Politecnico di Milano

Internet of Things

Project report

# Lightweight publish-subscribe application protocol

*Authors:*
Le Calvez Antoine    Ibrahimi Mëmëdhe

*Project advisor:*
Assistant Professor A.Redondi

*Course instructor:*
Professor M.Cesana

June 22, 2017

## Contents

# List of Figures

## 1 Introduction

Among the 3 projects that are proposed we have chosen the subject 1 (8 points): Lightweight publish-subscribe application protocol. The goal of the project is to mimic a MQTT-like application with a MQTT Broker and 8 clients (Figure 1) with 3 phases: connection, subscription and publishing. We also have to deal with some QoS features and design choices that we're going to present in this report. We used TinyOS as the operating system while the simulation environment used to implement our application was TOSSIM.

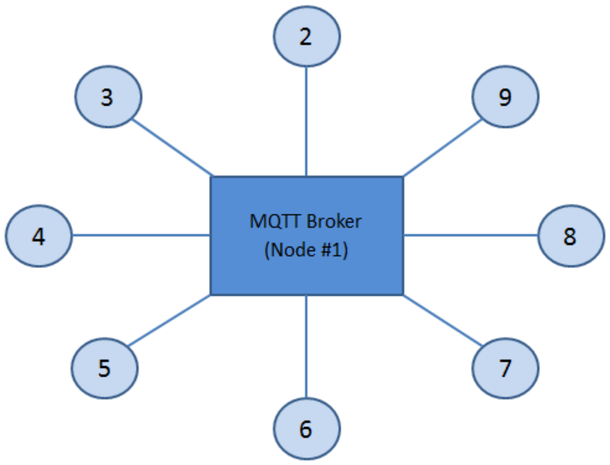

Figure 1: Topology of the MQTT model

## 2 Implementation of the MQTT model with TinyOS

### 2.1 Booting phase

The first phase of the implementation is booting which is performed by all the nodes to set up their radio. For the MQTT broker it also initializes variables that are useful to manage the network (connected nodes, subscriptions...) while for the clients it triggers the connection phase.

### 2.2 Connection of the clients to the MQTT Broker

After successful booting the clients need to get connected to the MQTT broker. To do so each node transmits a CONNECT message to the MQTT Broker which contains only one field which is the type of the message. We have defined the following message types: CONNECT=1, SUBSCRIBE=2, PUBLISH=3, FORWARD=4 which will be introduced progressively. In our implementation each CONNECT message has to be acknowledged by the MQTT Broker otherwise the node tries to send it again after 1 second (Figure 2). When the message is properly received, the Broker updates an array in which it stores the state of all the nodes ("0" = not connected, "1" = connected). For instance the array [1,0,0,1,0,0,1,1] indicates that nodes 2, 5, 8 and 9 (node 2 corresponds to the first column since the node 1 is the MQTT Broker) are connected while the others are not.

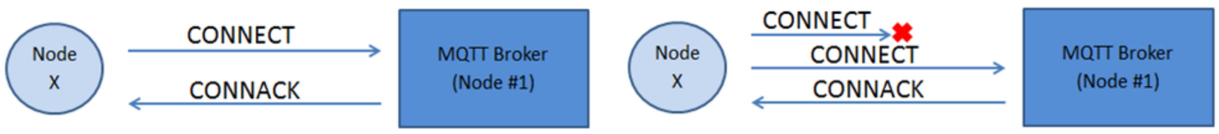

Figure 2: Acknowledgment policy for the connect phase

Once the acknowledgment of the CONNECT message has been received by the client, he waits 1 second before initiating the subscription procedure.

### 2.3 Subscription of the clients to the topics

In the MQTT model a node has to subscribe to some topics with a given QoS (Low QoS = 0, High QoS = 1). Here we have defined 3 topics: TEMPERATURE=1, HUMIDITY=2, LUMINOSITY=3. The SUBSCRIBE message is composed of:

- Message type: SUBSCRIBE ;
- Topic: array indicating the topics the node wants to subscribe to, it means each node can subscribe to more than one topic with only one SUBSCRIBE message ;
- QoS: array indicating the QoS for each of the topics ;

To determine which node subscribes to the different topics with the different QoS, we simply applied the following rules that are totally arbitrary:

- Nodes with even index subscribe to all the topics, odd index nodes subscribe to Temperature and Humidity only ;
- Nodes with index lower than 5 have High QoS, others have Low QoS ;

As we did for the connect phase, we have to deal with the acknowledgment issue. We used the same approach meaning that if the acknowledgement has not been received the client waits 1 second before sending again a SUBSCRIBE message to the MQTT Broker (Figure 3).

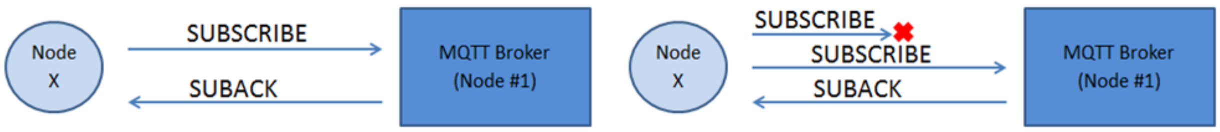

Figure 3: Acknowledgment policy for the subscribe phase

In order for the MQTT Broker to store the different subscriptions we have defined a matrix that is used only by the Broker. The rows represent the topics while the columns indicate the client node. The stored value $A_{ij}$ (i being the topic, j being the node) can be:

- "-1" the node j has not subscribed to the topic i ;
- "0" the node j has subscribed to the topic i with a Low QoS ;
- "1" the node j has subscribed to the topic i with a High Qos ;

An example of the subscriptions matrix stored by the Broker could be:

$$\begin{bmatrix} 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

In this simple case node 2 (corresponding to the first column since the node 1 is the MQTT Broker) has subscribed to the topic 1 (Temperature) with High QoS and to the topic 3 (Luminosity) with Low QoS. This matrix will be useful for the Broker in the publish phase because it has to know to which nodes it needs to forward the PUBLISH message (and the QoS).

Once the acknowledgment of the SUBSCRIBE message has been received by the client, he can start the publish phase which is performed by each node every 20 seconds (arbitrary choice).

## 2.4 Publishing a message

In order to implement the publish phase, we have decided to split it into 2 subphases: publish and forward (Figure 4). In our implementation the publish subphase corresponds to the transmission from the client to the MQTT Broker while the forward phase corresponds to the distribution of that message from the Broker to the subscribed nodes.

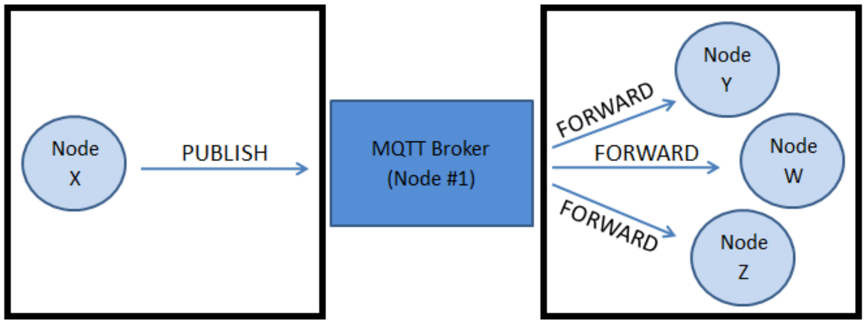

Figure 4: Publish and Forward subphases of our implementation

### 2.4.1 Publish subphase

In the publish subphase a client node sends a PUBLISH message to the Broker which is composed of the following fields:

- Message type: PUBLISH ;
- Value: the random value to be published on the topic. In case the QoS is High and the acknowledgement is not received correctly the node doesn't generate another random value, it tries to transmit the same value until the ACK is properly received ;
- Message ID: this field is useful to avoid forwarding two times the same message (in case the QoS is high and the ACK fails, the node would retransmit and the MQTT Broker would forward again the same message) ;
- Topic: topic concerned by the PUBLISH message ;
- QoS: if High, PUBLISH message needs to be acknowledged by the Broker; if Low, no acknowledgement required. In case the ACK is not received, the node sends again the message until it gets acknowledged (Figure 5) ;

Since each node can only publish on one topic, we decided to choose the topic arbitrarily: nodes 2 and 5 are Temperature sensors, nodes 4 and 7 are Humidity sensors, the others are Luminosity sensors. We also chose arbitrarily the QoS to be applied between the publishing node and the Broker: from the node 2 to 4 the QoS is High while for the others the QoS is Low. Once the MQTT Broker has received properly the PUBLISH message and the QoS has been fulfilled, the forward subphase can be performed.

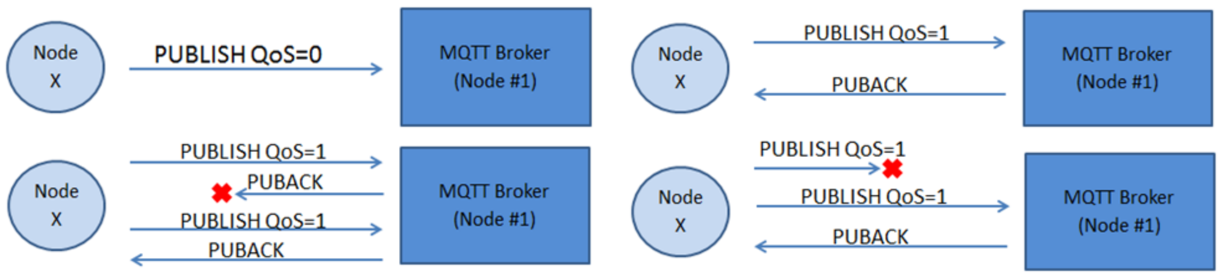

Figure 5: Acknowledgment policy for the publish subphase

### 2.4.2 Forward subphase

The FORWARD subphase deals with the distribution of the message to the nodes that have subscribed to the specific topic of the message. To perform it, the MQTT Broker copies the row of the subscriptions matrix corresponding to the topic concerned by the publish message in a temporary array. Then it starts to read it and everytime a "0" (Low QoS) or a "1" (High QoS) is found, the Broker tries to transmit a FORWARD message to the corresponding node with the specified QoS. Whenever a transmission is successful the temporary array is updated by overwriting "-1" instead of the previous value indicating that this forwarding is not to be done anymore. In case a transmission fails, the array is not updated, the Broker reads the next value and so on so forth till the end of the array and starts reading it again from the beginning until all the values are "-1". In case another node tries to publish a message while the broker is already forwarding another one, the publish message is discarded (displaying that the Broker is busy).

In our implementation, a FORWARD message is a little bit different from a PUBLISH message, it contains the following fields:

- Message type: FORWARD (and not PUBLISH anymore) ;
- Value: same as the one of the PUBLISH message ;
- Topic: same as the one of the PUBLISH message ;
- Source: original source of the message, it is not a fundamental field but it adds some information to the logs and was useful to debug ;
- QoS: QoS coming from the subscriptions matrix ;

## 3 Simulation with TOSSIM

To simulate our application with TOSSIM, we have simply used the Python script to create 9 nodes (MQTT Broker + 8 clients) that boot at different times (the MQTT Broker is the first to boot). All the links between clients and the Broker are characterized by the same quality (-60dBm) which is an arbitrary choice and can be modified as wished in the topology file. Running the application triggers the display of all the details from the booting phase to the publish one including transmissions and receptions together with the message fields and information on the acknowledgement.

## 4 Conclusion

This project was a great opportunity to deal with a practical implementation of an IOT application and to become more familiar with TinyOS and TOSSIM. The most interesting parts were without any doubt the design choices dealing with QoS and the consideration of the different problematic situations in which the application could get stuck.