DATA RELAY USING MQQT PROTOCOL
IMPLEMENTATION OF ECLIPSE MOSQUITTO LIBRARY

Introduction

MQTT Protocol employs a “**Broker**” as a central communication port, data is **published** to this broker in the form of messages by a **Client**. The routing information for the broker to store and locate a particular message is in the form of hierarchical strings called **Topics**, with “/” used as delimiters separating each level in the chain. Another client **subscribes** to the topic that the required data was published to to receive the data.

This project employs the library **“Eclipse Mosquitto”** to implement the above publisher->broker->subscriber architecture. There are 50 publisher clients on different floors of a house, 30 of them being sensors that publish data such as temperature, humidity, air pressure, pH, etc. and the remaining 20 being sensors that publish distance, switch states(“ON” or “OFF”), etc. to the topic “home/sensors”. The data is published in the form of Json Streams, at a frequency of 500 milliseconds.

There is one subscriber client which subscribes to “home/sensors” to receive the sensor data, parses the received Json streams using this jsonparser library and segregates it into various data types. This data is then compressed using this run-length-encoding algorithm at a frequency of 2 minutes, and written into a text file.

IMPLEMENTATION OF PUBLISHER CLIENT

1. Algorithm:-

i. 50 pointers to the structure mosquitto are created.

ii. The clients are looped through.

iii. For the first 30 clients, messages are generated as follows :-

   1. Integers for temperature, humidity, air pressure and pH are generated in the given range using rand and the formula rand()%(upper limit - lower limit + 1) + lower limit (added for offsetting)
   2. Doubles for Humidity, pH and temperature are generated by adding random values of 3 decimal places to the respective integer variables.
   3. These are then converted to a string using the sprintf() operator.
   4. These random values with the necessary delimiters for JSON are then concatenated into the final character array for a message using the strcat() function.

iv. The current pointer to the mosquitto structure, the number of clients and the generated message are then passed into the client_define function.

v. Within this function, mosquitto_new() is called which creates a new mosquitto client instance. The parameters passed to this are

1. struct mosquitto *mosquitto_new(const char *id,bool clean_session,void *obj)
2. Boolean clean_session is set to true to instruct the broker to clean all messages on client disconnect.

vi. Mosquitto_connect is called which connects the client to hostname “localhost” on port 1883, with a keepalive time of 60 seconds. Keepalive time defines the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time.

vii. Mosquitto_loop_start is called which starts a **thread** for the current client.

viii. If the return code of the above processes is zero, i.e. if there has been a successful connection of the client with the broker, then mosquitto_publish() is called which publishes the generated message as payload to the specified topic.

1. int mosquitto_publish(struct mosquitto *mosq,int *mid,const char *topic,int payloadlen,const void *payload,int qos,bool retain)
2. Int * mid represents the pointer to the message_id integer of the current message. In this code it has been set to NULL.
3. Char * topic represents the topic string for publishing the given message, here it has been published to “home/sensordata”.
4. Payloadlen and payload represent the length and content of the payload to be published, respectively.
5. Qos represents **Quality of Service**. It is set to 1 which guarantees that the message is delivered at least once to the

broker. The client stores the message till it gets a PUBACK packet from the broker. A QoS of 1 means it is possible for a message to be sent more than once, in which case the DUP flag is set to true.

6. Client -> publishes with QoS 1 to broker -> Broker receives and immediately processes the message by sending it to the clients that subscribed for it -> Broker then sends a PUBACK to the publisher client -> On receiving PUBACK, publisher client deletes the stored message.

ix. Upon publishing the message, the client disconnects from the broker, and the loop is stopped, effectively ending this thread.

x. This function calls usleep() for 500 milliseconds so as to wait for that time before publishing another message to the broker.

xi. Steps iii to x are also followed for the remaining 20 clients, except the payload contains distance and switch_state instead.

xii. This process happens in an infinite loop.