ECE 445L Lab 9

IEE802.11 Wi-Fi communication, TCP, Web sockets, MQTT Publish-Subscribe, IoT

This laboratory assignment accompanies the book, [*Embedded Systems: Real-Time Interfacing to ARM Cortex M Microcontrollers, ISBN-13: 978-1463590154*](https://www.amazon.com/Embedded-Systems-Real-Time-Interfacing-Microcontrollers/dp/1463590156), by Jonathan W. Valvano, copyright © 2021.

# Table of Contents

[Table of Contents 1](#_Toc179733935)

[Figures 2](#_Toc179733936)

[Team Size 3](#_Toc179733937)

[Goals 3](#_Toc179733938)

[Review 3](#_Toc179733939)

[Starter Files 3](#_Toc179733940)

[Required Hardware 3](#_Toc179733941)

[Lab Overview 3](#_Toc179733942)

[Preparation 7](#_Toc179733943)

[Procedure 10](#_Toc179733944)

[ESP8266 Procedure 11](#_Toc179733945)

[Web App Procedure 14](#_Toc179733946)

[LaunchPad Procedure 16](#_Toc179733947)

[Merge Lab 8 Code 19](#_Toc179733948)

[Deliverable 1 20](#_Toc179733949)

[Deliverable 2 20](#_Toc179733950)

[Deliverable 3 20](#_Toc179733951)

[Deliverable 4 21](#_Toc179733952)

[Deliverable 5 21](#_Toc179733953)

[Deliverable 6 (5pts Extra Credit) 22](#_Toc179733954)

[Lab Checkout 22](#_Toc179733955)

[Lab Report 22](#_Toc179733956)

[Hint (A deep dive into MQTT) 23](#_Toc179733957)

[Hint (Arduino Setup) 24](#_Toc179733958)

[Steps: 24](#_Toc179733959)

[Hint (MQTT Debugging) 27](#_Toc179733960)

[Browser Debugging 27](#_Toc179733961)

[Internet Traffic Analysis & Debug 28](#_Toc179733962)

[Hint (Notable Project Files) 29](#_Toc179733963)

[LaunchPad Starter Project 29](#_Toc179733964)

[ESP8266 Starter Project 30](#_Toc179733965)

[MQTT Web Application Starter Project 30](#_Toc179733966)

# Figures

[Figure 9.1: Lab 9 Data Flow Diagram 4](#_Toc179736851)

[Figure 9.2: The ESP8266 WiFi module. 5](#_Toc179736852)

[Figure 9.3 MQTT Publish-Subscribe Model 6](#_Toc179736853)

[Figure 9.4 Data Flow Diagram between the TM4C and the Web App 8](#_Toc179736854)

[Figure 9.5A (Left) & 9.5B (Right): ESP 8266 Pinout numbering 10](#_Toc179736855)

[Figure 9.6 Picture of OLED display on the RSLK Board 10](#_Toc179736856)

[Figure 9.7 Serial Monitor showing ESP8266 boot up process (debug only) 11](#_Toc179736857)

[Figure 9.8 Listing of Pub-Sub topics. 12](#_Toc179736858)

[Figure 9.9: Lab 9 MQTT Web App interface. 14](#_Toc179736859)

[Figure 9.10. MQTTX interface. 15](#_Toc179736860)

[Figure 9.11. Listing of “onConnect” 16](#_Toc179736861)

[Figure 9.12 Listing of “onMessageArrived” 16](#_Toc179736862)

[Figure 9.13. Listing of While loop on flag. 17](#_Toc179736863)

[Figure 9.14 Listing of Arduino side UART capture and parsing of the credential 17](#_Toc179736864)

[Figure 9.15. TM4C pub-sub command dataflow diagram. 18](#_Toc179736865)

[Figure 9.16 Listing of Authentication fields. 19](#_Toc179736866)

[Figure 9.17 Listing of the Command parser 19](#_Toc179736867)

[Figure 9.18 Example of TM4C\_to\_MQTT CSV generation 20](#_Toc179736868)

[Figure 9.19: A screenshot of an excerpt from the MQTT writeup in the resource folder. 23](#_Toc179736869)

[Figure 9.20: Adding Board Manager URL 24](#_Toc179736870)

[Figure 9.21: Installing the ESP8266 Board Packages 24](#_Toc179736871)

[Figure 9.22: Installing MQTT Library 25](#_Toc179736872)

[Figure 9.23: Installing Blynk Library 25](#_Toc179736873)

[Figure 9.25: MQTT explorer. 27](#_Toc179736874)

[Figure 9.26: Browser console. 27](#_Toc179736875)

[Figure 9.27: Wireshark frame. 28](#_Toc179736876)

[Figure 9.28: Hex packet data. 28](#_Toc179736877)

[Figure 9.29 Associated subscribe topic. 28](#_Toc179736878)

# Team Size

The team size for this lab is **4.**

# Goals

* Implement a “smart object” that connects to a Web Application using the MQTT broker.
* Use the ESP8266 to interface between the internet and the TM4C123.
* Remotely control the robot motors developed in Lab 8 via the MQTT Publish/Subscribe interface.

# Review

* Valvano Section 11.4 on IoT
* MQTT architecture (see Lab Overview)

# Starter Files

* Starter project:
  + Lab 9 template provided on the GitHub Classroom repo.

# Required Hardware

|  |  |  |  |
| --- | --- | --- | --- |
| Parts | Datasheet | Price | Source (**price source)** |
| EK-TM4C123GXL | [EK-TM4C123GXL datasheet](https://github.com/ECE445L/ECE445L-Lab5/blob/main/resources/TM4C_Datasheet.pdf) | $16.99 | **TI** |
| RSLK2 Robot | [RSLK2\_Datasheet](https://www.ti.com/lit/ml/sekp166/sekp166.pdf) |  |  |
| ESP8266-ESP01 | [ESP8266 Datasheet](https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf) | 3.50 | Get it from the TA’s |
| ESP01 breadboard adapter | N/A | N/A | Get it from the TA’s |
| ESP8266 Programmer |  | 10.99 | EER Checkout Desk |
| LM2937-3.3 Voltage Regulator **or** any 3.3V regulator | [LM3927 Datasheet](https://www.ti.com/lit/ds/symlink/lm2937.pdf?ts=1695011187491) | 1.68 | EER Checkout Desk |

# Lab Overview

In this lab your team will create a “smart object” that connects to a MQTT Broker. This Broker will allow communication to a Web Application as shown below in Figure 9.1. This figure primarily shows the data flow for the TM4C code.



Figure 9.1: Lab 9 Data Flow Diagram

Your system will include:

* A PC or Phone running a MQTT Web App that you will develop using starter code we provide.
* A TM4C123 Launchpad running your Lab 8 software plus a UART peripheral that talks to the ESP8266 Wi-Fi module.
* An ESP8266 Wi-Fi module (Fig. 9.2) that is configured with the MQTT network starter code built using the Arduino IDE
* Existing hardware reused from Lab 8, including the display and any switches.

Your system will be able to:

* Have the phone or laptop running the MQTT Web App change the four PI variables and the desired RPM (X-star)
* Have the TM4C send the following status data for each motor: U(t), E(t) and X(t) to the MQTT web application in response to the control variables from the MQTT web application.
* A nice to have would be another display showing the current values of the four PI variables.

You are free to define the “look and feel” of your system if you meet the specifications listed above. **NOTE: You will need access to a wireless access point, configured with or without security to use the MQTT broker running in the EER. You may also use any online cloud brokers. Wi-Fi** is the name given to devices that communicate wirelessly employing the IEEE 802.11 standard. They typically operate in the 2.4 GHz Industrial-Scientific-Medical (ISM) unlicensed band that is shared with ZigBee (IEEE 802.15.4) and Bluetooth (IEEE 802.15.1) communications. The IEEE 802.11 standard describes how information is represented via radio frequency signals, i.e., the **physical** or **PHY** layer, and how communications are formatted and controlled, i.e., the **media access control** or **MAC** layer. Communication requirements beyond the PHY and MAC, e.g., flow control, error recovery, and routing, are handled at higher levels of the internet stack. We will be using the ESP8266 WiFi module (Figure 9.2) to interface to the internet.

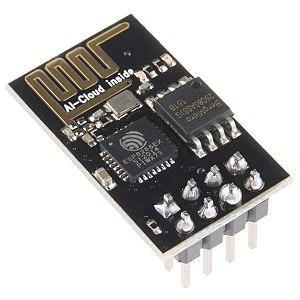


Figure 9.2: The ESP8266 WiFi module.

MQTT ([which stands for nothing](https://www.oasis-open.org/committees/download.php/49028/OASIS_MQTT_TC_minutes_25042013.pdf)) is a message passing mechanism that uses a centralized broker (server) to receive published data and send the data to any number of subscribers as shown below in Figure 9.3.



Figure 9.3 MQTT Publish-Subscribe Model

**Key points about MQTT:**

* **All clients** can Publish (broadcast) and subscribe (receive).
* Clients **do not have addresses** like in email systems; messages are not sent directly to clients.
* Clients become **subscribers** to “topic(s)” to get messages published to the broker about the “topic”.
* Messages are **published to a broker about a particular “topic”**. In this Lab all topics MUST begin with your EID. A topic example is *rg7677/b2w* where:
  + *rg7677* is the student EID,
  + *b2w* is a subtopic indicating that the message is from the LaunchPad board to the web application, and
* The job of an MQTT broker is to **collect and sort messages** based on the “topic”, so that the broker can **distribute messages to subscribers of the “topic”**.
* There is **no direct connection** between a publisher and subscriber.
* MQTT brokers do NOT normally store messages. All messages are sent via TCP which guarantees delivery (vs. UDP which does not).
* The publisher dictates the data format. The subscriber must know the format to decipher the data.
* You will be given suggested topics and data formats. You are free to generate your own.
* You have several options for MQTT brokers:
  + There is broker in the EER that can be used for this lab. The configuration is:
    - IP address: 10.159.177.133
    - Port: 9001
    - SSID: Utexas, Utexas-IoT, <your hot spot>
    - Password: you will need one to connect to these Wi-Fi SSIDs
    - Go to <https://network.utexas.edu> to register the MAC address of the ESP8266
  + You can set up your own broker on a laptop or a server; an option is [MOSQUITTO](https://mosquitto.org/) It is considered one of the best open-source MQTT brokers.
  + An online hosted MQTT broker: EMQX is a free public broker with no sign up that you can use instantly.
    - IP address/domain name: broker.emqx.io
    - Port: 8083 (for websockets, 1883 for TCP port)
    - **The starter code has been set up with to talk to this broker so you can get it running instantly.**

Here are some useful MQTT tutorials and projects:

* <http://www.steves-internet-guide.com/mqtt-works/>
* <http://www.steves-internet-guide.com/download/mqtt-study-guide/>
* <http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/>
* <http://www.steves-internet-guide.com/into-mqtt-python-client/>
* <http://www.steves-internet-guide.com/mqtt-websockets/>
* <http://www.steves-internet-guide.com/using-javascript-mqtt-client-websockets/>
* <http://www.steves-internet-guide.com/mosquitto-broker/>
* <http://www.steves-internet-guide.com/install-mosquitto-broker/>
* <http://www.steves-internet-guide.com/mqtt-broker-testing/>
* <http://www.steves-internet-guide.com/monitoring-mqtt-brokers/>

# Preparation

Preparation is performed during or before the W/TH lab session.

1. Software Review
   1. Review the Lab 9 starter code provided in the GitHub classroom repository. There is a hint at the end of this document enumerating notable files in the project. There are three sets of software you should be familiar with which are listed below.
      * The RLSK/TM4C starter project you expanded in Lab 8
      * The ESP8266 starter project, which performs the intermediate parsing and handshakes with the MQTT broker and LaunchPad (via UART)
      * The MQTT web application starter project which displays the data collected.
2. ESP Software Setup
   1. Demo to the TA that you can flash the ESP8266 Starter project successfully. A guide to installing the Arduino IDE and project dependencies can be found in the hints section at the end of this lab document. Be prepared to show the ESP interacting with the web interface.
   2. You will need to checkout an ESP8266 programmer from the checkout counter if you intend to change the functionality of the ESP8266. The orientation of the ESP8266 is displayed on the programmer. **You will destroy the ESP8266 if you install it in the other orientation. The programmer MUST be stored in the box as it is vulnerable to damage from electrostatic discharge (ESD).** The programmer MUST be returned in working order and is only available for short-term checkout.
3. MQTT API and communication scheme
   1. Explain to your TA what MQTT topics you will need to interface to the robot from Lab 8. MQTT Topics are key in communication between MQTT clients and brokers. Properly defining your MQTT Topics and the API regarding them is a crucial part of this lab.
   2. Understand that in MQTT, a topic is a string that is a filter that a MQTT broker uses for MQTT message delivery. The broker filters all clients according to their subscription and forwards the message to subscribers. A publisher and subscriber can create MQTT topics. Here is an example of a MQTT topic:
      * EER/floor1/embeddedlabroom/temperature - This topic represents the temperature in the embedded lab room, which is part of floor 1, which is part of the EER.
   3. The suggested dataflow for Lab 9 is shown below in Figure 9.4:

A diagram of a computer system

Description automatically generated

Figure 9.4 Data Flow Diagram between the TM4C and the Web App

The TM4C communicates via UART to the ESP8266 using CSV packets. The ESP8266 converts outgoing (Publish) data from CSV to a JSON packet. The JSON packet is sent the WEB APP (via the Broker) for display. The ESP8266 code that is provided does not check the published data to determine if all the data needs to be sent the WEB APP. It is advised that the following piece of code in the ESP8266 be modified such that only data that has changed be sent to the Broker. NOTE: If the broker gets too much data, too fast, it will shut down the connection.

The JSON packet format for this lab is:

{"A":300, "B":345, "C":4567, "D":12343, "E":23, "F":75, "G":75, "H":75, "I":75, "J":75, "K":75}

The following commands assign the corresponding TM4C signal to "A-K":

doc["A"] = U\_left;

doc["B"] = U\_right;

doc["C"] = error\_left;

doc["D"] = error\_right;

doc["E"] = rpm\_left;

doc["F"] = rpm\_right;

doc["G"] = Kp1;

doc["H"] = Kp2;

doc["I"] = Ki1;

doc["J"] = Ki2;

doc["K"] = Xstar;

The follow command "packs" the data for transmission.

serializeJson(doc, ser\_buf);

Prior to running this command determine which data needs to be sent to the Web App and only execute the corresponding "doc["?"] command.

1. Hardware Setup
   1. Figure 9.5 shows the two different numbering schemes that are found in the vendor documentation for the ESP8266. It can be confusing. Double check which footprint you are using in KiCad before sending your project board out to be manufactured.
   2. For this Lab you will be using a pre-wired board setup. It is critical though that you **DO NOT** put the ESP8266 in backwards. The ESP8266 should be hanging off the RLSK board and not covering the SSD1306 OLED device.

A screenshot of a cell phone

Description automatically generated A blue circuit board with red lines

Description automatically generated

Figure 9.5A (Left) & 9.5B (Right): ESP 8266 Pinout numbering

* 1. There is an OLED ([SSD1306](https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf&ved=2ahUKEwi1yoPdl4mJAxWS4ckDHZK1ELoQFnoECAgQAQ&usg=AOvVaw295piYr-tzt5CnBsNVzI7X)) on the RSLK board that can be used to display debug information about the MQTT interface to the RSLK board.



Figure 9.6 Picture of OLED display on the RSLK Board

# Procedure

*Note This procedure is performed during or before the W/TH lab session.*

This lab has a lot of moving parts, so we suggest you perform the lab procedure in a specific order to minimize heartburn and pain.

1. Run the starter code for the ESP8266 and the Web App. Make sure this interface is working correctly before trying to interface the TM4C to the ESP8266 in the board.
2. Communicate with the ESP8266 via your laptop’s serial terminal (Arduino IDE or Putty) and the web app console/monitor, seeing changes when topics are published or subscribed to.
3. Read the code to get an understanding of the data flow between devices.
   1. PC -> **UART** -> ESP -> **WIFI** -> MQTT Broker -> **WIFI (or Ethernet)** -> Web App
4. Extend the system to the TM4C replacing the PC. View communications sent by the TM4C using a logic analyzer, PuTTY, or the SSD1306 (or ST7735) display.
   1. Some code modifications may be required here. At this point students can split up the work, with one person working on the web interface and another on the TM4C.
5. When the TM4C "interfacing student" has a working unit test that can send messages to the other student’s web interface, then do we encourage students to (1) extend the endpoint API to their desired specifications and (2) begin integrating Lab 8 into Lab 9.

## ESP8266 Procedure

The ESP must be configured to connect to Wi-Fi. This can be done in two ways:

1. Update the Arduino Sketch with your group’s EIDs. By default, this will be **xxx**, and groups that do not change the EID may result in name collision with other groups.
2. After flashing the device, you can then connect to the ESP via UART. The ESP will prompt the PC/TM4C to provide comma separated credentials, or to skip this and use the preloaded values via a newline/enter. An example of a credential string is shown below in Figure 9.7 where format is: EID, SSID, PASSWORD, MQTT\_BROKER\_IP, PORT\_NUMBER

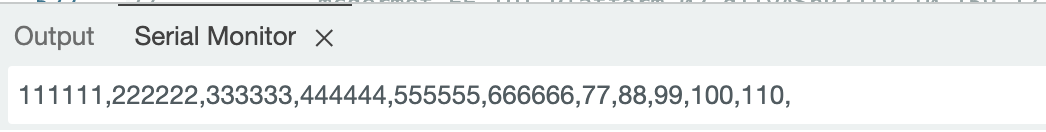
**A screenshot of a computer

Description automatically generated**

Figure 9.7 Serial Monitor showing ESP8266 boot up process (debug only)

Note that this UART interface is the same one that the TM4C will use to send data to the ESP8266. Thus, as a debugging measure, you can send premade messages to the ESP to confirm functionality of any changes you make to the Arduino code or Web app code. For example, in the figure above, the credential is being sent to the Arduino that is connected to a PC, emulating what will eventually be sent by the TM4C.

For example, sending the following CSV string:



should display the corresponding values on the Web App:

A screenshot of a computer

Description automatically generated

Data sent to the ESP8266 is separated by commas, with the following format: “U\_right, U\_left, error\_right, error\_left, rpm\_right, rpm\_left, Kp1, Kp2, Ki1, Ki2, Xstar”. This data is converted to the JSON format described above and is then published to the "b2w" topic as shown below in Figure 9.8 At this stage, test that the ESP can receive messages sent to its subscribed topic. Additionally, send comma separated (CSV) data to the ESP via UART to see how they show up in the published topics.

**// ----------------------------------------------------------------**

**// -------------- Publish topic --------------------------**

**//**

**const char \*pub\_b2w = "/b2w";**

**char topic\_publish[100];**

**// ----------------------------------------------------------------**

**// -------------- Subscribe topic ------------------------**

**//**

**char topic\_w2b[64] = "/w2b";**

**char topic\_subscribe[100];**

Figure 9.8 Listing of Pub-Sub topics.

## Web App Procedure

By default, the web app should connect to the [EMQX](https://www.emqx.com/en/mqtt/public-mqtt5-broker) server regardless of the network you are on (unless it is private and restricted to outgoing traffic).

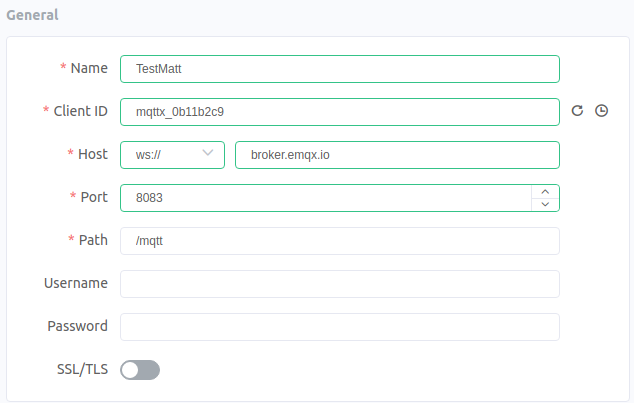
Opening the index.html in the *sw/web* folder of your starter code on your web browser, you should see Figure 9.9. below Pressing connect (even without a username and password) should connect you to the server. For a quick loopback test, you can subscribe and Publish to the same topic, with some random message, and you should get a message received response.

*A screenshot of a computer

Description automatically generated*

Figure 9.9: Lab 9 MQTT Web App interface.

We also encourage you to download [MQTTX](https://mqttx.app/) (an all-in-one client toolbox that supports MacOS, Windows, and Linux), to also connect and send/receive messages between the ESP and the Web App. This is like the Postman equivalent of HTTP and can be very useful for testing your web interface. The application should be relatively straightforward to set up; just make sure you have the host and port set properly, and you should be able to connect as in Figure 9.10. You can add subscriptions in the center pane and Publish to arbitrary topics on the bottom right console.



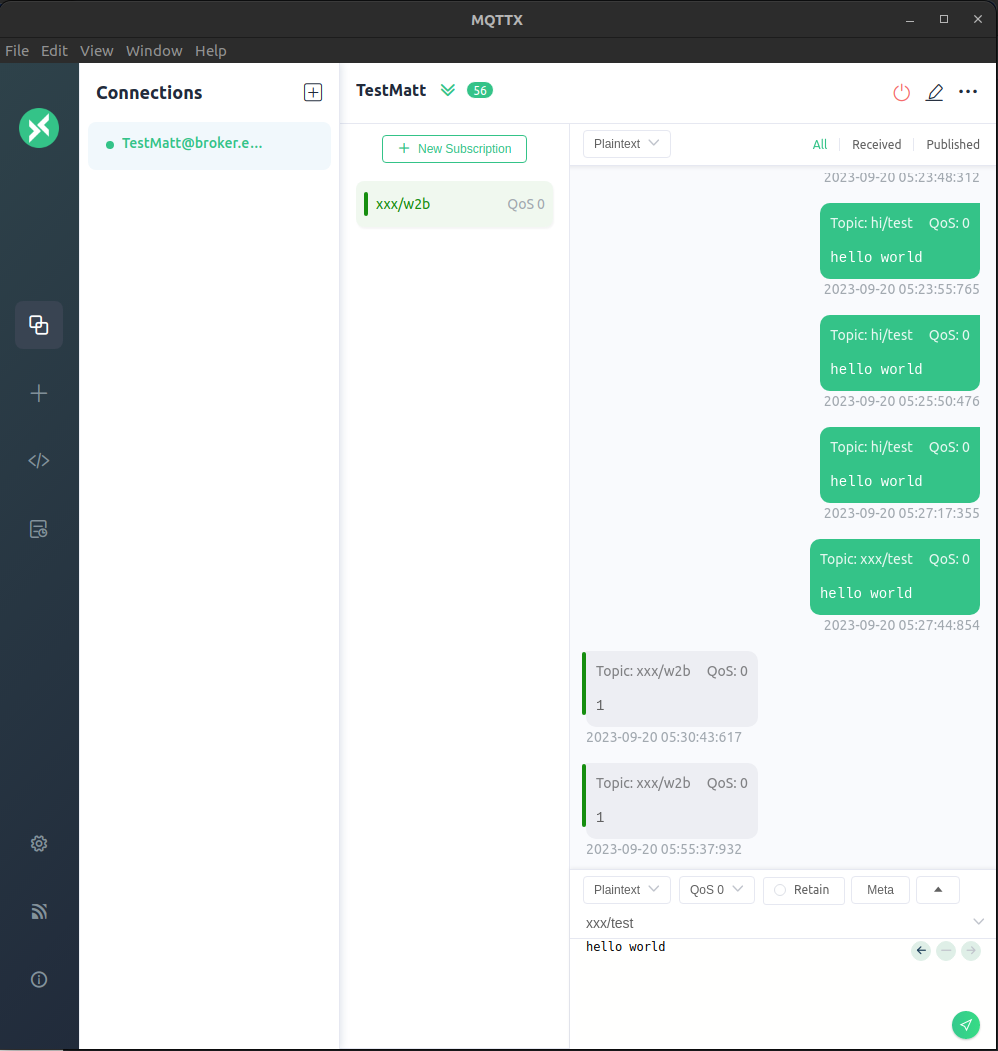


Figure 9.10. MQTTX interface.

Students who work on this portion of the lab, we suggest making a copy of this code so you can use it to continue debugging ESP8266 and TM4C functionality. With one set of code (the code you will keep), you will need to make the following modifications:

1. Modify the default configuration values in ***mqtt\_monitor.htm*** for your environment. This will involve setting the default IP address for the Broker.
2. Modify ***mqtt\_app.htm***to display the 11 values (“U\_right, U\_left, error\_right, error\_left, rpm\_right, rpm\_left, Kp1, Kp2, Ki1, Ki2, Xstar”.)

(See [w3schools](https://www.w3schools.com/html/default.asp) for lots of HTML tutorials on adding buttons or displays, etc).

1. Copy **clock\_page.js** to **motor\_ctl.js**
2. Modify the default configuration values in **motor\_ctl.js** for your environment.
3. Modify the Subscribe and Publish variables in **motor\_ctl.js** to comply with the data protocol outlined above. Modify the following onConnect and onMessageArrived functions for handling more APIs.

**// --------------------------------------------------------------------**

**//**

**function onConnect(context) {**

**console.log("Client Connected");**

**// Subscribe to our topics -- both with the same callback function**

**options = {qos:1, onSuccess:function(context){ console.log("subscribed"); } }**

**client.subscribe(b2w, options);**

**}**

Figure 9.11. Listing of “onConnect”

**// This function uses JSON.parse to extract data from the MQTT websocket payload.**

**// The TM4C composes the JSON packet and sends it to the ESP8266 which transmits**

**// it to this routine.**

**//**

**function onMessageArrived(message) {**

**console.log( "Incoming Message", message.destinationName, message.payloadString);**

**if (message.destinationName == sub\_b2w)**

**{**

**const obj = JSON.parse(message.payloadString);**

**U\_left\_local = eval(obj.A);**

**U\_right\_local = eval(obj.B);**

**error\_left\_local = eval(obj.C);**

**error\_right\_local = eval(obj.D);**

**rpm\_left\_local = eval(obj.E);**

**rpm\_right\_local = eval(obj.F);**

**kp1\_val = eval(obj.G);**

**kp2\_val = eval(obj.H);**

**ki1\_val = eval(obj.I);**

**ki2\_val = eval(obj.J);**

**xstar\_val = eval(obj.K);**

**}**

**}**

Figure 9.12 Listing of “onMessageArrived”

## LaunchPad Procedure

The next step is to replace the human in front of the PC sending commands via the serial port with the TM4C. To validate that we get the same connection outputs from the ESP8266, we can use PuTTY (Windows only) to talk to the TM4C through the ICDI USB-UART debug chip via UART0. Reminder that this only work when the USB cable is plugged into the debug microUSB connector. The starter code uses 115200 bits/sec, 1 stop, no parity, and no flow control, which can be set in the connection properties for PuTTY.

You may also output the results on the OLED display. There are two #defines in esp8266.h you can select either one or the other, both or neither of the UART0 or the OLED for debugging. DEBUG1 activates UART0 debugging and DEBUG3 activates OLED debugging. Use the Windows device manager to determine the COM port used to communication communicate with your LaunchPad (in Device Manager, click View -> Show hidden devices to view COM ports).

1. You will need to debug the interface between the ESP8266 and the TM4C. The ESP8266 will assert the RDY flag once it is ready to accept commands from the TM4C. Make sure all debug messages from the ESP8266 are sent to the UART0 debug port.

The following code in the TM4C loops waiting for the flag:

**while (!RDY) { // Wait for ESP8266 to indicate it is ready for programming data**

**#ifdef DEBUG1**

**UART0\_OutString(".");**

**#endif**

**DelayWait10ms(30);**

**}**

Figure 9.13. Listing of While loop on flag.

Reminder that the ESP8266 is expecting the following sequence of data to be sent to it after it asserts the RDY flag: **EID, SSID, PASSWORD, MQTT\_BROKER\_IP\_ADDRESS, PORT\_NUMBER,**

**NOTE! The sequence must be terminated with a comma and a NEWLINE (\n) as shown below:**

**char eid[32] = "your-EID-goes-here";**  **// Your EID goes here**

**char ssid[32] = "EE-IOT-Platform-03";**  **// WAP in the 445L Lab**

**char pass[32] = "";** **// Get the password from the TA**

**char mqtt\_broker[16] = "10.159.177.113";** **// EER based broker**

**char mqtt\_port[8] = "1883";** **// TCP/IP MQTT port number**

**UART2\_OutString(eid); // Student EID – Used for MQTT Topics**

**UART2\_OutChar(',');**

**UART2\_OutString(ssid); // Send WiFi SSID to ESP8266**

**UART2\_OutChar(',');**

**UART2\_OutString(pass); // Send WiFi Password to ESP8266**

**UART2\_OutChar(',');**

**UART2\_OutString(mqtt\_broker); // Send IP address of MQTT Broker**

**UART2\_OutChar(',');**

**UART2\_OutString(mqtt\_port); // Send MQTT port number**

**UART2\_OutChar(','); // Extra comma needed for ESP8266 parser code**

**UART2\_OutChar('\n'); // Send NewLine to indicate EOT**

Figure 9.14 Listing of Arduino side UART capture and parsing of the credential

Once you can make a connection to the MQTT Broker you will need to confirm that you can subscribe to a topic and Publish to a topic. Using the supplied MQTT Monitor (that your lab partner should have already gotten up and running), you will Publish data to a topic that the TM4C has subscribed to. The TM4C will in turn republish the data to a different topic than the Monitor has subscribed to. This will confirm that the roundtrip from the TM4C -> Broker -> Monitor -> Broker -> TM4C is functioning.

The next step is to set up the topics and data format between the TM4C and the Webapp. All topics MUST begin with your EID. A topic example is rg7677/b2w where rg7677 is your EID and b2w is a subtopic indicating that the message is from the LaunchPad board to the Web Application.

The topic implementation for the example code is:

**// ----------------------------------------------------------------**

**// -------------- Publish topic --------------------------**

**//**

**const char \*pub\_b2w = "/b2w";**

**char topic\_publish[100];**

**// ----------------------------------------------------------------**

**// -------------- Subscribe topic ------------------------**

**//**

**char topic\_w2b[64] = "/w2b";**

**char topic\_subscribe[100];**

The TM4C publishes 11 values in this order once a second: “U\_right, U\_left, error\_right, error\_left, rpm\_right, rpm\_left, Kp1, Kp2, Ki1, Ki2, Xstar”.

The ESP8266 converts the CSV stream into a JSON packet as shown above in Figure 9.4 and sends it to the Broker. The broker sends JSON packet to the Web Application which displays the data as shown above in Figure 9.14. The Web Application publishes a single command to the TM4C since there isn’t a need to send multiple commands.

A diagram of a computer

Description automatically generated

Figure 9.15. TM4C pub-sub command dataflow diagram.

## Merge Lab 8 Code

Finally, you will need to merge your Lab 8 code with the program you just got up and running.

1. Merge your Lab 8 code with the starter code.

You will find these five lines around line 35 in esp8266.c. Modify the lines of code for the environment you are working in.

**char eid[32] = ""; // Your EID goes here**

**char ssid[32] = ""; // SSID for the WAP you are using**

**char pass[32] = ""; // Password for the access to the WAP**

**char mqtt\_broker[32] = ""; // IP Address to your Broker**

**char mqtt\_port[8] = "1883"; // Port for TCP/IP access to the Broker**

Figure 9.16 Listing of Authentication fields.

1. Modify the MQTT\_to\_TM4C routine in the MQTT.c starter code. Modify the example code to parse commands from the Web App based on the data protocol that you have decided to use for the lab.

**if ((UART2\_FR\_R & UART2\_FR\_RXFE) ==0 )**

**{**

**input\_char =(UART2\_DR\_R & 0xFF);**

**if ((input\_char == ',') | ((input\_char >= '0') & (input\_char <= '9')) )**

**{**

**serial\_buf[bufpos] = input\_char;**

**bufpos++;**

**}**

**else if (input\_char == '\n')**

**{**

**serial\_buf[bufpos] = ',';**

**bufpos++;**

**if (bufpos > 0)**

**{**

**char\* w2b\_cmd = strtok(serial\_buf, ",");**

**char\* w2b\_num = strtok(NULL, "");**

**bufpos = 0;**

**uint8\_t cmd\_type = atoi(w2b\_cmd);**

**uint16\_t cmd\_num = atoi(w2b\_num);**

**// ---- Command #1: Set Kp1**

**//**

**if(cmd\_type == 0x1)**

**{**

**MC\_SetKp1(cmd\_num);**

**}**

**}**

**}**

}

Figure 9.17 Listing of the Command parser

1. The TM4C\_to\_MQTT routine in MQTT.c needs to be modified before it can be used.

**void TM4C\_to\_MQTT (void) {**

**char msp[24] = "";**

**sprintf(msp, "%d", MC\_GetLeftU());**

**sprintf(msp + strlen(msp),",");**

**sprintf(msp + strlen(msp),"%d", MC\_GetRightU());**

**sprintf(msp + strlen(msp),",");**

**sprintf(msp + strlen(msp),"%d", MC\_GetLeftE());**

**sprintf(msp + strlen(msp),",");**

**:**

:

:

**}**

Figure 9.18 Example of TM4C\_to\_MQTT CSV generation

1. Add calls to the corresponding initialization functions before interrupts are enabled.

## Deliverable 1

Create a schematic or figure showing all external components connected to the TM4C123 board on KiCad. You do not need to show hardware components on the TM4C123 LaunchPad board. Your schematic should include at least the TM4C, The ST7735, the LDO, and the ESP.

## Deliverable 2

Create a system call graph including all endpoints that you added for this lab. Your graph should show how time is updated, how time is displayed on the screen, and how data gets to the web interface.

## Deliverable 3

~~Take screenshots in the debugger showing incoming data dumps, and jitter measurements. This data should be collected using your Lab 2 dump.c code. Specifically, you should use:~~

1. **~~Dump\_Capture~~**~~() to record some of the outgoing data (Such as what value of second is being sent) in the~~ **~~TM4C\_to\_MQTT~~**~~() function calls.~~
2. **~~Jitter\_Measure~~**~~() to record the variance in timing of the~~ **~~MQTT\_to\_TM4C~~**~~() function calls.~~

## Deliverable 4

~~Take at least three pictures of the final alarm clock demonstrating the features of your system. These pictures should show that:~~

1. ~~The clock and the web interface show the same time after initialization. Use the MQTT monitor to show what data is coming in for the seconds.~~
2. ~~The unique features that you demonstrated in lab 3 are still present in the updated lab~~.

## Deliverable 5

~~Characterize the voltage and current on the 3V3 and 5V0 lines.~~

1. (~~In Lab) Disconnect the USB cable to the PC. If you have access to the lab bench supply, you can adjust the voltage output to +5V and connect it to VBUS on the LaunchPad. Verify 3.3V on the LaunchPad and 3.3V on the ESP8266. There should be between 40 and 200 mA of current on the 5V line, depending on what hardware you have connected and which software you are running. Take current measurements (both on 5V and on 3.3V to ESP8266) with~~ 
   1. ~~just the clock;~~
   2. ~~alarm on; and~~
   3. ~~during Wi-Fi communication.~~
2. ~~(Outside of Lab) Disconnect the USB cable to the PC. If you do NOT have access to the lab bench supply, you will remove the 2-pin jumper from the LaunchPad and connect a current meter between the pins. Reconnect the USB cable to the PC. Verify 3.3V on the LaunchPad and 3.3V on the ESP8266. There should be between 40 and 200 mA of current on the 5V line, depending on what hardware you have connected and which software you are running. Take current measurements (both on 5V and on 3.3V to ESP8266) with~~ 
   1. ~~just the clock;~~
   2. ~~alarm on; and~~
   3. ~~during Wi-Fi communication.~~

~~If you only have one current meter you can perform the experiment twice repeating the same conditions.~~

1. ~~Turn off power and remove the current meter(s). Connect the regulator output to the ESP8266 supply. Make sure the 2-pin jumper on the LaunchPad is connected and verify the regulator still works using a voltage measurement on the ESP8266 3.3V supply.~~

~~Document how you measured currents and report the current values (5V total and 3.3V to ESP8266) for (1) just the clock; (2) alarm on; and (3) during Wi-Fi.~~

## Deliverable 6 (5pts Extra Credit)

~~Interface any sensor to the ADC of the TM4C and have it report on the web interface. The type of sensor and interface is not important. A valid solution will have a code that calibrates the sensors to real results and reports these results in a human readable format to the end user. This measurement must be sent to Web App and be displayed along with the time.~~

~~One example is a distance sensor. Here, the values read by the ADC when interface with an IR sensor are recorded for several points, then a function is created based on these measurements to report the distance.~~

~~Another example is a digital protractor. Use a potentiometer interfaced to the ADC. Measure several angles and ADC values, then create a function/map that is able to report the angle to which the potentiometer has been turned.~~

# Lab Checkout

The lab checkout is performed during the M/T lab session.

~~Demonstrate that your system can control the Clock on the TM4C display data using the MQTT Web Application. Demonstrate that your system can read the sensor (extra credit) and time on the TM4C. Students should be able to display understanding of the data flow through the system and between the web interface.~~

# Lab Report

The lab report shall be submitted by the Friday after the second lab section.

You should complete the Lab9 Report.docx file with your data and answers then submit the completed file to Canvas.

# Hint (A deep dive into MQTT)

For a deep dive into the details of the MQTT protocol, there is a document in the “resources” folder of the lab starter code titled “MQTT and Lab 9.pdf”. This document has a detailed writeup and explanation of the MQTT protocol with examples of an implementation of it in an IOT system. Please refer to this resource to understand how the protocol works, as well as advice on how this is relevant to Lab 9.

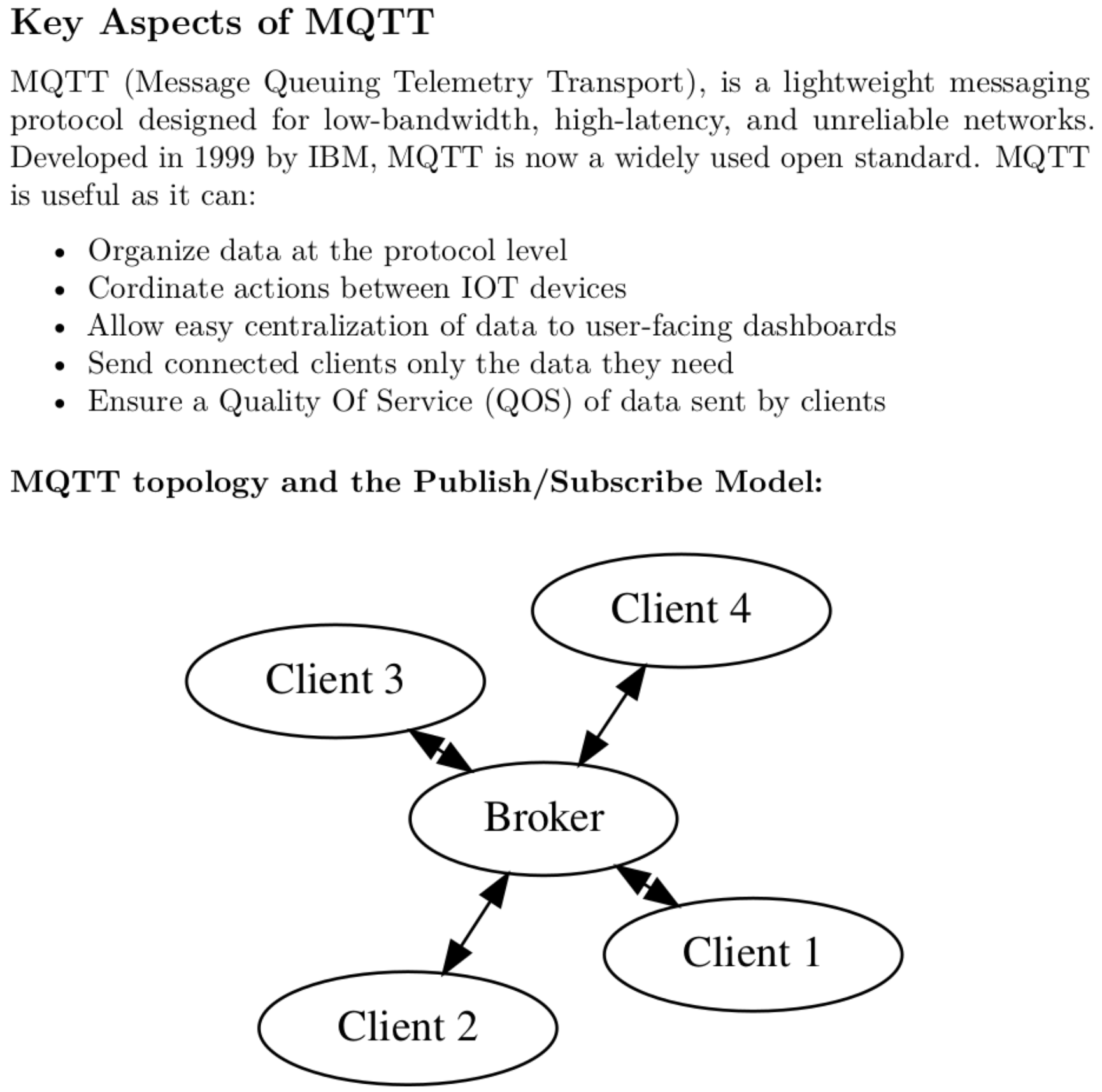


Figure 9.19: A screenshot of an excerpt from the MQTT writeup in the resource folder.

# Hint (Arduino Setup)

To use the ESP8266-ESP01 for this class, you will need to install the [Arduino IDE](https://docs.arduino.cc/software/ide-v2) (preferably version 2.0) to flash the .ino file provided in the starter files using the Arduino IDE. You will need the following dependencies to program the ESP8266, which we will show how to install:

* Adding the [ESP8266](https://randomnerdtutorials.com/how-to-install-esp8266-board-arduino-ide/) boards to the board manager
* Adding the following libraries to the library manager:
  + *PubSubClient* by Nick O’Leary
  + *Blynk* by Volodymyr Shymanskyy including dependencies.

### Steps:

1. To program the ESP8266 via the programmer, you must have board support in the Arduino IDE. In the Arduino IDE, navigate to File-->Preferences. In the Additional board manager URL’s, add the following URL:

<http://arduino.esp8266.com/stable/package_esp8266com_index.json>

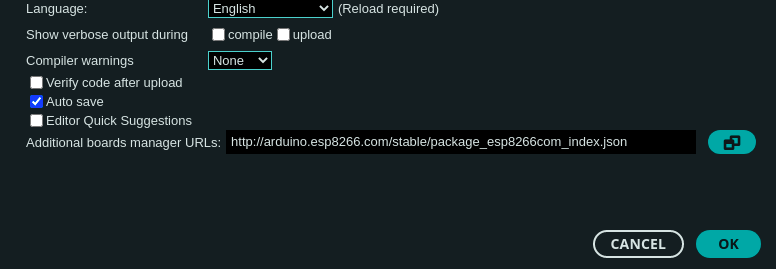


Figure 9.20: Adding Board Manager URL

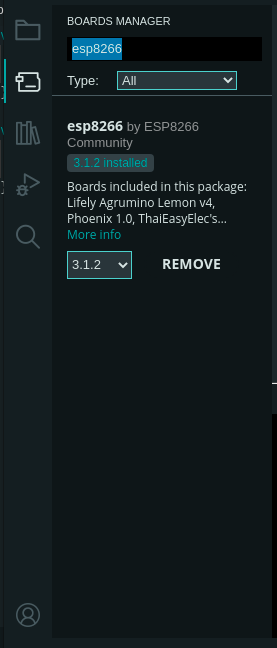
1. Open the Board Manager and search for “ESP8266”. If the previous step worked as it should, you should see the following package from ESP8266 Community. After installing, connect the programmer to your computer via USB and select the board as a Generic ESP8266 module.

Figure 9.21: Installing the ESP8266 Board Packages

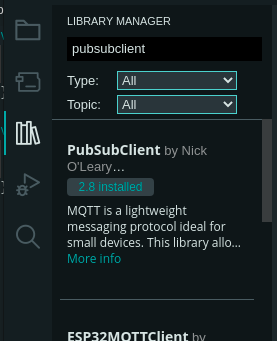
1. Install the MQTT Drivers: In the library manager tab, search for “pubsubclient”, install the package called PubSubClient by Nick O’Leary.

Figure 9.22: Installing MQTT Library

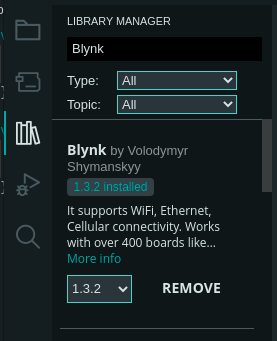
1. Install the Blynk Driver: In the library manager tab, search for “Blynk”. Install the library titled Blynk by Volodymyr Shymanskyy.

Figure 9.23: Installing Blynk Library

A screenshot of a computer

Description automatically generated

1. The following library (ArduinoJSON) needs to be installed so that the ESP6266 can convert the CSV packets to JSON packets:

Figure 9.24: Installing Arduino JSON library

1. To program the ESP, plug in the ESP into the programmer such that it is over the bent pins. Plug in the programmer, press the reset button on the programmer, then press D15 button twice to enter the ESP Bootloader. Within the Arduino IDE, press upload. If you are unable to connect to the board, your pins may be loose.

# Hint (MQTT Debugging)

There are more advanced monitors (such as MQTT\_Explorer) available if this homebrew tool is not adequate to debug an issue:

A screenshot of a computer screen

Description automatically generated

Figure 9.25: MQTT explorer.

## Browser Debugging

Use the WEB browser to receive console messages from the HTML/JS code. Embed console.log messages in your code: console.log("Connection Lost: " + responseObject.errorMessage);

A screenshot of a computer

Description automatically generated

Figure 9.26: Browser console.

## Internet Traffic Analysis & Debug

If the MQTT monitors are not able help debug an issue, the next step is to look at the internet traffic between the MQTT Broker and the Web App or the TM4C. You will need to use a TCP/IP analysis tool such as WireShark. You will need to know the IP address of the Broker the computer that is running the Web App. An example of a typical MQTT packet is shown below.

A screenshot of a computer

Description automatically generated

Figure 9.27: Wireshark frame.

The packet details are shown below. The subscribed data for this packet is “mcdermot/b2w”.

A screenshot of a computer

Description automatically generated

Figure 9.28: Hex packet data.

This corresponds to the subscription shown in the previous page.

A screenshot of a phone

Description automatically generated

Figure 9.29 Associated subscribe topic.

Refer to these three web pages for more details on how to use WireShark.

* Installation - instructions are [here](https://www.wireshark.org/docs/wsug_html_chunked/ChIntroPlatforms.html)
* WireShark - User Guide is [here](https://www.wireshark.org/download/docs/Wireshark%20User%27s%20Guide.pdf)
* WireShark - YouTube tutorial is [here](https://www.youtube.com/watch?v=TkCSr30UojM)

# Hint (Notable Project Files)

## LaunchPad Starter Project

The LaunchPad starter project contains the following modules:

* **Lab9Main.c** is a very basic template containing some basic initialization of the various modules used for Lab 9. Incorporate your Lab8 into this file.
* **MQTT.c** contains the TM4C\_to\_MQTT and MQTT\_to\_TM4C routines. This code is used to bridge the TM4C123 board and the MQTT Web Application via the ESP8266 Wi-Fi board. You will be extending the W2B parser, TM4C\_to\_MQTT, and MQTT\_to\_TM4C routines to send/receive and interpret commands between the TM4C and the MQTT WebApp. We communicate with the ESP via UART2. UART2 is used for PC debugging to see that data is sent to the ESP via serial terminal on your PC.
* **Timers:**
  + **Timer4A** is used to check subscriber data from MQTT Web App.
  + **Timer5A** is used to Publish data to the MQTT Web App.
  + If you use any of these timers for functionality in your Lab8, you will need to change those timers.
* **ESP8266 files**
  + You are given two separate files to flash to the esp8266: **esp8266\_fifo.c** and **esp8266.c**. These files will need to be merged in order to complete the lab. The file labeled **esp8266\_fifo.c** implements a low level UART interface which is required to send data between the esp8266 and the TM4C. The implementation is interrupt driven and uses a FIFO to buffer data to and from the 8266. Similarly, **esp8266.c** implements the MQTT interface that enables communication between the esp8266 and the MQTT broker. By merging these files, you will create the bridge that allows messages sent over UART by the TM4C to be sent to the MQTT broker.
* **UART.c** is an interface to PC for debugging (interrupt driven). Additionally, another UART channel is used to communicate with the esp8266.
* **Timers.c** contains code for the 3 system timers:
  + Timer0 is the 1ms (SysTick) PIT (Programmable Interval Timer).

## ESP8266 Starter Project

The ESP8266 contains a single starter file, **ESP\_to\_MQTT\_LAB\_9.ino**. Observe the following:

* Declaration of variables for authentication, MQTT connection, Publish and subscribe topics
* Routines for setting up WiFi that looks for auth data from the TM4C
* Routines for connecting to the MQTT broker and subscribing to a known topic (default is ‘<eid>*/w2b*’)
* Callback for receiving messages and sending it back to the TM4C
* Periodic function call to Publish any available TM4C data to the MQTT broker.

You will want to modify the high-level data variables that store your relevant topics and topic values, as well as relevant Publish calls in the function *tm4c2mqtt*.

## MQTT Web Application Starter Project

The MQTT Web Application (from Lab 4E) starter project contains the following modules:

* **Index.html** is the entry-point for the web application. Open this in your web browser.
* **clock\_ctl.js (motor\_ctl.js)** is a JavaScript that handles the MQTT WebSocket.It talks to the MQTT broker and performs UI and data logic.
* **mqtt\_app.htm** is the html script that displays the clocks and the buttons.
* **mqtt\_monitor.htm** is a simple MQTT Monitor that will help you with debugging.

You will need to modify motor\_ctl.js and mqtt\_app.htm for your application specifications.