Lightweight publish-subscribe application protocol

Luca Fochetta, Andrea Martino

September 16, 2017

1 Introduction

In this document we will try to summarise how we implemented

2 Implementation choices

2.1 ActiveMessages

In this project we need to handle different kind of messages, with different content and different purposes. One of the possible choices was to add a *packetId* field in the packet payload. Doing so we would have been able to understand the packet type just by simply looking at the first four bits of the payload. For example we could have used *0010* as the type id for the *PUBLISH* message and *0011* as the type id for the *SUBSCRIBE* message.
We didn't particularly like the idea of having one single component doing this check passing through a single *Receive*. So we decided to follow a different approach. We use as many components as the number of different packet we need to sort. For example in the *PanC* we use three different modules (*PublishModule*, *SubscribeModule*, *ConnectionModule*) that implements different *Receive.receive* each. Every *AMReceiverC* is built with different *Active Message ID*. So when *PanC* receive a *Publish* message only one of those *receive* event will be signaled.
Doing this kind of check at *Active Message* level makes our code cleaner and more expandable. It's very easy to add another type of message. You only need to initialize a *AMReceiverC* with a not used *Active Message ID*.

2.2 Events

Every component in our project heavily relies on events signaling in order to never wait for some data to be available. For example a publication is received in *PublishModule* the *PublishModule* itself will signal *PublishModule.OnPublishReceive* that is implemented by the *PanC* or the *Client*. Doing so allow us to split some of the logic between modules and the main component without ever incurring in heavy coupling between components. For example in the *PanC PublishModule.OnPublishReceive* publish message data is handled by *SubscribeModule* to get the list of subcribed nodes and then sent to the proper nodes via *PublishModule*.

2.3 Acks

We use explicit *SUBACK* and *CONNACK* messages in SubscribeModule and ConnectionModule but we have decided to use implicit *Active Message* ack to easily handle ack request and check for messages with QoS 1.

2.4 Messages

We use three different message structures. *struct* details can be seen in *Common/packets.h*.

- **simple_msg_t**: it only contains the sender ID. It is used for "simple" messages like *CONNECT*, *CONNACK* and *SUBACK*;
- **sub_msg_t**: contains every information for *SUBSCRIBE* needed to handle subscriptions.
- **pub_msg_t**: contains every information for *PUBLISH* needed to handle publishes.

3 Modules

In this chapter we will discuss about the module we implemented in order to achieve a good separation between all the different operation that the Node and the PanC have to do.

3.1 Common

We have two components that are in common between the *PanC* and the *Client*:

- **QueueSender**: a special sender that receives a generic message, inserts in a queue and then sends it. QueueSender potentially can be used by any components that needs to send a message.
- **PublishModule**: This module sends the publish message(using the *QueueSender*) and signals trough an event whenever a publish message is received.

3.2 Client

For the *Node* we have a principal module that is **ClientC**, that will integrate the components that are in common and the three components that will be explained later on. It will handle the signaled event from the integrated component and tell them when a new message must be sent. In the Client we have implemented three specific components:

- **ConnectionModuleC**: sends *CONNECT* message to the *PanC* and receives *CONNACK* message. When received, it signals an event to the *Client* main component.
- **Subscribe ModuleC**: sends *SUBSCRIBE* message (if the node wants to subscribe to some topic) and receives *SUBACK* message. When received, it signals an event to the *Client* main component.
- **FakeSensorP**: simulates different sensors and signals when a specific sensor is read to the *Client* main component.

For the *PanC* we have a principal module that is the *ServerC*, that will integrate the components that are in common and the two components that will be explained later on. It will handle the signaled event from the integrated component and when a publish message is received, it will retrieve the subscribed node to a specific topic and pass them to the *PublishModule* in order to redirect the message.

3.3 Server

In the Server we have implemented two specific components:

- **ConnectionModule**: handles the receive of the *CONNECT* message, signals it to the *Client* main component and implements the necessary method to send *CONNACK* to the node and add the node to the list of connected devices.
- **SubscribeModule**: handles the receive of the *SUBSCRIBE* message, signals it to the *Client* main component and implements the necessary method to send *SUBACK* to the node and add the node to the list of subscribed devices.