EE513: Assignment 2 2023/2024

[**Aims and Objectives**](#_bv3s02vrj2bs) **1**

[**Introduction to MQTT**](#_blb5bzlokh68) **1**

[**Assignment Tasks**](#_u8ormboxl2ej) **2**

[**Submission Requirements**](#_qtvhsk9gqumx) **4**

[**Supporting Materials**](#_aif2dw6xlqby) **4**

[Installing VirtualBox (help for new users)](#_9o72myjey6p2) 4

[Installing Mosquitto -- An MQTT Broker](#_hlnqhcrs6emz) 5

[Testing the MQTT Broker using mosquitto-clients](#_nhff82nomnbn) 6

[Adding a username/password to the mosquitto server](#_8pwrffus2i4p) 7

[Installing Paho on the Raspberry Pi/BBB](#_7y800fv8g9ey) 7

[A Paho Publisher Example in C (password based)](#_jq1ulrfnjquo) 8

[Listing 1. Code for publishing a message to the MQTT Broker](#_lf7azios6fk3) 8

[A Paho Subscriber Example in C (password based)](#_7cytabweniti) 10

[Listing 2. Code for subscribing to a MQTT Broker topic](#_50ub2kremxqn) 10

[Qt](#_s6uleorrzvnb) 12

[Installing Qt Creator](#_9nnvumgzouem) 12

[GL Error under VirtualBox](#_9oe18fdffsrb) 14

[Parsing JSON data within Qt](#_k807z0xh197f) 15

[Listing 3. Parsing JSON data in Qt.](#_pli83z10r5b6) 15

[The Qt Example Program](#_swsf50y1jn2h) 15

[Mqtt-spy](#_asxpnvoiqqp) 16

[ADXL345](#_xkd9j9owo9r9) 19

[NAT Port Forwarding](#_qzntpzd80l8z) 21

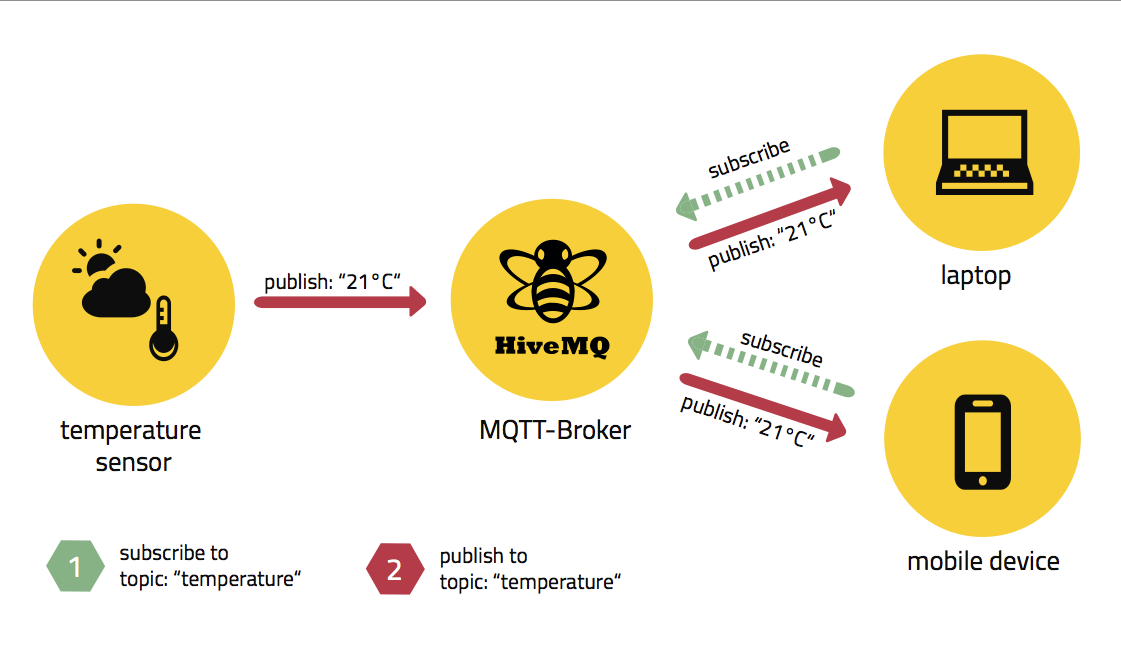
# Aims and Objectives

Internet Connected Embedded: In this assignment students must develop a full-stack IoT solution that interfaces to real-world sensors. This assignment is worth 15% of the overall module mark.

# Introduction to MQTT

Message Queue Telemetry Transport[[1]](#footnote-0) (MQTT) is a lightweight connectivity protocol for machine-to-machine (M2M) communications. It was conceived in 1999 and has been used by industry since then, however its applicability to the emerging IoT domain has placed it firmly in the spotlight, and in 2014 MQTT (version 3.1.1) became an OASIS standard. The lightweight nature of MQTT means that it can be used with low-level embedded devices and that it makes efficient use of network resources, while still providing reliable transactions. TCP/IP port 1883 is reserved for the MQTT protocol and 8883 is reserved for the protocol over SSL. In addition to SSL, MQTT supports username/password transactions.

With MQTT a client sends a connect message to a broker (never to another client) and the broker responds with an acknowledgement message and a status code (e.g., 0 for success, and 1–5 for different levels of failure). The connection then persists until the client disconnects. The client sends an MQTT packet that must contain a client ID, a clean session flag that indicates if a persistent session is to be created, and a keep alive time interval. The MQTT packet may also contain a username, password, and a last will message. The last will message can be used to notify other clients should this client be abruptly disconnected.



There are some useful guides on MQTT online. For example:

* http://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt

# Assignment Tasks

You should complete the following tasks to achieve full marks. You must write up your approach, implementation details, and testing/evaluation against the following tasks [100%]:

* **Establish an MQTT framework architecture**: it should consist of a mosquitto server (broker) that has been tested using the mosquitto-client tools. [10%]
  + Note that all payloads should be in JSON format for this architecture.
  + Adapt the server configuration to require a username/password.
  + In demonstrating that your framework works, use the -d option to display the full debug messages when using the mosquitto-client tools (not used below).
  + I have provided support for this task in the Support Materials section below.
* **Design and develop an MQTT publisher sensor application**: this application should run on the RPi/BBB and publish sensor data from the ADXL345: [30%]
  + It should read data from the sensor connected to your RPi/BBB and integrate time and CPU load/temperature from your RPi/BBB.
  + Demonstrate the use of a “last will” message under normal and abnormal disconnections.
  + Demonstrate the use of different MQTT QoS levels.
  + I have provided C code for a simple publisher application in the Support Materials below.
* **Design and develop multiple MQTT subscriber actuator applications**: these applications (minimum two) should run on the RPi/BBB and should process events (even actuate) based on the data that is published by the sensor application. [30%]
  + The events could be as simple as lighting an LED should a sensor value to which an application is subscribed exceeds a limit.
  + Demonstrate the use of persistent connections.
  + I have provided C code for a simple application in the Support Materials below.
* **Design and develop a Qt MQTT visualisation GUI application**: this application should use Qt to build a GUI that displays the sensor data that is being published in graphical form. [30%]
  + You should be able to manually choose the topic using the GUI.
  + The GUI should plot sensor data with respect to time (see http://www.qcustomplot.com/)
  + The Qt application should parse the JSON payload.
  + The application should demonstrate subject-based filtering.

**Please note that the application itself is up to you!** That is, the type of sensor data, the events you are triggering and the graphical display can be of your own design. For example, when using the accelerometer, the pitch and roll angles can be plotted and you can use threshold angle values to trigger events.

Please note that to achieve full marks in the sections above, each task needs to be complete, but it should also include good quality concise documentation, good quality code and a testing/evaluation. **You must use the Mosquitto Server and the C code listed in the Supporting Materials section below as the starting point for your solution.** In line with the plagiarism policy, any external references must be fully cited.

**Note on plagiarism:** Please note that you can work together to understand the assignment description, and/or to understand the course materials, but **you cannot work together to complete the assignment tasks**. The work completed **should be entirely your own** and I will be personally evaluating all submissions. All work is subject to interview, which can be arranged over Zoom.

# Submission Requirements

As part of your assignment submission, you must submit the following to Loop:

* A **single PDF document** that is structured against the task list above, which concisely describes the work performed and demonstrates an evaluation of your solution. You must also append your GitLab/GitHub commit history to your report.
* A **screencast video (3-5 mins MAX)** that demonstrates your system working. You must narrate the video to explain what you have achieved. The video must be in MP4 format. A screencast is video is sufficient but describe how you are manipulating the accelerometer as the output changes. You do not have to explain your code, just demonstrate the functionality. Please edit your video down if it is too long.
* A **Git code repository** which has been exported as a **.zip** file. Please see Assignment 1 for support documentation on this task. (You may receive an Urkund error message on uploading this file that can be safely ignored.)

Do not share any documents directly with me, e.g., Google Docs, Google sheets, code repositories, editing rights and please do not send me email versions.

The submission due date is **Monday the 15th April 2024 at 10am[[2]](#footnote-1).**

Late submissions will be reduced in the following way:

* 5% (gross) reduction for 1 minute to 1 hour late.
* 10% (gross) reduction for 1 hour to 24 hours late.
* 10% (gross) additional deduction for each day late[[3]](#footnote-2).
* No submission permitted 7 days after the assignment deadline.

Please note that you can submit your work multiple times before the deadline, which will help you avoid late penalties.

Please use the main discussion forum if you have any questions on the assignment. Do not paste your code! If you know the answer to someone else's question then please feel free to respond.

# Supporting Materials

## Installing VirtualBox (help for new users)

Please note that VirtualBox is one option to facilitate running Linux on a desktop PC and hosting the Mosquitto server. If you are familiar with other tools like VMWare/Docker, or are using Linux for your day-to-day-computing then use your preferred option.

Desktop virtualization enables a single desktop computer to run multiple OS instances simultaneously. It uses technology called hypervisors, which consist of hardware, firmware, and software elements, to create and run software-emulated machines, which are termed virtual machines (VMs). If you wish to run multiple OS instances on a single computer, VMs provide an alternative to creating a multi-boot configuration.

****

In virtualization, there are usually two or more distinct OS instances. The host OS is the one that was first installed on the physical machine. The hypervisor software is then used to create a guest OS within a virtual machine. If you install the VirtualBox “Guest Additions” you are able to copy-and-paste text between your guest and host OSs, share directories, and even resize the window dynamically. I recommend that you then install a Debian 64-bit desktop distribution, as this aligns with the RPi and BBB images.

Here are some tips on installing a Linux guest VirtualBox VM under a Windows host:

* Enable VT-x/AMD-V in the system BIOS settings of your physical computer in order to run a 64-bit guest OS.
* Install VirtualBox (virtualbox.org) on your host OS.
* Download your favourite Linux distribution as an ISO file. 64-bit Debian is often used for embedded development.
  + Debian Bullseye (Dec 2022) Stable: Download the netinst CD image or equivalent from the page https://www.debian.org/releases/stable/debian-installer/. Do not use the ia64 branch unless you are using a high-end Itanium-based server.
  + Ubuntu: Download an ISO image of Ubuntu LTS or equivalent from https://ubuntu.com/download/desktop
* Start Oracle VM VirtualBox Manager and choose New. If available, give the installation 2GB+ memory. Create a virtual hard drive (VDI format, dynamic allocated) and make the size much greater than the 8GB it recommends. Choose the location on your computer to store the disk and use a minimum of 25–30GB! It is difficult to resize a disk after the fact. If you choose “dynamically allocated”, it will only occupy real disk space when you fill the Linux disk, so feel free to choose a large disk size.
* You now have a new VM that is powered off. Select it and click on settings. Under storage you should see your VDI (under SATA) and an “Empty” entry (under IDE). Select the “Empty” entry and click the disk icon under Attributes. Find and select the Linux ISO file that you downloaded.
* Start your VM and install Linux by booting off the virtual drive. If you see a message “No Boot Device,” go to Settings System and disable the floppy drive.
* Once your guest OS is running, choose Devices Install Guest Additions from the menu bar of your VirtualBox window. If all goes well, you will now be able to resize the window and cut-and-paste text between the host and guest OSs (by enabling Devices ⇒ Shared ⇒ Clipboard Bidirectional). Choose “Devices”->”Insert Guest Additions CD image…”
* If the guest additions do not install automatically, follow these steps (for Debian desktop) in a terminal window:  
  molloyd@debian:~$ **su -**  
  root@debian:~# **apt update**  
  root@debian:~# **apt install build-essential module-assistant**  
  root@debian:~# **m-a prepare**  
  root@debian:~# **cd /media/cdrom**  
  root@debian:~# **sh ./VBoxLinuxAdditions.run**  
  root@debian:~# **reboot**

## Installing Mosquitto -- An MQTT Broker

You will need to install the Mosquitto server (it used to be called a Mosquitto broker) on your desktop machine for this assignment. If you are running Windows, it is possible to install the mosquitto server directly without Linux. However, I strongly recommend you install VirtualBox and a Debian or Ubuntu guest OS on your machine if you are running Windows or Mac OS.

Under Debian/Ubuntu you can install the server quite easily (always remember to perform an update before performing any install) using:

molloyd@desktop:~$ **sudo apt update**

molloyd@desktop:~$ **sudo apt install mosquitto**

[sudo] password for molloyd:

Reading package lists... Done

….

Processing triggers for systemd (215-17+deb8u3) …

molloyd@desktop:~$ **sudo apt install mosquitto-clients net-tools**  
Reading package lists... Done  
...  
Setting up mosquitto-clients (1.3.4-2) ...  
Processing triggers for libc-bin (2.19-18+deb8u3) …

The server will usually be executed automatically. In which case you should be able see that the server is bound to port 1883 on your Linux desktop machine:

molloyd@desktop:~$ **netstat -at**

Active Internet connections (servers and established)

Proto Recv-Q Send-Q Local Address Foreign Address State

tcp 0 0 localhost:smtp \*:\* LISTEN

**tcp 0 0 \*:1883 \*:\* LISTEN**

tcp 0 0 \*:sunrpc \*:\* LISTEN

tcp 0 0 \*:39701 \*:\* LISTEN

tcp6 0 0 localhost:smtp [::]:\* LISTEN

tcp6 0 0 [::]:56921 [::]:\* LISTEN

**tcp6 0 0 [::]:1883 [::]:\* LISTEN**

tcp6 0 0 [::]:sunrpc [::]:\* LISTEN

tcp6 0 0 localhost:51880 localhost:1883 TIME\_WAIT

If it does not start automatically you should be able to start it using sudo start mosquitto. The configuration files for the mosquitto server are as below should you need them.

molloyd@desktop:~$ **cd /etc/mosquitto**

molloyd@desktop:/etc/mosquitto$ **ls -l**

total 16

drwxr-xr-x 2 root root 4096 Mar 17 11:42 ca\_certificates

drwxr-xr-x 2 root root 4096 Mar 17 11:42 certs

drwxr-xr-x 2 root root 4096 Mar 17 11:42 conf.d

-rw-r--r-- 1 root root 348 Aug 16 2014 mosquitto.conf

And the log file is at /var/log/mosquitto/mosquitto.log if you need to identify server problems.

## Testing the MQTT Broker using mosquitto-clients

In a first terminal window, use the mosquitto subscriber client to subscribe to a topic “ee513/test”. This tool defaults to a server running on localhost:

molloyd@desktop:~$ **mosquitto\_sub -v -t 'ee513/test'**

You will not see any output until a message is published on this topic. To do this you can used the mosquitto publisher client. So, in a second terminal window you can publish a message on the topic:

molloyd@desktop:~$ **mosquitto\_pub -t 'ee513/test' -m 'Hello World'**

molloyd@desktop:~$

If the mosquitto server (broker) is working correctly then you should see the output change in the first terminal window. It should be updated to:

molloyd@desktop:~$ **mosquitto\_sub -v -t 'ee513/test'**

ee513/test Hello World

Any further messages sent by the publisher client will also appear in the server window. If you see the message above then your server is working correctly and you have tools that you can use to debug communications.

You will need to identify the IP address of your Virtual Machine so that you can communicate to your MQTT Broker from your Raspberry Pi. You can do this by making a call to ifconfig (or to /sbin/ifconfig). In my case the IP address of my desktop machine is 192.168.1.19.

## Adding a username/password to the mosquitto server

You need to edit the configuration file for the mosquitto server. The following should be of some assistance:

molloyd@desktop:/etc/mosquitto$ **tail -2 mosquitto.conf**

password\_file /etc/mosquitto/passwd

allow\_anonymous false

molloyd@desktop:/etc/mosquitto$ **sudo mosquitto\_passwd -c passwd molloyd**

Password:

Reenter password:

molloyd@desktop:/etc/mosquitto$ **more passwd**   
molloyd:$6$6QR+2DqWP41rYh0d$zWiR7qhvkzw1fEjigmLdDZh4sN+nmFpH72I9jR9NQnAufkqE6vv  
uT6vQkeWAxtLEyrmAOV7EU+ySsklP5YUJVg==

molloyd@desktop:/etc/mosquitto$ **sudo systemctl restart mosquitto**

At this point the previous test applications should fail to connect to the server. For example:

molloyd@desktop:~$ **mosquitto\_sub -t 'ee513/test'**

Connection Refused: not authorised.

You can use the tools by providing the usernames and passwords (yes, my password is password!):

molloyd@desktop:~$ **mosquitto\_sub -d -u molloyd -P password -t 'ee513/test'**

## Installing Paho on the Raspberry Pi/BBB

Paho is an open source <http://www.eclipse.org/paho/> client implementation of the MQTT and MQTT-SN messaging protocols for IoT applications. Paho provides open-source implementations of MQTT in C/C++, Java, Python, Javascript, and other languages that can be used to build small footprint reliable MQTT client applications. In addition, the Eclipse IoT Working Group (iot.eclipse.org) provides strong support documentation and tools for developing open-source IoT solutions.

You can install Paho directly on your RPi/BBB as follows:

pi@raspberrypi:~ $ **sudo apt install libssl-dev git**

Reading package lists... Done

Building dependency tree

Reading state information…

pi@raspberrypi:~ $ **git clone https://github.com/eclipse/paho.mqtt.c**

Cloning into 'paho.mqtt.c'...

remote: Counting objects: 2647, done.

remote: Compressing objects: 100% (89/89), done.

remote: Total 2647 (delta 56), reused 2 (delta 2), pack-reused 2556

Receiving objects: 100% (2647/2647), 1.03 MiB | 236.00 KiB/s, done.

Resolving deltas: 100% (1860/1860), done.

Checking connectivity... done.

pi@raspberrypi:~ $ **cd paho.mqtt.c/**

pi@raspberrypi:~/paho.mqtt.c $ **make**

…

cc -g -o build/output/test/test5 src/../test/test5.c -lpaho-mqtt3as -I src -Wl,--start-group -lpthread -lssl -lcrypto -Wl,--end-group -L build/output

There may be warnings during the make process but there should be no errors.

pi@raspberrypi:~/paho.mqtt.c $ **sudo make install**

mkdir -p build/output/samples

mkdir -p build/output/test

…

You can test that the libraries are in the correct location using:

pi@raspberrypi:~/paho.mqtt.c $ **ls /usr/local/lib/libpaho\***

/usr/local/lib/libpaho-mqtt3a.so /usr/local/lib/libpaho-mqtt3c.so

/usr/local/lib/libpaho-mqtt3a.so.1 /usr/local/lib/libpaho-mqtt3c.so.1

/usr/local/lib/libpaho-mqtt3a.so.1.0 /usr/local/lib/libpaho-mqtt3c.so.1.0

/usr/local/lib/libpaho-mqtt3as.so /usr/local/lib/libpaho-mqtt3cs.so

/usr/local/lib/libpaho-mqtt3as.so.1 /usr/local/lib/libpaho-mqtt3cs.so.1

/usr/local/lib/libpaho-mqtt3as.so.1.0 /usr/local/lib/libpaho-mqtt3cs.so.1.0

## A Paho Publisher Example in C (password based)

The next step is to write code for your RPi/BBB that publishes data to your MQTT broker. To do this you can use the example code below to publish a message to your MQTT broker. In this example, my broker is running at 192.168.1.19 and the username/password for the broker is molloyd/password.

The full explanation of all of the Paho function calls and data structures is available at: <https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/index.html>

This example reads the CPU temperature, packages it up in JSON format and sends the message to the MQTT Broker under the topic heading ee513/CPUTemp.

### Listing 1. Code for publishing a message to the MQTT Broker

// Based on the Paho C code example from www.eclipse.org/paho/

#include <iostream>

#include <sstream>

#include <fstream>

#include <string.h>

#include "MQTTClient.h"

#define CPU\_TEMP "/sys/class/thermal/thermal\_zone0/temp"

using namespace std;

//Please replace the following address with the address of your server

#define ADDRESS "tcp://192.168.1.19:1883"

#define CLIENTID "rpi1"

#define AUTHMETHOD "molloyd"

#define AUTHTOKEN "password"

#define TOPIC "ee513/CPUTemp"

#define QOS 1

#define TIMEOUT 10000L

float getCPUTemperature() { // get the CPU temperature

int cpuTemp; // store as an int

fstream fs;

fs.open(CPU\_TEMP, fstream::in); // read from the file

fs >> cpuTemp;

fs.close();

return (((float)cpuTemp)/1000);

}

int main(int argc, char\* argv[]) {

char str\_payload[100]; // Set your max message size here

MQTTClient client;

MQTTClient\_connectOptions opts = MQTTClient\_connectOptions\_initializer;

MQTTClient\_message pubmsg = MQTTClient\_message\_initializer;

MQTTClient\_deliveryToken token;

MQTTClient\_create(&client, ADDRESS, CLIENTID, MQTTCLIENT\_PERSISTENCE\_NONE, NULL);

opts.keepAliveInterval = 20;

opts.cleansession = 1;

opts.username = AUTHMETHOD;

opts.password = AUTHTOKEN;

int rc;

if ((rc = MQTTClient\_connect(client, &opts)) != MQTTCLIENT\_SUCCESS) {

cout << "Failed to connect, return code " << rc << endl;

return -1;

}

sprintf(str\_payload, "{\"d\":{\"CPUTemp\": %f }}", getCPUTemperature());

pubmsg.payload = str\_payload;

pubmsg.payloadlen = strlen(str\_payload);

pubmsg.qos = QOS;

pubmsg.retained = 0;

MQTTClient\_publishMessage(client, TOPIC, &pubmsg, &token);

cout << "Waiting for up to " << (int)(TIMEOUT/1000) <<

" seconds for publication of " << str\_payload <<

" \non topic " << TOPIC << " for ClientID: " << CLIENTID << endl;

rc = MQTTClient\_waitForCompletion(client, token, TIMEOUT);

cout << "Message with token " << (int)token << " delivered." << endl;

MQTTClient\_disconnect(client, 10000);

MQTTClient\_destroy(&client);

return rc;

}

You can build this program directly on your RPi using the following commands (remember to install Paho as above first of all):

pi@raspberrypi:~/ass2 $ **nano publish.cpp**

pi@raspberrypi:~/ass2 $ **g++ publish.cpp -o publish -lpaho-mqtt3c**

pi@raspberrypi:~/ass2 $ **./publish**

Failed to connect, return code 5

A return code of 5 indicates that the username/password is incorrect. If necessary, correct your code and try again. If you get an output of -1 check the IP address of your server again and check that the mosquitto server is running. Then:

pi@raspberrypi:~/ass2 $ **./publish**

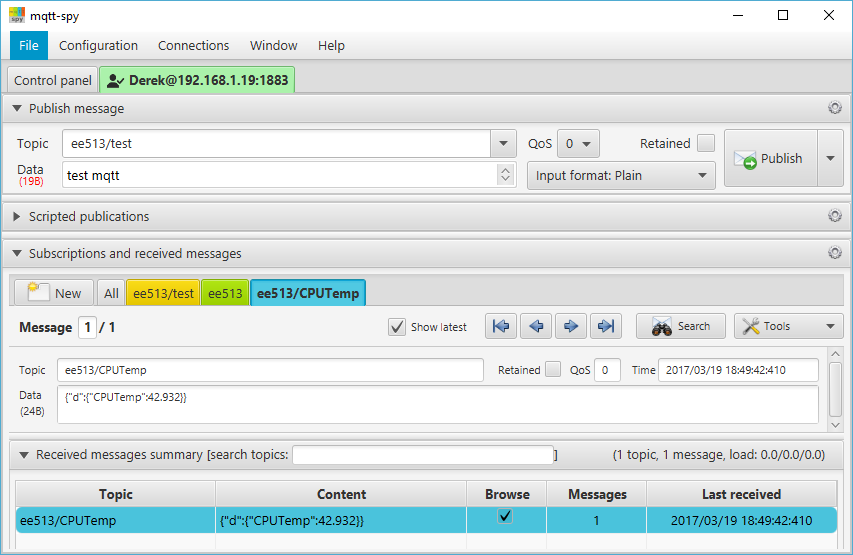
Waiting for up to 10 seconds for publication of {"d":{"CPUTemp":44.007}}

on topic ee513/CPUTemp for ClientID: rpi1

Message with token 1 delivered.

The data is sent in JavaScript Object Notation (JSON) format (e.g., {"d": {"CPUTemp" : 32.552}}. The “d” value identifies the client as a device, which is necessary for some services such as IBM Thingspeak.

If you have installed mqtt-spy, [MQTTLens](https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkbcookmlgmdigohjobjm?hl=en), or a similar application, you will see the output of this message in the window, as illustrated in Figure 1. The installation of this tool is described at the end of the Support Materials section.



**Figure 1**. mqtt-spy receiving the message send by the publish.cpp program.

## A Paho Subscriber Example in C (password based)

The next step is to write code for your RPi/BBB that subscribes to a data feed from your MQTT broker. You can use the example code below to subscribe to a topic on your MQTT broker. In this example, my broker is running at 192.168.1.19 and the username/password for the broker is molloyd/password.

It is best if you work on this by opening a second SSH connection to your RPi/BBB.

### Listing 2. Code for subscribing to a MQTT Broker topic

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

#include "MQTTClient.h"

#define ADDRESS "tcp://192.168.1.19:1883"

#define CLIENTID "rpi2"

#define AUTHMETHOD "molloyd"

#define AUTHTOKEN "password"

#define TOPIC "ee513/CPUTemp"

#define PAYLOAD "Hello World!"

#define QOS 1

#define TIMEOUT 10000L

volatile MQTTClient\_deliveryToken deliveredtoken;

void delivered(void \*context, MQTTClient\_deliveryToken dt) {

printf("Message with token value %d delivery confirmed\n", dt);

deliveredtoken = dt;

}

int msgarrvd(void \*context, char \*topicName, int topicLen, MQTTClient\_message \*message) {

int i;

char\* payloadptr;

printf("Message arrived\n");

printf(" topic: %s\n", topicName);

printf(" message: ");

payloadptr = (char\*) message->payload;

for(i=0; i<message->payloadlen; i++) {

putchar(\*payloadptr++);

}

putchar('\n');

MQTTClient\_freeMessage(&message);

MQTTClient\_free(topicName);

return 1;

}

void connlost(void \*context, char \*cause) {

printf("\nConnection lost\n");

printf(" cause: %s\n", cause);

}

int main(int argc, char\* argv[]) {

MQTTClient client;

MQTTClient\_connectOptions opts = MQTTClient\_connectOptions\_initializer;

int rc;

int ch;

MQTTClient\_create(&client, ADDRESS, CLIENTID, MQTTCLIENT\_PERSISTENCE\_NONE, NULL);

opts.keepAliveInterval = 20;

opts.cleansession = 1;

opts.username = AUTHMETHOD;

opts.password = AUTHTOKEN;

MQTTClient\_setCallbacks(client, NULL, connlost, msgarrvd, delivered);

if ((rc = MQTTClient\_connect(client, &opts)) != MQTTCLIENT\_SUCCESS) {

printf("Failed to connect, return code %d\n", rc);

exit(-1);

}

printf("Subscribing to topic %s\nfor client %s using QoS%d\n\n"

"Press Q<Enter> to quit\n\n", TOPIC, CLIENTID, QOS);

MQTTClient\_subscribe(client, TOPIC, QOS);

do {

ch = getchar();

} while(ch!='Q' && ch != 'q');

MQTTClient\_disconnect(client, 10000);

MQTTClient\_destroy(&client);

return rc;

}

pi@raspberrypi:~/ass2 $ **nano subscribe.cpp**

pi@raspberrypi:~/ass2 $ **g++ subscribe.cpp -o subscribe -lpaho-mqtt3c**

pi@raspberrypi:~/ass2 $ **./subscribe**

Subscribing to topic ee513/CPUTemp

for client rpi2 using QoS1

Press Q<Enter> to quit

Message arrived

topic: ee513/CPUTemp

message: {"d":{"CPUTemp":45.084}}

Message arrived

topic: ee513/CPUTemp

message: {"d":{"CPUTemp":44.546}}

Further documentation on the Paho MQTT C library is available at:

* <https://www.eclipse.org/paho/files/mqttdoc/Cclient/index.html>

## Qt

Qt is a full cross-platform development framework that is written in C/C++. It is used in for UI programming, but it also provides support for databases, threads, timers, networking, multimedia, XML processing, and more. Qt extends C++ by adding macros and introspection, code that examines the type and properties of an object at run time, which is not natively available in C++. It is important to note that *all the code is still just plain C++*!

### Installing Qt Creator

You can install Qt Creator on your desktop machine. You can do this within the VirtualBox Linux guest or the Window/Mac OS host OS, **however for this assignment you should install Qt Creator under Linux in the VirtualBox image**. You should be careful to download the open source version of Qt Creator from the address:

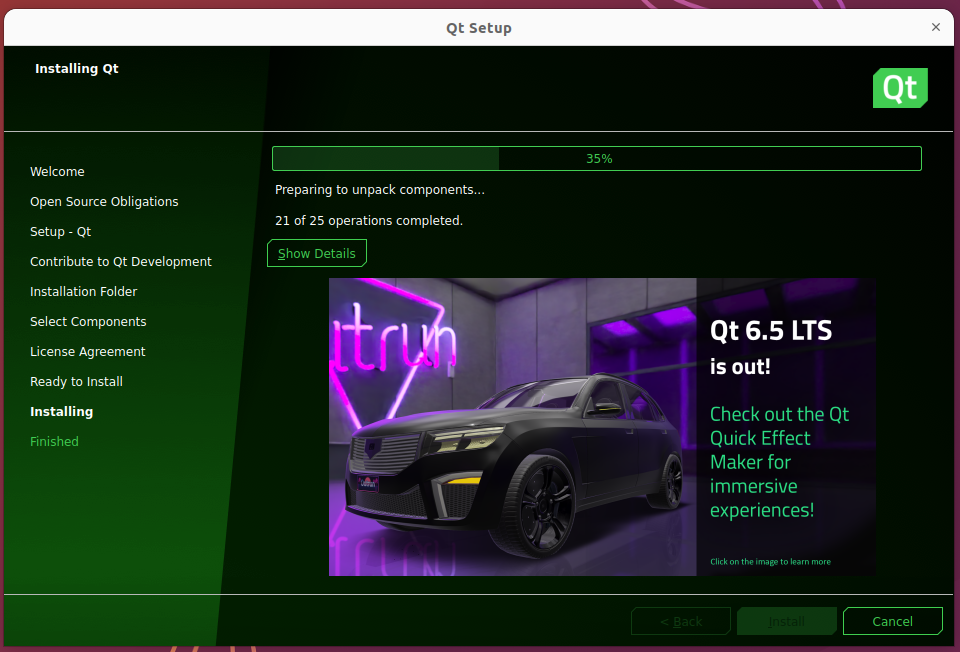
<https://www.qt.io/download-open-source/>.

There are many licensing options for Qt Creator, so be careful.

You may have to make the downloaded installer executable, as follows:

| molloyd@x:~/Downloads$ chmod u+x qt-unified-linux-x64-4.5.2-online.run  molloyd@x:~/Downloads$ ./qt-unified-linux-x64-4.5.2-online.run |
| --- |

Choose the default options.. The overall install is large and will take quite some time.



**Figure 3**. The Installer in action

Once Qt Creator is installed you can choose to create a new project. Choose a Qt Widgets Application and a suitable location for the project. You should then choose a Kit, as in Figure 4. You only need to choose one of the kits for your application.

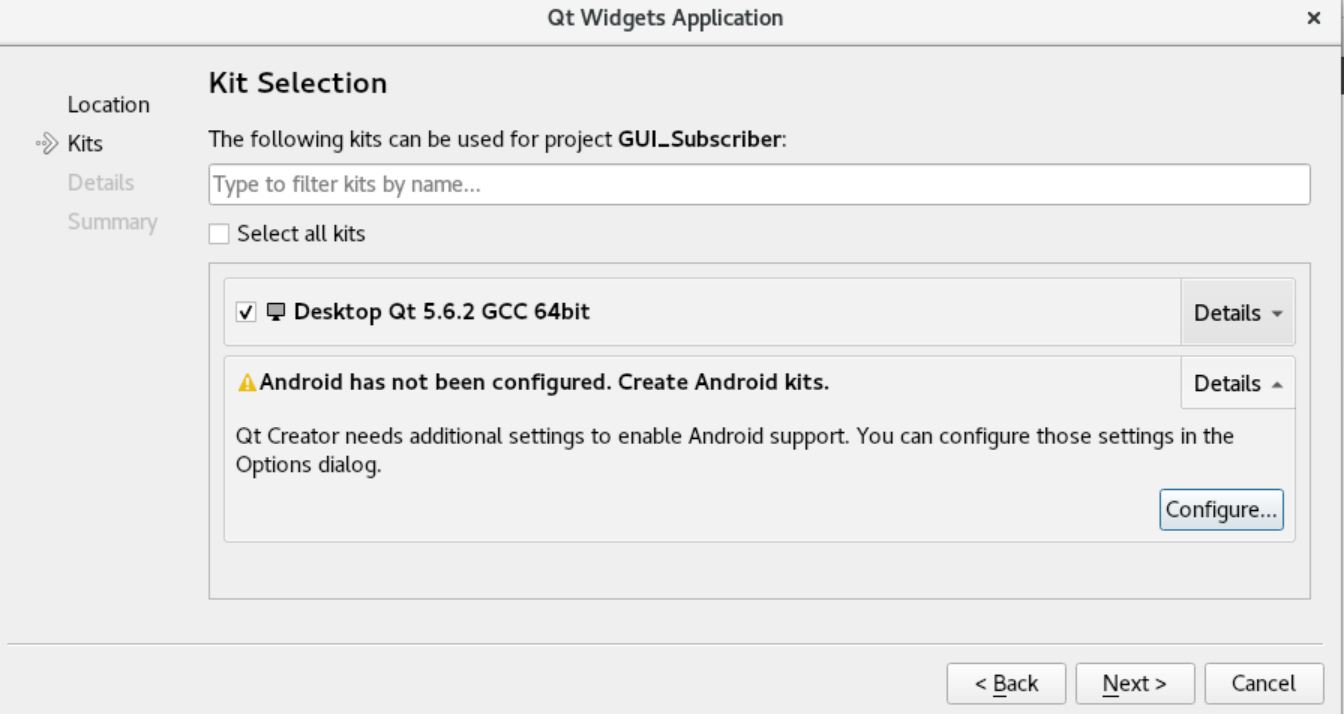
If you have problems with this graphical installer, I suggest you try the following to use a command-line only method:

| **molloyd@OfficePC-Ubuntu:~$ sudo apt install python3-pip molloyd@OfficePC-Ubuntu:~$ pip install -U pip molloyd@OfficePC-Ubuntu:~$ sudo pip install aqtinstall molloyd@OfficePC-Ubuntu:~$ aqt list-qt linux desktop .... shows all of the qt versions available molloyd@OfficePC-Ubuntu:~$ sudo aqt install-qt linux desktop 5.13.2 -m qtcharts molloyd@OfficePC-Ubuntu:~$ sudo apt install qtcreator** |
| --- |

I suggest that you use Qt5, not Qt6 as above. Then clone the assignment repository:

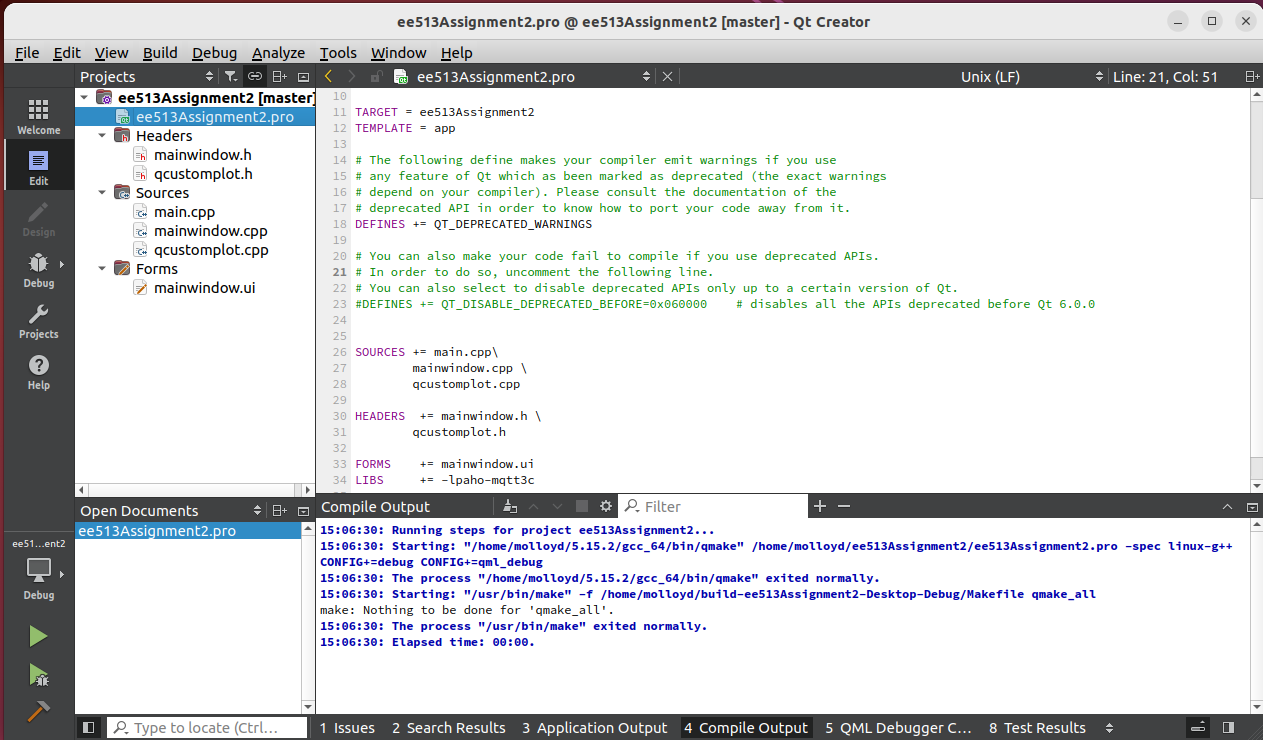
| molloyd@OfficePC-Ubuntu:~$ git clone https://github.com/derekmolloy/ee513Assignment2 Cloning into 'ee513Assignment2'... remote: Enumerating objects: 29, done. remote: Total 29 (delta 0), reused 0 (delta 0), pack-reused 29 Receiving objects: 100% (29/29), 288.37 KiB | 2.34 MiB/s, done. Resolving deltas: 100% (10/10), done. molloyd@OfficePC-Ubuntu:~$ cd ee513Assignment2/ molloyd@OfficePC-Ubuntu:~/ee513Assignment2$ rm ee513Assignment2.pro.user  molloyd@OfficePC-Ubuntu:~$ qtcreator & |
| --- |

It’s important that you delete the pro.user file, but not the pro file. I will remove this from the repository, so that step may no longer be required.

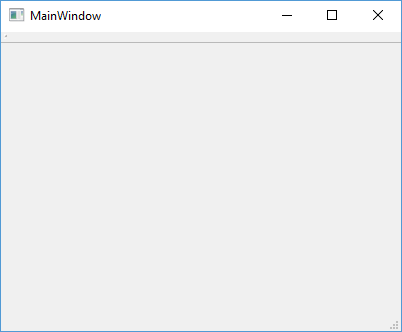


**Figure 4**. Choosing the Kit. Just choose one for your new project.

You will now have the Qt Creator main window (as illustrated in Figure 5) along with a default project that displays a blank window on the screen (see Figure 6). You now have everything you need in order to build a Qt project



**Figure 5**. The Qt Creator Main Development IDE running in VirtualBox



**Figure 6**. The default application window that appears when the project is executed

### GL Error under VirtualBox

If you get an error when you execute your application under VirtualBox in linking your application of the for “cannot find -lGL” then you need to open a terminal and use:

molloyd@debian:~$ **sudo apt install libgl1-mesa-dev**

[sudo] password for molloyd:

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following extra packages will be installed:

libdrm-dev libpthread-stubs0-dev libx11-dev libx11-doc…

Your application should now work without having to leave and reenter Qt Creator.

### Parsing JSON data within Qt

The assignment requires that all payloads are in JSON format. Usefully, the Qt framework has full support for parsing JSON data using the QJsonDocument class. Listing 3 below is a segment of code that parses JSON data in the format:

{

"sample": {

"temperature" : 18.2,

"humidity": 45.4

}

}

and retrieves the floating-point temperature and humidity values. By converting the byte data into an sample object of the QJsonObject class, the data values can be retrieved by calling sample["name"].toDouble(), where name is the string name of the value to be retrieved. There are similar functions for other data types, for example toInt(), toString(), toBool(), and toArray().

#### Listing 3. Parsing JSON data in Qt.

int MainWindow::parseJSONData(QString str){

QJsonDocument doc = QJsonDocument::fromJson(str.toUtf8());

QJsonObject obj = doc.object();

QJsonObject sample = obj["sample"].toObject();

this->temperature = (float) sample["temperature"].toDouble();

this->humidity = (float) sample["humidity"].toDouble();

cout << "The temperature is " << temperature << " and humidity is "

<< humidity << endl;

return 0;

}

### Parsing JSON data in a C program (non Qt -- e.g., for the Pi actuator)

Please note that you can use whatever format of JSON messages that you like. You do not need to use the {"d" : {"CPUTemp" : 21.54 }} format that I use in the examples. You can use something more simple: { "CPUTemp" : 21.54 } for example.

Please keep the construction and parsing of the data very simple. I have given you the code for Qt and I would suggest that you use Json-C to parse the code for the actuator nodes (or parse it yourself using string functions).

If you would like to do this more neatly then please see: <https://linuxprograms.wordpress.com/2010/05/20/json-c-libjson-tutorial/> for help.

You can use the C libjson library. This will be something like:

$ sudo apt install libjson0 libjson0-dev

or:

$ sudo apt-get install libjson-glib-1.0-0 libjson-glib-dev

or something close to that...!

If you are having difficulties (as I just had under Debian), you can build it yourself using the instructions at: <https://github.com/json-c/json-c>

A program example to generate a JSON string:

#include <json/json.h>

#include <stdio.h>

int main() {

/\*Creating a json object\*/

json\_object \* jobj = json\_object\_new\_object();

/\*Creating a json double\*/

json\_object \*jdouble = json\_object\_new\_double(3.14);

/\*Form the json object\*/

json\_object\_object\_add(jobj,"PI", jdouble);

/\*Now printing the json object\*/

printf ("The json object created: %sn",json\_object\_to\_json\_string(jobj));

}

$ **gcc test.c -o test -ljson**

$ **./test**

The json object created: { "PI": 3.140000 }

A program to read in a JSON double value from the same string could be something like this (compile it in the same way with -ljson):

#include <json/json.h>

#include <stdio.h>

void json\_parse(json\_object \* jobj) {

enum json\_type type;

json\_object\_object\_foreach(jobj, key, val) {

type = json\_object\_get\_type(val);

switch (type) {

case json\_type\_double: printf("type: json\_type\_double, ");

printf("value: %fn", json\_object\_get\_double(val));

break;

}

}

}

int main() {

char \* string = "{ \"PI\" : 3.140000 }";

printf ("JSON string is: %s\n", string);

json\_object \* jobj = json\_tokener\_parse(string);

json\_parse(jobj);

}

$ **gcc testread.c -o testread -ljson**

$ **./testread**

JSON String is: { "PI" : 3.140000 }

ntype: json\_type\_double, value: 3.140000

## The Qt Example Program

The Qt Example MQTT Program can be downloaded from GitHub using the following commands. You may need to sudo apt install git the first time you use git on your Linux virtual machine. **Please note that this example assumes that you have installed paho on the Linux virtual machine exactly as described in this document**. When executed on the Linux Virtual Machine, the window should appear as illustrated in Figure 7. You can also view the code online at: <https://github.com/derekmolloy/ee513Assignment2>

molloyd@debian:~/temp$ **git clone** <https://github.com/derekmolloy/ee513Assignment2>

Cloning into 'ee513Assignment2'...

remote: Counting objects: 15, done.

remote: Compressing objects: 100% (12/12), done.

remote: Total 15 (delta 1), reused 12 (delta 1), pack-reused 0

Unpacking objects: 100% (15/15), done.

Checking connectivity... done.

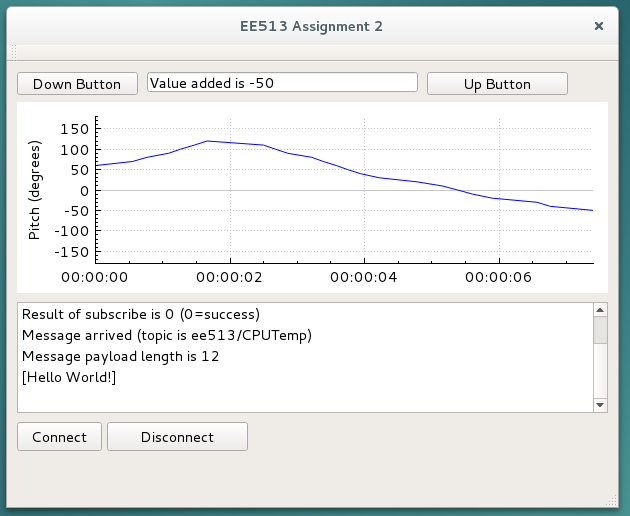
molloyd@debian:~/temp$ **cd ee513Assignment2/**

molloyd@debian:~/temp/ee513Assignment2$ **ls**

ee513Assignment2.pro mainwindow.cpp qcustomplot.cpp

ee513Assignment2.pro.user mainwindow.h qcustomplot.h

main.cpp mainwindow.ui README.md



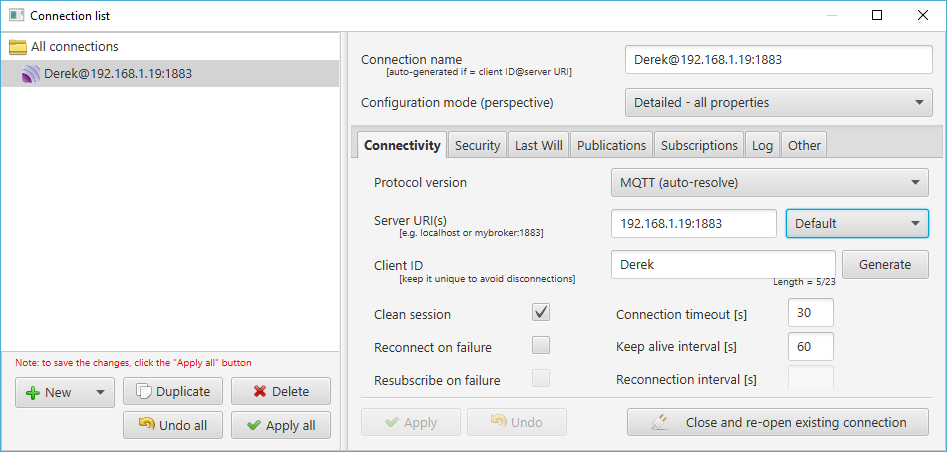
**Figure 7**. The Qt MQTT Sample Program

## Mqtt-spy

There are many useful tools for debugging mqtt applications. I recommend that you download and install the cross-platform mqtt-spy application. You can download it from: <https://github.com/kamilfb/mqtt-spy/wiki/Downloads>. It is a Java application, so you will need a JVM (e.g., Oracle JDK) on your machine. You should be able to double-click the .jar file to execute it.

Once you open it, you can configure a connection to your MQTT broker that is running on your Linux (or VirtualBox Linux) host. In my case, the IP address of my broker is 192.168.1.19, so the connection can be configured as in Figure 8. Remember to add a username and password to the Security tab for connecting to a secured server.

**Please note that you must hit the Enter key when you change a value in any of the fields**. If you leave the field (e.g., using your mouse) then it will revert to its previous value. I made this mistake many times so that you don’t have to :-)



**Figure 8**. The mqtt-spy connection configuration.

You can use this tool to publish or to subscribe to a topic. You can even configure more advanced options such as Last Will, which can be very useful. Figure 9 illustrate the publication of a message to a topic with the data “Testing mqtt-spy”. In this case the mqtt-spy tool is configured as both a publisher and a subscriber.

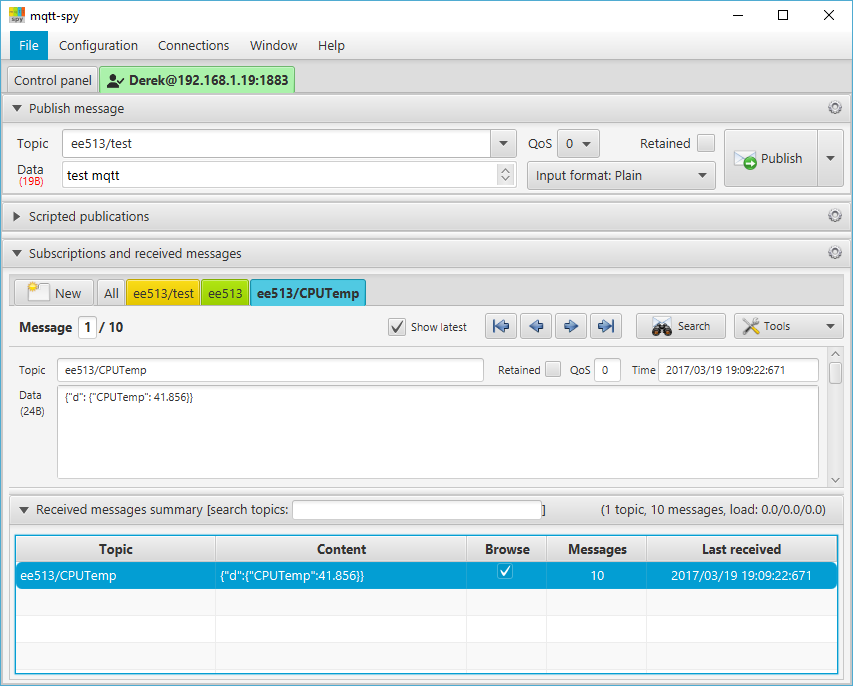
If you have the test subscriber client open on your Linux server, it will give the output below:

molloyd@desktop:~/paho.mqtt.c$ **mosquitto\_sub -v -t 'ee513/test'**

ee513/test Hello World

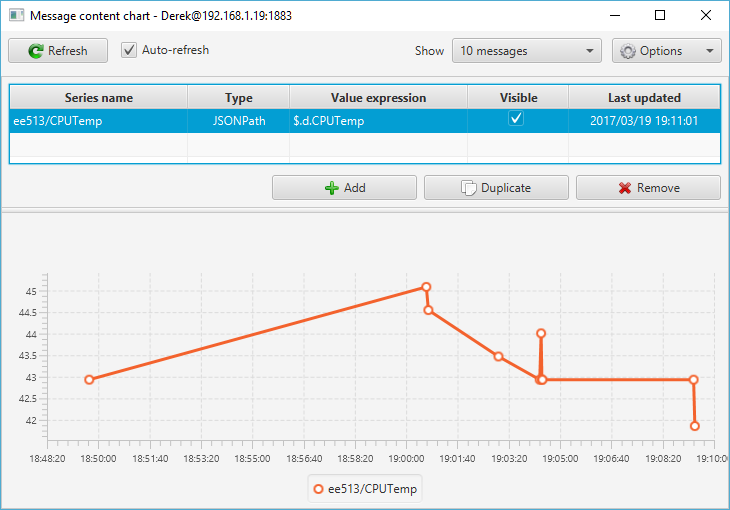
ee513/test Testing mqtt-spy

If you notice that messages that are being sent by your applications are not being picked up by mqtt-spy, please ensure that your subscription tab is actually subscribed to the topic. It’s not that obvious when it is not. Right-click the tab and choose Re-subscribe if the option is not greyed out (i.e., you are already subscribed.)



**Figure 9**. mqtt-spy publishing a message to the topic

If you right-click the ee513/CPUTemp line entry, you can choose Charts->Show Payload topics for this Topic and plot the data. In my case the JSON format is of the form: {“d”:{“CPUTemp”:41.856}}. The “d” is so that the data is compatible with online services such as IBM ThingSpeak. Therefore the value expression to use for the plot is $.d.CPUTemp. If you had a more simplified JSON payload of the form {“CPUTemp”:41.856} then you would just use $.CPUTemp.



**Figure 10**. mqtt-spy plotting the data values from the topic.

## ADXL345

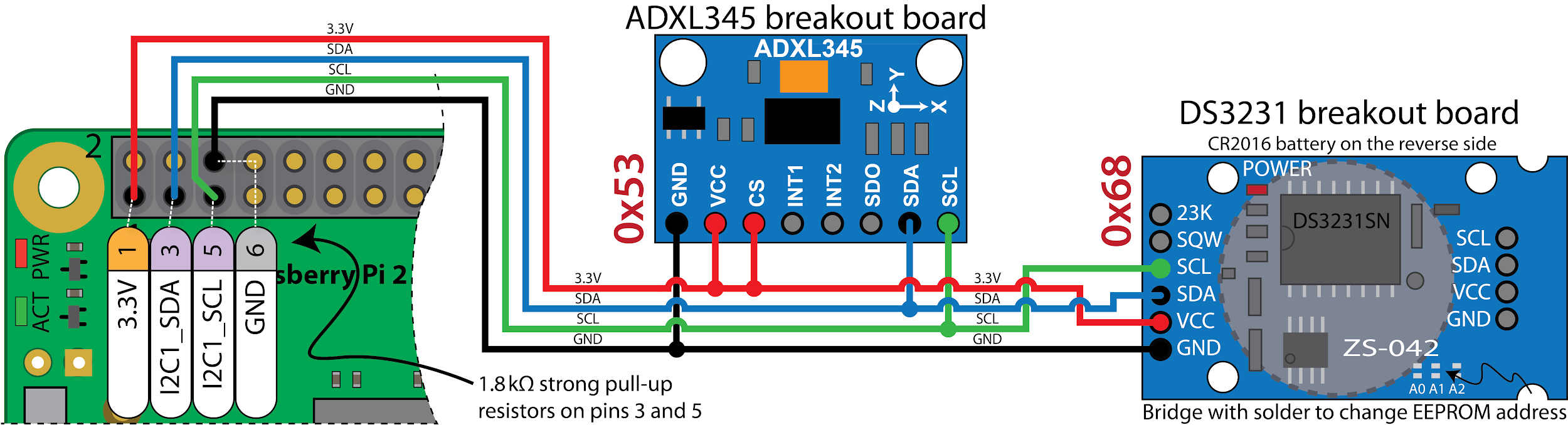
The Analog Devices ADXL345 is a small, low-cost *accelerometer* that can measure angular position with respect to the direction of Earth’s gravitational force. For example, a single-axis accelerometer at rest on the surface of Earth, with the sensitive axis parallel to Earth’s gravity, will measure an acceleration of 1*g* (9.81 m/s2) straight upward. While accelerometers provide absolute orientation measurement, they suffer from high-frequency noise, so they are often paired with gyroscopes for accurate measurement of change in orientation (e.g., in game controllers)—a process known as *sensor fusion*. However, accelerometers have excellent characteristics for applications in which low-frequency absolute rotation is to be measured. For simplicity, an accelerometer is used on its own in the following discussions, because the main aim is to impart an understanding of the I2C bus.

The ADXL345 can be set to measure values with a fixed 10-bit resolution, or using a 13-bit resolution at up to ±16 *g*. Digital accelerometers such as the ADXL345 include analog-to-digital conversion circuitry along with real-time filtering capabilities—they are more complex devices with many configurable options, but it is actually easier to attach them to the RPi than their analog equivalents. The ADXL345 can be interfaced to the RPi using an I2C or SPI bus, which makes it an ideal sensor to use as an example for both bus types.

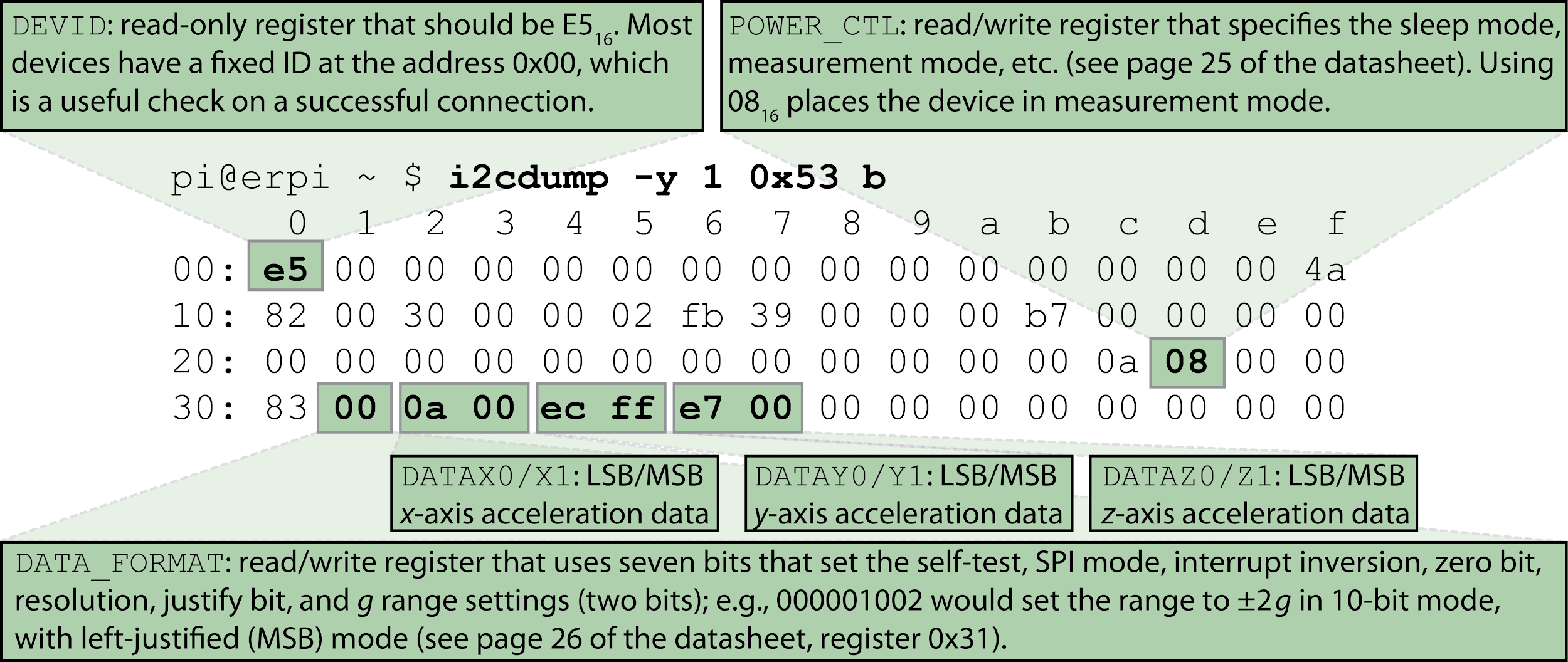
The I2C slave address is determined by the slave device itself. For example, the ADXL345 breakout board has the address 0x53, which is determined at manufacture. Many devices, including the ADXL345, have selection inputs that allow you to alter this value within a defined range[[4]](#footnote-3). If the device does not have address selection inputs then you could not connect two of them to the same bus, as their addresses would conflict. However, there are I2C multiplexers available that would enable you to overcome this problem. The data sheet for the ADXL345 is an important document that should be read when using this sensor. It is available at [www.analog.com/ADXL345](http://www.analog.com/ADXL345)

**The full source code for working with the ADXL345 is available at:**

<https://github.com/derekmolloy/exploringrpi/tree/master/chp08/i2c/cpp>



**Figure 11.** ADXL345 with the RTC



**Figure 12.** The register set for the ADXL345

## NAT Port Forwarding

If you are having difficulties with connecting to the Mosquitto server in your Virtual Machine over a Bridged Connection then you could use NAT with port forwarding. The key point here is that you have to map the MQTT broker on port 1883 to port 1883 on your host OS (i.e., Windows or Mac OS). It’s reasonably straightforward to do this using the Network settings in the VM.

Choose advanced settings under NAT (as in Figure 13) and then add a new rule as illustrated in Figure 14. I have added a rule for MQTT and also for SSH so that I can SSH from the RPi/BBB to the VM if required. You may need to install the ssh service in order to do this:

molloyd@desktop:~$ **sudo apt install openssh-server**

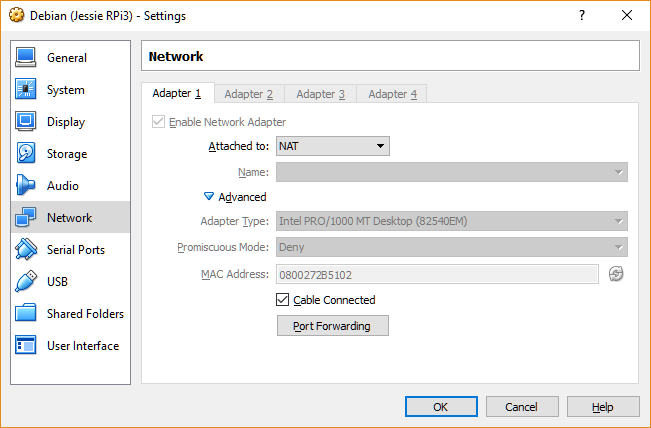
molloyd@desktop:~$ **sudo systemctl start ssh.service**

molloyd@desktop:~$ **nc -zv 127.0.0.1 22**

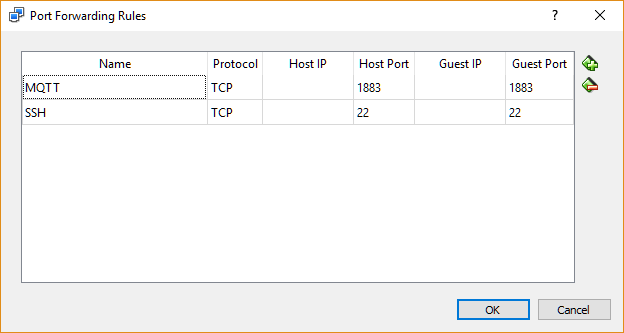
localhost [127.0.0.1] 22 (ssh) open

molloyd@desktop:~$ **nc -zv 127.0.0.1 1883**

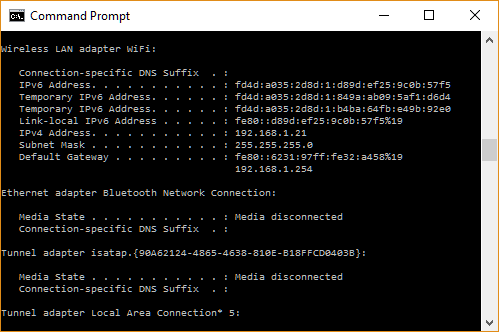
localhost [127.0.0.1] 1883 (?) open



**Figure 13**. VirtualBox Network Settings



**Figure 14**. NAT Port Forwarding Settings



**Figure 15.** You are now using the IP address of your Host OS to connect to the MQTT Broker. Mine is 192.168.1.21, as illustrated.

1. This acronym doesn’t make that much sense anymore as it doesn’t really support queue management, so MQTT just means MQTT without expansion. [↑](#footnote-ref-0)
2. Changed on 3/4/24 in response to numerous requests. [↑](#footnote-ref-1)
3. Negative marks limited to 0% -- e.g., a score of 40/100 will receive a mark of 0% after incurring a penalty for being 5 days late. [↑](#footnote-ref-2)
4. The ADXL345’s alternative address pin ALT is tied to GND on this particular breakout board, fixing the device at I2C address 0x53, despite the capability of the device itself to be configured for an alternative address. [↑](#footnote-ref-3)