Electronics for IoT

MQTT

Bernhard E. Boser
University of California, Berkeley
boser@eecs.berkeley.edu
So Far …

• INA219 to measure V, I, P
  – Photocell characterization
  – Many other applications

• I2C to send data from INA219 to ESP32
  – Great for local (short range) communication (ex: between chips on a board)
  – Will be used again for other sensors!

• Connect MCU to WiFi
Data Communication Options

• How do we send data to/from ESP32 to remote host computer?

• Need a protocol … i.e. established set of rules for communication

• HTTP protocol
  + ubiquitous
  - verbose
    - visual output difficult to parse
  - one way
    - needs webserver

• We will use MQTT over WiFi
MQTT

• **Message Queuing Telemetry Transport**
  – Misnomer: no queuing
  – Lightweight machine-machine messaging protocol
  – Lightweight ➔
    • low communication bandwidth,
    • suitable for implementation on resource-constrained devices

• Standard
  – ISO/IEC PRF 20922
  – Lot’s of support available:
    • Implementations in many computer languages (e.g. Python)
    • Tutorials, documentation …
      – E.g. [https://www.hivemq.com/blog/how-to-get-started-with-mqtt](https://www.hivemq.com/blog/how-to-get-started-with-mqtt)
      – Many others
Facebook Messenger uses MQTT


Why?
- Low bandwidth use
- Low power use
- Supports many-2-many
- Scalable: supports millions of users
MQTT Publish/Subscribe
A Cable TV Analogy

- Publisher: cable company
- Broker: internet provider
- Subscriber: household
- Topic: channel you can buy (ex: ESPN)
- Message: an episode (SportsCenter on 9/10/2018)

- Publisher sends out all messages on a topic
- All subscribers receive all messages on topics they subscribe to
- Routing of messages managed by broker
- Publisher doesn’t need to know details of subscribers (ex: IP address)
MQTT Features

• Clients need only know broker, not each other
  – No “what’s your IP address”
  – Asynchronous:
    • No connection issues (“turn this on first, then …”)

• Text messages
  – Lightweight – good for low bandwidth situations
  – Easy parsing (you choose format)

• Topics
  – Organization
  – Hierarchical, separated by /, e.g.
    • solar/current, solar/voltage, kitchen/temperature, stocks/DJIA
    • alice/solar/current, fred/solar/current
MQTT Messages

• Topic
  – Hierarchical, separated by /
  – E.g.
    • Kitchen/temperature
    • IBM/stockprice
    • ...

• Message
  – Arbitrary text
Putting it all together …

• Broker

• Client
  – Python library “MQTTClient”
  – Topics
  – Messages
  – QoS

• Security
MQTT Broker

• “Hub” of the service

• Many service offerings:
  – Amazon AWS, Microsoft Azure, IBM Watson, …
  – Free brokers for testing (no security)
    • iot.eclipse.org
    • iot49.eecs.berkeley.edu
  – Roll your own …
    • https://mosquitto.org
Sharing Brokers

• Typically, brokers are shared among many users
  – Thousands or millions on commercial servers, e.g. Amazon AWS

• That’s great for sharing data and collaboration, e.g.
  – boston/temperature
  – berkeley/temperature
  – …

• But if several users send unrelated information to the same topic, e.g. everybody in EE49
  – solar_panel_current

• … the result is a mess!
Cooperatively sharing MQTT Brokers

- Prefix all topics with unique identifier, e.g.
  - bernhardboser/current
  - alicequyo/curren

- Commercial brokers enforce this

- Others, e.g.
  - iot.eclipse.org
  - iot49.eecs.berkeley.edu

- ... rely on users following an agreed convention

- EE49:
  - Prefix all topics with your name
  - Beware of multiple participants with same name ... (UID?)
from mqttclient import MQTTClient
import network
import sys
import time

session = "insert unique session here"  # change this!
BROKER = "iot.eclipse.org"
# check wifi connection!
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
if not wlan.isconnected():
    print("no wifi connection")
    sys.exit()
else:
    print("connected to WiFi at IP", ip)
# connect to MQTT broker
print("Connecting to MQTT broker", BROKER, "...", end="")
mqtt = MQTTClient(BROKER)
print("Connected!")

# Define function to execute when message is received on subscribed topic
def mqtt_callback(topic, msg):
    print("RECEIVE topic = {}", msg = {}").format(topic.decode('utf8'), ...  
              msg.decode('utf-8'))

# Set callback function
mqtt.set_callback(mqtt_callback)

# Set subscribe topic
mqtt.subscribe(session + "/host/hello"")
for t in range(100):
    # Microcontroller sends hello statements
    topic = "{}/mcu/hello".format(session)
    data = "hello" + str(t)
    print("send topic='{}' data='{}'".format(topic, data))
    mqtt.publish(topic, data)
    # Check for any messages in subscribed topics
    for _ in range(10):
        mqtt.check_msg()
        time.sleep(0.5)

# free up resources
mqtt.disconnect()
Lab 3, part B: Skeleton of IoT App

1. Establish Internet connection
   - E.g. boot.py

2. Initializations
   - MQTT client
   - I2C and INA219

3. On host:
   - Start MQTT client and subscribe …

4. Collect data
   - Measure solar cell I/V characteristic and send to cloud

5. Instruct host to create plot
Plotting

- Transfer data from ESP32 to host computer (ex: laptop)
- Then plotting is easy
  - Matlab, Excel, Python, …
- Within python, you can use matplotlib
Python Plotting Library

- [https://matplotlib.org](https://matplotlib.org)
- Similar to matlab
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
T = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(T, T**2, 'r--', T, T**3, 'bs', T, T**3, 'g^')
plt.xlabel('Time')
plt.ylabel('Power')
plt.title('Power versus Time')
plt.show()
What's wrong with these topics?
publish

```python
topic = "iot49/esp32"
message = "hi there!"
mqtt.publish(topic, message)
```
publish – subscribe loop

```python
for i in range(1000):
    topic = "iot49/esp32"
    message = "hello " + str(i)
    print("PUBLISH topic = {} message = {}".format(topic, message))
    mqtt.publish(topic, message)
for _ in range(10):
    mqtt.check_msg()
    sleep(0.5)
```
Python strings / byte arrays

• byte:
  – 8-Bits of data
  – Can hold $2^8 = 256$ values

• char:
  – ~ 127 (ASCII) latin, decimals, and punctuation
  – Thousands with other alphabets (Greek, …)

• Python 3 treats bytes and chars different
  – Python 2 “blurs the lines”
  – Code that works in Python 2 won’t necessarily in Python 3

• Literals:
  – String: ‘this is a string literal’
  – Byte array: b‘this is a byte array literal’
string / byte array conversions

```python
>>> b'byte array'.decode('utf-8')
'byte array'

>>> 'some string'.encode('utf-8')
b'some string'
```

• **Encoding:**
  – E.g. utf-8 (many others)
  – For only latin characters, this rarely matters
  – For others, get funny symbols if incorrect …

• **MQTT library uses byte arrays (not strings) …**
Putting it all together ...

• Broker

• Client
  – Python library “MQTTClient”
  – Topics
  – Messages
    – QoS

• Security
MQTT QoS

• QoS
  – 1: deliver at most one time
  – 2: deliver at least one time
  – 3: deliver exactly one time

• Optional arguments to publish and subscribe:
  – mqtt.publish(topic, message, qos=0)
  – mqtt.subscribe(topic, qos=0)

• Not all brokers and clients support all QoS levels

• MQTT has a few other features
  – E.g. last will
  – Check the online documentation