Programming Project

Introduction

The Internet of things (IoT) has become more and more popular thanks to the development and deployment of computer networks and sensors. Nowadays people are surrounded by numerous sensors and network connected devices in their daily life. Most of these IoT devices are cheap, such as temperature sensors or web cameras, hence they don't come with powerful computation capability. Not only so, the data generated by IoT devices are usually small in terms of size but large in terms of quantity. As a result, a lightweight yet efficient message transport protocol needs to be proposed in IoT environments.

Message Queuing Telemetry Transport (MQTT) is one of the publish subscribe message transport protocol designed for this specific environment. The protocol is lightweight and reliable which can be used in low network bandwidth environment with minimal power consumption. There are three roles in a MQTT network, publisher, subscriber, and broker. The publisher and subscriber are both client software where publisher publishes data to the broker while subscriber subscribes data from the broker. The broker on the other hand is the server software which stores and handles the message transmission for the clients. The broker acts as a relay for the publishers and subscribers hence the clients don't need to know the IP addresses of each other. The relationships between these roles are shown in Figure 1.

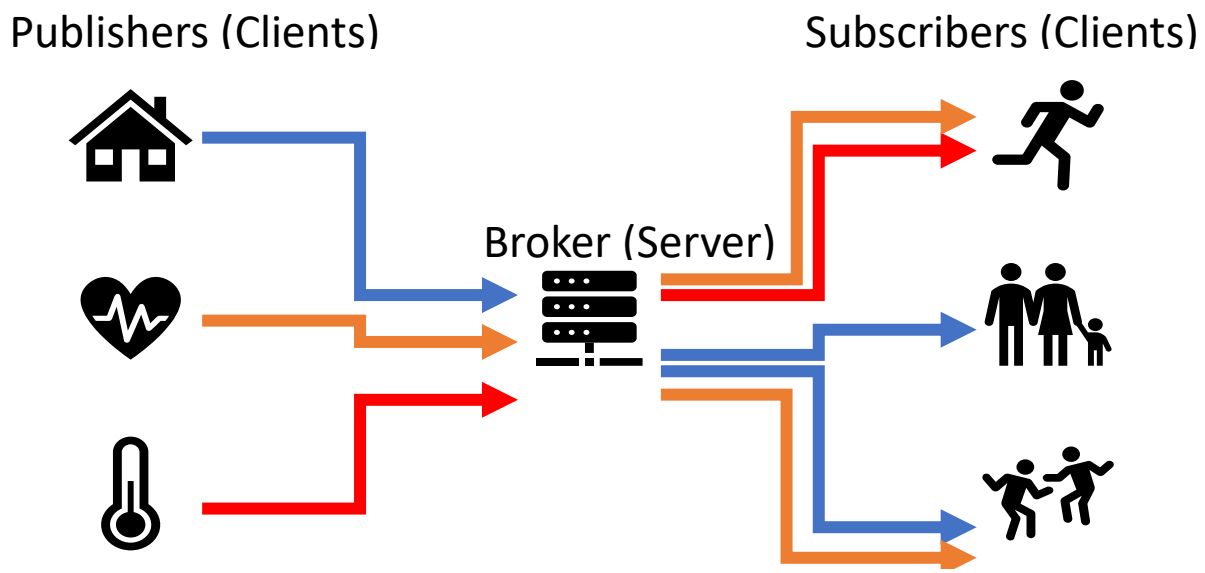

Figure 1 The relationships between different roles. The arrows in the figure shows the topic being published or subscribed. Different color shows different topics and subscribers can subscribe to any number of topics they are interested in. Note that since both publisher and subscriber are client software, a client can be both publisher and subscriber at the same time.

A topic is a unique identification of where a message should be published. When a publisher wants to distribute a new message of a certain topic, it sends a control message with the data to the broker. The broker will then distribute the message to all the subscribers who subscribed to this topic. In MQTT, the topics are managed in a hierarchical manner where special characters can be used when subscribing.

Phase I

There will be two phases for this programming project. In the first phase, you will implement a simple message passing framework through socket programming. This system includes both server-side application and client-side application.

Server

The server-side application listens on a specific port to handle all the incoming clients. It is responsible for managing all the requests from the connected clients. The subscription of each client is maintained in the server only when the server is up and running. You don't have to remember the states after shutting down the server. The server needs to service multiple clients at the same time, which means you may have to create multiple threads for different clients.

Client

The client-side application connects to the server and publish/subscribe to topics of interests. Clients need to show the messages of the subscribed topics received from the server. There is no specific requirement on the message shown but you need to at least present the topic and the message received.

Topic

There will be four default topics when you start the server, which are WEATHER, NEWS, HEALTH, and SECURITY. The server needs to reply with error messages when publishing or subscribing to a topic which doesn't exist. In this phase, the clients can only publish or subscribe to the default topics.

Message

The following features are required to be implemented.

Feature	Description	Message Format
CONNECT	Clients can connect to a server and exchange messages with it. This message will be sent from client to server after socket connection being established.	Client: <CONN> Server: <CONN_ACK>
DISCONNECT	When clients disconnect from a server, they need to notify the server and wait for the server to respond to the disconnect request. The server must respond with an ACK to the request before closing the socket. Clients can safely close the socket after receiving the response from the server.	Client: <DISC> Server: <DISC_ACK>
PUBLISH	Clients can send request to publish messages to any topic. The server handles the publish request and distribute the message to the subscribed clients.	Client: <PUB, TOPIC, MSG>
SUBSCRIBE	Clients can send request to subscribe to topics they are interested in.	Client: <SUB, TOPIC>

There is no limitation on how you want to design the messages being transmitted between the server and clients. The message format column shows an example which you can choose to follow or design your own format.

Phase II

In the second phase, you will implement more functions for the publish subscribe system.

Topic

In this phase, clients can publish or subscribe to any topics of interests. Topics now have different levels and are case sensitive. There are three special characters to be implemented. Note that # and + can only be used when subscribing to topics.

1. Topic level separator (/)
   A topic level separator separates each level of a topic and provides a hierarchical structure to the topic names. The followings are some examples.
   - home/sensor/1: three levels, home, sensor, and 1
   - home/Sensor/1: topic is case sensitive; hence this is a different topic than the previous one
   - home//test: this is also valid. Level can be empty where in this example test is at the same level as 1

2. Multi-level wildcard (#)
   This character matches any number of levels within a topic. It must be specified either on its own or following a topic level separator. The followings are some examples.
   - home/#: this includes all the topics starting with home/, for example home/room, home/kitchen/faucet, or home/yard/sensor
   - home/test#: this is invalid, # should always follow a topic level separator
   - home/#/testing: this is invalid, # should be the last character

3. Single level wildcard (+)
   This character matches one topic level. The followings are some examples.
   - home/+/sensor: this includes home/yard/sensor, home/room/sensor; it doesn't include home/yard/pond/sensor, home/yard/sensor/test, home/room/temp

Retain

This is a flag which can be set when publishing a message. If a message is published with this flag set, this message will be stored on the server and any subscriber subscribes to this topic later will receive this message right after subscription. Only one message can be retained for each topic which means the old ones will be replaced by the latest retain message published to each topic.

Message

The following features are required in phase II.

Feature	Description	Message Format
CONNECT	Clients can connect to a server and exchange messages with it. This message will be sent from client to server after socket connection being established.	Client: <CONN> Server: <CONN_ACK>
DISCONNECT	When clients disconnect from a server, they need to notify the server and wait for the server to respond to the disconnect request. The server must respond with an ACK to the request before closing the socket. Clients can safely close the socket after receiving the response from the server.	Client: <DISC> Server: <DISC_ACK>
PUBLISH	Clients can send request to publish messages to any topic. The server handles the publish request and distribute the message to the subscribed clients.	Client: <PUB, TOPIC, RETAIN, MSG>
SUBSCRIBE	Clients can send request to subscribe to topics they are interested in.	Client: <SUB, TOPIC> Server:<SUCCESS> Server: <ERROR>
UNSUBSCRIBE	Clients can unsubscribe a topic from the server. Note server should return the topics unsubscribed or error message if this client has never subscribed to such topics.	Client: <UNSUB, TOPIC> Server: <SUCCESS> Server: <ERROR>
LIST	Clients can use this control message to query all the topics which have been published from the server.	Client: <LIST> Server: <Number of Topics, TOPIC, TOPIC, …>

Important Notes

For both phases, you will need to submit the following files:

- The source code of your implementation
- A document explaining how to build/run your program with your name, student ID, and email address
- For both phases, you need to include a document explaining how you implement the features including the message format and flow between server and client

You should make sure you can run your program on **CSELabs** machines. You can find the list here: https://cseit.umn.edu/computer-classrooms

Sample Test Cases

The followings are some sample test cases describing different use cases and expected results. These test cases are only samples and do not cover all the cases.

Test Case	Result	Explanation

1. Client_1 publishes a message “Testing” with the retain flag set to topic “test/test_topic” 2. Client_1 subscribes to “test/test_topic” 3. Client_2 subscribes to “test/+”	Both Client_1 and Client_2 receive “Testing” right after subscription	Retained message will be published to all the clients subscribe to this topic
1. Client_1 subscribes to “topic/#” 2. Client_2 publishes “connect” to “topic/test/con” 3. Client_2 publishes “random” to “toPic/test/con”	Client_1 receives “connect”	Topics are case sensitive

Due Date

- Phase I: 04/07/2021
- Phase II: 05/05/2021

Grading Policy

- Phase I: 0%
- Phase II: 90%
    - Message passing through sockets: 30%
    - Publish: 20%
    - Subscribe: 20%
    - Topics: 10%
    - Retain: 10%
- Others: 10%
    - Documentation
    - Readability
    - Error handling

Reference

1. https://en.wikipedia.org/wiki/MQTT
2. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf