Emerging Technologies I

MIS 284N | Fall 2019

Project Milestone 3

Overview: For this next milestone, we will enable the data to be sent from the Raspberry Pi to an Android device using a protocol called MQTT. This publish-subscribe protocol will run over a WiFi connection from the Raspberry Pi to the Android device. Initially, the Android device will host the MQTT *broker*. As a later (optional) step, we will shift the broker to the Raspberry Pi. Both the Raspberry Pi and Android devices will host MQTT *clients*. The Raspberry Pi will use a python MQTT library; the Android app will be written in Kotlin using a Java MQTT library. A template Android app will be provided that you can start from.

Step 1: Follow the following instructions (please read carefully and in detail) to install the needed MQTT python library on the Raspberry Pi if it is not installed in your Pi. To check if your Pi has paho-mqtt: sudo pip3 list. This should list 'paho-mqtt'. If that is the case, you can skip step 1. However, these steps will be useful if you would like to install additional libraries to your pi.

- Download the zip file containing paho.mqtt.python from https://github.com/eclipse/paho.mqtt.python
- Put the zip file on a USB stick
- Plug the USB stick into your Raspberry Pi
- Copy the zip file into your home directory. For me, this was the command:
  sudo cp /media/pi/48CB-DB4B/paho.mqtt.python-master.zip .
  but the name of your drive is likely different.
- Move the file into a temporary directory. I did:
  mkdir paho_temp
  cd paho_temp
  mv ../paho.mqtt.python-master.zip .
- Unzip the file. I typed
  unzip paho.mqtt.python-master.zip
- Enter the paho.mqtt.python-master directory
- Type sudo python3 setup.py install

Step 2: Follow the following instructions to install the MQTT broker on your Android device:

- On your Android device, connect to the Internet, then download and install an app called MQTT Broker App.
- Connect the Android phone to your Raspberry Pi's network.

- Open the app. Click Start Broker with the default settings. The IP address of your broker will be shown in the window. Note it down.

Step 3: Set up the starter python code on the Raspberry Pi:

- Access the PythonMQTTStarter Repository here: https://github.com/UT-APAD/PythonMQTTStarter
- Download the file client.py and get it to your Raspberry Pi. You can move the file using your USB thumb drive. You can retype it. You can copy and paste the contents through VNC. Whatever.
- On the Raspberry Pi, in the terminal, navigate to the folder containing client.py. You should change the broker address to the IP address of your broker from the previous step. Type
  sudo python3 client.py
  The program should run without error.

Step 4: Send some initial messages back and forth. In particular, you'll do the following:

- This timing is a bit "tight"; feel free to change it inside of client.py if you want more time to do things on the Android side.
- With the Android device connected to the Raspberry Pi's network, open the MQTT Broker App. If the Broker is not already started, start the broker.
- On the Raspberry Pi, make sure the broker address in client.py matches your broker address.
- Click on MQTT Client in the Android App. Click the + sign at the upper right.
- Create a new client. You can give it whatever name you want (just make it different than the one used in client.py on the Raspberry Pi). Select No Authentication. Select LocalHost Broker. Click Create.
- Click on the new client you have created.
- Under Publish, click the + sign. Enter whatever you want for publish name and value. For Topic name, enter, exactly, testTopic1. Click Add.
- Under Subscribe, click the + sign. Enter whatever you want for the subscriber name. For the Topic name, enter, exactly, testTopic2. Click Add.
- On the Android App, return to the Publish screen to prepare for the next step.
- Now, on the Raspberry Pi, run client.py (i.e., from the terminal, type
  sudo python3 client.py)
- Immediately after you run it (but not before), in the Android App, under the Publish screen touch the paper airplane next to the publication you created. You should see stuff happen on the VNC terminal to the Raspberry Pi.
- You have 5 seconds to do the following in the Android app. Click on Subscribe. Then click on the i in the circle for the subscription you created. You'll see a terminal. Wait, and you should see the number 1000 appear.

Step 5: Enter Kotlin. First, visit https://github.com/UT-APAD/SampleMQTTAndroidApp and clone or download the repository. We're providing a complete and working template, but you should note the following:

- build.gradle contains support for mqtt:
  implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0'
  implementation 'org.eclipse.paho:org.eclipse.paho.android.service:1.1.1'
- you've maybe not seen the lateinit keyword in Kotlin before
- three permissions are needed in the android manifest:
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
  <uses-permission android:name="android.permission.WAKE_LOCK"/>
- the mqtt service needs to be declared in the android manifest:
  <service android:name="org.eclipse.paho.android.service.MqttService" />

You should be able to clone this repo, build the project, and deploy it to your Android device. You should now be able to use this as your Android client rather than relying on the MQTT broker app's built in client. In particular, you should be able to:

- start the broker through the MQTT Android Broker App
- launch your Android app
- start client.py on the Raspberry Pi
- click the button in your Android app (shortly after starting client.py)

Step 6: Pause. What have you done so far? Draw an architecture diagram that depicts the architectural components of your "system" and how they fit together. Who is subscribing/publishing? Where's the broker? How do the software components relate to your hardware components? You should reference the diagrams in the lecture about MQTT, but you will have additional information. Write an accompanying paragraph for your diagram to describe (in your own words) what's happening.

Step 7 (optional but strongly recommended): We've also installed a library on your Raspberry Pi that will allow IT to host the broker. This will be useful later when you want to have your Android device switch between the local network (with the RPi) and the Internet.

- To make this work, type the following in a terminal on the Raspberry Pi.
  sudo systemctl enable mosquitto.service
  This will enable the mosquitto MQTT service on your Raspberry Pi.
- To make sure your broker is up and running, type:
  mosquitto -v
- Now you should be able to rerun client.py and your Android application. But now get them to connect not through the broker app on the Android device but instead through the broker that is now running on your Raspberry Pi. [Note: you'll need to change one thing in client.py and one thing in the Android app.]

Step 8: Reflect/restate: in your own words, why is it better to run the broker on the Raspberry Pi instead of the Android device? Give at least two reasons.

Step 9: Now it's back to our pedometer. Design a way for your steps to get from your microbit, to the Raspberry Pi (via EddyStone) and then on to the Android device (via MQTT). Draw a sequence diagram

depicting the different elements of your system (including the human user), how they interact (including any explicit interactions from the human, like pressing a button), and what messages are sent and how. Implement this solution. This is what you will demo to show completion of Milestone 3.

What to submit

Via Canvas, submit a brief writeup that includes the following information for each step:

- **Steps 1-5**: Nothing to submit.
- **Step 6**: Submit as described in Step 6.
- **Step 7**: Nothing to submit.
- **Step 8**: Submit as described in Step 8
- **Step 9**: Submit your design, including a brief textual description and the associated sequence diagram.

Wednesday, November 13, in office hours, demonstrate Step 9. Be prepared to explain what the code does in all three devices (the makecode or javascript on the microbit, the python on the Raspberry Pi, and the Kotlin on Android).