Group Project

1. Introduction

In the course we have seen two ways to send sensor data from IoT devices to MQTT subscribers:

1. With MQTT on TCP: The IoT device connects to an MQTT broker and publishes its sensor data.
2. With MQTT-SN on UDP: The IoT device sends the sensor data to an MQTT-SN gateway that forwards it to an MQTT broker.

Both solutions require an IP network (for example IPv6 with 6LoWPAN and RPL) on the device side. Despite being much more lightweight than a full IPv6 implementation, 6LoWPAN and RPL are still relatively complex beasts and it is not always easy to obtain good network performance, as you might have noticed in the exercise.

In this project you should build an IoT network using *Rime* where the sensor data is published through an MQTT-Rime gateway to normal MQTT subscribers. To do this, you have to implement

- a tree-based routing protocol using Rime
- an MQTT-Rime gateway

Please read the following sections carefully as they describe the requirements to the system you have to develop in this project.

2. System requirements

2.1 The sensor network

The sensor network consists of Zolertia embedded systems (Z1, Re-Mote and Firefly) communicating via IEEE 802.15.4. The sensor nodes must organize themselves into a tree with the gateway node as root of the tree. To select a parent node, a new sensor node either uses signal strength[1] or number of hops as criterion (you can pick one method for your implementation). New nodes can join the network at any time and nodes can also disappear. The routing must adapt to such changes. To simplify things, unlike RPL, a node selects exactly one parent, i.e., there is exactly one path from the root node to another node.

Important: In your implementation, you are only allowed to use the Rime[2] modules for

- single-hop (reliable) unicast, and
- best effort local area broadcast.

You are not allowed to use other protocol stacks nor the existing Rime modules for routing, multi-hop communication etc.

You can choose whatever message format you like inside the sensor network. Efficiency is important, of course. See what we said about MQTT-SN in the course. Your sensor nodes should produce at least two kinds of sensor data (e.g. temperature and something else). For tests, it might be useful to generate "fake" sensor measurements with random numbers.

2.2 The MQTT-Rime gateway

The gateway is a normal PC (e.g. running Linux). It receives the data packets from the IEEE 802.15.4 sensor network, translates them into MQTT publish messages and sends them to an MQTT broker which forwards them to the subscribers.

Sensors <-----> Gateway (PC) <-----> MQTT broker <-----> Subscriber

Subscribers can subscribe to the different kinds of sensor data provided by the sensors, e.g. with topics like "/node1/temperature" (this is just an example).

The gateway must also have some simple command line interface that allows sending configuration messages to the sensor nodes. It must be at least possible to configure nodes to either

a) send their sensor data periodically, or

[1] You can find several examples in the Internet how to get the signal quality of the last received packet in Contiki.
[2] See http://contiki.sourceforge.net/docs/2.6/a01798.html and the exercises for more information on Rime.

b) send their sensor data only when there is a change.

2.3 Bridging

If you want to run your software on real hardware (i.e. outside Cooja), some bridging functionality is needed in order to allow the gateway to communicate with the sensor network. As in the exercises, you can connect one of the Zolertia nodes to your PC via USB and use it as the physical root of the sensor network tree. There are several ways how the PC can communicate with the bridge node:

- The easiest is probably to use the "serialdump" tool shipped with Contiki. As explained in one of the exercises, that tool just forwards everything on stdin (PC→ node) and on stdout (node → PC). If you write your gateway software in Java, you can start the serialdump tool with Runtime.exec(…) and access stdin and stdout as input/output streams. There are similar ways in other programming languages.
- If you are familiar with C, an alternative can be to write your own tunnel software. If you take a look at the source code of the serialdump tool you can see that it does not do much more than opening the USB device as a serial interface.

Unfortunately, we can only give you a limited number of devices (around 4) per group for experiments with real hardware. This is enough to do some general tests. However, to test your solution for large sensor networks, you have to simulate the sensor nodes in Cooja. There are two ways how your gateway can communicate with a simulated bridge node:

- Over a network connection to Cooja. To this end, you have to start a Serial Socket Server on the simulated bridge node in Cooja (see the exercises).
- By installing the serial2pty plugin in Cooja: https://github.com/cmorty/cooja-serial2pty
  This plugin allows you to create a serial device for a simulated node in Cooja that you can access from outside. In that way, your simulated gateway node appears as a serial device on the PC like a real device connected over USB.

2.4 Optimizations

Stop sending if there is no subscriber

In MQTT, publishers always send data to the broker, even if there is no subscriber. This is a waste of energy and network bandwidth in sensor networks. Your gateway should automatically tell sensor nodes to stop sending data if there is no subscriber for the kind of data produced by the nodes. If you use mosquitto as broker, the gateway can subscribe to the logging information of the broker and receive a message when a subscriber arrives or leaves. This is explained on

http://www.steves-internet-guide.com/mosquitto-logging/

That is a simple method for the gateway to keep track of subscribers. However, it is also unreliable: What happens if a subscriber connects to the broker before the gateway has been started? In addition to the logging approach described above, implement also that subscribers can publish messages about the data they are interested in, so that the gateway can know what data is needed.

Aggregate messages

Intermediate nodes in the routing tree should look for opportunities to aggregate multiple sensor data messages into a single message instead of forwarding them one by one toward the root.

3. Deliverables

You have to deliver a short report in PDF format (~5 A4 pages, 10 or 11pt) describing

- the general structure of your system,
- the message format in the sensor network,
- how the network organization and the routing in the sensor network work,
- how the optimization works.

Keep the report short and only give the essential information, i.e. your design decisions.

The report must contain a link to a github or bitbucket repository read-accessible to us with the commented source code of the code for your sensor nodes, gateway, and other components you have developed.

You can use any programming language as long as it is not too exotic (e.g. Java, python, C, C++, kotlin are okay).