Implementation of MQTT using a public broker

Sharath S Chellappa	1PI13EC092
Sharath Kumar S	1PI13EC090
S Rohith	1PI13EC083

1 AIM

To send messages using the MQTT protocol.

2 THEORY

MQTT(**MQ Telemetry Transport** or **Message Queuing Telemetry Transport**) is an ISO standard (ISO/IEC PRF 20922)publish-subscribe-based "lightweight" messaging protocol for use on top of the TCP/IP protocol. It is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited. The publish-subscribe messaging pattern requires a message broker. The broker is responsible for distributing messages to interested clients based on the topic of a message. Andy Stanford-Clark and Arlen Nipper of Cirrus Link Solutions authored the first version of the protocol in 1999.

The specification does not specify the meaning of "small code footprint" or the meaning of "limited network bandwidth". Thus, the protocol's availability for use depends on the context. In 2013, IBM submitted MQTT v3.1 to the OASIS specification body with a charter that ensured only minor changes to the specification could be accepted. MQTT-SN is a variation of the main protocol aimed at embedded devices on non-TCP/IP networks, such as ZigBee.

Historically, the "MQ" in "MQTT" came from IBM's MQ Series message queuing product line. However, queuing itself is not required to be supported as a standard feature in all situations.

Alternative protocols include the Advanced Message Queuing Protocol, the IETF Constrained Application Protocol, XMPP and Web Application Messaging Protocol (WAMP).

MQTT defines methods (sometimes referred to as *verbs*) to indicate the desired action to be performed on the identified resource. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server.

Connect

Waits for a connection to be established with the server.

Disconnect

Waits for the MQTT client to finish any work it must do, and for the TCP/IP session to disconnect.

Subscribe

Waits for completion of the Subscribe or UnSubscribe method.

UnSubscribe

Requests the server unsubscribe the client from one or more topics.

Publish

Returns immediately to the application thread after passing the request to the MQTT client.

3 CODE IMPLEMENTATION

```
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import java.util.concurrent.TimeUnit;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import com.opencsv.CSVWriter;

class SimpleCallback implements MqttCallback {

      @Override
      public void connectionLost(Throwable cause) { //Called when the client lost the
connection to the broker
      }

      @Override
      public void messageArrived(String topic, MqttMessage message) throws Exception {
         System.out.println("-------------------------------------------------");
         System.out.println("| Topic:" + topic);
         System.out.println("| Message: " + new String(message.getPayload()));
         System.out.println("-------------------------------------------------");
      }

      @Override
      public void deliveryComplete(IMqttDeliveryToken token) { //Called when a outgoing
publish is complete
      }
}

public class PahoSendRecieve {
   public static void main(String[] args) throws InterruptedException, IOException{
   String csvFile = "C:/Users/Sharath/Desktop/messages.csv";
   CSVWriter writer = new CSVWriter(new FileWriter(csvFile));
```

```
        String topic = "Message array";
        String content[] = {"Message1","Message2","Message3","Message4","Message5"};
        int qos = 2;
        String broker = "tcp://broker.hivemq.com:1883";
        String clientId = "Sharath";
        MemoryPersistence persistence = new MemoryPersistence();
        Ceaser ceaser = new Ceaser(15);

        try {
           MqttClient Client = new MqttClient(broker, clientId, persistence);
           MqttClient subClient = new MqttClient(broker,clientId,persistence);
           MqttConnectOptions connOpts = new MqttConnectOptions();
           connOpts.setCleanSession(true);
           connOpts.setUserName("sharath");
           connOpts.setPassword("sharath".toCharArray());
           System.out.println("Connecting to broker: " + broker);
           Client.connect(connOpts);
           Client.simpleCallback();
           Client.subscribe(topic,1);
           System.out.println("Connected");
           subClient.connect(connOpts);
           subClient.subscribe(topic, 1);
           System.out.println("subClient subscribed to topic"+topic);

           for(int i=0;i<5;i++)
           {
           System.out.println("Publish message: " + content[i]);
           String codestring = ceaser.encode(content[i]);
           MqttMessage message = new MqttMessage(codestring.getBytes());
              message.setQos(qos);
              Client.setCallback(new SimpleCallback());
              Client.publish(topic, message);
              TimeUnit.SECONDS.sleep(1);
              System.out.println("Broker:Message "+(i+1)+" published\n");
              System.out.println("Broker:Message "+(i+1)+" is
"+ceaser.decode(message.toString())+"\n");
              List<String[]> data = new ArrayList<String[]>();
              data.add(new String[] {topic, ceaser.decode(message.toString())});
              writer.writeAll(data);
           subClient.setCallback(new SimpleCallback());
           subClient.publish(topic, message);
           System.out.println("Broker:Message "+(i+1)+" published");

           }


          try {
```

```
            Thread.sleep(4000);
            writer.close();
            subClient.disconnect();
            subClient.close();
            Client.disconnect();
            Client.close();
          } catch(Exception e) {
            e.printStackTrace();
          }
          System.out.println("Disconnected");
          System.exit(0);
        } catch(MqttException me){
          System.out.println("reason " + me.getReasonCode());
          System.out.println("msg " + me.getMessage());
          System.out.println("loc " + me.getLocalizedMessage());
          System.out.println("cause " + me.getCause());
          System.out.println("except " + me);
          me.printStackTrace();
        }
    }
}
```

4 APPLICATIONS

MQTT is designed to support wireless networks with varying levels of latency due to occasional bandwidth constraints or unreliable connections. There are several projects that implement MQTT.

- Facebook Messenger. Facebook has used aspects of MQTT in Facebook Messenger for online chat. However, it is unclear how much of MQTT is used or for what.
- IECC Scalable DeltaRail's latest version of their IECC Signaling Control System uses MQTT for communications within the various parts of the system and other components of the signaling system. It provides the underlying communications framework for a system that is compliant with the CENELEC standards for safety-critical communications.
- The EVERYTHNG IoT platform uses MQTT as an M2M protocol for millions of connected products.
- On October 8, 2015 Amazon Web Services announced Amazon IoT based on MQTT.
- The Open Geospatial Consortium SensorThings API standard specification has a MQTT extension in the standard as an additional message protocol binding. It was demonstrated in a US Department of Homeland Security IoT Pilot.
- The OpenStack Upstream Infrastructure uses MQTT as a unified message bus between services.

5 OUTPUT

Connecting to broker: tcp://iot.eclipse.org:1883

Connected

Client subscribed to topicMessage array

Publish message: Message1

| Topic:Message array

| Message: BTHHPVT1

Broker:Message 1 published

Broker:Message 1 is message1

Broker:Message 1 published

Publish message: Message2

| Topic:Message array

| Message: BTHHPVT2

Broker:Message 2 published

Broker:Message 2 is message2

Broker:Message 2 published

Publish message: Message3

| Topic:Message array

| Message: BTHHPVT3

Broker:Message 3 published

Broker:Message 3 is message3

Broker:Message 3 published

Publish message: Message4

| Topic:Message array

| Message: BTHHPVT4

Broker:Message 4 published

Broker:Message 4 is message4

Broker:Message 4 published

Publish message: Message5

| Topic:Message array

| Message: BTHHPVT5

Broker:Message 5 published

Broker:Message 5 is message5

Broker:Message 5 published

Disconnected