Data Science and Communication in Smart Cities

Day 3: Data Communication Basics

Chao Wang

Networked Cyber-Physical Systems Laboratory
Department of Computer Science and Information Engineering
National Taiwan Normal University

August 6, 2024

NATIONAL TAIWAN NORMAL UNIVERSITY

Outline

Broker-based data communication (pub/sub)

- Terminologies
  - Topic: the subject of interest
  - Message: a piece of data for a certain topic
  - Publishers: programs that publishes messages of certain topics
  - Subscribers: programs that subscribes to certain topics
  - Broker: a program that helps forward messages from publishers to subscribers
- Message delivery semantics
  1. Each message will be delivered to all who subscribed to the corresponding topic (example: MQTT)
  2. Each message will be delivered to only one of the corrsponding subscribers (example: NSQ)

Hands-on activity #1: simple pub/sub application

- Refer to the course webpage for the needed programming environment
- Let's work on this together:
  1. Connect to our server machine that runs a MQTT broker
  2. Publish a message of a certain topic
  3. Verify that our public subscriber has received your message
  4. Subscribe to a certain topic
  5. Verify that your subscriber can receive a message we published

The MQTT protocol

- History
  - Invented in 1999 by Andy Stanford-Clark and Arlen Nipper
  - Internal use at IBM until 2010 (MQTT 3.1)
  - In 2014, approved as an OASIS standard (MQTT 3.1.1)
  - March 2019, MQTT 5 specification
- Design objectives (i.e., requirements)
  - Simple implementation
  - Quality-of-Service data delivery
  - Lightweight and bandwidth efficient
  - Data agnostic
  - Continuous session awareness
- Implementation for MQTT clients
  - https://github.com/mqtt/mqtt.org/wiki/libraries
  - https://pypi.org/project/paho-mqtt/

MQTT terminology

- Client vs. broker
- Sender vs. receiver
- Control packets vs. data delivery
- A complete list of MQTT control packets (15 in total):
  - Connection establishment and destruction:
    - CONNECT, CONNACK, DISCONNECT
  - Message subscription:
    - SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK
  - Message delivery and QoS control:
    - QoS 0: PUBLISH
    - QoS 1: PUBLISH, PUBACK
    - QoS 2: PUBLISH, PUBREC, PUBREL, PUBCOMP
  - Liveness:
    - PINGREQ, PINGRESP
  - Authentication:
    - AUTH

Connection establishment

- A client sends a CONNECT control packet to the broker to request for connection (either to establish or resume a session).
- A CONNACK control packet from the broker acknowledges the connection request.

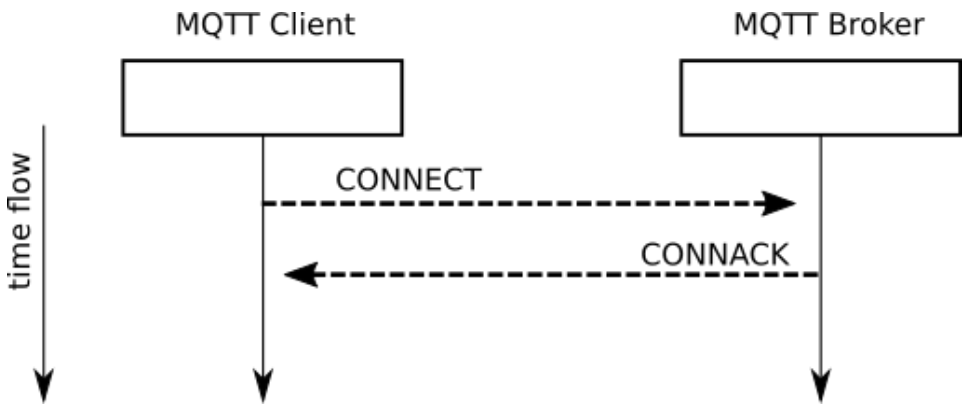

Figure: Sequence diagram of connection establishment.

Message subscription

- A client sends a SUBSCRIBE control packet to the broker to subscribe to a set of topics.
- A SUBACK control packet from the broker acknowledges the subscription request.
- A PUBLISH control packet carries a message (data) and is forwarded by the broker, from a publisher to a subscriber.

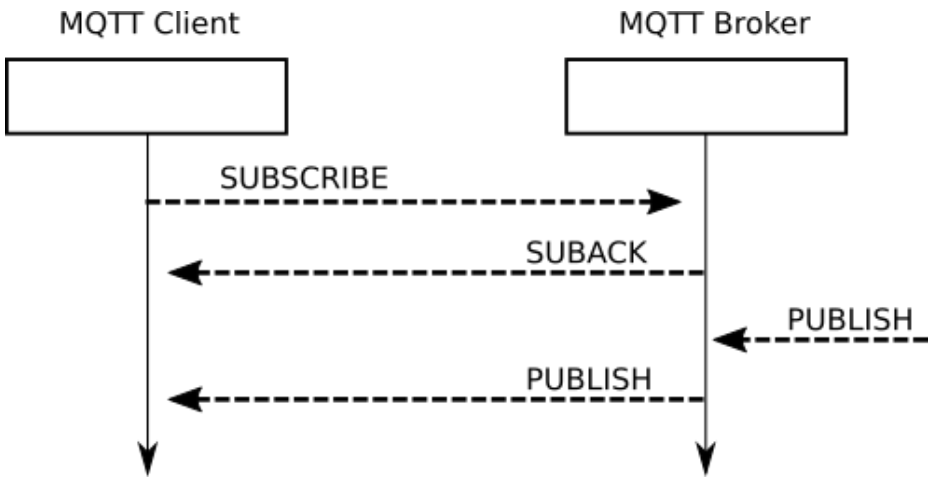

Figure: Sequence diagram of message subscription.

Code: a simple MQTT publisher

- `paho-mqtt`, an MQTT client library written in Python: https://pypi.org/project/paho-mqtt/

```python
import paho.mqtt.client as mqtt
import sys

client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2,"id") # a unique id
client.username_pw_set("name", "passwd") # your username and password
client.connect("ip", 1883, 60) # broker ip and port, and keepalive seconds
msg = sys.argv[1] # first command-line argument
client.publish(tc, msg, 0) # tc is a string for pub/sub topic
print("Publishing message '" + msg + "'")
```

Figure: A simple publisher.

Code: a simple MQTT subscriber

- Two concepts in network programming:
  1. callback functions
  2. event loops

```python
import paho.mqtt.client as mqtt

# a user-defined callback function, called upon each message arrival
def on_message(client, userdata, msg):
    payload_string = str(msg.payload, encoding='utf8')
    print(payload_string)

client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2,"id2")
client.on_message = on_message # set up the callback function
client.username_pw_set("name", "passwd")
client.connect("ip", 1883, 60)
client.subscribe(tc, 0)

# a blocking call: it processes network traffic,
# dispatches callbacks, and handles reconnecting.
client.loop_forever()
```

Figure: A simple subscriber.

Hands-on activity #2: Let's chat

- Let's implement a simple chatting service!
  - Requirement: publish our message while receiving the message from the other side
  - Challenges: do these two activities simultaneously (somewhat)
- Demo time

Code: multiple threads of execution

- One more concept in network programming: concurrency

```
6  def on_message(client, userdata, msg):
7      print(str(msg.payload, encoding='utf8'))
8
9  client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2, "")
10 client.on_message = on_message
11
12 client.loop_start()
13 print("Enter topic-to-subscribe: ", end='')
14 tp_sub = input()
15 client.subscribe(tp_sub, 0)
16 print("Enter topic-to-publish: ", end='')
17 tp_pub = input()
18 while True:
19     msg = input()
20     client.publish(tp_pub, msg, 0)
21 client.loop_stop()
```

Figure: An example multi-threaded program.

Client-server and request-response

- The client-server pattern
  - client: a program who calls out
  - server: a program who takes the call
- The request-response pattern
  - requester: a program who requests for some data or action
  - responder: a program who responds and replys to the requester
- Examples?

Example 1: an echo server

- Functionality: send back the message sent by the client.

```
(base) vboxuser$ python3 client.py
Connected to the broker.
Type your message: Hello
                                    server: You typed 'Hello'
Type your message: 'Nice to meet you'
                          server: You typed ''Nice to meet you''
Type your message: Could you not to repeat my words?
       server: You typed 'Could you not to repeat my words?'
Type your message:
```

Figure: An example echo server.

Example 2: a stateful server

- Functionality: return based on some internal system states.

```
(base) vboxuser$ python3 client.py
Connected to the broker.
Type your message: menu
                --- menu ---
                0: Get the server local time
                1: Show the current value of the magic number
                2: Increment the magic number by one
Type your message: 0
                                Mon, 05 Aug 2024 22:51:15 +0800
Type your message: 1
                                                          74
Type your message: 2
Type your message: 1
                                                          75
Type your message: 0
                                Mon, 05 Aug 2024 22:51:26 +0800
Type your message:
```

Figure: An example stateful server.

Implementation using MQTT

- How to implement such client-server applications using MQTT?

Code: a simple client feed

- Tip: print a prompt in a callback function.

```
15 def on_message(client, userdata, msg):
16     message = str(msg.payload, encoding='utf8')
17     s = message
18     print("\r"+s.rjust(60) + "\nType your message: ", end='')
19 
20 client.on_message = on_message
21 client.loop_start()
22 time.sleep(1)
23 while True:
24     print("Type your message: ", end='')
25     msg = input()
26     client.publish(to_server, msg, 0)
27 client.loop_stop()
```

Figure: An example client code.

Code: an echo server

- Key idea: a server can be implemented as a MQTT client.

```
 9 def on_message(client, userdata, msg):
10     payload_string = str(msg.payload, encoding='utf8')
11     client.publish(to_client, "server: You typed '" + payload_string +"'", 0)
12
13 client.on_message = on_message
14 print("This runs a simple server to echo client string.")
15 client.connect("ip", 1883, 60)
16 client.subscribe(to_server, 0)
17 client.loop_forever()
```

Figure: An example echo server implementation.

Code: a stateful server

- Key idea: add variables to keep some internal states.

```
 7 magic_number = 0
 8
 9 def on_message(client, userdata, msg):
10     payload_string = str(msg.payload, encoding='utf8')
11     global magic_number
12     match payload_string:
13         case "menu":
14             client.publish(to_client, "\
15                 --- menu ---\n\
16                 0: Get the server local time\n\
17                 1: Show the current value of the magic number\n\
18                 2: Increment the magic number by one", 0)
19         case "0":
20             client.publish(to_client, time.strftime("%a, %d %b %Y %H:%M:%S %
   z", time.localtime(time.time())))
21         case "1":
22             client.publish(to_client, str(magic_number))
23         case "2":
24             magic_number += 1
25         case _:
26             client.publish(to_client, "Unknown request. Try 'menu'.", 0)
```

Another useful MQTT feature: topic organization

- Use slash '/' to better organize topics
- Use wildcards (+ and #) to better manage subscriptions
  - + : single-level wildcard
  - # : multi-level wildcard
- Example
  - topic 1: sensor/noiseLevel/livingRoom
  - topic 2: sensor/noiseLevel/bedRoom
  - topic 3: sensor/headCount/livingRoom
  - topic 4: sensor/headCount/bedRoom
  - Subscribe to sensor/# ?
  - Subscribe to sensor/+/bedRoom ?

Bridging Data Science and Communication

- Key idea: the data items needed by data science can be encapsulated in MQTT messages.
- Example: report some moving vehicle locations in a smart city
  1. Write a Python script to cook up some fake GPS location data
  2. Put your data in a MQTT message and publish it
  3. Have another Python script to get the data via MQTT subscription
  4. Import the received data to QGIS for visualization and interpretation
- How to proceed?
  - Get your hands dirty: implement your idea bit by bit
  - Get your eyes busy: read the related documents
  - Some starters:
    - QGIS live GPS tracking (link)
    - A Python way to write to COM ports (link)
  - Discuss your questions and answers on our Moodle forum (link)