MQTT EXAMPLES RUNNING ON RP2040 PICOW

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Yunnie Kim

MEng Field Advisor: Van Hunter Adams, Bruce R. Land

Degree Date: May 2026

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: MQTT Examples Running on RP2040 PicoW

Author: Yunnie Kim

Abstract:

MQTT is a protocol for small systems to share data over TCP/IP communications. The MQTT client available on the PicoW can be investigated to create an IoT connection system. Using a Raspberry Pi as a broker, a PicoW can read from a sensor and publish data, effectively able to communicate with other devices subscribed to the broker. After setting up this system, teaching examples for ECE 4760 will be designed.

Executive Summary

This project investigated the feasibility of integrating the MQTT communication protocol into the ECE 4760 course curriculum using the Raspberry Pi PicoW. The primary objective was to create a reliable framework for wireless communication with MQTT while also creating an infrastructure for future students.

The project successfully demonstrated that MQTT can be implemented on the RP2040 PicoW using the lwIP networking stack and the Raspberry Pi Pico C SDK. Multiple Pico W devices were configured as MQTT clients and connected through a central server hosted on a Raspberry Pi 3 Model B. Testing confirmed that the devices were capable of publishing and subscribing to messages across a shared wireless network, validating the practicality of MQTT for the context of ECE 4760.

In addition to demonstrating technical feasibility, the project established a foundational MQTT infrastructure intended for long-term use within ECE 4760. This infrastructure includes networking configurations and system documentation that future students can use as a starting point for laboratory exercises and collaborative projects. By creating and evaluating a reusable client–broker framework, the project enables future coursework involving wireless communication, distributed sensing, and cooperative embedded systems development.

Overall, the project showed that MQTT is a viable communication solution for embedded systems education on the PicoW platform and provides a scalable foundation for future ECE 4760 student projects involving networked microcontrollers.

Acknowledgements

I would like to thank Hunter Adams and Bruce Land for all their guidance, support, and wisdom throughout this project. Thank you for your patience along with your life lessons that I will surely carry with me in my future endeavors.

Introduction

MQTT, which stands for Message Queuing Telemetry Transport, is a lightweight messaging transport protocol that runs over TCP/IP through the lwIP implementation. MQTT works well for many situations, particularly Internet of Things (IoT) contexts where network bandwidth and code headroom are limited. Therefore, it is perfect for the purposes of ECE 4760.

Figure 1: MQTT Communication System [5]

The MQTT protocol operates on a publish/subscribe system with two roles: a client and a broker (also known as a server). The key organization factor is through topic names. Messages are published to a topic and sent to devices subscribed to that topic. Clients can publish or subscribe to a topic, meaning that they can both send and receive messages through specifying a topic.

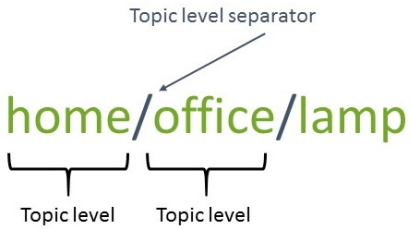

Figure 2: MQTT Topic Hierarchy [5]

The MQTT broker receives all the messages published by the devices connected in the network. Using the topic name, the broker filters through the messages and finds devices that are interested and listening to the topic. Then, it publishes messages to all subscribed clients accordingly.

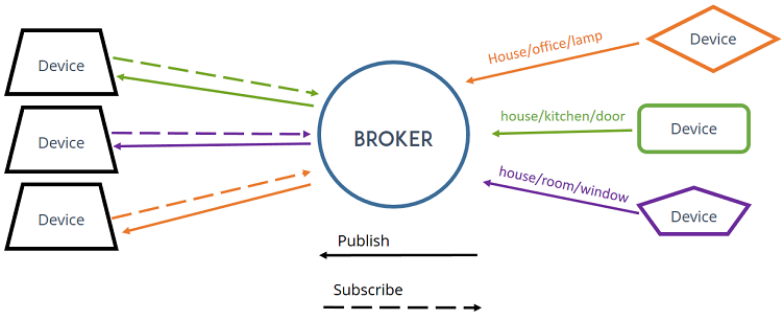

Figure 3: MQTT Broker Functionality [5]

Design

This project focuses on developing a lightweight MQTT communication infrastructure for the Raspberry Pi Pico W using the lwIP networking stack and the Raspberry Pi Pico C SDK. The primary design problem is enabling reliable MQTT communication on the RP2040 microcontroller while providing a framework that is accessible and reusable for future ECE 4760 students.

The system implements a distributed client–broker architecture. Each PicoW operates as an MQTT client capable of reading sensor data and publishing messages over a wireless network. A Raspberry Pi 3 Model B functions as the MQTT broker, coordinating communication between multiple PicoW devices connected through an isolated local network. The intended outcome is a scalable environment where multiple devices can exchange information in real time using the MQTT protocol.

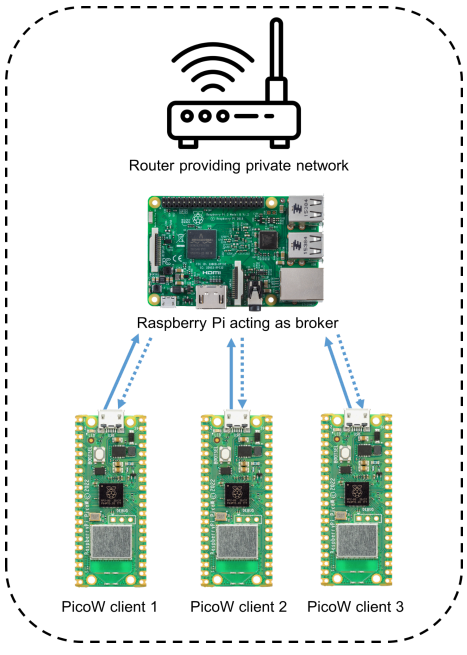

Figure 4: High Level System Overview

Several system requirements guided the design process:

- Integration of the MQTT client into the lwIP implementation for the Pico W platform
- Reliable communication between multiple PicoW clients through a central broker
- Operation on a private local network without dependence on external cloud services
- Minimal setup and maintenance requirements for classroom and laboratory use
- Clear documentation, example code, and demonstrations to support adoption by future students

To address these requirements, the project investigates existing MQTT implementations and reference designs for the Pico W platform. Resources considered include official Raspberry Pi Pico W networking examples, lwIP MQTT example implementations, and community-developed projects such as picow-iot. These references provide a foundation for evaluating communication reliability and ease of deployment.

The chosen design allows for future development and usage for ECE 4760. By hosting the broker locally on a Raspberry Pi and maintaining communication within a private network, the system can ensure security while improving accessibility for educational use. The resulting framework is intended to serve as a reusable platform for future ECE 4760 laboratory exercises and collaborative projects involving distributed embedded systems and MQTT communication.

Implementation

The beginning approach to this project began with the Raspberry Pi PicoW MQTT Client example project [3] and the open-source Eclipse Mosquitto MQTT package. Using the instructions provided in the example project and the official Mosquitto website [4], a Mosquitto broker and client were set up on a local Linux machine. To test the validity of the Mosquitto implementation, terminals connecting to the localhost IP address were tested. After connection was verified, where one terminal was able to subscribe and receive messages that the other terminal published, the next step was to implement the MQTT client on a PicoW.

When running the Raspberry Pi example code on a PicoW, some problems arose. It was initially unclear how to define the connection on the PicoW client, like the MQTT server and WiFi credentials. From tinkering with SSL/TLS server keys and certificates to trying multiple WiFi connection sources, error messages “Failed to connect” and “Failed to connect to mqtt server” were persistent.

Eventually, it was decided to isolate the sources of error by focusing on ensuring the PicoW client functionality. Instead of using a locally hosted broker, a commercial broker, specifically test.mosquitto.org, was used to host the connection. After connecting both the PicoW and local Linux machine to the IP address of the public Mosquitto broker, the MQTT protocol was indeed able to run. With the temperature function, the PicoW was able to publish messages and the Linux machine was able to receive them. Likewise, with the LED function, the Linux machine was able to publish and the PicoW was able to receive the

messages. After functionality of the PicoW client was confirmed through the commercial broker, then it was time to implement the Raspberry Pi broker.

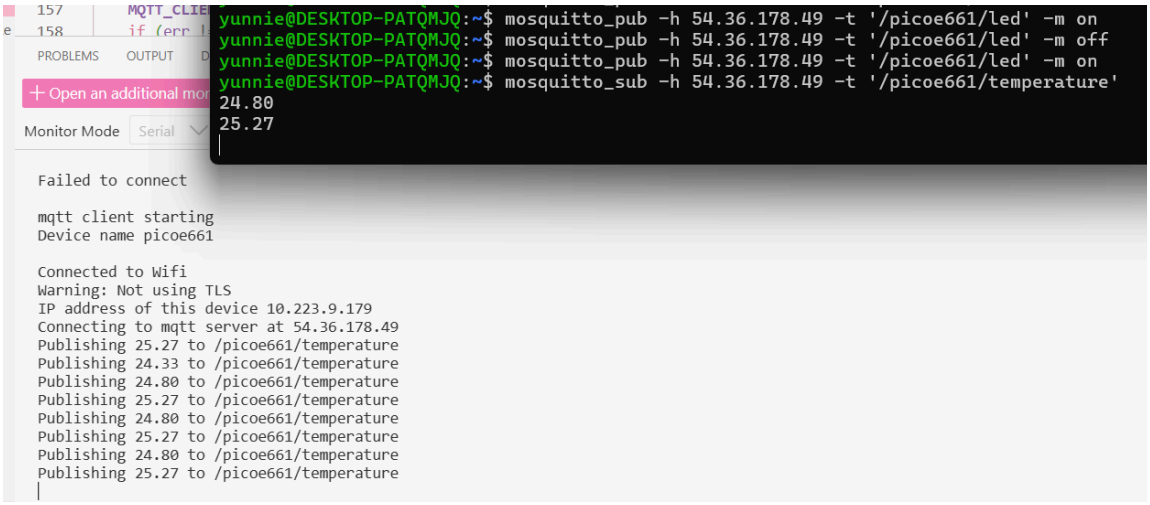

Figure 5: PicoW Client and Linux Machine Client using Mosquitto.org MQTT Server

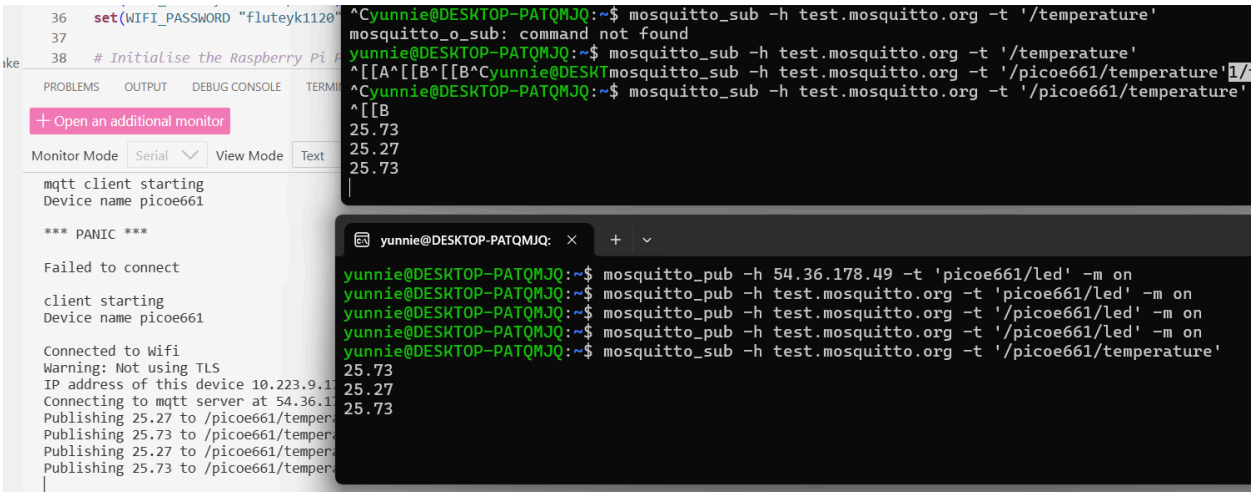

Figure 6: PicoW Client and Multiple Linux Machine Clients

The process of establishing the MQTT broker on the Raspberry Pi was relatively smooth, though it did not come without its faults. Particularly, an HDMI cable was not initially accessible. As a workaround, I attempted to use an HDMI to VGA adapter to connect to a VGA to DisplayPort cable. I learned, however, of the unidirectional aspect of these screen

connections. Once an HDMI cable was acquired, setup of the Raspberry Pi was able to continue.

After the Raspberry Pi broker had been set up, it was time to establish communication between the PicoW client and Raspberry Pi broker. Both the PicoW and Raspberry Pi were connected to a phone hotspot. Once the assigned IP address of the broker was determined, the host name MQTT_SERVER was set for the PicoW to communicate with. Along with acting as a MQTT broker, the Raspberry Pi also had MQTT client functionality, meaning that it could publish and subscribe to topic names. The Raspberry Pi subscribed to the PicoW's temperature topic, or /picoe661/temperature, and published to the PicoW's LED topic, or /picoe661/led. The broker received the recorded temperature messages from the PicoW and turned the PicoW board LED on and off, confirming a successful MQTT connection between a PicoW client and Raspberry Pi broker.

```
mqtt client starting
Device name picoe661

Connected to Wifi
Warning: Not using TLS
IP address of this device 10.64.217.179
Connecting to mqtt server at 10.64.217.19
Publishing 27.14 to /picoe661/temperature
Publishing 26.67 to /picoe661/temperature
Publishing 27.14 to /picoe661/temperature
Publishing 27.61 to /picoe661/temperature
Publishing 27.14 to /picoe661/temperature
Publishing 27.61 to /picoe661/temperature
Publishing 27.14 to /picoe661/temperature
Publishing 27.61 to /picoe661/temperature
Publishing 27.14 to /picoe661/temperature
```

Figure 7: PicoW Connecting to Broker and Publishing to /picoe661/temperature Topic

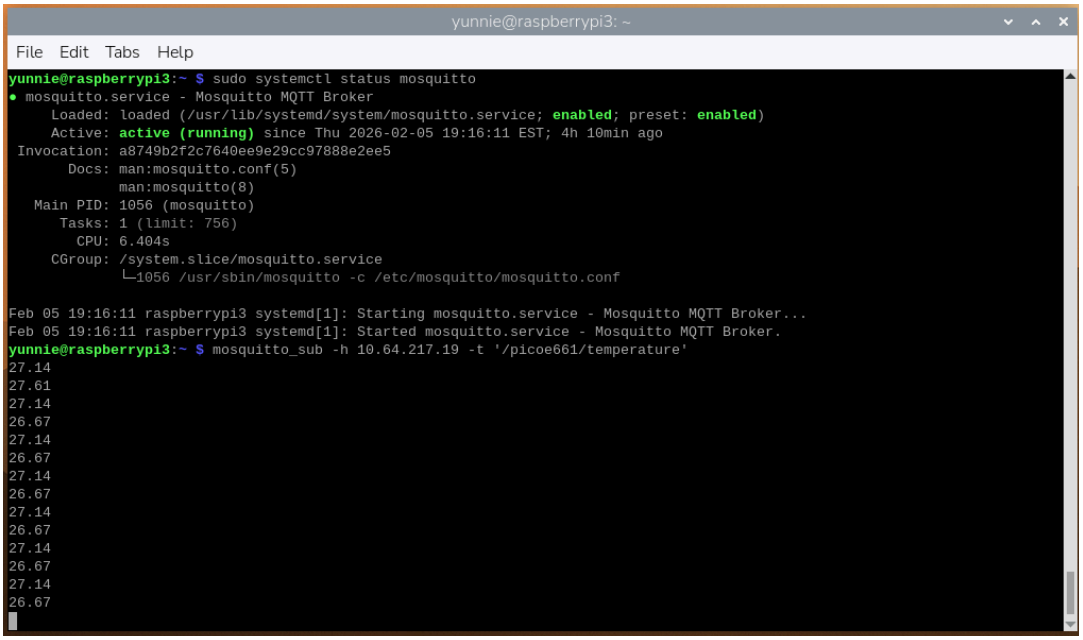

Figure 8: Raspberry Pi Broker Subscribing to /picoe661/temperature Topic

Figure 9: Raspberry Pi Broker Publishing to /picoe661/led Topic

Once the connection between a single PicoW client and the broker had been established, the next step was to connect multiple PicoW clients to the broker and confirm that each PicoW could communicate with each other. Communication between devices was confirmed by transmitting messages from separate PicoWs to the Raspberry Pi broker and with each other, demonstrating that the MQTT infrastructure functioned correctly.

Additional implementation work involved validating the compatibility of VGA graphic animation and the PicoW. This ensured that MQTT communication could coexist with

graphics-related applications running on the RP2040, which was important because many future student projects may combine wireless communication with graphical interfaces for real-time visual output.

The next step for this system was adding support for a private standalone network. This involved two stages: setting up the router and integrating the router network into the existing system. The initial approach was to manually configure a static IP address and its associated static DNS servers. This approach, however, proved to be difficult to execute correctly.

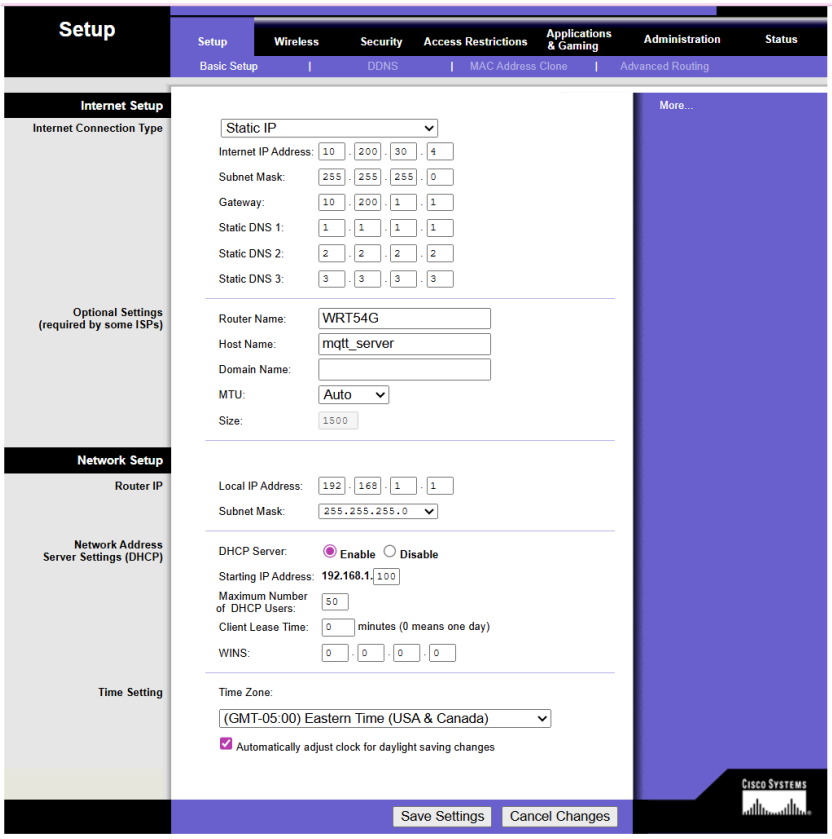

Figure 10: Setup Page for Linksys WRT54G Router

Instead, the router was configured using DHCP (Dynamic Host Configuration Protocol), where the router automatically assigns temporary IP addresses to devices that connect. DHCP handles IP allocation, prevents address conflicts, and simplifies network management, and therefore worked better for the project. Phones and computers were able to connect to and ping through the WiFi network produced by the router, which verified correct setup.

After the router setup was completed, the network needed to be integrated into the PicoW client and Raspberry Pi broker system. During this process, however, several networking issues were encountered while attempting to establish connection on the PicoW. With the MQTT system, while the broker could connect to the network, the PicoW client could not connect to either the network or IP address assigned to the broker. To isolate the problem, debugging was performed using a UDP networking example included with the Pico SDK. The debugging process revealed that the PicoW devices were unable to progress beyond the DHCP stage responsible for assigning IP addresses. As a result, the devices could not successfully join the network or communicate with the MQTT broker.

One possible explanation for this issue was firmware incompatibility between the wireless router and the PicoW networking stack. Since the PicoW utilizes the CYW43 wireless chipset and associated firmware, differences in router security settings, wireless standards, or DHCP behavior may have interfered with successful network initialization. Investigating and resolving these compatibility issues became an important part of the implementation process, as reliable wireless connectivity is essential for maintaining MQTT communication between embedded devices.

Conclusion

This project successfully demonstrated the feasibility of implementing MQTT communication on the Raspberry Pi PicoW platform for use in the ECE 4760 curriculum. By integrating the MQTT client into the lwIP networking stack and establishing communication between multiple PicoW devices through a centralized Raspberry Pi 3 Model B broker, the project validated that lightweight TCP/IP communication can be effectively incorporated into embedded systems coursework.

In addition to demonstrating technical viability, the project established a foundational infrastructure that future ECE 4760 students can use for laboratory exercises and independent projects. The resulting framework provides students with an accessible introduction to distributed embedded systems, wireless networking, and real-time communication protocols. Expanding ECE 4760 to include MQTT and TCP/IP-based communication significantly broadens the range of possible projects and encourages collaborative system design between student groups.

Future Work

Future work will focus on completing and refining the MQTT infrastructure for long-term classroom use within ECE 4760. One of the most important remaining tasks is configuring the system to operate reliably on a standalone local network using a dedicated wireless router. While testing demonstrated successful MQTT communication, integrating the local network generated by the router introduced networking challenges that prevented the PicoW devices from consistently obtaining IP addresses through DHCP. One possible cause

of this issue is firmware incompatibility between the router and the PicoW wireless networking stack. Additional debugging and testing with different router configurations, firmware versions, and security settings will be necessary to find the root cause and achieve a stable connection.

Another important future step is the creation of comprehensive documentation and instructional material for students. This includes preparing an MQTT handout describing the networking architecture, MQTT publish–subscribe concepts, setup instructions, and editing the Raspberry Pi Pico example code for integrating MQTT into PicoW projects. Providing reusable templates and clear documentation will make the infrastructure more accessible to future ECE 4760 students and instructors.

Adding support for TCP/IP communication and MQTT also opens the door for a wide range of collaborative and interactive laboratory exercises. One potential lab project is a class-wide collaborative LED display, where each student group controls an LED color and timing through MQTT messages. Another possibility is a multiplayer maze game where groups communicate indirectly through network messages while navigating using only a first-person VGA display. Additional ideas include hide-and-seek multiplayer game variants, such as Marco Polo, Mummy Tag, and Sardines, all of which could leverage wireless communication between multiple devices. These types of projects would encourage collaboration, networking concepts, and large-scale system integration while enabling more creative potential for students in ECE 4760.

References

[1] MQTT Version 5.0. Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. 07 March 2019. OASIS Standard. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html. Latest version: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[2] raspberrypi/pico-examples, https://github.com/raspberrypi/pico-examples

[3] raspberrypi/pico-examples/pico_w/wifi/mqtt, https://github.com/raspberrypi/pico-examples/tree/master/pico_w/wifi/mqtt

[4] R. A. Light, "Mosquitto: server and client implementation of the MQTT protocol," The Journal of Open Source Software, vol. 2, no. 13, May 2017, DOI: 10.21105/joss.00265; https://www.mosquitto.org/

[5] “Raspberry Pi Pico W: Getting Started with MQTT (MicroPython),” Random Nerd Tutorials, https://randomnerdtutorials.com/raspberry-pi-pico-w-mqtt-micropython/

[6] vha3/Hunter-Adams-RP2040-Demos, https://github.com/vha3/Hunter-Adams-RP2040-Demos

[7] “UDP communication from Pico W to PC,” V. Hunter Adams, https://vanhunteradams.com/Pico/UDP/UDP_Chat.html

Bill of Materials

Item	Quantity	Price	Notes
Raspberry Pi PicoW	2	$6.00	As of 12/19/25
Raspberry Pi 3 Model B	1	$35.00	
MicroSD card (32GB)	1	$15.39	
Linksys WRT54G Wireless-G Router	1	$54.97	
Total		$111.36	

Appendix

PicoW Client

For the PicoW client, the Raspberry Pi picow_mqtt_client example was used. The two main files are shown below.

To connect the PicoW to the broker and WiFi network, the following parameters need to be set in the CMakeLists: MQTT_SERVER, WIFI_SSID, and WIFI_PASSWORD.

- MQTT_SERVER: hostname (IP address) of the server/broker as a String
- WIFI_SSID: name of WiFi network as a String
- WIFI_PASSWORD: WiFi password as a String (set to “” if no password)

The mqtt_client.c file includes two functions that send or receive messages: a temperature function and an LED function. The temperature function records the PicoW’s temperature using its onboard sensor and publishes the data periodically, the length of which is determined by TEMP_WORKER_TIME_S and the unit of measurement determined by TEMPERATURE_UNITS. For MQTT configuration parameters, the MQTT_KEEP_ALIVE_S sets the maximum length of time allowed between communication packets for the client to listen for.

A key detail to note is that if MQTT_UNIQUE_TOPIC is set to 1, then the PicoW will publish to a topic under its unique device name. So, the temperature function for picoe661 publishes to /picoe661/temperature. If MQTT_UNIQUE_TOPIC is set to 0, then the topic name is simply /temperature.

CMakeLists.txt

```
# Generated Cmake Pico project file
cmake_minimum_required(VERSION 3.13)

set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)

# Initialise pico_sdk from installed location
# (note this can come from environment, CMake cache etc)

# == DO NOT EDIT THE FOLLOWING LINES for the Raspberry Pi Pico VS Code
Extension to work ==
if(WIN32)
    set(USERHOME $ENV{USERPROFILE})
else()
    set(USERHOME $ENV{HOME})
endif()
set(sdkVersion 2.2.0)
set(toolchainVersion 14_2_Rel1)
set(picotoolVersion 2.2.0-a4)
set(picoVscode ${USERHOME}/.pico-sdk/cmake/pico-vscode.cmake)
if (EXISTS ${picoVscode})
    include(${picoVscode})
endif()
#
==========
set(PICO_BOARD pico_w CACHE STRING "Board type")

# Pull in Raspberry Pi Pico SDK (must be before project)
include(pico_sdk_import.cmake)

project(picow_mqtt_client C CXX ASM)

set(MQTT_SERVER "yk_server") # Change this to the host name of your MQTT
server

set(WIFI_SSID "mqtt_router")
```

```
set(WIFI_PASSWORD "")

# Initialise the Raspberry Pi Pico SDK
pico_sdk_init()

# Add executable. Default name is the project name, version 0.1

# Define the host name of the MQTT server in an environment variable or
pass it to cmake,
# e.g. cmake -DMQTT_SERVER=myserver ..

# cmake MQTT_SERVER=yk_server ..

if (DEFINED ENV{MQTT_SERVER} AND (NOT MQTT_SERVER))
    set(MQTT_SERVER $ENV{MQTT_SERVER})
    message("Using MQTT_SERVER from environment ('${MQTT_SERVER}')")
endif()
if (NOT MQTT_SERVER)
    message("Skipping MQTT example as MQTT_SERVER is not defined")
    return()
endif()
# Define the name of an MQTT broker/server to enable this example
set(MQTT_SERVER "${MQTT_SERVER}" CACHE INTERNAL "MQTT server for
examples")

if (DEFINED ENV{MQTT_USERNAME} AND (NOT MQTT_USERNAME))
    set(MQTT_USERNAME $ENV{MQTT_USERNAME})
    message("Using MQTT_USERNAME from environment ('${MQTT_USERNAME}')")
endif()
set(MQTT_USERNAME "${MQTT_USERNAME}" CACHE INTERNAL "MQTT user name for
examples")
if (DEFINED ENV{MQTT_PASSWORD} AND (NOT MQTT_PASSWORD))
    set(MQTT_PASSWORD $ENV{MQTT_PASSWORD})
    message("Using MQTT_PASSWORD from environment")
endif()
set(MQTT_PASSWORD "${MQTT_PASSWORD}" CACHE INTERNAL "MQTT password for
examples")

# Set path to the certificate include file
if (NOT MQTT_CERT_PATH)
```

```
    set(MQTT_CERT_PATH ${CMAKE_CURRENT_LIST_DIR}/certs/${MQTT_SERVER})
endif()

# Set the name of the certificate include file
if (NOT MQTT_CERT_INC)
    set(MQTT_CERT_INC mqtt_client.inc)
endif()

add_executable(picow_mqtt_client
    mqtt_client.c
    )
target_link_libraries(picow_mqtt_client
    pico_stdlib
    hardware_adc
    pico_cyw43_arch_lwip_threadsafe_background
    pico_lwip_mqtt
    pico_mbedtls
    pico_lwip_mbedtls
    )
target_include_directories(picow_mqtt_client PRIVATE
    ${CMAKE_CURRENT_LIST_DIR}
    ${CMAKE_CURRENT_LIST_DIR}/.. # for our common lwipopts or any other
standard includes, if required
    )
target_compile_definitions(picow_mqtt_client PRIVATE
    WIFI_SSID=\"${WIFI_SSID}\"
    WIFI_PASSWORD=\"${WIFI_PASSWORD}\"
    MQTT_SERVER=\"${MQTT_SERVER}\"
    )
if (EXISTS "${MQTT_CERT_PATH}/${MQTT_CERT_INC}")
    target_compile_definitions(picow_mqtt_client PRIVATE
        MQTT_CERT_INC=\"${MQTT_CERT_INC}\" # contains the tls certificates
for MQTT_SERVER needed by the client
        ALTCP_MBEDTLS_AUTHMODE=MBEDTLS_SSL_VERIFY_REQUIRED
        )
    target_include_directories(picow_mqtt_client PRIVATE
        ${MQTT_CERT_PATH}
        )
endif()
if (MQTT_USERNAME AND MQTT_PASSWORD)
```

```
    target_compile_definitions(picow_mqtt_client PRIVATE
        MQTT_USERNAME=\"${MQTT_USERNAME}\"
        MQTT_PASSWORD=\"${MQTT_PASSWORD}\"
    )
endif()
pico_add_extra_outputs(picow_mqtt_client)

# Ignore warnings from lwip code
set_source_files_properties(
    ${PICO_LWIP_PATH}/src/apps/altcp_tls/altcp_tls_mbedtls.c
    PROPERTIES
    COMPILE_OPTIONS "-Wno-unused-result"
    )
```

mqtt_client.c

```
/**
 * Copyright (c) 2022 Raspberry Pi (Trading) Ltd.
 *
 * SPDX-License-Identifier: BSD-3-Clause
 */
//
// Created by elliot on 25/05/24.
//
#include "pico/stdlib.h"
#include "pico/cyw43_arch.h"
#include "pico/unique_id.h"
#include "hardware/gpio.h"
#include "hardware/irq.h"
#include "hardware/adc.h"
#include "lwip/apps/mqtt.h"
#include "lwip/apps/mqtt_priv.h" // needed to set hostname
#include "lwip/dns.h"
#include "lwip/altcp_tls.h"

// Temperature
#ifndef TEMPERATURE_UNITS
#define TEMPERATURE_UNITS 'F' // Set to 'F' for Fahrenheit
```

```
#endif

#ifndef MQTT_SERVER

#endif

// This file includes your client certificate for client server
authentication
#ifdef MQTT_CERT_INC
#include MQTT_CERT_INC
#endif

#ifndef MQTT_TOPIC_LEN
#define MQTT_TOPIC_LEN 100
#endif

typedef struct {
    mqtt_client_t* mqtt_client_inst;
    struct mqtt_connect_client_info_t mqtt_client_info;
    char data[MQTT_OUTPUT_RINGBUF_SIZE];
    char topic[MQTT_TOPIC_LEN];
    uint32_t len;
    ip_addr_t mqtt_server_address;
    bool connect_done;
    int subscribe_count;
    bool stop_client;
} MQTT_CLIENT_DATA_T;

#ifndef DEBUG_printf
#ifndef NDEBUG
#define DEBUG_printf printf
#else
#define DEBUG_printf(...)
#endif
#endif

#ifndef INFO_printf
#define INFO_printf printf
#endif
```

```c
#ifndef ERROR_printf
#define ERROR_printf printf
#endif

// how often to measure our temperature
#define TEMP_WORKER_TIME_S 5

// keep alive in seconds
#define MQTT_KEEP_ALIVE_S 60

// qos passed to mqtt_subscribe
// At most once (QoS 0)
// At least once (QoS 1)
// Exactly once (QoS 2)
#define MQTT_SUBSCRIBE_QOS 1
#define MQTT_PUBLISH_QOS 1
#define MQTT_PUBLISH_RETAIN 0

// topic used for last will and testament
#define MQTT_WILL_TOPIC "/online"
#define MQTT_WILL_MSG "0"
#define MQTT_WILL_QOS 1

#ifndef MQTT_DEVICE_NAME
#define MQTT_DEVICE_NAME "pico"
#endif

// Set to 1 to add the client name to topics, to support multiple devices
using the same server
#ifndef MQTT_UNIQUE_TOPIC
#define MQTT_UNIQUE_TOPIC 1
#endif

/* References for this implementation:
 * raspberry-pi-pico-c-sdk.pdf, Section '4.1.1. hardware_adc'
 * pico-examples/adc/adc_console/adc_console.c */
static float read_onboard_temperature(const char unit) {

    /* 12-bit conversion, assume max value == ADC_VREF == 3.3 V */
    const float conversionFactor = 3.3f / (1 << 12);
```

```
    float adc = (float)adc_read() * conversionFactor;
    float tempC = 27.0f - (adc - 0.706f) / 0.001721f;

    if (unit == 'C' || unit != 'F') {
        return tempC;
    } else if (unit == 'F') {
        return tempC * 9 / 5 + 32;
    }

    return -1.0f;
}

static void pub_request_cb(__unused void *arg, err_t err) {
    if (err != 0) {
        ERROR_printf("pub_request_cb failed %d", err);
    }
}

static const char *full_topic(MQTT_CLIENT_DATA_T *state, const char *name)
{
#if MQTT_UNIQUE_TOPIC
    static char full_topic[MQTT_TOPIC_LEN];
    snprintf(full_topic, sizeof(full_topic), "/%s%s",
state->mqtt_client_info.client_id, name);
    return full_topic;
#else
    return name;
#endif
}

static void control_led(MQTT_CLIENT_DATA_T *state, bool on) {
    // Publish state on /state topic and on/off led board
    const char* message = on ? "On" : "Off";
    if (on)
        cyw43_arch_gpio_put(CYW43_WL_GPIO_LED_PIN, 1);
    else
        cyw43_arch_gpio_put(CYW43_WL_GPIO_LED_PIN, 0);
```

```
    mqtt_publish(state->mqtt_client_inst, full_topic(state, "/led/state"),
message, strlen(message), MQTT_PUBLISH_QOS, MQTT_PUBLISH_RETAIN,
pub_request_cb, state);
}

static void publish_temperature(MQTT_CLIENT_DATA_T *state) {
    static float old_temperature;
    const char *temperature_key = full_topic(state, "/temperature");
    float temperature = read_onboard_temperature(TEMPERATURE_UNITS);
    if (temperature != old_temperature) {
        old_temperature = temperature;
        // Publish temperature on /temperature topic
        char temp_str[16];
        snprintf(temp_str, sizeof(temp_str), "%.2f", temperature);
        INFO_printf("Publishing %s to %s\n", temp_str, temperature_key);
        mqtt_publish(state->mqtt_client_inst, temperature_key, temp_str,
strlen(temp_str), MQTT_PUBLISH_QOS, MQTT_PUBLISH_RETAIN, pub_request_cb,
state);
    }
}

static void sub_request_cb(void *arg, err_t err) {
    MQTT_CLIENT_DATA_T* state = (MQTT_CLIENT_DATA_T*)arg;
    if (err != 0) {
        panic("subscribe request failed %d", err);
    }
    state->subscribe_count++;
}

static void unsub_request_cb(void *arg, err_t err) {
    MQTT_CLIENT_DATA_T* state = (MQTT_CLIENT_DATA_T*)arg;
    if (err != 0) {
        panic("unsubscribe request failed %d", err);
    }
    state->subscribe_count--;
    assert(state->subscribe_count >= 0);

    // Stop if requested
    if (state->subscribe_count <= 0 && state->stop_client) {
        mqtt_disconnect(state->mqtt_client_inst);
```

```
    }
}

static void sub_unsub_topics(MQTT_CLIENT_DATA_T* state, bool sub) {
    mqtt_request_cb_t cb = sub ? sub_request_cb : unsub_request_cb;
    mqtt_sub_unsub(state->mqtt_client_inst, full_topic(state, "/led"),
MQTT_SUBSCRIBE_QOS, cb, state, sub);
    mqtt_sub_unsub(state->mqtt_client_inst, full_topic(state, "/print"),
MQTT_SUBSCRIBE_QOS, cb, state, sub);
    mqtt_sub_unsub(state->mqtt_client_inst, full_topic(state, "/ping"),
MQTT_SUBSCRIBE_QOS, cb, state, sub);
    mqtt_sub_unsub(state->mqtt_client_inst, full_topic(state, "/exit"),
MQTT_SUBSCRIBE_QOS, cb, state, sub);
}

static void mqtt_incoming_data_cb(void *arg, const u8_t *data, u16_t len,
u8_t flags) {
    MQTT_CLIENT_DATA_T* state = (MQTT_CLIENT_DATA_T*)arg;
#if MQTT_UNIQUE_TOPIC
    const char *basic_topic = state->topic +
strlen(state->mqtt_client_info.client_id) + 1;
#else
    const char *basic_topic = state->topic;
#endif
    strncpy(state->data, (const char *)data, len);
    state->len = len;
    state->data[len] = '\0';

    DEBUG_printf("Topic: %s, Message: %s\n", state->topic, state->data);
    if (strcmp(basic_topic, "/led") == 0)
    {
        if (lwip_stricmp((const char *)state->data, "On") == 0 ||
strcmp((const char *)state->data, "1") == 0)
            control_led(state, true);
        else if (lwip_stricmp((const char *)state->data, "Off") == 0 ||
strcmp((const char *)state->data, "0") == 0)
            control_led(state, false);
    } else if (strcmp(basic_topic, "/print") == 0) {
        INFO_printf("%.*s\n", len, data);
    } else if (strcmp(basic_topic, "/ping") == 0) {
```

```
        char buf[11];
        snprintf(buf, sizeof(buf), "%u",
to_ms_since_boot(get_absolute_time()) / 1000);
        mqtt_publish(state->mqtt_client_inst, full_topic(state,
"/uptime"), buf, strlen(buf), MQTT_PUBLISH_QOS, MQTT_PUBLISH_RETAIN,
pub_request_cb, state);
    } else if (strcmp(basic_topic, "/exit") == 0) {
        state->stop_client = true; // stop the client when ALL
subscriptions are stopped
        sub_unsub_topics(state, false); // unsubscribe
    }
}

static void mqtt_incoming_publish_cb(void *arg, const char *topic, u32_t
tot_len) {
    MQTT_CLIENT_DATA_T* state = (MQTT_CLIENT_DATA_T*)arg;
    strncpy(state->topic, topic, sizeof(state->topic));
}

static void temperature_worker_fn(async_context_t *context,
async_at_time_worker_t *worker) {
    MQTT_CLIENT_DATA_T* state = (MQTT_CLIENT_DATA_T*)worker->user_data;
    publish_temperature(state);
    async_context_add_at_time_worker_in_ms(context, worker,
TEMP_WORKER_TIME_S * 1000);
}
static async_at_time_worker_t temperature_worker = { .do_work =
temperature_worker_fn };

static void mqtt_connection_cb(mqtt_client_t *client, void *arg,
mqtt_connection_status_t status) {
    MQTT_CLIENT_DATA_T* state = (MQTT_CLIENT_DATA_T*)arg;
    if (status == MQTT_CONNECT_ACCEPTED) {
        state->connect_done = true;
        sub_unsub_topics(state, true); // subscribe;

        // indicate online
        if (state->mqtt_client_info.will_topic) {
```

```
            mqtt_publish(state->mqtt_client_inst,
state->mqtt_client_info.will_topic, "1", 1, MQTT_WILL_QOS, true,
pub_request_cb, state);
        }

        // Publish temperature every 10 sec if it's changed
        temperature_worker.user_data = state;
        async_context_add_at_time_worker_in_ms(cyw43_arch_async_context(),
&temperature_worker, 0);
    } else if (status == MQTT_CONNECT_DISCONNECTED) {
        if (!state->connect_done) {
            panic("Failed to connect to mqtt server");
        }
    }
    else {
        panic("Unexpected status");
    }
}

static void start_client(MQTT_CLIENT_DATA_T *state) {
#if LWIP_ALTCP && LWIP_ALTCP_TLS
    const int port = MQTT_TLS_PORT;
    INFO_printf("Using TLS\n");
#else
    const int port = MQTT_PORT;
    INFO_printf("Warning: Not using TLS\n");
#endif

    state->mqtt_client_inst = mqtt_client_new();
    if (!state->mqtt_client_inst) {
        panic("MQTT client instance creation error");
    }
    INFO_printf("IP address of this device %s\n",
ipaddr_ntoa(&(netif_list->ip_addr)));
    INFO_printf("Connecting to mqtt server at %s\n",
ipaddr_ntoa(&state->mqtt_server_address));

    cyw43_arch_lwip_begin();
```

```
    if (mqtt_client_connect(state->mqtt_client_inst,
&state->mqtt_server_address, port, mqtt_connection_cb, state,
&state->mqtt_client_info) != ERR_OK) {
        panic("MQTT broker connection error");
    }
#if LWIP_ALTCP && LWIP_ALTCP_TLS
    // This is important for MBEDTLS_SSL_SERVER_NAME_INDICATION

mbedtls_ssl_set_hostname(altcp_tls_context(state->mqtt_client_inst->conn),
MQTT_SERVER);
#endif
    mqtt_set_inpub_callback(state->mqtt_client_inst,
mqtt_incoming_publish_cb, mqtt_incoming_data_cb, state);
    cyw43_arch_lwip_end();
}

// Call back with a DNS result
static void dns_found(const char *hostname, const ip_addr_t *ipaddr, void
*arg) {
    MQTT_CLIENT_DATA_T *state = (MQTT_CLIENT_DATA_T*)arg;
    if (ipaddr) {
        state->mqtt_server_address = *ipaddr;
        start_client(state);
    } else {
        panic("dns request failed");
    }
}

int main(void) {
    stdio_init_all();
    INFO_printf("mqtt client starting\n");

    adc_init();
    adc_set_temp_sensor_enabled(true);
    adc_select_input(4);

    static MQTT_CLIENT_DATA_T state;

    if (cyw43_arch_init()) {
        panic("Failed to inizialize CYW43");
```

```
    }

    // Use board unique id
    char unique_id_buf[5];
    pico_get_unique_board_id_string(unique_id_buf, sizeof(unique_id_buf));
    for(int i=0; i < sizeof(unique_id_buf) - 1; i++) {
        unique_id_buf[i] = tolower(unique_id_buf[i]);
    }

    // Generate a unique name, e.g. pico1234
    char client_id_buf[sizeof(MQTT_DEVICE_NAME) + sizeof(unique_id_buf) -
1];
    memcpy(&client_id_buf[0], MQTT_DEVICE_NAME, sizeof(MQTT_DEVICE_NAME) -
1);
    memcpy(&client_id_buf[sizeof(MQTT_DEVICE_NAME) - 1], unique_id_buf,
sizeof(unique_id_buf) - 1);
    client_id_buf[sizeof(client_id_buf) - 1] = 0;
    INFO_printf("Device name %s\n", client_id_buf);

    state.mqtt_client_info.client_id = client_id_buf;
    state.mqtt_client_info.keep_alive = MQTT_KEEP_ALIVE_S; // Keep alive
in sec
#if defined(MQTT_USERNAME) && defined(MQTT_PASSWORD)
    state.mqtt_client_info.client_user = MQTT_USERNAME;
    state.mqtt_client_info.client_pass = MQTT_PASSWORD;
#else
    state.mqtt_client_info.client_user = NULL;
    state.mqtt_client_info.client_pass = NULL;
#endif
    static char will_topic[MQTT_TOPIC_LEN];
    strncpy(will_topic, full_topic(&state, MQTT_WILL_TOPIC),
sizeof(will_topic));
    state.mqtt_client_info.will_topic = will_topic;
    state.mqtt_client_info.will_msg = MQTT_WILL_MSG;
    state.mqtt_client_info.will_qos = MQTT_WILL_QOS;
    state.mqtt_client_info.will_retain = true;
#if LWIP_ALTCP && LWIP_ALTCP_TLS
    // TLS enabled
#ifdef MQTT_CERT_INC
    static const uint8_t ca_cert[] = TLS_ROOT_CERT;
```

```
    static const uint8_t client_key[] = TLS_CLIENT_KEY;
    static const uint8_t client_cert[] = TLS_CLIENT_CERT;
    // This confirms the indentity of the server and the client
    state.mqtt_client_info.tls_config =
altcp_tls_create_config_client_2wayauth(ca_cert, sizeof(ca_cert),
            client_key, sizeof(client_key), NULL, 0, client_cert,
sizeof(client_cert));
#if ALTCP_MBEDTLS_AUTHMODE != MBEDTLS_SSL_VERIFY_REQUIRED
    WARN_printf("Warning: tls without verification is insecure\n");
#endif
#else
    state->client_info.tls_config = altcp_tls_create_config_client(NULL,
0);
    WARN_printf("Warning: tls without a certificate is insecure\n");
#endif
#endif

    cyw43_arch_enable_sta_mode();
    if (cyw43_arch_wifi_connect_timeout_ms(WIFI_SSID, WIFI_PASSWORD,
CYW43_AUTH_WPA2_AES_PSK, 30000)) {
        panic("Failed to connect");
    }
    INFO_printf("\nConnected to Wifi\n");

    // We are not in a callback so locking is needed when calling lwip
    // Make a DNS request for the MQTT server IP address
    cyw43_arch_lwip_begin();
    int err = dns_gethostbyname(MQTT_SERVER, &state.mqtt_server_address,
dns_found, &state);
    cyw43_arch_lwip_end();

    if (err == ERR_OK) {
        // We have the address, just start the client
        start_client(&state);
    } else if (err != ERR_INPROGRESS) { // ERR_INPROGRESS means expect a
callback
        panic("dns request failed");
    }
```

```
    while (!state.connect_done ||
mqtt_client_is_connected(state.mqtt_client_inst)) {
        cyw43_arch_poll();
        cyw43_arch_wait_for_work_until(make_timeout_time_ms(10000));
    }

    INFO_printf("mqtt client exiting\n");
    return 0;
}
```

Raspberry Pi Broker

The Raspberry Pi was configured with the Mosquitto MQTT package using the following steps.

README

```
# Quick start

To use this example you will need to install an MQTT server on your
network.
To install the Mosquitto MQTT client on a Raspberry Pi Linux device run
the following...

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

Check it works...

```
mosquitto_pub -t test_topic -m "Yes it works" -r
mosquitto_sub -t test_topic -C 1
```
```

To allow an external client to connect to the server you will have to change the configuration. Add the following to /etc/mosquitto/conf.d/mosquitto.conf

```
allow_anonymous true
listener 1883 0.0.0.0
```

Then restart the service.

```
sudo service mosquitto restart
```

When building the code set the host name of the MQTT server, e.g.

```
export MQTT_SERVER=myhost
cmake ..
```

The example should publish its core temperature to the /temperature topic. You can subscribe to this topic from another machine.

```
mosquitto_sub -h $MQTT_SERVER -t '/temperature'
```

You can turn the led on and off by publishing messages.

```
mosquitto_pub -h $MQTT_SERVER -t '/led' -m on
mosquitto_pub -h $MQTT_SERVER -t '/led' -m off
```

Security

If your server has a username and password, you can set these with the following variables.

```
export MQTT_USERNAME=user
export MQTT_PASSWORD=pass
```

Be aware that these details are sent in plain text unless you use TLS.

Using TLS

To use TLS and client server authentication you need some keys and certificates for the client and server.
The `certs/makecerts.sh` script demonstrates a way to make these.
It creates a folder named after `MQTT_SERVER` containing all the required files.
From these files it generates a header file `mqtt_client.inc` included by the code.
Your server will have to be configured to use TLS and the port 8883 rather than the non-TLS port 1883.

```
listener 1883 127.0.0.1

listener 8883 0.0.0.0
allow_anonymous true
cafile /etc/mosquitto/ca_certificates/ca.crt
certfile /etc/mosquitto/certs/server.crt
keyfile /etc/mosquitto/certs/server.key
require_certificate true
```

To connect to your server with the mosquitto tools in linux you will have to pass extra parameters.

```
mosquitto_pub -h $MQTT_SERVER --cafile $MQTT_SERVER/ca.crt --key
$MQTT_SERVER/client.key --cert $MQTT_SERVER/client.crt -t /led -m on
mosquitto_sub -h $MQTT_SERVER --cafile $MQTT_SERVER/ca.crt --key
$MQTT_SERVER/client.key --cert $MQTT_SERVER/client.crt -t "/temperature"
```

There are some shell scripts in the certs folder to reduce the amount of typing assuming `MQTT_SERVER` is defined.

```
cd certs
./pub.sh /led on
./sub.sh /temperature
```