Sistema de Detecção de Vazamentos de Fluídos

Autores: Jonatas da Silva, Thiago Sassaki, Vitor Lopes Data: 05/2025  
Git: <https://github.com/FiredDurden/FiredDurden-SistemaDeteccaoVazamentoFluidos.git>

# Objetivo

Propor um método de detecção de vazamento de fluídos em imóveis (sejam eles residenciais, comerciais, industriais etc..) que tome proveito das possibilidades abertas com a Internet das Coisa (IoT – Internet Of Things) e protocolo MQTT.

O projeto foi criado inicialmente para detecção de vazamentos de água, no entanto a abordagem torna possível também adaptá-lo para a detecção de vazamentos de gás: é só uma questão de substituição do sensor de fluxo para o que mais se adeque ao fluído a ser monitorado.

# Abordagem do problema e Solução Proposta

Abordamos esse problema como uma questão de lógica booleana: aqui os sensores de fluxo de água são classificados de duas formas:

* Sensores **fornece**: representa sensores que verificam se há vazão nas principais admissões de água da residência. Um exemplo é o ponto em que o encanamento residencial recebe a água do encanamento de distribuição da rua;
* Sensor **consome:** são sensores que verificam se há vazão nos pontos de consumo de água do imóvel. Exemplos são torneiras e descargas.
* Cada sensor **consome** deve estar associado a um sensor **fornece**;

Consideramos haver vazamento de água **SE** em um conjunto **fornece-consome** nenhum sensor **consome** registrar fluxo de água **E** o sensor **fornece** registrar o fluxo de água.

Vale observar que, pela abordagem booleana, essa solução não tem o intuito de indicar a vazão do vazamento, mas sim somente sinalizar a existência do vazamento.

# Materiais

|  |  |
| --- | --- |
| **COMPONENTE** | **DESCRIÇÃO** |
| **ESP32** | Microcontrolador com entrada para sensores (General Input Output Interfaces - **GPIO**) e módulo WiFi. É responsável pela monitoração dos sensores e também é o MQTT Publisher na solução. |
| **Sensor YF-S201** | Sensor de fluxo de água conectado às GPIOs do ESP32. Sinaliza presença do fluxo de água. |
| **Diodo LED** | Indicação visual de vazamento que não depende da internet. |
| **Moskitto MQTT Broker** | MQTT Broker para repasse de mensagens entre MQTT Publhiser e os MQTT Subscribers. |
| **Network Time Protocol Server (NTP)** | Fornece data e hora) que será parte das mensagens em publicações MQTT. |

# Protocolo MQTT

O Message Queuing Telemetry Transport (MQTT) (MQTT ORG, 1999) é um protocolo mantido pela Organization for Advancements of Structured Information Standars (OASIS, 1993) e desenvolvido para fornecer comunicação de forma eficiente, sem desperdício de banda de dados.

A tabela abaixo apresenta um resumo dos jargões comuns dessa tecnologia que são necessários para a compreensão deste documento. Para maiores detalhes sobre o funcionamento do protocolo

|  |  |
| --- | --- |
| **JARGÃO** | **DESCRIÇÃO** |
| **topic** | é o assunto (tópico) que receberá as mensagens publicadas |
| **MQTT Publisher** | MQTT Client que publica a mensagem em um topic para um MQTT Broker |
| **MQTT Subscriber** | MQTT Client que se se inscreve(assina) em um MQTT Broker para receber mensagens sobre um topic. |
| **MQTT Broker** | MQTT Server que recebe publicações dos publishers e as repassa para os subscribers. Vale lembrar que topics não são criados pelo Broker: quem cria os topics no MQTT Server são os publishers e os subscribers, via mensagens de publicação e assinatura respectivamente |

Não entraremos nos detalhes do funcionamento do MQTT pois esse documento é focado na solução proposta. Detalhes do protocolo estão amplamente disponíveis na Internet.

# Protocolo NTP

O Network Time Protocol é um protocolo de sincronismo de data e hora: nele NTP clients se conectam a NTP Servers que fornecem essas informações.

O NTP não é necessário para as funcionalidades principais da solução: nesse caso o usamos somente para que a mensagem publicada pelo ESP32 também tenha a data e a hora da detecção da falha.

Não entraremos nos detalhes do funcionamento do NTP pois esse documento é focado na solução proposta. Detalhes do protocolo estão amplamente disponíveis na Internet.

# Ambientes de Desenvolvimento (IDE)

O ambiente de desenvolvimento referência para controladores como o ESP32 ou Arduino é o Arduino IDE, que fornece suporte nativo a C/C++ além de conter bibliotecas específicas para interagir com controladores e sensores, e até mesmo com o IoT em mente.

Por isso qualquer IDE em compliance com o Arduino IDE é válida para a programação. Exemplos são o Arduino IDE, Platform IO, ESP-IDF. Ambientes virtualizados, como o Wokwi, também possuem sua própria IDE (Wokwi IDE), que também é compatível com o Arduino IDE.

# Funcionamento da Solução

1. Cada pontos chave em que há fluxo de água é classificado como **fornece** ou **consome**, com cada um recebendo um sensor;
2. Internamente ao sensor de fluxo de vazão de água existem pás que giram quando submetidas à água corrente: essas pás são ligadas a um minigerador interno ao próprio sensor que transforma a energia fluvial em energia elétrica;
   1. - Se não há fluxo de água as pás, não gerando tensão (0V, ou LOW, nível 0).
   2. - Se há fluxo de água essas pás giram, gerando uma tensão elétrica que pode variar a depender do fluxo da água (entre 5V e 35V).
3. Essa tensão elétrica é baixada para garantir que cheguem somente 3.3V (HIGH, ou nível lógico 1) ao GPIO no ESP32;
4. O ESP32, via programação, tem a associação dos GPIOs aos sensores **fornece** ou **consome**, e constantemente monitora o status de cada um deles (0 ou 1);
5. Com base no nível de tensão nos GPIOs (LOW ou High), a programação no ESP32 determina o status de **vazamento** (true ou false);
6. A programação também mantém em memória o último estado de **vazamento** enviado ao MQTT Broker. Quando a lógica do ESP32 verifica que o status calculado do vazamento é diferente do último estado de vazamento enviado para o MQTT Broker, as seguintes ações são disparadas:  
     
   6.1 - A conexão com a internet é estabelecida. No nosso protótipo é via Wi-Fi;  
   6.2 - O ESP32 se conecta ao Broker MQTT (Mosquito) e publica a mensagem referente ao novo status (vazamento ou sem vazamento) no **topic** MQTT configurado;  
   6.3 – Em toda atualização o ESP32 busca o horário em um NTP para criar o timestamp do evento na mensagem.
7. MQTT Broker repassa a mensagem para os subscribers registrados no **topic**.

# Código Fonte da Programação do ESP-32

A seguir disponibilizamos o código de nossa implantação. Segue tabela de referência de como estão as conexões no ambiente desenvolvido:

|  |  |
| --- | --- |
| **COMPONENTE** | **Número da GPIO CONECTADA** |
| **YF-S201**(Fornece01) | 2 |
| **YF-S201**(Consome01) | 35 |
| **YF-S201**(Consome02) | 34 |
| **LED**(Atuador) | 12 |

Na página a seguir disponibilizamos o código da implantação.

#include <WiFi.h> // Biblioteca para conexão WiFi em placas ESP32  
#include <PubSubClient.h> // Biblioteca cliente MQTT  
#include <ezTime.h> // Biblioteca cliente NTP  
  
// Variáveis da loop()

boolean vazamentoMQTT = true;   
int contaCiclo = 1;  
boolean vazamento;  
String dataHoraStatus;  
String dataHoraPublish;  
String statusVazamento;  
String motivoAtualiza;  
String mensagem;  
Timezone dadoNTP;  
  
// Mapeamento variável -> GPIOs

const int fornece01 = 2;  
const int consome01 = 35;  
const int consome02 = 34;  
const int atuador = 12;  
  
//Variáveis da função conectawifi  
#define WIFI\_SSID "NOME\_REDE\_WIFI"  
#define WIFI\_PASSWORD "SENHA\_REDE\_WIFI"

// Definir o canal do WiFi acelera a conexão:  
#define WIFI\_CHANNEL # // Canal WiFi definido como #  
int status = WL\_IDLE\_STATUS; // Status inicial do rádio WiFi

//Variáveis da função publicaMqttBroker

const char \*mqtt\_broker = "IP\_MQTTBroker";  
const char \*topic = "/TOPICO\_MQTT";  
const char \*mqtt\_username = "USUÁRIO\_MQTTBroker";  
const char \*mqtt\_password = "SENHA\_USUÁRIO\_MQTTBroker ";  
const int mqtt\_port = 1883;

// Objeto WiFi

WiFiClient internet; // Objeto cliente WiFi usado na conexão  
// Objeto MQTT

PubSubClient client(internet); // Objeto para conexão MQTT

// Iniciando o hardware ESP32

void setup() {

// put your setup code here, to run once:

Serial.begin(115200);  
 pinMode(2, INPUT);  
 pinMode(34, INPUT);  
 pinMode(35, INPUT);  
 pinMode(12, OUTPUT);  
 digitalWrite(atuador, HIGH);  
}

//Função para mudar Atuador  
void setAtuador(boolean vazamento) {  
 if (!vazamento){  
 digitalWrite(atuador, LOW);

} else {

digitalWrite(atuador, HIGH);  
 }

}

// Função para conectar ao WiFi  
void conectaWifi() {

WiFi.begin(WIFI\_SSID, WIFI\_PASSWORD, WIFI\_CHANNEL) Serial.print("\nConectando ao WiFi ");  
Serial.print(WIFI\_SSID);   
Serial.println(" Conectado!"); // Mensagem de sucesso na conexão  
Serial.print("IP address: ");  
Serial.println(WiFi.localIP()); // Imprime o endereço IP obtido  
WiFi.mode(WIFI\_STA); // Configura o modo WiFi como estação (não ponto de acesso)

}

// Função para desconectar ao WiFi  
void desconectaWifi() {  
 WiFi.disconnect();

}

//Função para publicar no MQTT Broker  
void publicaMqttBroker(String mqttMensagem) {  
 client.setServer(mqtt\_broker, mqtt\_port);  
 client.setCallback(callback);  
 while (!client.connected()) {  
 String client\_id = "esp32-MQTTPublisher";  
 client\_id += String(WiFi.macAddress());  
 Serial.printf("\nPublisher %s conectando ao MQTT broker\n", client\_id.c\_str());  
 if (client.connect(client\_id.c\_str(), mqtt\_username, mqtt\_password)) {  
 Serial.print("Conectado ao MQTT broker wIoTer ");  
 Serial.println(mqtt\_broker);  
 } else {  
 Serial.print("Failed with state ");  
 Serial.println(client.state());  
 delay(2000);  
 }  
 }  
 Serial.print("\nTransmitirá seguinte mensagem via MQTT: ");  
 Serial.println(mqttMensagem);  
 client.publish(topic, mqttMensagem.c\_str());

}

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

//Função para pegar horário de server NTP

String getDataHora() {  
 char buffer[20];  
 waitForSync();   
 dadoNTP.setLocation("America/Sao\_Paulo");   
 snprintf(buffer, sizeof(buffer), "%02d/%02d/%04d %02d:%02d:%02d",  
 dadoNTP.day(), dadoNTP.month(), dadoNTP.year(),  
 dadoNTP.hour(), dadoNTP.minute(), dadoNTP.second());  
return String(buffer);  
}

//Função Loop (principal)

void loop() {

delay(2000); // espera 2 segundos antes de inciar nova execução  
Serial.print("\nCiclo de Medição número ");  
Serial.println(contaCiclo);  
int estadoFornece01 = digitalRead(fornece01);  
Serial.print("Status Fornece01: ");  
Serial.println(estadoFornece01);  
int estadoConsome01 = digitalRead(consome01);  
Serial.print("Status Consome01: ");  
Serial.println(estadoConsome01);  
int estadoConsome02 = digitalRead(consome02);  
Serial.print("Status Consome02: ");  
Serial.print(estadoConsome02);  
//Lógica para verificar vazamento  
vazamento = (estadoFornece01 == 1) && (estadoConsome01 == 0) && (estadoConsome02 == 0);

//Decide se haverá publicação  
if((vazamento != vazamentoMQTT) || (contaCiclo==60)) {  
conectaWifi();  
dataHoraStatus = getDataHora();  
vazamentoMQTT = vazamento;  
setAtuador(vazamentoMQTT);  
statusVazamento = (vazamentoMQTT?"COM VAZAMENTO": "Sem Vazamento");

if (contaCiclo < 60) {

motivoAtualiza = "POR EVENTO";  
} else {  
motivoAtualiza = "por periodo";  
contaCiclo = 0;

}

mensagem = "\nAtualizado " + motivoAtualiza + " .Status: " + statusVazamento + " .\nData e Hora: " + dataHoraStatus;  
dataHoraPublish= getDataHora();  
publicaMqttBroker(mensagem);  
desconectaWifi();  
Serial.print("\nHora da detecção: ");  
Serial.print(dataHoraStatus);  
Serial.print("\nHora da transmissão: ");  
Serial.print(dataHoraPublish);  
}  
Serial.print("\nStatus Vazamento: ");  
Serial.print(vazamentoMQTT?"True" : "False");  
contaCiclo = contaCiclo + 1;

}