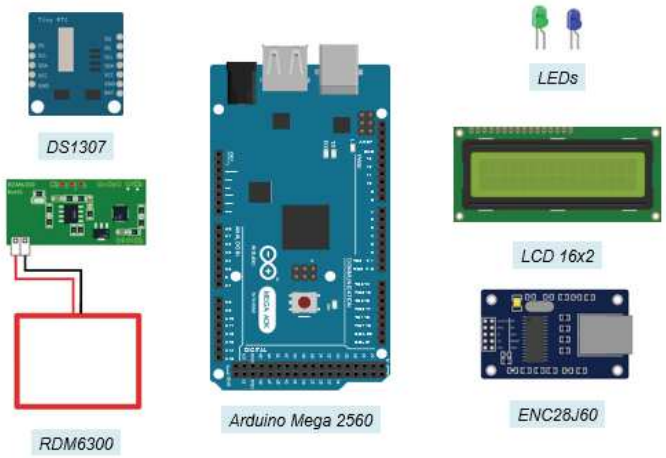
Apesar de não ser preciso a instalação de *softwares* como feito na placa BBB, foi necessária a configuração inicial do horário armazenado no módulo RTC. Para tal, executou-se o código descrito no Anexo A, desenvolvido por René Wennekes (2020). Foi necessário que para esta configuração a rede estivesse conectada à Internet para o acesso ao servidor NTP online. Depois de armazenado no módulo, o horário correto é mantido.

3.1.2.1 Primeiro protótipo

Depois de realizados a instalação da IDE Arduino e o *download* das bibliotecas necessárias, iniciou-se a montagem de um protótipo inicial do dispositivo cliente. Idealizou-se a construção de um circuito composto pelos componentes esquematizados na Figura 32: a placa Arduino Mega 2560 como central de processamento, o módulo DS1307 como relógio de tempo real, o módulo RDM6300 como leitor RFID, o *display* LCD 16x2 para informar ao usuário o status da solicitação de acesso e LEDs para representar a resposta da solicitação e a abertura

da porta. Adicionalmente, utilizou-se componentes auxiliares como um *protoboard*, *jumpers*, resistores e um botão. Este processo foi efetuado de forma semelhante ao efetuado para o dispositivo servidor.

Figura 32 – Principais componentes do protótipo inicial servidor

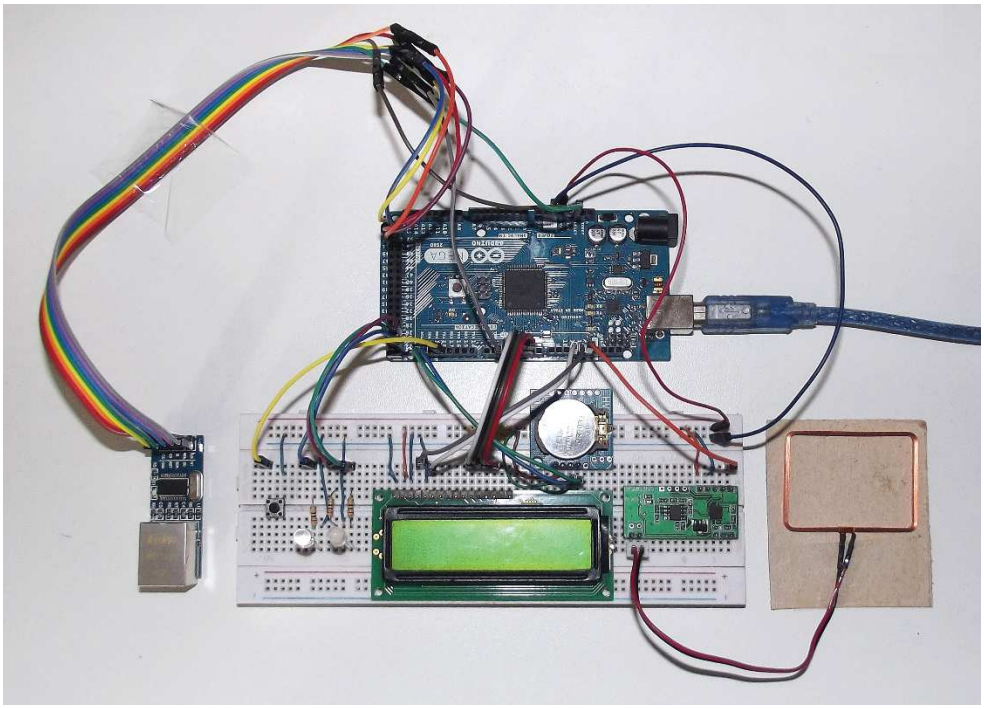


Fonte: Elaborado pela autora, 2020.

Os módulos foram fixados ao *protoboard* e conectados à Arduino com o auxílio de *jumpers*. Para esta etapa, a Arduino permaneceu conectada ao computador via USB.

A seguir, a Figura 33 apresenta o circuito montado de forma análoga ao feito com o protótipo inicial do dispositivo servidor.

Figura 33 – Protótipo inicial do dispositivo cliente



Fonte: Elaborado pela autora, 2020.

Após a montagem de ambos os protótipos iniciais, foi possível planejar as estruturas física e lógica da rede e visualizar como esses dispositivos se comunicariam. Este processo é detalhado a seguir.

3.2 ESTRUTURA DA REDE DE DISPOSITIVOS

Após construir os protótipos iniciais servidor e cliente, a primeira etapa na estruturação da rede dispositivos é a atribuição de um endereço de IP fixo à placa BBB. Os roteadores geralmente usam o Protocolo de Configuração Dinâmica de Host (de sigla DHCP, do inglês *Dynamic Host Configuration Protocol*) para alocação de endereços IP, o que indica que endereços diferentes podem ser definidos a cada vez que dispositivos se conectam a rede. Entretanto, para padronizar o acesso ao servidor MQTT e à página *web*, o dispositivo servidor deve ter um endereço de IP fixo.

Este endereço deve ser definido mediante a um estudo prévio da rede ligada ao roteador. Deve-se escolher um endereço não utilizado por outros dispositivos e, caso o roteador continue com a alocação por DHCP ativada, é necessário criar uma exceção DCHP para este endereço.

Em seguida, prosseguiu-se com a implementação do protocolo de troca de mensagens e a definição do meio de comunicação (sem fio ou com fio) entre os dispositivos.

A topologia física da rede de dispositivos é a topologia em estrela, na qual os dispositivos são conectados individualmente a um roteador. Optou-se pela conexão via Ethernet, com cabos RJ45, visando maior estabilidade e confiabilidade de rede. A conexão via Wi-Fi foi descartada visto que a qualidade da conexão pode ser influenciada pela distância e barreiras físicas entre o dispositivo e o roteador ou condições climáticas, por exemplo. O roteador escolhido foi o modelo DIR-600 da marca D-Link. Este modelo possui uma porta WAN (Internet) e 4 portas LAN (Ethernet), além de Wi-Fi.

A topologia lógica, por sua vez, é a topologia cliente/servidor, estabelecida pelo protocolo MQTT. Optou-se por padronizar as estruturas de tópico e de mensagens do sistema, conforme descrito a seguir.

3.2.1 Configuração MQTT

Para padronizar a troca de mensagens entre os dispositivos do sistema, criaram-se estruturas fixas para os tópicos MQTT e para as mensagens enviadas pelos dispositivos. Em suma, os tópicos são responsáveis por fornecer as informações do dispositivo cliente ao dispositivo servidor e as mensagens contém as informações da solicitação de acesso em curso. A estrutura básica dos tópicos é apresentada abaixo.

```
ControleDeAcesso/area/[ID_area]/sala/[ID_sala]/[tipo_de_mensagem]
```

As informações são separadas por barras e o primeiro campo, ControleDeAcesso, indica a finalidade do sistema. Em seguida, o local onde o dispositivo está instalado é definido pelos campos [ID_area] e [ID_sala]. Esses campos podem ser preenchidos com setores e salas específicas da instituição, respectivamente. Por fim, é definido o campo [tipo_de_mensagem]. Este campo pode ter dois valores: “acesso” ou “permissao”. Os dispositivos clientes publicam somente em tópicos terminados em “acesso” e estão inscritos para somente receber mensagens de autorização dos tópicos terminados em “permissao”. Portanto, a exemplo de um

dispositivo cliente localizado no Bloco M controlando o acesso à sala M204, seus tópicos seriam:

Publicar em: `ControleDeAcesso/area/BlocoM/sala/M204/acesso`

Inscrito em: `ControleDeAcesso/area/BlocoM/sala/M204/permissao`

O dispositivo servidor, por sua vez, funciona de forma diferente. Apesar de ter em sua nomenclatura a palavra “servidor” (visto que esta é a sua principal funcionalidade), este dispositivo pode atuar também como cliente e pode ser instalado em um ambiente para controle de acesso. Como cliente, está inscrito no tópico “ControleDeAcesso/#”, o símbolo “#” atua como uma “carta coringa” e abrange todos os tópicos abaixo do nível “ControleDeAcesso”. Como servidor, ele é responsável por receber as mensagens enviadas por todos os dispositivos clientes e tratar as solicitações de acesso buscando as credenciais em seu banco de dados.

A mensagens de solicitação de acesso enviadas pelos clientes nos tópicos terminados em “acesso” tem a seguinte estrutura:

```
[ID_tag]/[estado_porta]/[data_hora]
```

O campo [ID_tag] refere-se ao código de identificação da *tag* RFID apresentada. O campo [estado_porta] indica se a porta do ambiente está aberta ou fechada no momento da solicitação e pode assumir os valores “P1” ou “P0”, respectivamente. O estado da porta, neste trabalho, é detectado através da leitura do estado do LED indicativo (LED acesso indica porta aberta e LED apagado indica porta fechada). Por fim, o campo [data_hora] indica a data e horário da solicitação sob o formato “dd-MM-aaaa hh:mm:ss”. O exemplo abaixo indica a solicitação de acesso feita pela *tag* de código “01000BCAE6” no dia 10-10-2020 às 14:55:20, enquanto a porta estava fechada.

```
01000BCAE6/P0/10-10-2020 14:55:20
```

A mensagens de autorização de acesso enviadas pelo servidor nos tópicos terminados em “permissao” tem a seguinte estrutura:

```
[ID_tag]/[matricula]/[autorizacao]
```

O campo [ID_tag] refere-se ao código de identificação da *tag* RFID apresentada e o campo [matricula] identifica o número de identificação do usuário, que pode ser a matrícula de aluno ou número do servidor. Por fim, o campo [autorizacao] pode assumir os valores “LIBERADO” e “NEGADO” caso o acesso seja autorizado ou não. O exemplo abaixo indica que a *tag* de código “01000BCAE6” está associada a matrícula “20001EN0000” e o acesso ao ambiente está autorizado.

```
01000BCAE6/20001EN0000/LIBERADO
```

Caso o usuário não esteja cadastrado no sistema ou não tenha permissão de acesso ao ambiente, não haverá um número de matrícula e somente a mensagem “ACESSO NEGADO” será exibida no visor LCD.

A estrutura dos tópicos e das mensagens foi projetada para evitar erros na resposta a uma solicitação. Ambas as mensagens de acesso e de permissão contêm o número da *tag* RFID. Assim, é improvável que uma solicitação receba a resposta destinada a uma solicitação de uma *tag* diferente.

3.3 ELABORAÇÃO DAS APLICAÇÕES

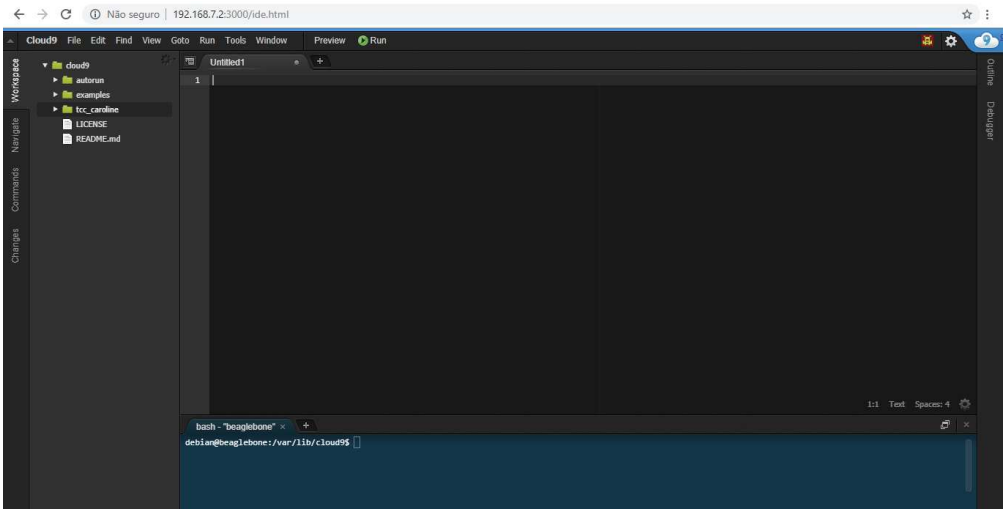
Em seguida, fez-se necessária a criação de aplicações que definem as ações e funcionalidades do sistema de controle de acesso. As aplicações são programas principais executados por ambos os dispositivos e a página *web* hospedada no dispositivo servidor.

3.3.2 Os programas principais

O programa principal do dispositivo cliente foi desenvolvido utilizando a Arduino IDE e os programas do dispositivo servidor foram desenvolvidos utilizando a Cloud9 IDE, apresentada na Figura 34.

Os códigos são apresentados nos Apêndices B e C e estão disponibilizados online na plataforma Github através do link https://github.com/carolineslopes/TCC-Caroline-Lopes .

Figura 34 – Interface da Cloud9 IDE, embarcada na Beaglebone Black



Fonte: Elaborado pela autora, 2020.

3.3.2 O programa como serviço do sistema

Para que o dispositivo servidor funcione de forma autônoma, é necessário que suas aplicações sejam iniciadas automaticamente com a inicialização do SO Debian. Para tal, é necessária a criação e habilitação de arquivos de serviço do sistema. Inicialmente, criou-se serviço para a inicialização do servidor Mosquitto, nomeado como “mosquitto.service”. O serviço de inicialização do MySQL, denominado “mysql.service”, é padrão do Debian e não foi alterado.

Adicionalmente, mostrou-se necessário criar um serviço para autorizar a leitura de um arquivo de sistema utilizado pela biblioteca “board” do Python. Sem esta etapa, o programa principal apresenta falhas em sua execução.

Por fim, criou-se o serviço para a execução do programa principal do controle de acesso, nomeado como “controle-de-acesso.service”. O arquivo de serviço criado define que o programa principal depende dos servidores Mosquitto e MySQL e deve ser executado exclusivamente depois da inicialização completa dos serviços “mosquitto.service” e “mysql.service”.

3.3.2 Página web

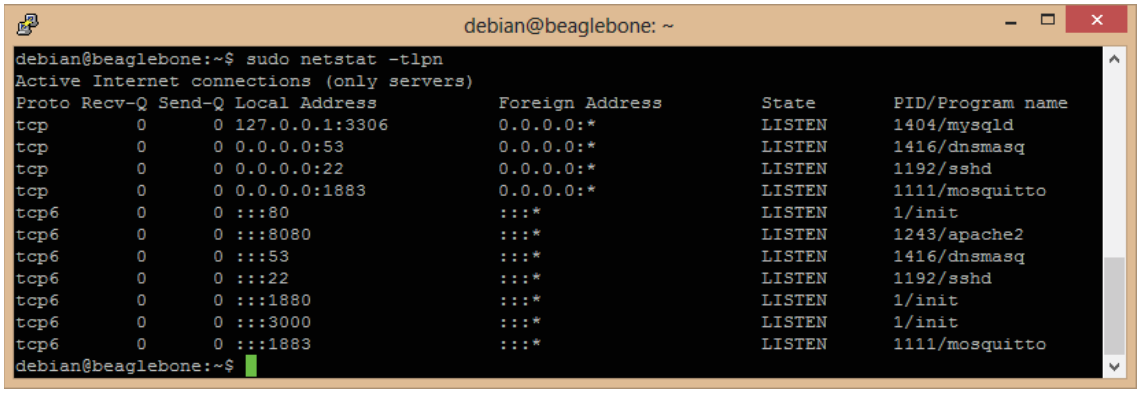
Como observado na etapa criação do banco de dados, o acesso direto ao MySQL requer conhecimento na linguagem SQL e apresenta um risco de segurança, visto

que o SO do dispositivo servidor deve ser acessado diretamente. Portanto, a fim de facilitar o gerenciamento dos dados armazenados e aumentar a segurança do sistema, optou-se pelo desenvolvimento de uma página *web* na qual um ou mais usuários possam cadastrar novos usuários, alterar permissões de acesso e gerar relatórios de sistema, através de uma interface amigável.

Nesta etapa, foi necessário habilitar a placa BBB para atuar como servidor *web* e a versão 2.4.25-3 do Apache foi instalada. Prosseguiu-se com a instalação da versão 7.3 do pacote PHP, que é uma linguagem de programação voltada para o desenvolvimento de aplicações *web*.

Conforme exemplificado pela Figura 35, o servidor Apache ocupa a porta 8080, geralmente usada por servidores HTTP.

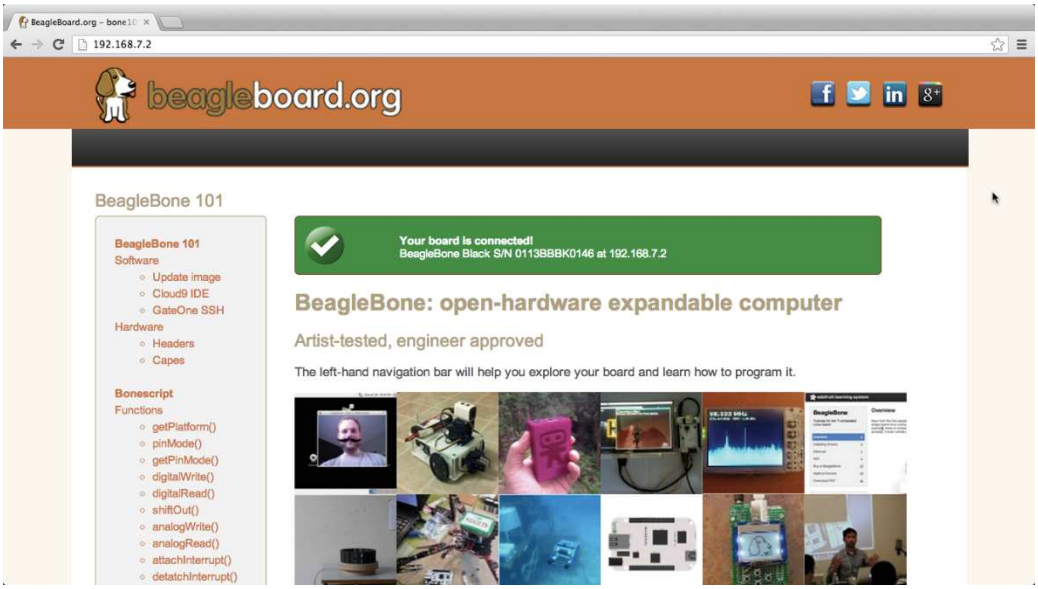
Figura 35 – Portas TCP utilizadas pelos serviços da BBB



Fonte: Elaborado pela autora, 2020.

A porta padrão para aplicações *web* é a porta 80, entretanto ela já é ocupada pelo servidor BeagleBone 101 (serviço init) que proporciona informações valiosas para quem deseja começar a utilizar a BBB. A página, apresentada na Figura 36, é exibida quando o usuário conecta a BBB ao computador via USB e digita o endereço http://192.168.7.2/ no navegador.

Figura 36 – Página BeagleBone 101



Fonte: Retirado de BeagleBone.org (2019c).

Após a configuração inicial do Apache, prosseguiu-se com a criação das páginas *web*. O propósito é criar uma plataforma de gerenciamento do sistema sem conectar-se diretamente ao dispositivo servidor. Para acessar esta página, bastar ter um computador conectado a mesma rede do sistema de controle de acesso e digitar no navegador o endereço de IP do dispositivo servidor.

A primeira página a ser exibida é a página de login, como exposto na Figura 37.

Figura 37 – Página de login do servidor *web*

Fonte: Elaborado pela autora, 2020.

Após inserir suas informações de usuário e senha, o usuário é redirecionado para a página principal “Acessos”, apresentada na Figura 38. No menu acima do relatório de acessos, o usuário tem acesso à funcionalidades de gerenciamento como visualização e exportação de relatórios, cadastro de novos usuários e alteração de usuários já cadastrados. Os relatórios podem ser exportados como arquivos de extensão CSV, XLS e PDF, além da possibilidade de impressão e alteração de colunas vísiveis nos relatórios.

Figura 38 – Página “Acessos” do servidor *web*

Fonte: Elaborado pela autora, 2020.

Na página “Usuários”, apresentada na figura Figura 39, uma lista de todos os usuários cadastrados no sistema de controle de acesso é exibida, sendo possível adicionar novos usuários e exportar relatórios.

Figura 39 – Página "Usuários" do servidor *web*

Fonte: Elaborado pela autora, 2020.

Similarmente, a página “Web”, apresentada na figura Figura 40, exibe a lista de todos os usuários que tem acesso ao próprio servidor *web*, sendo possível adicionar novos usuários e exportar relatórios.

Figura 40 – Página "Web" do servidor *web*

Fonte: Elaborado pela autora, 2020.

Por fim, há a página “Ambientes”, apresentada na figura , que lista todos os ambientes cadastrados no sistema de controle de acesso.

Figura 41 – Página "Ambientes" do servidor *web*

Fonte: Elaborado pela autora, 2020.

Os códigos das páginas *web* são extensos e numerosos, sendo disponibilizados online na plataforma Github através do link https://github.com/carolineslopes/TCC-Caroline-Lopes.

3.4 SEGUNDO PROTÓTIPO

Após a elaboração dos protótipos iniciais e suas aplicações, optou-se por aplicar melhorias aos protótipos servidor e cliente que facilitassem a realização dos testes finais de usabilidade. Dessa forma, os componentes dos dispositivos foram encapsulados em uma caixa de plástico preta, comercializada para uso em projetos de prototipagem, como apresentado na Figura 42.

Figura 42 – Segundo protótipo dos dispositivos servidor e cliente

Fonte: Elaborado pela autora, 2020.

Todas as conexões físicas entre as placas microcontroladoras e os módulos periféricos foram feitas com *jumpers* e são apresentadas no Apêndice D.

Depois de finalizados, os protótipos foram conectados a um roteador, conforme apresentado na Figura 43, e os testes finais de usabilidade foram iniciados.

Figura 43 – Protótipos conectados ao roteador

Fonte: Elaborado pela autora, 2020.

3.5 TESTES

Como última etapa do desenvolvimento deste projeto, iniciou-se a fase de testes para avaliar o funcionamento da rede de dispositivos de acesso de acordo com alguns índices de performance tais como o tempo médio de inicialização do sistema e o tempo médio de resposta a uma solicitação de acesso. Não se objetivou utilizar conceitos aprofundados de análise estatística.

Situações que podem ocorrer na utilização do sistema, como a solicitações de acesso simultâneas ao dispositivo servidor por parte dos dispositivos clientes, também foram contempladas na etapa de testes. Para tal, construiu-se um segundo dispositivo cliente, idêntico ao primeiro, totalizando três dispositivos construídos.

Os primeiros testes foram realizados na condição de uso considerada ideal, na qual somente os dispositivos servidor e cliente estão conectados ao roteador da rede.

Os testes foram realizados ao decorrer de duas semanas e os primeiros parâmetros analisados foram o tempo médio de inicialização do dispositivo servidor (Ti_S) e o tempo médio de inicialização do dispositivo cliente (Ti_C). Estes parâmetros contabilizam o tempo desde a energização do dispositivo, carregamento do programa principal até a conexão bem-sucedida ao servidor MQTT.

O próximo parâmetro analisado foi o tempo médio de resposta à solicitação de acesso (Ts). Para o dispositivo servidor, Ts_S compreende o tempo necessário para a leitura e decodificação da *tag* RFID e busca no banco de dados para autorização. Para o dispositivo cliente, Ts_C compreende o tempo necessário para a leitura e decodificação da *tag* RFID e comunicação com o dispositivo servidor (envio e recebimento de mensagens MQTT) para autorização. Salienta-se que, para Ts_S e Ts_C, não é contabilizado o período de 2 s no qual a autorização é exibida para o usuário pelo *display* LCD.

O parâmetro consumo médio de memória do programa principal, Cm_S, indica a porcentagem de memória RAM da placa BBB utilizada pelo programa principal do dispositivo servidor. Conforme informado na seção 2.6.1, a memória RAM da BBB possui 512 MB. Por sua vez, o parâmetro consumo médio de memória do programa principal, Cm_C, indica a porcentagem de memória *flash* da placa Arduino utilizada pelo programa principal do dispositivo servidor.

Outro parâmetro observado foi o nível médio de utilização da CPU, Uc. Salienta-se que esse parâmetro não tem relevância para o dispositivo cliente, visto que a placa Arduino tem somente a execução do programa principal em loop e dedica todo a sua capacidade de processamento para tal.

Os valores observados para os parâmetros anteriormente citados são apresentados no Quadro 10.

Quadro 10 – Parâmetros de funcionamento do sistema de controle de acesso

Descrição	Servidor		Cliente	
	Notação	Valor	Notação	Valor
Tempo médio de inicialização	Ti_S	1 min 50 s	Ti_C	50 s
Tempo médio de resposta à solicitação de acesso	Ts_S	< 1 s	Ts_C	< 1 s
Consumo médio de memória do programa principal	Cm_S	2,9%	Cm_C	14%
Nível médio de utilização da CPU	Ucs	0,1%	Ucc	100%

Fonte: Elaborado pela autora, 2020.

Iniciando a análise dos resultados, é notável a diferença notável entre Ti_S e Ti_C. Essa disparidade é atribuída a complexidade do SO da placa BBB, as diferentes capacidades de processamento das placas Arduino e BBB e a quantidade de serviços que precisam ser iniciados para o funcionamento do programa, tais como os servidores MySQL, Mosquitto e Apache. Portanto, caso os dispositivos sejam energizados simultaneamente, o dispositivo cliente aguardará a inicialização completa do dispositivo servidor, para assim poder se conectar ao Mosquitto.

Em seguida, o Ts observado para ambos os dispositivos é inferior a 1 s. A inexatidão do valor se deve ao fato de que não foi possível monitorá-lo através dos dispositivos, foi necessário cronometrar o tempo manualmente.

Em seguida, avaliaram-se os fatores qualitativos da performance do sistema. Erros de leitura das *tags* RFID ocorreram somente durante o desenvolvimento dos protótipos iniciais dos dispositivos, enquanto o funcionamento do leitor e das bibliotecas eram testados. Depois dessa fase, nenhum erro de leitura foi identificado. A biblioteca “rdm6300.h” do Arduino possui funções embutidas de comprovação e o

código escrito em Python para a BBB impede o envio de *tags* que não estejam corretas.

Todavia, erros podem ocorrer durante o envio e recebimento das mensagens MQTT, visto que devido a falhas na conexão e perdas de pacotes de dados transmitidos. Em caso de falha na rede, o dispositivo cliente possui rotina para reconexão ao servidor. Além de possíveis oscilações na rede, realizou-se o teste por meio da desconexão dos cabos de rede.

Simulando a situação de falta de energia, realizaram-se testes desconectando simultaneamente os dispositivos da tomada e os dispositivos iniciam automaticamente seus programas. A execução automática do programa principal da placa BBB é definida pelas especificações do arquivo “controle-de-acesso.service”.

4 CONCLUSÃO

Este trabalho se propôs a desenvolver um sistema de controle de acesso, composto por uma rede de dispositivos que utiliza o RFID como tecnologia de identificação e o MQTT como protocolo de comunicação.

Projetar a rede de dispositivos se mostrou uma tarefa desafiadora devido ao forte caráter interdisciplinar deste trabalho. Através deste projeto, foi possível aperfeiçoar os conhecimentos adquiridos ao longo do curso de graduação, principalmente conhecimentos em eletrônica digital, programação e redes de computadores. Além do desafio da montagem física dos protótipos, foi indispensável o aprendizado de diversas linguagens de programação para a implementação das aplicações tais como Python, C, SQL, PHP e HTML.

Após a conclusão da etapa de testes, observou-se que a tecnologia RFID e mais precisamente os leitores RFID 125 kHz de modelo RDM6300 se mostraram como uma tecnologia de identificação rápida e eficaz, com reconhecimento de *tag* em menos de 1 s e existência mínima de erros de leitura.

Aliado a estrutura física da rede de dispositivos, foi desenvolvido o servidor *web* para o gerenciamento de credencias e geração de relatórios. Essa ferramenta se mostrou extremamente valiosa, por oferecer diversas funcionalidades centralizadas em uma interface amigável ao usuário. Adicionalmente, permite que o administrador do sistema realize o gerenciamento de um computador ligado à rede e não diretamente do dispositivo servidor, oferecendo mais comodidade e segurança à operação diária do sistema.

Assim, concluiu-se que é tecnicamente viável o desenvolvimento de um sistema de controle de acesso eficaz utilizando *open-source* e materiais facilmente encontrados no mercado.

Por fim, a estruturação deste trabalho foi pensada para que este projeto possa ser replicado futuramente por alunos do Ifes e entusiastas de telecomunicações e sistemas embarcados. Espera-se que este projeto sirva como base para um futuro estudo de implementação de um sistema de controle de acesso para garantir mais segurança a alunos e servidores.

4.1 TRABALHOS FUTUROS

A partir do estudo feito e das conclusões obtidas, espera-se que este trabalho inspire a produção de novos projetos de pesquisa. Dessa forma, são descritas a seguir algumas sugestões de melhoria ao sistema desenvolvido visando sua futura implementação:

a) estudo de mecanismos de segurança para a rede como a implantação de encriptação de mensagens e transmissão via SSL TLS;
b) otimização da constituição física dos protótipos, com o desenvolvimento de placas de circuito impresso (PCI) e caixas plásticas personalizadas;
c) inclusão de um segundo leitor RFID, no interior do ambiente, para cadastrar a saída dos usuários;
d) inserção de mais dispositivos e realização de testes de escalabilidade;
e) aperfeiçoamento da página *web* de gerenciamento para exibição de alertas em caso de falhas no sistema de controle de acesso;
f) aperfeiçoamento da página *web* de gerenciamento para edição de usuários em lote;
g) criação de relatórios para monitoramento de rede, como registros de falhas, falta de energia e instabilidade de conexão;
h) adição de um cartão micros para expansão de armazenamento;
i) testes de robustez através do cadastro de grandes quantidades de usuários;
j) adição de módulo de conexão sem fio (Bluetooth ou WiFi) aos dispositivos e criação de aplicação de gerenciamento para aparelho celular.

REFERÊNCIAS

AL-FUQAHA, A. et al. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. **IEEE Communications Surveys & Tutorials**, v. 17, n. 4, p. 2347–2376, 2015.

ARDUINO.CC. **Arduino - About Us**. Disponível em: <https://www.arduino.cc/en/Main/AboutUs>. Acesso em: 13 nov. 2019a.

ARDUINO.CC. **Arduino Mega 2560 Rev3**. Disponível em: <https://store.arduino.cc/usa/mega-2560-r3>. Acesso em: 13 nov. 2019b.

ARDUINO.CC. **Arduino Uno Rev3**. Disponível em: <https://store-usa.arduino.cc/products/arduino-uno-rev3?selectedStore=us>. Acesso em: 8 abr. 2021.

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: A survey. **Computer Networks**, v. 54, n. 15, p. 2787–2805, out. 2010.

AUGUSTO, J. C.; NAKASHIMA, H.; AGHAJAN, H. Ambient Intelligence and Smart Environments: A State of the Art. In: **Handbook of Ambient Intelligence and Smart Environments**. Boston, MA: Springer US, 2010. p. 3–31.

BEAGLEBOARD.ORG. **BeagleBone Black**. Disponível em: <https://elinux.org/Beagleboard:BeagleBoneBlack>. Acesso em: 4 nov. 2019a.

BEAGLEBOARD.ORG. **System Reference Manual**. Disponível em: <https://github.com/beagleboard/beaglebone-black.wiki.git>. Acesso em: 4 nov. 2019b.

BEAGLEBOARD.ORG. **BeagleBord.org - bone101**. Disponível em: <https://beagleboard.org/Support/bone101>. Acesso em: 15 nov. 2019c.

CISCO. **Internet of Things At a Glance**. [s.l: s.n.].

COOPER, J. **Adding a Real Time Clock to BeagleBone Black**. Disponível em: <https://learn.adafruit.com/adding-a-real-time-clock-to-beaglebone-black?view=all>. Acesso em: 20 ago. 2019.

EASYTRONICS. **Módulo Tiny RTC DS1307 - Real Time Clock**. Disponível em: <https://www.easytronics.com.br/modulo-tiny-rtc-ds1307-real-time-clock>. Acesso em: 23 nov. 2019.

ELECROW. **Tiny RTC**, 2014.

ELETROGATE. **Módulo Ethernet ENC28J60 - Mini**. Disponível em: <https://www.eletrogate.com/modulo-ethernet-enc28j60-mini>. Acesso em: 16 nov. 2019.

ELTY ELETRONIKA. **RDM630 Specification**. Disponível em: <https://elty.pl/upload/download/RFID/RDM630-Spec.pdf>. Acesso em: 15 nov. 2019.

GODSE, A. P.; MULANI, A. O. **Embedded Systems**. 1. ed. Pune: Technical Publications, 2009.

HEATH, S. **Embedded Systems Design**. 2. ed. Oxford: Elsevier Science, 2002.

HILLAR, G. C. **MQTT Essentials - A Lightweight IoT Protocol**. 1. ed. Birmingham: Packt Publishing, 2017.

HWANG, I.; BAEK, J. Wireless access monitoring and control system based on digital door lock. **IEEE Transactions on Consumer Electronics**, v. 53, n. 4, p. 1724–1730, 2007.

INTRELBRAS. **Leitor de cartão RFID 125 kHz LE 130**. Disponível em: <https://www.intelbras.com/pt-br/leitor-de-cartao-rfid-125-khz-le-130>. Acesso em: 15 nov. 2019.

KOREN, I.; KLAMMA, R. Enabling visual community learning analytics with Internet of Things devices. **Computers in Human Behavior**, v. 89, p. 385–394, 2018.

LAMMLE, T. **CompTIA Network+ Deluxe Study Guide**. 1. ed. Indianapolis: Wiley Publishing Inc., 2009.

LAMPKIN, V. et al. **Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry**. 1. ed. Poughkeepsie: IBM Redbooks, 2012.

LUOTO, A.; SYSTÄ, K. Fighting network restrictions of request-response pattern with MQTT. **IET Software**, v. 12, n. 5, p. 410–417, 2018.

MAHMOUD, R. et al. **Internet of things (IoT) security: Current status, challenges and prospective measures**. 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST). **Anais**...IEEE, dez. 2015.

MAXIM INTEGRATED. **DS1307**, 2015.

MEYER, M. et al. **Mike Meyers' Comptia RFID+ Certification Passport**. 1. ed. Nova Iorque: McGraw-Hill Osborne Media, 2007.

MICROCHIP TECHNOLOGY INC. **ENC28J60 Data Sheet**, 2008.

OASIS OPEN. **MQTT Version 3.1.1 - OASIS Standard**, 2014.

ORACLE CORPORATION. **MySQL :: MySQL 8.0 Reference Manual :: 1 General Information**. Disponível em: <https://dev.mysql.com/doc/refman/8.0/en/introduction.html>.

PALMA, D. et al. An Internet of Things Example: Classrooms Access Control over Near Field Communication. **Sensors**, v. 14, n. 4, p. 6998–7012, 21 abr. 2014.

PHIPPS ELECTRONICS. **RDM6300 RFID Reader Mod – 125KHz**. Disponível em: <https://www.phippselectronics.com/product/rdm6300-125khz-em4100-rfid-reader-module-uart-output-access-control/>. Acesso em: 20 nov. 2019.

RED HAT INC. **Creating and Modifying systemd Unit Files**. Disponível em: <https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/sect-managing_services_with_systemd-unit_files>. Acesso em: 15 nov. 2019.

REDDY, P. M. Embedded systems. **Resonance**, v. 7, n. 12, p. 20–30, 2002.

REIJULA, J. et al. Human well-being and flowing work in an intelligent work environment. **Intelligent Buildings International**, v. 3, n. 4, p. 223–237, 19 out. 2011.

TANIM, M. M. Z. **How does passive RFID works, briefly explained.**, 2016.

WENNEKES, R. **ENC28J60 NTP Client (and DHCP) Demo**. Disponível em: <https://www.wennekes.info/scripts.php?chapter=Arduino&article=NTP Client - ENC28J60https://www.wennekes.info/scripts.php?chapter=Arduino&article=NTP Client - ENC28J60>. Acesso em: 25 fev. 2020.

WORTMANN, F.; FLÜCHTER, K. Internet of Things. **Business & Information Systems Engineering**, v. 57, n. 3, p. 221–224, jun. 2015.

YANG, J.-C. et al. An Intelligent Automated Door Control System Based on a Smart Camera. **Sensors**, v. 13, n. 5, p. 5923–5936, 10 maio 2013.

APÊNDICE A – CONFIGURAÇÃO INICIAL DO DISPOSITIVO SERVIDOR

Neste apêndice, encontra-se documentada a configuração inicial completa da placa BBB utilizada no dispositivo servidor. A disposição do conteúdo foi idealizada para que posteriormente possa ser utilizada como referência para a reprodução deste projeto, seguindo a estrutura de um manual “passo-a-passo”.

Seguindo a ordem dos procedimentos descritos na Seção 3, encontram-se documentados os comandos executados para as configurações e instalações feitas no SO da placa BBB, bem como as versões de *software* instaladas. Os comandos apresentados são precedidos pelo símbolo “$” quando executados em nível de usuário do sistema e o comando “sudo” indica a execução em nível de superusuário (usualmente referido como *root*).

Salienta-se que as versões instaladas e mencionadas a seguir eram as mais atualizadas durante o período de desenvolvimento do projeto. É possível que durante o período de testes e escrita deste texto outras atualizações mais recentes tenham sido lançadas.

A primeira etapa para a utilização da placa BBB neste projeto foi a instalação do SO Debian, na versão 9.5, e o *download* de *softwares* e bibliotecas necessários à integração dos demais módulos periféricos. A seguir, é exposta a sequência das configurações e instalações feitas no sistema da BBB via terminal de comando.

1. Atualização do *kernel*

Executando os comandos abaixo, a versão de *kernel* atualizada instalada foi a versão 3.14.108-ti-r124.

```
$ cd /opt/scripts/tools/
$ git pull
$ sudo ./update_kernel.sh
$ sudo reboot
```

2. Sincronização do módulo RTC

O mapeamento do endereço do módulo RTC no barramento I²C foi feito pelo comando abaixo:

```
$ sudo i2cdetect -y -r 2
```

O endereço detectado foi “0x68”. Foi verificado dispositivo é visível ao usuário:

```
$ echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-2/new_device
```

Em sequência, foi feita a sincronização do módulo ao servidor “pool.ntp.org” e a definição do fuso horário correto:

```
$ sudo apt install ntp
$ sudo apt install ntpdate
$ sudo ntpdate -b -s -u pool.ntp.org
$ sudo timedatectl set-timezone America/Sao_Paulo
$ sudo hwclock -w -f /dev/rtc2
```

É possível verificar o horário atualizado através do comando a seguir:

```
$ hwclock -r -f /dev/rtc2
```

Posteriormente, foi criado um serviço do sistema Debian que será executado toda vez que o sistema for inicializado. Primeiramente, criou-se um diretório e um arquivo, dedicados ao serviço:

```
$ mkdir /usr/share/rtc_ds1307
$ nano /usr/share/rtc_ds1307/clock_init.sh
```

O seguinte script foi adicionado ao arquivo:

```
#!/bin/bash
sleep 15
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-2/new_device
hwclock -s -f /dev/rtc2
hwclock -w
```

Em seguida criou-se o serviço que executará o script acima na inicialização do sistema:

```
$ nano /etc/systemd/system/rtc-ds1307.service
```

E adicionou-se o seguinte conteúdo:

```
[Unit]
Description=DS1307 RTC Service
[Service]
Type=simple
WorkingDirectory=/usr/share/rtc_ds1307
ExecStart=/bin/bash clock_init.sh
SyslogIdentifier=rtc_ds1307
[Install]
WantedBy=multi-user.target
```

O serviço foi habilitado e o sistema foi reinicializado para efetivar as mudanças feitas:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable rtc-ds1307.service
$ sudo systemctl start rtc-ds1307.service
$ sudo shutdown -h now
```

3. Instalação do MySQL e do Mosquitto

A instalação dos servidores MySQL e Mosquitto foram feitas através dos comandos abaixo. As versões 10.1.41-MariaDB-0+deb9u1 e 1.4.10-3 foram instaladas, respectivamente.

```
$ sudo apt update
$ sudo apt install mysql-server
$ sudo apt install mosquitto
$ sudo apt install mosquitto-clients
```

Criou-se um arquivo de senha chamado “pwd.txt”, incluindo o usuário “debian”:

```
$ mosquitto_passwd -c pwd.txt debian
```

Em seguida, foi criado um arquivo de configuração chamado “mqtt.conf”:

```
$ nano mqtt.conf
```

Adicionou-se o seguinte texto ao arquivo:

```
allow_anonymous false
password_file pwd.txt
```

O servidor Mosquitto pode ser executado a partir do seguinte comando:

```
$ mosquitto -c mqtt.conf -d
```

4. Instalação de bibliotecas e atualização do pacote Python

O compilador Python 3 e seu gerenciador de pacotes “pip” foram atualizados para as versões 3.5 e 9.0.1-2, respectivamente, através dos comandos abaixo:

```
$ sudo apt install python3
$ sudo apt install python3-pip
```

A versão 3 foi definida como padrão:

```
python –version
update-alternatives       --install       /usr/bin/python       python
/usr/bin/python2.7 1
update-alternatives       --install       /usr/bin/python       python
/usr/bin/python3.5 2
```

A biblioteca Paho MQTT foi instalada, na versão 1.5.0:

```
$ sudo pip3 install paho-mqtt
```

Instalou-se também a versão 3.4 da pySerial e a versão 1.1.1 da Adafruit BBIO:

```
$ pip3 install Adafruit_BBIO
$ pip3 install pyserial
```

Em adição, as versões mais atualizadas do Blinka e do CircuitPython foram instaladas, 3.5.1 e 3.2.1, respectivamente:

```
$ sudo pip3 install adafruit-blinka
$ sudo pip3 install adafruit-circuitpython-charlcd
```

Também foi instalada a biblioteca MySQL Connector/Python:

```
$ pip3 install mysql-connector-python
```

5. Criação do banco de dados

A seguir, será descrita a criação de um banco de dados dentro do servidor MySQL previamente instalado. As linhas de comando precedidas por “[mysql]>” representam os comandos executados no interpretador do servidor MySQL.

Primeiramente, é necessária a criação de um banco de dados e de um usuário para acessá-lo. Para tal, o servidor foi acesso via usuário *root*:

```
$ sudo mysql -u root -p
```

O banco de dados “Registro” foi criado:

```
[mysql]> CREATE DATABASE Registro;
```

Definiu-se o usuário “debian” e sua senha, além de sua permissão de acesso ao banco de dados:

```
[mysql]> CREATE USER debian@localhost IDENTIFIED BY ‘debian’;
[mysql]> SET PASSWORD FOR debian@localhost=password(***);
[mysql]> GRANT ALL ON Registro.* TO debian@localhost;
[mysql]> FLUSH PRIVILEGES;
```

Em seguida, criaram-se as tabelas que registram as informações necessárias ao funcionamento do sistema de controle de acesso, informando inicialmente em qual banco de dados elas devem ser criadas:

```
[mysql]> USE Registro;

[mysql]> CREATE TABLE Cadastro (tag_id VARCHAR(12) PRIMARY KEY,
matricula VARCHAR(12), nome VARCHAR(50), coordenadoria VARCHAR(50),
perfil VARCHAR(20), nivel INT(1), email VARCHAR(50), data_adicao
VARCHAR(22));

[mysql]> CREATE TABLE Acesso (num INT PRIMARY KEY INT AUTO_INCREMENT,
tag_id VARCHAR(12), area VARCHAR(20), sala VARCHAR(20),
inicio_vigencia VARCHAR(22), fim_vigencia VARCHAR(22), data_adicao
VARCHAR(22));
```

```
[mysql]> CREATE TABLE RegistroLog (num INT(11) AUTO_INCREMENT PRIMARY KEY, hora VARCHAR(22), tag_id VARCHAR(12), area VARCHAR(20), sala VARCHAR(10), acesso VARCHAR(20), entrada INT(1));

[mysql]> CREATE TABLE LoginWeb (num INT(5) AUTO_INCREMENT PRIMARY KEY, tag_id VARCHAR(12), usuario VARCHAR(50), senha VARCHAR(50), inicio_vigencia VARCHAR(22), fim_vigencia VARCHAR(22), data_adicao VARCHAR(22));
```

Para verificar as tabelas criadas:

```
[mysql]> SHOW TABLES;
```

Para verificar os atributos das colunas de uma tabela:

```
[mysql]> SHOW COLUMNS FROM Cadastro;
```

6. O programa como serviço do sistema

Inicialmente, criou-se um arquivo com extensão .service para a inicialização do servidor Mosquitto:

```
$ sudo nano /etc/systemd/system/mosquitto.service
```

O conteúdo do arquivo, apresentado a seguir, descreve como o sistema operacional deve executá-lo:

```
[Unit]
Description=Mosquitto MQTT Broker
ConditionPathExists=/etc/mosquitto/conf.d/mqtt.conf
After=network.target
Requires=network.target
[Service]
Type=forking
RemainAfterExit=no
StartLimitInterval=0
PIDFile=/run/mosquitto.pid
ExecStart=/usr/sbin/mosquitto -c /etc/mosquitto/conf.d/mqtt.conf -d
ExecReload=/bin/kill -HUP $MAINPID
```

```
Restart=on-failure
RestartSec=2
[Install]
WantedBy=multi-user.target
```

Posteriormente, o serviço foi habilitado:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable mosquitto.service
$ sudo systemctl start mosquitto.service
```

Adicionalmente, Criou-se o seguinte arquivo para autorizar a leitura de arquivo de sistema utilizado pela biblioteca “board” do Python:

```
$ sudo nano /etc/systemd/system/nvmem.service
```

O conteúdo do arquivo é apresentado a seguir:

```
[Unit]
Description=Adiciona permissao ao arquivo de sistema nvmem
After=mosquitto.service
[Service]
Type=oneshot
User=root
ExecStart=/bin/bash -c "/bin/chmod a+rw /sys/devices/platform/ocp/
44e0b000.i2c/i2c-0/0-0050/0-00500/nvmem"
[Install]
WantedBy=multi-user.target
```

Posteriormente, o serviço foi habilitado:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable nvmem.service
$ sudo systemctl start nvmem.service
```

Em seguida, criou-se o serviço para a execução do programa principal do controle de acesso:

```
$ sudo nano /etc/systemd/system/controle-de-acesso.service
```

O seguinte conteúdo foi adicionado:

```
[Unit]
Description=Script que inicia a rotina de controle de acesso
After=mosquitto.service
After=mysql.service
Requires=mosquitto.service
Requires=mysql.service
[Service]
User=debian
ExecStart=/var/lib/cloud9/tcc_caroline/codigo_principal.py
Restart=always
[Install]
WantedBy=multi-user.target
```

Em seguida, transformou-se o arquivo em executável:

```
$ sudo chmod a+rwx /var/lib/cloud9/tcc_caroline/codigo_principal.py
```

Posteriormente, o serviço foi habilitado:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable controle-de-acesso.service
$ sudo systemctl start controle-de-acesso.service
```

7. Página web

A atualização do servidor Apache 2 foi realizada:

```
$ sudo apt install apache2
```

Prosseguiu-se com a instalação do pacote PHP:

```
$ sudo apt install ca-certificates apt-transport-https
$ wget -q https://packages.sury.org/php/apt.gpg -O- | sudo apt-key add -
$ echo "deb https://packages.sury.org/php/ stretch main" | sudo tee /etc/apt/sources.list.d/php.list
$ sudo apt install php7.3
$ sudo apt install php7.3-cli php7.3-common php7.3-curl php7.3-mbstring php7.3-mysql php7.3-xml
```

O servidor Apache foi reiniciado:

```
$ sudo service apache2 restart
```

É possível verificar quais portas TCP estão sendo utilizadas através do comando abaixo:

```
$ sudo netstat -tlpn
```

É necessário escolher um endereço adequado para a rede na qual a BBB está conectada e, em seguida, este endereço deve ser adicionado ao arquivo que contém as configurações de rede da BBB.

```
$ sudo nano /etc/network/interfaces
```

O arquivo interfaces foi alterado através do comando nano acima e o trecho modificado é apresentado abaixo em negrito:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 192.168.0.105
        netmask 255.255.255.0
        gateway 192.168.0.1
        dns-nameservers 8.8.8.8
        dns-nameservers 8.8.4.4

# Example to keep MAC address between reboots
#hwaddress ether DE:AD:BE:EF:CA:FE

##connman: ethX static config
#connmanctl services
```

```
#Using the appropriate ethernet service, tell connman to setup a
static IP ad$
#sudo connmanctl config <service> --ipv4 manual <ip_addr> <netmask>
<gateway>$

##connman: WiFi
#
#connmanctl
#connmanctl> tether wifi off
#connmanctl> enable wifi
#connmanctl> scan wifi
#connmanctl> services
#connmanctl> agent on
#connmanctl> connect wifi_*_managed_psk
#connmanctl> quit

# Ethernet/RNDIS gadget (g_ether)
# Used by: /opt/scripts/boot/autoconfigure_usb0.sh
iface usb0 inet static
    address 192.168.7.2
    netmask 255.255.255.252
    network 192.168.7.0
    gateway 192.168.7.1
```

APÊNDICE B – PROGRAMA PRINCIPAL DO DISPOSITIVO SERVIDOR

```
1.  #!/usr/bin/python3
2.
3.  # Bibliotecas para configuracao de tempo
4.  from time import time, sleep
5.  from datetime import datetime
6.
7.  # Biblioteca para tratamento dos sinais enviados pelo sistema operacional da
    BBB
8.  import signal
9.
10. # Bibliotecas para uso do serial
11. import serial
12. import Adafruit_BBIO.UART as UART
13.
14. # Biblioteca para uso dos pinos da BBB
15. import Adafruit_BBIO.GPIO as GPIO
16.
17. # Biblioteca para execucao de um cliente MQTT
18. import paho.mqtt.client as paho
19.
20. # Biblioteca para conexao ao MySQL
21. import mysql.connector as mysql
22.
23. # Bibliotecas para uso do LCD
24. import board
25. import digitalio
26. import adafruit_character_lcd.character_lcd as characterlcd
27.
28. # CONSTANTES:
29.
30. # DISPOSITIVO
31. BBB_operacao = 1 # 0 para atuar somente como servidor, 1 como srevidor e
    dispositivo de acesso tbm
32. BBB_area = "BlocoM"
33. BBB_sala = "M204"
34.
35. # Configuracao do LCD
36. lcd_columns = 16
37. lcd_rows = 2
38. lcd_d7 = digitalio.DigitalInOut(board.P8_8)
39. lcd_d6 = digitalio.DigitalInOut(board.P8_10)
40. lcd_d5 = digitalio.DigitalInOut(board.P8_12)
41. lcd_d4 = digitalio.DigitalInOut(board.P8_14)
42. lcd_en = digitalio.DigitalInOut(board.P8_16)
43. lcd_rs = digitalio.DigitalInOut(board.P8_18)
44. lcd = characterlcd.Character_LCD_Mono(lcd_rs, lcd_en, lcd_d4, lcd_d5,
    lcd_d6, lcd_d7, lcd_columns, lcd_rows)
45.
46. # Funcao para exibicao de texto no LCD
47. def LCD_txt(frase, espera = 0):
48.     lcd.clear()
```

```
49.      lcd.message = frase
50.      sleep(espera)
51.
52.  LCD_txt('INICIANDO A\nROTINA')
53.
54.  def handle_iterrupt():
55.        LCD_txt('SERVICO PARADO')
56.        sys.exit(0)
57.
58.  signal.signal(signal.SIGTERM, handle_iterrupt)
59.
60.  # Constantes do cliente MQTT
61.  mqtt_broker = "192.168.0.105"
62.  mqtt_port = 1883
63.  mqtt_user = "debian"
64.  mqtt_password = "temppwd"
65.  mqtt_clientID = "M207"
66.  mqtt_sub = "ControleDeAcesso/#"
67.
68.  # Constantes do cliente SQL
69.  sql_user = 'debian'
70.  sql_pwd = 'temppwd'
71.  sql_host = 'localhost'
72.  sql_db = 'Registro'
73.
74.  # Configuracao dos demais pinos LEDS
75.  pino_neg = "P8_15"
76.  pino_lib = "P8_17"
77.  pino_porta = "P8_9"
78.  pino_botao = "P8_7"
79.  GPIO.setup(pino_neg, GPIO.OUT)
80.  GPIO.setup(pino_lib, GPIO.OUT)
81.  GPIO.setup(pino_porta, GPIO.OUT)
82.  GPIO.setup(pino_botao, GPIO.IN, pull_up_down=GPIO.PUD_UP)
83.  estado_porta = GPIO.LOW
84.
85.  # Rotina a ser executada quando uma interrupcao for detectada
86.  def interrupt_callback (pino_botao):
87.      estado_porta = GPIO.input(pino_porta)
88.      estado_porta = GPIO.HIGH if estado_porta == GPIO.LOW else GPIO.LOW
89.      GPIO.output(pino_porta, estado_porta)
90.      return None
91.
92.  # Configuracao do evento de interrupcao do botao
93.  GPIO.add_event_detect(pino_botao, GPIO.FALLING)
94.  GPIO.add_event_callback(pino_botao,
     callback=interrupt_callback,bouncetime=200)
95.
96.  # Inicializacao da porta serial UART1 da BB
97.  UART.setup("UART1")
98.  ser = serial.Serial(port = "/dev/ttyS1", baudrate=9600)
99.  ser.close()
100. ser.open()
```

```
101. sleep(0.2)
102.
103. # Conexao ao banco de dados MySQL
104. db = mysql.connect(user=sql_user, password=sql_pwd, host=sql_host,
     database=sql_db)
105. sql_cursor = db.cursor() # Get a cursor
106.
107. # FUNCOES
108.
109. # Retorna um numero especificado de caracteres do lado esquerdo de uma
     string
110. def left(s, amount):
111.     return s[:amount]
112.
113. # Retorna um numero especificado de caracteres do lado direito de uma string
114. def right(s, amount):
115.     return s[-amount:]
116.
117. # Retorna um numero especificado de caracteres do meio de uma string
118. def mid(s, offset, amount):
119.     return s[offset:offset+amount]
120.
121. # Funcao de leitura e decodificacao da tag RFID
122. def RFID (rfid_payload, formato):
123.     a=datetime.now()
124.     rfid_hex = ''
125.     rfid_dec = ''
126.     rfid_chr = ''
127.     sucesso = 0
128.     for c in rfid_payload:
129.         if rfid_hex == '' :
130.             aux = ''
131.         else:
132.             aux = ' '
133.         rfid_hex = rfid_hex + aux + hex(int(c))
134.         rfid_dec = rfid_dec + aux + str(c)
135.         #rfid_chr = rfid_chr + str(c)
136.     if (rfid_dec[0] == '2' and rfid_dec[len(rfid_dec) - 1] == '3'): # se a
     msg foi recebida propriamente
137.         sucesso = 1
138.         rfid_chr = rfid_payload[1:11]
139.         rfid_chr = rfid_chr.decode('utf-8')
140.         if formato == 'CHR':
141.                 RX_tag_id = str(rfid_chr)
142.         elif formato == 'HEX':
143.                 RX_tag_id = rfid_hex[5:54]
144.         elif formato == 'DEC':
145.                 RX_tag_id = rfid_dec[3:32]
146.         else:
147.             RX_tag_id = 'Erro!'
148.     return sucesso, RX_tag_id
149.
150. # Query de busca de permissao da tag no banco de dados
```

```
151. def sql_busca (RX_tag_id, RX_area, RX_sala):
152.     query_acesso = "SELECT inicio_vigencia, fim_vigencia FROM Acesso \
153.             WHERE tag_id = '" + RX_tag_id + "' AND (area = '" + RX_area + "'
     OR area = '*')  AND (sala = '" + RX_sala + "' OR sala = '*');"
154.     sql_cursor.execute(query_acesso)
155.     res_acesso = sql_cursor.fetchone()
156.     query_cadastro = "SELECT matricula FROM Cadastro WHERE tag_id = '" +
     RX_tag_id + "';"
157.     sql_cursor.execute(query_cadastro)
158.     res_cadastro = sql_cursor.fetchone()
159.     # status_acesso = 0 : com acesso com cadastro
160.     # status_acesso = 1 : com acesso sem cadastro
161.     # status_acesso = 2 : sem acesso com cadastro
162.     # status_acesso = 3 : sem acesso sem cadastro
163.     matricula = str(res_cadastro[0]) if res_cadastro != None else ''
164.     if res_acesso != None :
165.         # formatacao do horario
166.         inicio = datetime.strptime(str(res_acesso[0]), '%d-%m-%Y %H:%M:%S')
167.         final = datetime.strptime(str(res_acesso[1]), '%d-%m-%Y %H:%M:%S')
168.         hoje = datetime.now().replace(microsecond=0)
169.         access = "LIBERADO" if (inicio < hoje and hoje < final) else
     "NEGADO"
170.     else:
171.         access = "NEGADO"
172.
173.     if access == "LIBERADO" and matricula != '': # com acesso e com cadastro
174.         status_acesso = 0
175.         resposta = RX_tag_id + "/" + matricula + "/" + access
176.     elif access == "LIBERADO" and matricula == '': # com acesso sem cadastro
177.         status_acesso = 1
178.         resposta = RX_tag_id + "/TEMPORARIO/" + access
179.     elif access == "NEGADO" and matricula != '': # sem acesso com cadastro
180.         status_acesso = 2
181.         resposta = RX_tag_id + "/" + matricula + "/" + access
182.     elif access == "NEGADO" and matricula == '': # sem cadastro sem acesso
183.         status_acesso = 3
184.         resposta = RX_tag_id + "/SEMACESSO/NEGADO"
185.     return resposta, status_acesso, matricula
186.
187. # Insere a solicitacao de acesso no banco de dados
188. def sql_registro (tempo, RX_tag_id, RX_area, RX_sala, status_acesso,
     RX_porta):
189.     log_entry = "INSERT INTO RegistroLog
     (hora,tag_id,area,sala,acesso,entrada) VALUES(%s, %s, %s, %s, %s, %s);"
190.     val = (tempo, RX_tag_id, RX_area, RX_sala, status_acesso, RX_porta)
191.     sql_cursor.execute(log_entry, val)
192.     db.commit()
193.     return None
194.
195. # Callback executada quando o cliente MQTT se conecta ao servidor
196. def on_connect(client, userdata, flags, rc): #(rc = return code)
197.     if rc==0:
198.         client.subscribe(mqtt_sub)
```

```
199.     else:
200.         # print("Conexao ruim Returned code= ", rc)
201.         pass
202.     return None
203. 
204. # Callback executada quando o cliente publica uma mensagem
205. # def on_publish(client,userdata,result):
206. #     #print("on publish \n")
207. #     return None
208. 
209. # Callback executada quando o cliente MQTT recebe uma mensagem
210. def on_message(client, userdata, msg):
211.     RX_topic = str(msg.topic)
212.     RX_msg = str(msg.payload.decode('utf-8'))
213.     if right(RX_topic,6) == "acesso":
214.         #definicao do topico de resposta: (assunto muda de 'acesso' para
    'permissao')
215.         TX_topic = left(RX_topic, len(RX_topic) - 6) + "permissao"
216.         c0 = RX_topic.find("/area/", 0)
217.         c1 = RX_topic.find("/sala/", c0 + 1) #contagem comeca em 0
218.         RX_area  = mid(RX_topic, c0 + 6 , c1 - (c0 + 6))
219.         RX_sala = mid(RX_topic, c1 + 6, len(RX_topic) - (c1 + 6) - 7)
220.         # segmentacao da resposta
221.         c2 = RX_msg.find("/", 0)
222.         c3 = RX_msg.find("/", c2 + 1)
223.         RX_tag_id = mid(RX_msg, 0, c2)
224.         RX_porta = mid(RX_msg, c2 + 1, c3 - 1 - c2)
225.         RX_tempo = mid(RX_msg, c3 + 1, len(RX_msg) - c3 - 1)
226.         ret3 = sql_busca (RX_tag_id, RX_area, RX_sala)
227.         resposta = str(ret3[0])+ "o"
228.         status_acesso = ret3[1]
229.         # publicar mensagem
230.         mqttClient.publish(TX_topic,resposta)
231.         0 if RX_porta == "P0" else 1
232.         sql_registro (RX_tempo, RX_tag_id, RX_area, RX_sala, status_acesso,
    RX_porta)
233.     return None
234. 
235. # Callback executada quando o cliente MQTT e disconectado do servidor
236. # def on_disconnect(client, userdata, rc):
237. #     # print ("disconneted")
238. #     pass
239. #     return None
240. 
241. # Inicializacao do cliente MQTT
242. mqttClient = paho.Client(mqtt_clientID)
243. # Atribuicao de callbacks
244. mqttClient.on_connect = on_connect
245. mqttClient.on_message = on_message
246. #mqttClient.on_publish = on_publish
247. #mqttClient.on_disconnect = on_disconnect
248. # Conexao ao servidor
249. mqttClient.username_pw_set(mqtt_user, mqtt_password)
```

```
250. mqttClient.connect(mqtt_broker, port=mqtt_port)
251.
252. # Inicio do loop MQTT
253. mqttClient.loop_start()
254.
255. # Rotina executada quando o acesso é liberado
256. def liberar(matricula, pino_neg=pino_neg, pino_lib=pino_lib,
     pino_porta=pino_porta):
257.     estado_porta == GPIO.HIGH
258.     # Acionamento LED
259.     GPIO.output(pino_neg, GPIO.LOW)
260.     GPIO.output(pino_lib, GPIO.HIGH)
261.     # Acionamento do atuador
262.     GPIO.output(pino_porta, GPIO.HIGH)
263.     setenca = 'ACESSO LIBERADO\n' + str(matricula)
264.     LCD_txt(setenca, 3)
265.     GPIO.output(pino_lib, GPIO.LOW)
266.     return estado_porta
267.
268. # Rotina executada quando o acesso é negado
269. def negar(pino_neg=pino_neg, pino_lib=pino_lib, pino_porta=pino_porta):
270.     estado_porta == GPIO.LOW
271.     # Acionamento LED
272.     GPIO.output(pino_lib, GPIO.LOW)
273.     GPIO.output(pino_neg, GPIO.HIGH)
274.     setenca = 'ACESSO NEGADO\n' + str(matricula)
275.     LCD_txt(setenca, 3)
276.     GPIO.output(pino_neg, GPIO.LOW)
277.     return estado_porta
278.
279. # Loop principal do código
280. try:
281.     anterior = ''
282.     if BBB_operacao == 0 :
283.         while True:
284.             pass
285.     elif BBB_operacao == 1:
286.         millis = datetime.now()
287.         while True:
288.             LCD_txt('SALA ' + BBB_sala +'\nAPRESENTE CARTAO')
289.             if ser.isOpen():
290.                 rfid_payload = ser.read(14)
291.                 #tempo = datetime.now().strftime("%d-%m-%Y %H:%M:%S")
292.                 retorno = RFID(rfid_payload,'CHR')
293.                 if retorno[0] == 1: # se sucesso
294.                     RX_tag_id = retorno[1]
295.                     c = datetime.now() - millis
296.                     if RX_tag_id != anterior or (RX_tag_id == anterior and
    c.seconds > 4 ):
297.                         anterior = RX_tag_id
298.                         estado_porta = GPIO.input(pino_porta)
299.                         porta = 'P0' if estado_porta == GPIO.LOW else 'P1'
300.                         ret1 = sql_busca (RX_tag_id, BBB_area, BBB_sala)
```

```
301.                            status_acesso = ret1[1]
302.                            matricula = ret1[2]
303.                            tempo = datetime.now().strftime("%d-%m-%Y %H:%M:%S")
304.                            sql_registro (tempo, RX_tag_id, BBB_area, BBB_sala,
    status_acesso, porta)
305.                            if status_acesso == 0 or status_acesso == 1:
306.                                liberar(matricula)
307.                            elif status_acesso == 2 or status_acesso == 3:
308.                                negar()
309.                            millis = datetime.now()
310.                            ser.reset_input_buffer()
311.                else:
312.                        pass
313.
314.     mqttClient.loop_stop()
315.
316. # Rotina de tratamento caso o programa seja interrompido
317. except KeyboardInterrupt:
318.     LCD_txt("CODIGO PARADO\nREINICIAR!")
319.     GPIO.output(pino_porta, GPIO.LOW)
320.     # print ('KeyboardInterrupt')
```

APÊNDICE C – PROGRAMA PRINCIPAL DO DISPOSITIVO CLIENTE

```
// DECLARAÇÃO DAS BIBLIOTECAS:
#include <UIPEthernet.h>      //Biblioteca para o módulo ENC28J60
#include <PubSubClient.h>     //Biblioteca para o protocolo MQTT
#include <LiquidCrystal.h>    //Biblioteca para o display LCD 16x2
#include <SoftwareSerial.h>   //Biblioteca para a leitura de Serial
#include <RDM6300.h>          //Biblioteca para o módulo RDM6300
#include <Wire.h>             //Biblioteca para o módulo DS1307
#include <RTClib.h>           //Biblioteca para o módulo DS1307

// Inicialização da biblioteca RTC:
RTC_DS1307 RTC;

// Inicialização do cliente Ethernet:
EthernetClient ethClient;

// Atribuição dos endereços MAC e IP ao módulo ENC28J60:
uint8_t myMAC[6] = {0x54, 0x55, 0x58, 0x10, 0x00, 0x24};
IPAddress myip(192, 168, 0, 109);

// Inicialização do cliente MQTT:
IPAddress mqttServer(192, 168, 0, 105);
void callback(char* rx_topic, byte* rx_payload, unsigned int rx_length);
PubSubClient mqttClient(mqttServer, 1883, callback, ethClient);
char MQTT_payload [34];
#define MQTT_CLIENT     "M205"
#define MQTT_USER       "debian"
#define MQTT_PWD        "temppwd"
#define MQTT_PUB        "ControleDeAcesso/area/BlocoM/sala/M205/acesso"
#define MQTT_SUB        "ControleDeAcesso/area/BlocoM/sala/M205/permissao"

// Tempo de espera de resposta a uma solicitação de acesso: 5 s
#define MQTT_WAIT1      5000

// Pinos de LEDs e Botão:
#define PINO_BOTAO     18
#define PINO_PORTA     22
#define PINO_LIBERADO  25
#define PINO_NEGADO    2

// Pinos do LCD:
#define LCD_RS 12
#define LCD_E 11
#define LCD_D4 3
#define LCD_D5 4
#define LCD_D6 5
#define LCD_D7 6
```

```
47.
48.  // Inicialização da biblioteca LiquidCrystal:
49.  LiquidCrystal LCD(LCD_RS, LCD_E, LCD_D4, LCD_D5, LCD_D6, LCD_D7);
50.
51.  // Variáveis de estado do progama (temporizadores, contadores...)
52.  #define WAIT0 0
53.  #define WAIT1 1
54.  int estado = WAIT0;
55.  unsigned long temp_WAIT1;
56.  int cont_espera = 0;
57.  int estado_LCD0 = 1;
58.  int estado_LCD1 = 1;
59.  volatile int estado_porta = LOW;
60.  volatile int estado_porta0 = HIGH;
61.  String porta = "P0";
62.  String TX_tag_id = ""; // numero da tag enviada
63.
64.  // Inicialização da biblioteca SoftwareSerial:
65.  SoftwareSerial RFID(13, 10);
66.  uint8_t RFID_payload[6];
67.
68.  // Inicialização da biblioteca RDM6300:
69.  RDM6300 RDM6300(RFID_payload);
70.
71.  // Inicialização da funçao de exibição do LCD:
72.  void LCDtxt (int x, String matricula = " ");
73.
74.  // Função de callback do cliente MQTT:
75.  void callback(char* rx_topic, byte* rx_payload, unsigned int rx_length) {
76.    //Serial.println ("Callback:");
77.    String str_payload = "";
78.    // A mensagem recebida só será analisada caso o programa esteja a espera
     da resposta de uma solicitação:
79.    if (estado == WAIT1) {
80.      if (strcmp(rx_topic, MQTT_SUB) == 0) {
81.        for (int i = 0; i < rx_length; i++) {
82.          str_payload = str_payload + String((char)rx_payload[i]);
83.        }
84.        // EXEMPLO DE MENSAGEM RECEBIDA: 10BCAE6/20201EN20110/LIBERADO
85.        int cont_barra0 = str_payload.indexOf("/");
86.        int cont_barra1 = str_payload.indexOf("/", cont_barra0 + 1);
87.        String RX_tag_id = str_payload.substring(0, cont_barra0);
88.        String matricula = str_payload.substring(cont_barra0 + 1, cont_barra1
     - 1);
89.        String permissao = str_payload.substring(cont_barra1 + 1, cont_barra1
     + 4);
90.        //Serial.print("Mensagem recebida: ");
91.        //Serial.println(str_payload);
92.
93.        // Caso o número da tag recebida seja idêntico ao número envidado:
```

```
94.         if (RX_tag_id == TX_tag_id) {
95.           if (permissao == "LIBERADO") {
96.             //Serial.println("ACESSO LIBERADO!");
97.             digitalWrite(PINO_LIBERADO, HIGH);
98.             LCDtxt (2, matricula); // Display: ACESSO LIBERADO
99.             estado_porta = HIGH;
100.            digitalWrite(PINO_LIBERADO, LOW);
101.          } else if (permissao == "NEGADO") {
102.            //Serial.println("ACESSO NEGADO!");
103.            digitalWrite(PINO_NEGADO, HIGH);
104.            LCDtxt (3); // Display: ACESSO NEGADO
105.            digitalWrite(PINO_NEGADO, LOW);
106.          }
107.          estado = WAIT0;
108.        }
109.      }
110.    }
111.    cont_espera = 0;
112.    discard();
113. }
114.
115. void setup() {
116.    // Inicialização do LCD:
117.    LCD.begin(16, 2);
118.    LCDtxt(0); // Display: INICIANDO O DISPOSITIVO!
119.
120.    // Inicialização do SerialMonitor:
121.    //Serial.begin(9600);
122.
123.    // Configuração I/O dos pinos:
124.    pinMode(PINO_NEGADO, OUTPUT); //led vermelho
125.    pinMode(PINO_LIBERADO, OUTPUT); //led ver
126.    pinMode(PINO_PORTA, OUTPUT); //led ver
127.    pinMode(PINO_BOTAO, INPUT_PULLUP); //led ver
128.
129.    // Configuração do interrupção:
130.    attachInterrupt(digitalPinToInterrupt(PINO_BOTAO), interrupcao, FALLING);
131.
132.    // Inicialização do módulo RTC:
133.    Wire.begin();
134.    RTC.begin();
135.
136.    // Inicialização do serial do RFID:
137.    RFID.begin(9600);
138.
139.    // Conexão do cliente Ethernet:
140.    Ethernet.begin(myMAC, myip);
141.
142.    // Conexão ao servidor MQTT:
```

```
143.   mqtt_connexao();
144. }
145.
146. void loop() {
147.   // Exibe a mensagem padrão do sistema:
148.   if (estado_LCD0 != estado_LCD1) {
149.     LCDtxt (1); // Display: APRESENTE CARTAO
150.   }
151.
152.   if (estado_porta0 != estado_porta) {
153.     digitalWrite(PINO_PORTA, estado_porta);
154.   }
155.
156.   // Caso o programa esteja a espera da leitura de uma nova tag:
157.   if (estado == WAIT0) {
158.     estado_LCD1 = 0;
159.
160.     // Verifica se uma tag é apresentada:
161.     if (RFID.available()) {
162.       LCDtxt(5); // Display: AGUARDE...
163.       uint8_t c = RFID.read();
164.
165.       // Decodificação da tag:
166.       if (RDM6300.decode(c)) {
167.         String str_msg = "";
168.         String str_aux = "";
169.         for (int i = 0; i < 5; i++) {
170.           if (String(RFID_payload[i], HEX).length() == 1) {
171.             str_aux = "0" + String(RFID_payload[i], HEX);
172.           } else {
173.             str_aux = String(RFID_payload[i], HEX);
174.           }
175.           str_msg = str_msg + str_aux;
176.         }
177.
178.         // Horário:
179.         String horario = rtc_tempo("DD-MM-YYYY hh:mm:ss");
180.
181.         // Verifica se a porta está fechada ou aberta:
182.         if (digitalRead(PINO_PORTA) == LOW) {
183.           porta = "P0";
184.         } else {
185.           porta = "P1";
186.         }
187.
188.         // EXEMPLO DE ENVIO: 01000BCAE6/P0/10-10-2020 10:10:10
189.
190.         // Formatação da mensagem:
191.         TX_tag_id = str_msg;
```

```
192.          str_msg.concat("/");
193.          str_msg.concat(porta);
194.          str_msg.concat("/");
195.          str_msg.concat(horario);
196.          str_msg.toUpperCase();
197.
198.          // Conversão de string para char:
199.          str_msg.toCharArray(MQTT_payload, sizeof(MQTT_payload));
200.
201.          // Verifica se o cliente está conectado e envia a mensagem de
      solicitação:
202.          while (!mqttClient.connected()) {
203.            mqtt_connexao();
204.          }
205.          while (!mqttClient.publish(MQTT_PUB, MQTT_payload)) {
206.            mqtt_connexao();
207.            mqttClient.publish(MQTT_PUB, MQTT_payload);
208.          }
209.          //Serial.print("Mensagem enviada: ");
210.          //Serial.println(MQTT_payload);
211.
212.          estado = WAIT1;
213.          cont_espera = 0;
214.          temp_WAIT1 = millis();
215.        }
216.      }
217.    }
218.
219.    // Verifica se o programa está a espera da resposta de uma solicitação:
220.    if (estado == WAIT1) {
221.      if (millis() - temp_WAIT1 > MQTT_WAIT1) {
222.        if (cont_espera == 0) {
223.          //Serial.println("Tente novamente!");
224.          cont_espera = 1;
225.          estado = WAIT0;
226.          LCDtxt (4); // Display: TENTE NOVAMENTE
227.          delay(1500);
228.          LCD.clear();
229.          discard();
230.        }
231.      }
232.    }
233.    mqttClient.loop();
234. }
235.
236. // Função para exibição de texto no display LCD:
237. void LCDtxt (int x, String matricula) {
238.    estado_LCD0 = estado_LCD1;
239.    LCD.clear();
```

```
240.   LCD.setCursor(0, 0);
241.   switch (x) {
242.     case 0:
243.       estado_LCD1 = 0;
244.       LCD.print("INICIANDO O");
245.       LCD.setCursor(0, 1);
246.       LCD.print("DISPOSITIVO!");
247.       break;
248.     case 1:
249.       estado_LCD1 = 1;
250.       LCD.print("SALA M205");
251.       LCD.setCursor(0, 1);
252.       LCD.print("APRESENTE CARTAO");
253.       break;
254.     case 2:
255.       estado_LCD1 = 2;
256.       LCD.print("ACESSO LIBERADO");
257.       LCD.setCursor(0, 1);
258.       LCD.print(matricula);
259.       delay(2000);
260.       break;
261.     case 3:
262.       estado_LCD1 = 3;
263.       LCD.print("ACESSO NEGADO");
264.       delay(2000);
265.       break;
266.     case 4:
267.       estado_LCD1 = 4;
268.       LCD.print("TENTE NOVAMENTE");
269.       break;
270.     case 5:
271.       estado_LCD1 = 5;
272.       LCD.print("AGUARDE...");
273.       break;
274.   }
275. }
276.
277. // Função para descartar dados seriais após a leitura de uma tag:
278. void discard() {
279.   while (RFID.available()) {
280.     uint8_t lixo = RFID.read();
281.   }
282.   memset(RFID_payload, 0, sizeof(RFID_payload));
283. }
284.
285. // Rotina de interrupção do botao:
286. void interrupcao() {
287.   estado_porta0 = estado_porta;
288.   estado_porta = !estado_porta;
```

```
289. }
290.
291. // Função para obter o horário do RTC:
292. String rtc_tempo(String rtc_str) {
293.   DateTime now = RTC.now();
294.   char format[20];
295.   rtc_str.toCharArray(format, 20);
296.   rtc_str = now.toString(format);
297.   return rtc_str;
298. }
299.
300. // Função de nova conexão ao servidor MQTT:
301. void mqtt_connexao() {
302.   while (!mqttClient.connected()) {
303.     //Serial.println("Conectando ao servidor MQTT... ");
304.     if (mqttClient.connect(MQTT_CLIENT, MQTT_USER, MQTT_PWD)) {
305.       //Serial.println("Conectado!");
306.       mqttClient.subscribe(MQTT_SUB);
307.     } else {
308.       //Serial.print("Falha! Estado: ");
309.       //Serial.println(mqttClient.state());
310.       delay(1000);
311.     }
312.   }
313. }
```

APÊNDICE D – QUADROS DE CONEXÃO

Este apêndice apresenta os quadros que reúnem as conexões dos entre as placas microcontroladoras e os módulos periféricos, para ambos os dispositivos.

1. Dispositivo Servidor

Componente	Pino componente		Pino BBB	
	Número	Função	Número	Função
RDM6300	1	TX	P9_26	UART1_RXD
	4	GND	P9_1	DGND
	5	VCC	P9_5	VDD_5V
Display LCD 16x2	1	Vss	P9_1	DGND
	2	Vdd	P9_5	VDD_5V
	3	V0	P9_1	DGND
	4	RS	P8_18	GPIO_65
	5	RW	P9_1	DGND
	6	E	P8_16	GPIO_46
	11	D4	P8_14	GPIO_26
	12	D5	P8_12	GPIO_44
	13	D6	P8_10	GPIO_68
	14	D7	P8_8	GPIO_67
	15	A	P9_5	VDD_5V
	16	K	P9_1	DGND
RTC	1	SCL	P9_19	I2C2_SCL
	2	SDA	P9_20	I2C2_SDA
	3	VCC	P9_5	VDD_5V
	4	GND	P9_1	DGND
LED STATUS	R		P8_15	GPIO_47
	G		P8_17	GPIO_45
	B		P8_11	GPIO_27
LED PORTA	(+)		P8_9	GPIO_69
BOTAO	(+)		P8_7	GPIO_66

2. Dispositivo Cliente

Componente	Pino componente		Pino Arduino Mega
	Número	Função	
RDM6300	1	TX	13
	4	GND	GND
	5	VCC	5V
Display LCD 16x2	1	Vss	GND
	2	Vdd	5V
	3	V0	GND
	4	RS	12
	5	RW	GND
	6	E	11
	11	D4	3
	12	D5	4
	13	D6	5
	14	D7	6
	15	A	5V
	16	K	GND
ENC28J60	CS		53
	SI		51
	SO		50
	SCK		52
	RESET		RESET
	INT		D2
	VCC		3V3
	GND		GND
RTC	SCL		21
	SCA		20
	VCC		5V
	GND		GND
LED STATUS	R		24

	G	25
	B	26
LED PORTA	(+)	22
BOTAO	(+)	18

ANEXO A – PROGRAMA DE CONFIGURAÇÃO RTC NTP DO DISPOSITIVO CLIENTE

```
#include <EtherCard.h>
#include <Wire.h>
#include "RTClib.h"  // Credit: Adafruit

RTC_DS1307 RTC;

static byte ntpServer[] = {194,109,6,2}; //ntp.xs4all.nl but you can cange in any public ntp server.
uint8_t ntpMyPort = 123;
long timeZoneOffset = -10800L; //
boolean networkConnection = true;
static byte mymac[] = {0x54, 0x55, 0x58, 0x10, 0x00, 0x24};
byte Ethernet::buffer[700];

void setup () {
  Serial.begin(9600);

  // Instantiate the RTC
  Wire.begin();
  RTC.begin();

  // Check if the RTC is running.
  if (! RTC.isrunning()) {
    Serial.println("RTC is NOT running");
  } else {
    Serial.println("RTC present");
  }

  Serial.println(F("\n[testDHCP]"));

  Serial.print("MAC: ");
  for (byte i = 0; i < 6; ++i) {
    Serial.print(mymac[i], HEX);
    if (i < 5)
      Serial.print(':');
  }
  Serial.println();
```

```
int i = 1;
int DHCP = 0;
 //Try to get dhcp settings 3 times before giving up
Serial.println("Resolve dhcp ...");

while( DHCP == 0 && i < 3){
  Serial.print("Trying to resolve DHCP attempt nr. ");
  Serial.println(i);
  delay(1000);
  DHCP = ether.begin(sizeof Ethernet::buffer, mymac);
  i++;
  }

//Serial.println(F("Setting up DHCP"));
if (!ether.dhcpSetup()) {
  Serial.println(F("DHCP failed"));
  networkConnection = false;
  } else {
    Serial.println(F("DHCP SUCCESS"));
  }

ether.printIp("My IP: ", ether.myip);
ether.printIp("Netmask: ", ether.netmask);
ether.printIp("GW IP: ", ether.gwip);
ether.printIp("DNS IP: ", ether.dnsip);

//you can output this to lcd, oled or 7-segment
uint8_t  *ipPtr = ether.myip;
String ip0 = String(ipPtr[0], DEC);
String ip1 = String(ipPtr[1], DEC);
String ip2 = String(ipPtr[2], DEC);
String ip3 = String(ipPtr[3], DEC);


//Get time from DS3231
DateTime now = RTC.now();
Serial.println();
unsigned long epoch  = now.unixtime();

//If network connection is true, sync with ntp server
```

```
  if (networkConnection) {
    Serial.println(F("TEMPO DA INTERNET"));
}
epoch  = getNtpTime();
  Serial.println("ntptime:");
  Serial.println(epoch);

  if (epoch == 0) {
    Serial.println("tempo do rtc:");
  DateTime now = RTC.now();
epoch  = now.unixtime();
} //sync failed, use DS3231 time instead

  //Check summer / winter time
  if (adjustDstEurope()) {epoch = epoch + 3600; timeZoneOffset = 7200L;}
  RTC.adjust(epoch);
  now = RTC.now();
  Serial.println("epoch:");
  Serial.println(epoch);
} //End of setup

unsigned long getNtpTime() {
  unsigned long timeFromNTP;
  const unsigned long seventy_years = 2208988800UL;
  int i = 60; //Number of attempts
  Serial.println("NTP request sent");
  while(i > 0) {
  ether.ntpRequest(ntpServer, ntpMyPort);
      Serial.print("."); //Each dot is a NTP request
      word length = ether.packetReceive();
      ether.packetLoop(length);
      if(length > 0 && ether.ntpProcessAnswer(&timeFromNTP,ntpMyPort)) {
        Serial.println();
        Serial.println("NTP reply received");
        return timeFromNTP - seventy_years + timeZoneOffset;
      }
      delay(500);
      i--;
    }
  Serial.println();
  Serial.println("NTP reply failed");
```

```
  return 0;
}

boolean adjustDstEurope()
{
  // Get the current time
  DateTime now = RTC.now();
  // last sunday of march
  int beginDSTDate= (31 - (5* now.year() /4 + 4) % 7);
  int beginDSTMonth=3;
  //last sunday of october
  int endDSTDate= (31 - (5 * now.year() /4 + 1) % 7);
  int endDSTMonth=10;
  // DST is valid as:
  if (((now.month() > beginDSTMonth) && (now.month() < endDSTMonth))
    || ((now.month() == beginDSTMonth) && (now.day() >= beginDSTDate))
    || ((now.month() == endDSTMonth) && (now.day() < endDSTDate)))
    return true; // DST europe = GMT +2
  else return false; // nonDST europe = GMT +1
}

void loop () {
  // Get the current time
  DateTime now = RTC.now();

  unsigned long newHour = now.hour();
  unsigned long newMinute = now.minute();
  unsigned long newSecond = now.second();
  unsigned long newYear = now.year();
  unsigned long newMonth = now.month();
  unsigned long newDay = now.day();

  Serial.print("Hour ");
  Serial.print(newHour);
    Serial.print(" minute ");
  Serial.print(newMinute);
    Serial.print(" second ");
  Serial.print(newSecond);
    Serial.print(" year ");
  Serial.print(newYear);
    Serial.print(" month ");
```

```
  Serial.print(newMonth);
    Serial.print(" day ");
  Serial.print(newDay);

  int hour1    = (newHour%10);
  int hour10   = (newHour/10);
  int minute1  = (newMinute%10);
  int minute10 = (newMinute/10);
  int second1  = (newSecond%10);
  int second10 = (newSecond/10);
  int year1    = (newYear%10);
  int year10   = ((newYear/10)%10);
  int year100  = ((newYear/100)%10);
  int year1000 = (newYear/1000);
  int month1   = (newMonth%10);
  int month10  = (newMonth/10);
  int day1     = (newDay%10);
  int day10    = (newDay/10);
  delay(1000);
  }
```