

MONITOR DE FLUXO DE PESSOAS

CESAR School

MONITOR DE FLUXO DE PESSOAS

Desenvolvido por:

- André Luiz Alves de Sousa (alas@cesar.school)
- Henrique Cordeiro Pereira (hcp@cesar.school)
- Leonardo Menezes Soares de Azevedo (lmsa@cesar.school)
- Luiza Omena Suassuna (los2@cesar.school)

Recife, 05 de junho de 2025

1. INTRODUÇÃO

1.1 Contexto

O monitoramento de fluxo de pessoas em ambientes diversos é uma necessidade crescente em diversos setores, desde estabelecimentos comerciais até instituições acadêmicas. A capacidade de coletar, processar e visualizar dados sobre a movimentação de pessoas proporciona insights valiosos para otimização de recursos, planejamento de espaços e tomada de decisões estratégicas.

1.2 Motivação

A motivação para o desenvolvimento deste projeto surge da necessidade de criar uma solução de IoT acessível e eficiente para monitoramento de fluxo de pessoas. Utilizando componentes de baixo custo e tecnologias abertas, buscamos demonstrar como é possível implementar um sistema completo de coleta, transmissão e visualização de dados em tempo real.

1.3 Objetivos

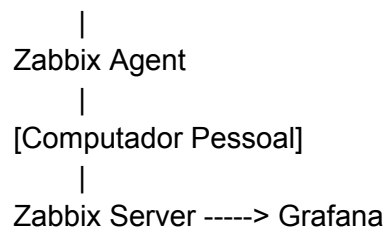
O objetivo principal deste trabalho é desenvolver um sistema de monitoramento de fluxo de pessoas utilizando dispositivos IoT conectados via protocolo MQTT. Os objetivos específicos incluem:

- Implementar a coleta de dados através de sensores conectados a microcontrolador ESP8266
- Estabelecer comunicação via protocolo MQTT entre dispositivos
- Configurar um broker MQTT para gerenciamento das mensagens
- Integrar o sistema com ferramentas de monitoramento e visualização (Zabbix e Grafana)
- Validar o funcionamento completo da solução através de testes práticos

2. METODOLOGIA

2.1 Diagrama do Sistema

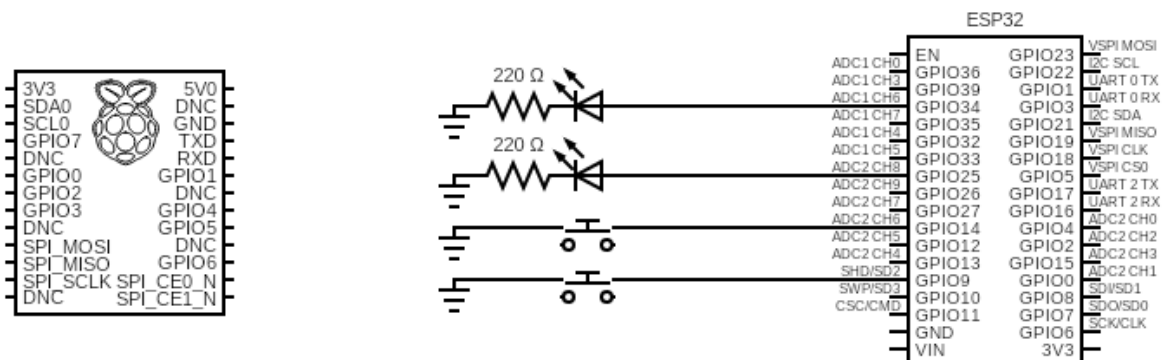
[ESP8266 + Sensores] ----WiFi/MQTT----> [Raspberry Pi 3 + Mosquitto Broker]



2.2 Lista de Hardware e Software

Hardware utilizado:

- 1x Raspberry Pi 3
- 1x ESP8266
- 2x Push-down buttons
- 2x LEDs
- 2x Resistores
- 7x Jumper cables



Software utilizado:

- Mosquitto MQTT Broker (Raspberry Pi)
- Zabbix Agent (Raspberry Pi)
- Zabbix Server (Computador pessoal)
- Grafana (Visualização de dados)
- VSCode e PlatformIO (Programação ESP8266)

2.3 Fluxo de Comunicação

O sistema implementa uma arquitetura distribuída baseada no protocolo MQTT sobre rede Wi-Fi. O fluxo de comunicação segue as seguintes etapas:

1. **Coleta de Dados:** A ESP8266 monitora continuamente os sensores (push-down buttons) para detectar eventos de passagem de pessoas
2. **Publicação MQTT:** Os dados coletados são publicados no tópico "monitor/traffic-flow" através do broker MQTT
3. **Recepção e Processamento:** A Raspberry Pi 3, atuando como broker Mosquitto, recebe e processa as mensagens MQTT
4. **Transmissão para Monitoramento:** O Zabbix Agent instalado na Raspberry Pi coleta os dados processados e os envia para o Zabbix Server
5. **Visualização:** O Zabbix Server transfere os dados para o Grafana através de itens dependentes, permitindo a visualização em dashboards interativos

Ambas as placas (ESP8266 e Raspberry Pi 3) operam na mesma rede Wi-Fi, garantindo a conectividade necessária para a comunicação MQTT.

3. RESULTADOS

3.1 Implementação do Sistema

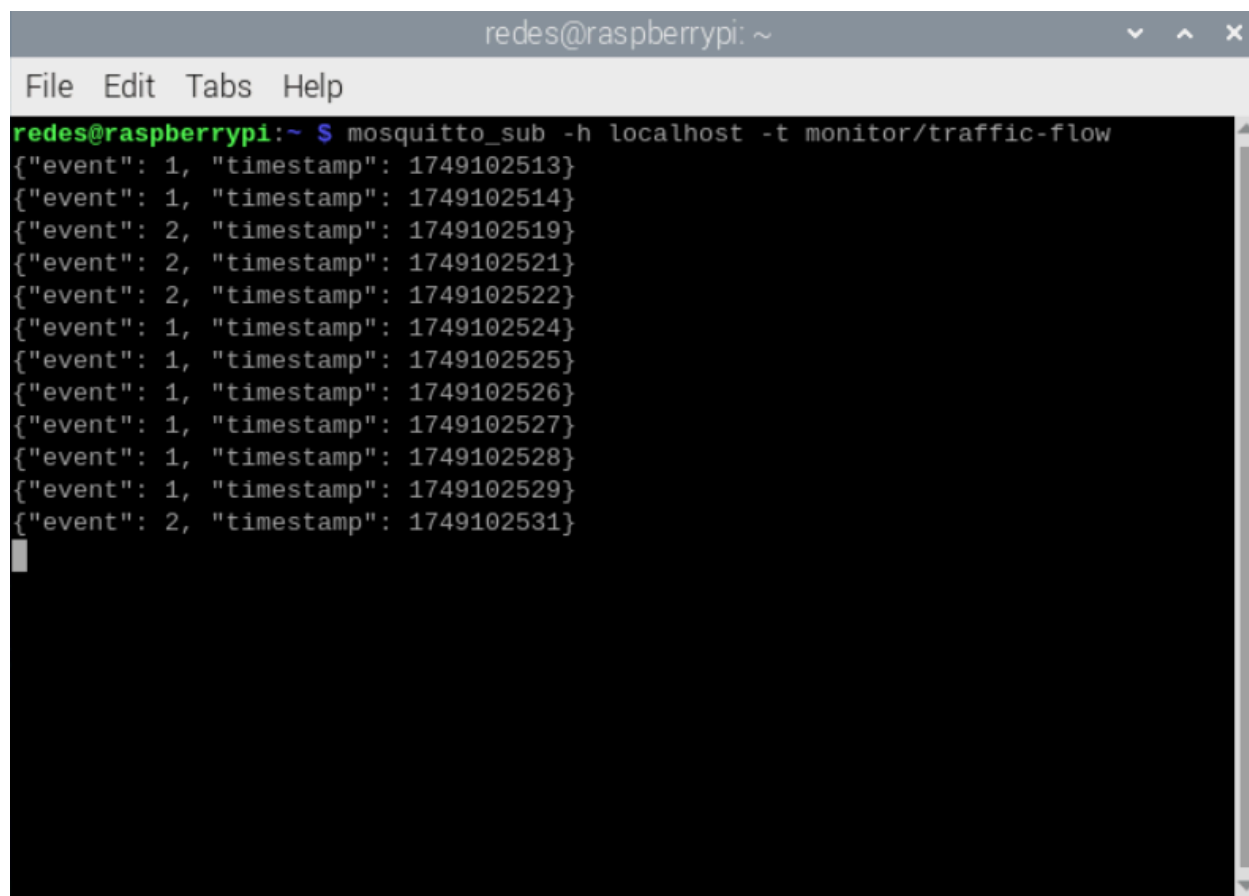
O sistema foi implementado com sucesso, demonstrando a viabilidade da solução proposta. A ESP8266 conseguiu estabelecer conexão estável com a rede Wi-Fi e manter comunicação contínua com o broker MQTT na Raspberry Pi.

3.2 Coleta e Transmissão de Dados

Os dados foram enviados ao Zabbix, onde eram armazenados e pré-processados

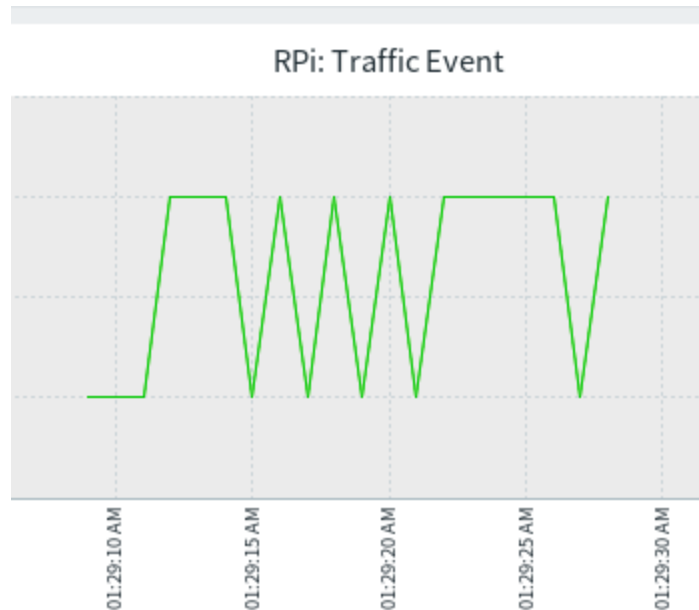
Name ▲	Triggers	Key
MQTT Traffic Flow Data (Raw)		mqtt.get["tcp://192.168.0.54:1883","monitor/traffic-flow"]
MQTT Traffic Flow Data (Raw): Traffic Event		event
MQTT Traffic Flow Data (Raw): Traffic Timestamp		timestamp

Itens no Zabbix

A terminal window titled 'redes@raspberrypi: ~' with a menu bar containing 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows the command 'mosquitto_sub -h localhost -t monitor/traffic-flow' being executed. Below the command, a series of JSON messages are displayed, each containing an 'event' number and a 'timestamp'. The messages are: {"event": 1, "timestamp": 1749102513}, {"event": 1, "timestamp": 1749102514}, {"event": 2, "timestamp": 1749102519}, {"event": 2, "timestamp": 1749102521}, {"event": 2, "timestamp": 1749102522}, {"event": 1, "timestamp": 1749102524}, {"event": 1, "timestamp": 1749102525}, {"event": 1, "timestamp": 1749102526}, {"event": 1, "timestamp": 1749102527}, {"event": 1, "timestamp": 1749102528}, {"event": 1, "timestamp": 1749102529}, {"event": 2, "timestamp": 1749102531}.

```
redes@raspberrypi: ~  
File Edit Tabs Help  
redes@raspberrypi:~$ mosquitto_sub -h localhost -t monitor/traffic-flow  
{"event": 1, "timestamp": 1749102513}  
{"event": 1, "timestamp": 1749102514}  
{"event": 2, "timestamp": 1749102519}  
{"event": 2, "timestamp": 1749102521}  
{"event": 2, "timestamp": 1749102522}  
{"event": 1, "timestamp": 1749102524}  
{"event": 1, "timestamp": 1749102525}  
{"event": 1, "timestamp": 1749102526}  
{"event": 1, "timestamp": 1749102527}  
{"event": 1, "timestamp": 1749102528}  
{"event": 1, "timestamp": 1749102529}  
{"event": 2, "timestamp": 1749102531}
```

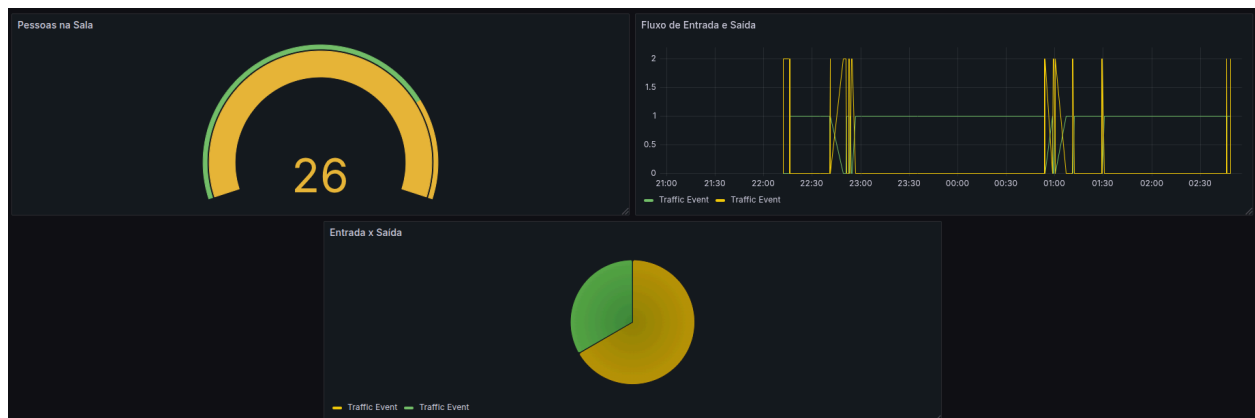
Mensagens recebidas em um *subscriber* na Raspberry PI



Gráficos no Zabbix

3.3 Visualização e Monitoramento

A integração com Zabbix e Grafana proporcionou uma interface de visualização robusta e em tempo real. Os dashboards criados permitem acompanhar o fluxo de pessoas através de diversos gráficos.



4. CONCLUSÃO

4.1 Desafios Encontrados

Durante o desenvolvimento do projeto, alguns desafios foram identificados e superados. A configuração inicial da comunicação MQTT entre os dispositivos exigiu ajustes nos parâmetros

de rede e autenticação. A integração entre Zabbix Agent e o Zabbix Server demandou configurações específicas para garantir a coleta adequada dos dados MQTT.

4.2 Aprendizados Obtidos

O projeto proporcionou conhecimentos valiosos sobre arquiteturas IoT, protocolos de comunicação M2M (Machine-to-Machine) e integração de sistemas de monitoramento. A experiência prática com MQTT demonstrou sua eficácia para aplicações IoT, especialmente em cenários que demandam comunicação assíncrona e baixo consumo de recursos.

4.3 Melhorias Futuras

Para versões futuras do sistema, sugerem-se as seguintes melhorias:

- Implementação de sensores mais sofisticados
- Desenvolvimento de algoritmos de machine learning para análise preditiva do fluxo de pessoas
- Criação de triggers para monitoramento remoto
- Implementação de sistema de alertas via email ou SMS

O projeto demonstrou com sucesso a viabilidade de implementar soluções IoT acessíveis para monitoramento de fluxo de pessoas, utilizando tecnologias abertas e componentes de baixo custo. A arquitetura desenvolvida serve como base sólida para expansões futuras e adaptações para diferentes cenários de aplicação.

5. APÊNDICE

5.1 Código ESP8266

```
#include <ctime>
#include <time.h>
#include <sys/time.h>
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
```

```
struct MQTTPayload
{
    int event;
    std::time_t timestamp;
};
```

```
MQTTPayload constructPayload(int event);
void publishEvent(int event);
```

```

void connectMQTT();
void setupWiFi();
void handleEnter();
void handleExit();

const char *ssid = "uaifai-tiradentes";
const char *password = "bemvindoaoacesar";

const char *mqtt_server = "172.17.0.1";
const int mqtt_port = 1883;
const char *mqtt_topic = "monitor/traffic-flow";
const char *mqtt_client_id = "esp8266-traffic-monitor";

const int BUTTON_1_PIN = D3; // GPIO 0
const int BUTTON_2_PIN = D5; // GPIO 14
const int LED_1_PIN = D6; // GPIO 12
const int LED_2_PIN = D7; // GPIO 13

const int ENTER = 1;
const int EXIT = 2;

WiFiClient espClient;
PubSubClient client(espClient);

volatile unsigned long lastPress1 = 0;
volatile unsigned long lastPress2 = 0;

MQTTPayload constructPayload(int event)
{
    MQTTPayload payload;
    payload.event = event;
    payload.timestamp = time(nullptr);
    return payload;
}

void publishEvent(int event)
{
    MQTTPayload payload = constructPayload(event);

    char msg[128];
    snprintf(msg, sizeof(msg), "{\"event\": %d, \"timestamp\": %lld}", payload.event,
    payload.timestamp);

    client.publish(mqtt_topic, msg);

```

```
}
```

```
void connectMQTT()
```

```
{  
  while (!client.connected())  
  {  
    Serial.print("Connecting to MQTT...");  
    if (client.connect(mqtt_client_id))  
    {  
      Serial.println("connected");  
    }  
    else  
    {  
      Serial.print("failed, rc=");  
      Serial.print(client.state());  
      delay(1000);  
    }  
  }  
}
```

```
void setupWiFi()
```

```
{  
  delay(10);  
  Serial.println();  
  Serial.print("Connecting to ");  
  Serial.println(ssid);  
  WiFi.begin(ssid, password);  
  while (WiFi.status() != WL_CONNECTED)  
  {  
    delay(500);  
    Serial.print(".");  
  }  
  Serial.println("WiFi connected");  
}
```

```
void setupTime()
```

```
{  
  configTime(0, 0, "pool.ntp.org", "time.nist.gov");  
  
  Serial.print("Waiting for NTP sync...");  
  while (time(nullptr) < 100000)  
  {  
    delay(500);  
    Serial.print(".");  
  }  
}
```



```
}  
Serial.println("NTP time acquired.");  
}
```

```
void IRAM_ATTR handleEnter()  
{  
  if (millis() - lastPress1 > 300)  
  {  
    lastPress1 = millis();  
    Serial.println("Entrada detectada.");  
    digitalWrite(LED_1_PIN, HIGH);  
    publishEvent(ENTER);  
    delay(500);  
    digitalWrite(LED_1_PIN, LOW);  
  }  
}
```

```
void IRAM_ATTR handleExit()  
{  
  if (millis() - lastPress2 > 300)  
  {  
    lastPress2 = millis();  
    Serial.println("Saída detectada.");  
    digitalWrite(LED_2_PIN, HIGH);  
    publishEvent(EXIT);  
    delay(500);  
    digitalWrite(LED_2_PIN, LOW);  
  }  
}
```

```
void setup()  
{  
  Serial.begin(115200);  
  pinMode(BUTTON_1_PIN, INPUT_PULLUP);  
  pinMode(BUTTON_2_PIN, INPUT_PULLUP);  
  pinMode(LED_1_PIN, OUTPUT);  
  pinMode(LED_2_PIN, OUTPUT);  
  
  setupWiFi();  
  setupTime();  
  client.setServer(mqtt_server, mqtt_port);  
  connectMQTT();  
  
  attachInterrupt(digitalPinToInterrupt(BUTTON_1_PIN), handleEnter, FALLING);
```

```
attachInterrupt(digitalPinToInterrupt(BUTTON_2_PIN), handleExit, FALLING);

Serial.println("Sistema de monitoramento iniciado.");
}

void loop()
{
  if (!client.connected())
  {
    connectMQTT();
  }
  client.loop();
}
```

5.2 Configurações do Broker MQTT (Raspberry Pi)

Instalação do Mosquitto:

```
sudo apt update
sudo apt install mosquitto mosquitto-clients
```

Configuração do arquivo /etc/mosquitto/mosquitto.conf:

```
listener 1883
allow_anonymous true
persistence true
persistence_location /var/lib/mosquitto/
log_dest file /var/log/mosquitto/mosquitto.log
```

Comandos para iniciar o serviço:

```
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
```

5.3 Configuração do Zabbix Agent

Arquivo de configuração /etc/zabbix/zabbix_agentd.conf:

```
Server=<Ip de rede do PC contendo o Zabbix Server>
ServerActive=<Ip de rede do PC contendo o Zabbix Server>
```

Hostname=RPi

Reiniciar o serviço:

```
sudo systemctl restart zabbix-agent
```