# Lab5: Application Layer

## Introduction

For an IoT system, the Application layer is the main ‘variable’ in the system, since most IoT systems will use TCP/IP and either Ethernet, WiFi or one of the newer LoRa-type wireless systems.

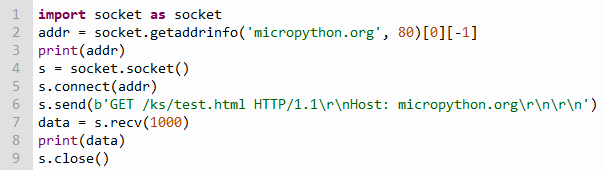
## Learning outcomes

1. Program/set up simple PC-based HTTP and MQTT systems and check operation
2. Analyse transactions using Wireshark

## Task 1: HTTP Connection

Since HTTP is text-based it is simple to write code to access an HTTP server. The example code below downloads a minimal web page and displays the HTML code in ASCII text. The website is micropython.org, which hosts a simple test page, test.html. Standard Python networking commands used throughout.

1. Open Thonny and enter and run the code – save it somewhere alongside your other Python programs. You will need network access on your machine.



1. Have a look at the HTML code and compare it to pointing your browser at the page, and seeing it rendered correctly (you will need to reassemble the URL from line 6).
2. We can get a better understanding (and a lot more detail) of what is happening by using Wireshark to capture the whole exchange of data between your program and the remote micropython.org website. Run Wireshark and use a display filter to ensure you only see exchanges with the IP address of micropython.org. Identify:
   1. Opening SYN/ACK TCP handshake
   2. HTTP GET request
   3. HTTP response from server. You can look at the raw HTML code by examining the ‘Line-based text data’ section
   4. The TCP close connection handshake.
3. Examine line 6 closely, noting the requirement that an HTTP1.1 request must provide a ‘Host’, and that the request must end with an empty line (two new lines). Try removing the ‘Host’ entry and note the result, especially the return code (4xx)  
     
   ‘No host’ error code =
4. Now try modifying the Host entry so that it does not relate to the actual host being addressed – try bbc.co.uk in the GET request. This security measure helps prevent ‘spoofing’ attacks. Note down the return code, 4xx  
     
   ‘Incorrect host’ error code =
5. Modify the program so that you can view the HTML code on the opening page of the UCLan or BBC web site. What problems arise, and why?
6. Line 6 sends a ‘GET’ request; in return the whole file specified in the request is included in the reply. One feature added with V1.x is the ‘HEAD’ request – replace ‘GET’ with ‘HEAD’ and suggest a purpose for request type.

## Task 2:MQTT Connection

This can best be done using an existing MQTT broker (server) running elsewhere.

**MQTT**

**Broker**

**Client**

**(your PC)**

Subscribe

Publish

(arrows show direction of message data)

We will use an Internet-based broker to begin with, and later on we will set up our own local broker.

IP address of MQTT broker: ­­­­­­­­­­­­­­­­­­­­­­\_\_\_\_5.196.95.208\_\_\_\_\_\_

1. **Installing the MQTT client and server on Linux**. The ‘mosquitto’ MQTT server and client software can be installed on Ubuntu from a terminal
   1. Run a terminal session. Write down your IP address for the session:  
        
      IP address:
   2. sudo apt update //you should always do this before installing anything new
   3. ppa
   4. sudo apt install mosquitto //this is the broker(server)
   5. sudo apt install mosquitto-clients //publish and subscribe clients
2. Open a second terminal session (you will need 2). You will use one to publish, the other to subscribe – decide which is which.
3. In the ‘subscribe’ window, we are going to subscribe to a topic called CM125/**xxx** (where xxx is your name, or some other identifiable unique string):  
     
   **mosquitto\_sub -h aaaa -t CM125/xxx** -d //aaaa is the IP host address of broker  
     
   -h = destination host -t = topic -d = debug mode

command must be exact, including spaces. See above for host address of broker to use. The ‘-d’ switch in the command above runs debugging messages, so that you can see what is happening at your end. Read these messages describing how the subscribing client is connecting and sending a subscribe message.

The subscribe process also causes messages to appear at the broker (which is in America – too far away to see). We will look at these messages later.  
  
Step back and think about what is happening here. A computer in America has been waiting all day long for you to contact it with a ‘subscribe’ message. It responds almost immediately because it is actively ‘listening’ on port 1883. You can see how busy a typical broker is by running your browser and going to <http://www.mqtt-dashboard.com/>

1. Use Wireshark to capture the ‘Subscribe’ message (use filters if necessary). Make sure you can identify the topic when you examine the MQ Telemetry Transport Protocol section.
2. In the publish window, you will publish some data on a topic (same topic as above) to the broker:

**mosquitto\_pub -h aaaa -t CM125/xxx -m Hello -d** // -m is message to send

Note that this results in up to three sets of data appearing on the terminal windows (although you can only see two, the publish and the subscribe):

**Publisher:**   
Sends connect,   
Receives connect ack,   
Sends publish message   
Disconnects

**Broker:**

Reports new client and connection,

Ack’s connection

Receives publish message

Sends published message to subscriber

(remains connected)

**Subscriber:**

Receives publish

Prints message

(remains connected)

At this stage both broker and subscribing client maintain a connection. From time to time, the client will send a ‘keep alive’ PINGREQ to the broker, and the broker will respond with a PINGRESP.

The publish command can be repeated by clicking on the publish window (to give it the focus for input) and then pressing the UP arrow key (to recall last command), followed by Enter. The message (or a modified message, if you choose) is published to the broker, and received by the subscriber.

Experiment by sending different (polite!) messages

1. Try to capture the request-publish cycle using Wireshark – note that your terminal publishing client will send a ‘publish’ message, and then the broker will also send a ‘publish message back to your subscribing client.
2. Team up with one other student (if possible), and exchange the topic names you decided upon. Send each other messages via the broker by modifying the publish command (the subscribe command can be left running on both machines). If you have two computers, you can exchange over your ‘home’ network.  
     
   Topics are strings that identify messages – there is no fixed format, but a hierarchy is normally implemented using the forward slash ‘/’. A home automation system might use a topic like:

myhouse/lounge/lights  
  
Topics must be globally unique if you are using a public broker.

1. You can run the broker on your PC (it has already been downloaded) in a new (3rd) window:  
     
   mosquitto -v //-v is verbose mode

This may fail, because by default a mosquito ‘daemon’ may be already running in the background (note the error message). We want to see ours running and printing messages, so we have to stop the background one using:

sudo /etc/init.d/mosquitto stop //(I don’t write this stuff!)

You should see a message confirming it is stopped. Now start mosquitto again as described above.

1. Try to lay out your terminal windows neatly, as you should now have three open. Stop the subscribing client (typing *control-C* into a window stops the running process). Change the host (-h) parameter in both the subscribe and publish commands to localhost (this machine) or *your machine ip address*. As you run them, note the messages appearing in the mosquito broker window.
2. If possible, exchange IP addresses with another student. Send and receive messages using their broker. Try to visualise what is happening here by sketch a diagram (showing computers, programs and IP addresses) of how you are sending them a message using their broker.
3. There is a broker at *broker.hivemq.com*, publishing messages on the following topic:

tele/mysonoff/SENSOR  
  
Try to subscribe to it and read the content.

NOTE: this may not be ‘up’ at the moment, but was confirmed running on 15/11/22

‘Mosquitto’ is a freely available broker. It can run on devices as small as a Raspberry Pi zero W. The Eclipse/Mosquitto web site <https://mosquitto.org/> says:



Eclipse Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers.

The MQTT protocol provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for Internet of Things messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers.