MQTT PROTOCOL

WORKING

* The protocol uses a publish/subscribe architecture, which is event-driven and enables messages to be published to clients.
* In this the central communication is the MQTT BROKER. Here, each client that publishes a messages to the broker includes a topic { a topic is a simple string that can here more hierarchy levels, which are separated by slash. } into the message .
* Here the topic is the routing information for the broker. So, each client that wants to receive messages subscribe to the certain topic and the broker delivers all the messages with the matching topic to the clients. Hence, clients only communicate over the topic.
* This architecture enables highly scalable solution without dependencies between the data producers and the data consumers.
* Each Mqtt client has a permanently open TCP connections to the broker and if the connection is interrupted by any circumstances, the mqtt broker can buffer all the messages and send them to the client when it is back online.

IMPLEMENTATION

* FOR IMPLEMENTING THE SENSOR CLIENT

1. First step is to create an instance of the mqtt class.

Ex. Public class publisher

1. As a parameter for the constructor it is necessary to specify the URL of the broker and also client id.

Ex. Public static final String broker\_url = “ URL ”;

1. for the second part define the unique id for the broker. We can use the client id as the mac address of the computer because it is automatically uniques.
2. After creating the instance it is possible to try to connect to the broker with calling client.connect()
3. Apart from a simple connect, it is also possible to hand over the parameter.

For ex. For the clean session define a variable flag, when it is set to true the broker will wipe the session every time the client disconnects and otherwise it will keep the subscription and buffer the messages sent with quality of service.

For example :

Public class publisher

{

public static final String BROKER\_URL = "url";

private MqttClient client;

public Publisher()

{

String clientId = Utils.getMacAddress() + "-pub";

try

{

client = new MqttClient(BROKER\_URL, clientId);

}

catch (MqttException e)

{

e.printStackTrace();

System.exit(1);

}

}

}

* Now we have to implement the business retrieve the values and send them every second, therefore we use an infinite loop methods *.*Each method creates a MqttTopic object and a random value, which will then be published.
* Each message can be published with one of three quality of service levels (QoS). These levels are associated with different guarantees.
* A message send with level 0 doesn’t have a guarantee at all, it implies fire and forget.
* Level 1 guarantees that the message will at least arrive once, but can arrive more than once. Level 2 is the most sophisticated choice, which guarantees that the message arrives at the destination exactly once. The choice of QoS is a trade-off, between protocol overhead and the guarantee that the message arrives, because ensuring QoS 2 is using more bandwidth than QoS 0.
* The next step is implementing the subscribing client,

which is reading the values on the topics. Topic for the last will message to detect a failure of the sensor client. The initialization of the M qtt Client instance is almost the same, except we use *-sub* as a suffix for the client id. For receiving the messages sent by the sensor simulator, it is necessary to implement the MqttCallback interface. The MqttCallback interface defines three methods that need to implemented: connection Lost, message Arrived and delivery Complete. Connection Lost is called when the connection is unexpectedly closed form the MQTT broker. This method is the best place for a reconnect logic. The method message Arrived is called when the broker sends a new message to this particular client. Finally, there is the delivery Complete method that is called after a message with QoS 1 or 2 reaches the broker. For our use case implementing the message Arrived method is enough. All arriving messages should be print out with topic and payload.

* After implementing the call back, we have to make it known to the *Mqtt Client* before connecting. Also after the successful connection is established, it is necessary to subscribe to all topics, which should be sent to the client. In this case it should be everything that starts with *home*, so the usage of a multi-level wildcard saves us from subscribing to 3 different topics.