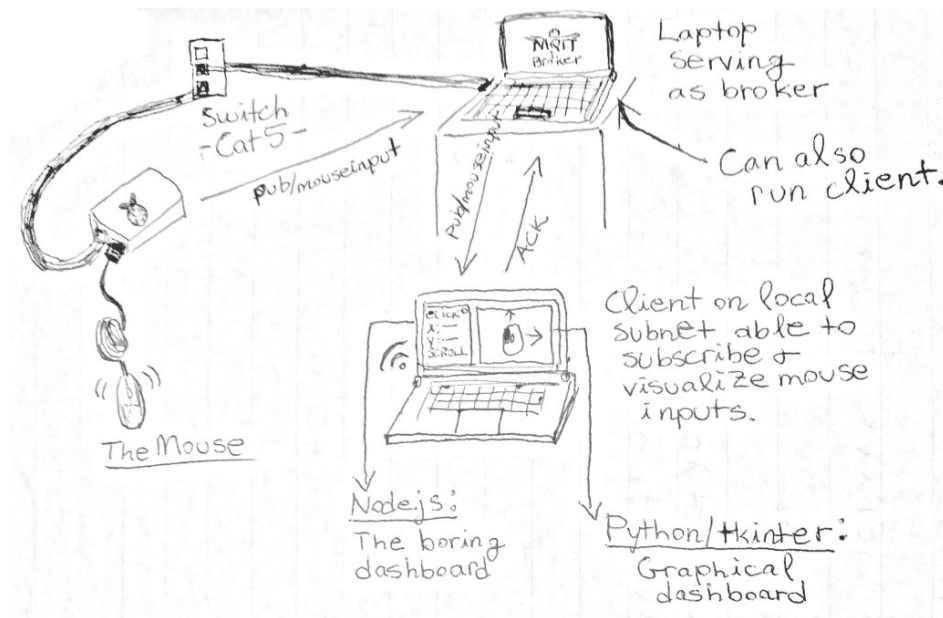


Ryan Haas
Dr. David Tarnoff
CSCI 4677
9 October, 2020

USB-HID Midterm Project

System design

I thought a diagram would be best to help explain my design:



I have taken advantage of my abundance of Cat 5e ethernet cables which I am using to keep the latency low between the Pi client sending the packets and my MQTT broker. A separate client may either be hosted on the same laptop running the broker or on any other device on the subnet.

You will notice that I have devised two dashboards. One is a simple CLI dashboard in Node.js to ensure I meet the project requirements. Still, I really want to visualize the mouse inputs graphically, if for no other reason than just to say that I can! Because I do not have much experience with Node.js, I will write a separate MQTT client in Python and use the tkinter library to graphically represent the mouse inputs being received by the client.

The USB mouse

The USB mouse I have chosen to use for this project is an onn.TM USB corded optical mouse. The mouse has left and right click buttons, a clickable scroll wheel, and a sensitivity of 1000 DPI.

The bits are mapped as follows:

Mouse Inputs

First Byte

Hexadecimal Value	Binary Value	Bit Used	Input
00	0000 0000	None	No clicking
01	0000 0001	Bit 1	Left click
02	0000 0010	Bit 2	Right click
04	0000 0100	Bit 3	Wheel click

Second Byte

The second byte is a 2's complement signed integer for X-direction movement. Negative values are left motion whereas positive values are right motion.

Third Byte

The third byte is a 2's complement signed integer for Y-direction movement. Negative values are upward motion whereas positive values are downward motion.

Fourth Byte

Hexadecimal Value	Binary Value	Bit Used	Input
00	0000 0000	None	No scrolling
01	0000 0001	Bit 1	Scroll up

ff 1111 1111 All bits used Scroll down

The...other bytes

My mouse has 4 remaining bytes. I am not sure exactly what bytes 5 and 7 indicate but I believe I have identified what the other bytes indicate.

The fifth byte seems to somehow indicate speed of the mouse movement. If I move the mouse gently right, for example, the fifth byte registers 01-03. If I jerk the mouse very quickly to the right however, the fifth byte reads around 20-40 depending on how quickly I am able to move it.

As for the sixth byte, I realized that it uses 00 to represent no x-axis movement or positive x-axis movement and it uses ff to represent negative x-axis movement.

The seventh byte appears to simply be a separate representation of the y-axis movement. I was wondering if perhaps the bits were stored left to right and that is what causes this seventh byte to sometimes be a slightly different value than the third byte.

Lastly, I have found that the eighth byte only uses two different values as well: ff is used for positive y-axis movement and/or positive x-axis movement and 00 is used for negative y-axis movement and/or negative x-axis movement.

Node packages used

I have organized all the code for this project in a central repository. My Node.js project used to broadcast the mouse inputs (hosted on my Raspberry Pi) is inside the **mqtt_broadcast_mouse** directory. My two dashboard projects are

located in the **mqtt_display_mouse** directory where the Node.js package is stored in the **node_cli/mqtt_mouse_event_client** directory.

broadcast_mouse packages

The publishing client uses the packages: **fs**, **node-hid**, **json5**, and **mqtt**.

Out of personal preference, I like to store configuration settings in a file outside of my code, usually in a JSON file. I soon realized that hexadecimal values (vendor and product identifiers, in my use case) were not supported in standard JSON and then I found the **json5** package which is used to parse in my MQTT and HID settings which are read using the **fs** package. Of course, the **hid** package is used to read the byte buffer from the mouse itself and the **mqtt** package is used to connect to the broker and then publish to **pub/mouseinput**.

mqtt_mouse_event_client packages

This dashboard uses the packages: **fs**, **json5**, and **mqtt**.

Again, I have my MQTT client configuration settings stored as JSON5 text which is read using **fs** and parsed using **json5**. Of course, the **mqtt** package is to establish a client connection which subscribes to **pub/mouseinput**.

No special instructions are necessary to install the required Node.js packages.

MQTT topic details

The topic I have chosen to publish my data to and to have it read from is **pub/mouseinput**. The first topic level, **pub** is used generally to denote that published data will fall under its hierarchy; that means if I had five other Raspberry Pis all publishing data of some kind, each published topic would be found at **pub**. Each message published to **pub/mouseinput** is an eight-byte value

stored in hexadecimal notation and according to the **mqtt** package's documentation, "everything is UTF-8".

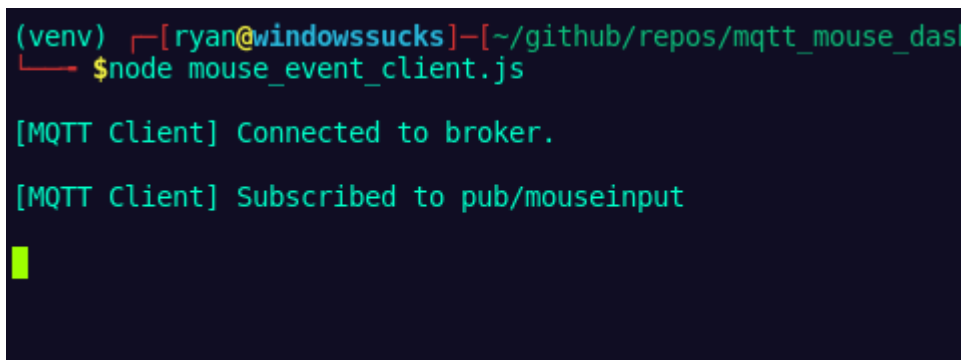
Because only one input is being published—my wired mouse input—I chose **mouseinput** for my second topic level although if I had multiple mouse inputs, I might even add a third topic level to identify each individual mouse (such as `pub/mouseinput/mouse<uniqueMouseID>`). Because the topics are case sensitive, I have left them entirely lowercase to avoid any errors.

The dashboards

The Node.js dashboard

My Node.js dashboard is written in the file, **mqtt_mouse_event_client/mouse_event_client.js** (I am not the best with naming conventions).

The first thing that is done after the packages are imported is the configuration settings for the MQTT client are read into a json5 object from the file **mqtt_mouse_event_client/config/config.json5**. The dictionary parsed in this JSON object is then passed into the mqtt class' connect function to initialize my MQTT client. The connect callback function is used so that the client immediately subscribes to **pub/mouseinput** following its connection. The connection and subsequent subscription are pictured below:



```
(venv) [ryan@windowssucks]-[~/github/repos/mqtt_mouse_dashboards]
└─$ node mouse_event_client.js

[MQTT Client] Connected to broker.
[MQTT Client] Subscribed to pub/mouseinput
```

Once subscribed, the message callback function is used so that the program will break apart the hexadecimal values from the message (I only make use of the first four bytes) convert them to signed integer values and then use

some quick branching logic to decide what the inputs mean (i.e. Did the mouse move left, right, up, down? Was the left, right or scroll wheel button pressed, was the scroll wheel scrolling up or down?) and print them in the console. For each input that is read, the original hexadecimal values for the first four bytes are also displayed before an acknowledgement is given back to the broker. Some of the output is displayed below:

```
No X movement
No Y movement
Wheel click
Scroll up

MOUSE INPUT:
Click=0x04 Scroll=0x01
X mov=0x00 Y mov=0x00

Acknowledgement sent to publisher.

No X movement
No Y movement
No click
No scroll

MOUSE INPUT:
Click=0x00 Scroll=0x00
X mov=0x00 Y mov=0x00

Acknowledgement sent to publisher.

No X movement
No Y movement
Right click
No scroll

MOUSE INPUT:
Click=0x02 Scroll=0x00
X mov=0x00 Y mov=0x00

Acknowledgement sent to publisher.
```

Top left: We can see a click of the wheel registered by the hex. value, 0x04.
Bottom left: We can see that a right mouse click was registered by the hex. value, 0x02.

```
Mouse left
Mouse up
No click
No scroll

MOUSE INPUT:
Click=0x00 Scroll=0x00
X mov=0xfb Y mov=0xfa

Acknowledgement sent to publisher.

Mouse left
Mouse up
No click
No scroll

MOUSE INPUT:
Click=0x00 Scroll=0x00
X mov=0xfd Y mov=0xfd

Acknowledgement sent to publisher.

Mouse right
No Y movement
No click
No scroll

MOUSE INPUT:
Click=0x00 Scroll=0x00
X mov=0x01 Y mov=0x00

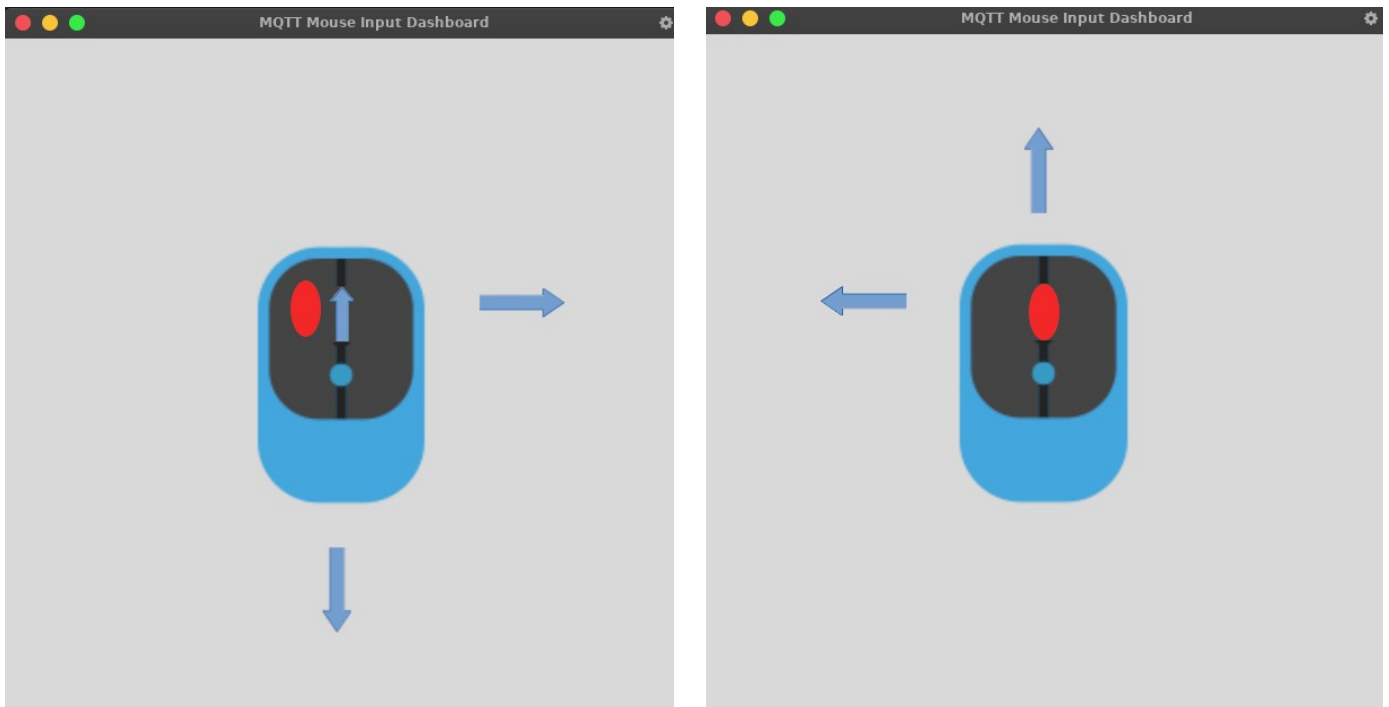
Acknowledgement sent to publisher.
```

Right: We notice that contrary to the output on the right the mouse was being moved about the X and Y axes. No click events or scroll events were recorded at that period in time.

The Python Dashboard

I will not go much in depth about this dashboard as I have not done it to meet the requirements but just, well, because I could. The MQTT package I used is called **paho-mqtt** which is maintained through the Eclipse Foundation. The client works very similarly to my Node.js implementation. The GUI components are not really worth discussing, in my opinion.

I will say that the organization of this code is a bit quirky because the client needed to make regular calls to the window object but the window object is blocking (and so is the MQTT client object). All of this is to say, if you do run this one, close the GUI window *before* sending an interrupt to the console.



Above: On a separate thread, the client, subscribed to **pub/mouseinput**, is reading the inputs and overlaying icons onto a mouse image to indicate what movements are taking place. Left, we see that the mouse has downward right motion while the left button is being clicked and the scroll wheel is scrolling up. Right, we see that the mouse has upward left motion while the scroll wheel is being clicked.

Github

All the code I used is on my public Github repository (https://github.com/haasr/mqtt_mouse_dashboard). I haven't written a README yet but you can get the Python dashboard up and running on a linux machine with a desktop environment following these instructions...

Getting the Python dashboard running

- 1) Ensure Python 3.x is installed as well as the **python3-virtualenv** (Debian) or **python-virtualenv** (Arch). You will also need to install **tk**.
- 2) Clone my repository. CD to **mqtt_display_mouse/python_gui/mqtt_mouse_event_client**.
- 3) Create a virtualenv in this directory:

```
python3 -m venv venv
```
- 4) Activate the virtualenv:

```
source venv/bin/activate
```
- 5) Install the requirements:

```
pip3 install -r requirements.txt
```
- 6) Enter your MQTT settings in the config module in **config/mqtt_config.py**.
- 7) Run it. Of course if packets aren't being sent over **pub/mouseinput**, not much will happen but there you go.