Politecnico di Milano

Accademic year 2016 - 2017

Internet of Things

Project Report

Lightweight publish-subscribe application protocol

Claudio Scandella - 853781

Lorenzo Raimondi - 859001

1 Project Aim

The project consists in the design and the implementation of a publish-subscribe application using TinyOS framework. The implemented protocol is a lightweight version of MQTT, capable of managing up to 8 nodes connected in a star topology. The central node, the MQTT Broker, can handle Clients connections, subscriptions, and publications. The Broker also supports two Quality of Service constraints, requested by publishing and subscribing nodes: with QoS = 0, publications and acknowledgments are sent with an "at most one" fashion, while with QoS = 1 "at least one" approach is used.

2 Design

As main design choice, we decided to decouple nodes' application logic from radio communication logic, in order to allow an independent development/update/maintenance of the two parts, and by consequence an easy task assignment. This approach led also in having only one component responsible of the Radio, allowing its safe usage. Each node component, named `NodeC`, being it the Broker or a Client, owns a communicator component, named `CommunicatorBufferC`, which is wired to radio components, needed for wireless communication. The communicator contains a queue of messages managed with a FIFO policy; the time at which the packet is added in the queue is decoupled by the time it is actually sent .

Regarding communication, we decided to use 4 types of message: CONNECT, SUBSCRIBE, PUBLISH, ACKNOWLEDGEMENT. While the first three types of message are unique, the fourth one is disambiguated using a specific code convention.

As regards publishing data, we decided to design a simulated sensor component, named `MultiSensorC`, in order to provide temperature, humidity, and luminosity data to be published.

At last, we wanted to make the application configurable by using a header file containing all the possible choices regarding nodes publication/subscription topics and QoS requests, in order to have a unique and easy configuration mechanism not hard-coded with application logic.

3 Runtime architecture details

3.1 Booting sequence

Once booted every Client node configures its state variables using the configuration header file: the nodes save locally the topics to which they will have to subscribe, with the relative QoS constraint, as well as the (possible) topic on which the node will publish messages. Afterwards, sensor and communicator components are started, as shown in Figure 1. The latter one is in charge of starting the radio. If this starting sequence succeeds, Client nodes can begin their connection phase.

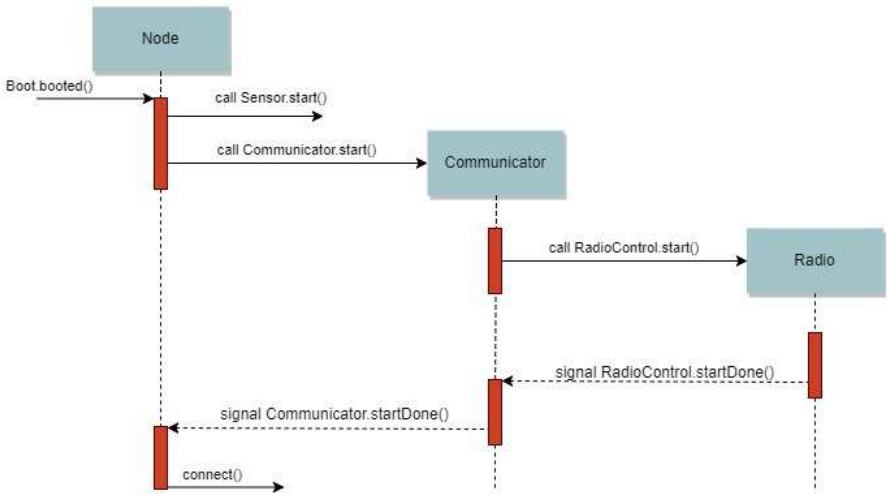

Figure 1: Booting sequence: radio is started by the Communicator.

3.2 Communicator

The communicator component is responsible for every task regarding radio communications. It is implemented as a wrapper for the Radio components (`AMSend`, `Receive`, `PacketAcknowledgements`), which allow the node to enqueue requests that will be processed, saved and sent as soon as possible. Requests can be added by the node specifying the request code (e.g. CONNECT, CONNACK, PUBLISH,...) and, if needed, a set of useful arguments. In such way the communicator is capable of creating the corresponding message and adding it to a FIFO queue: here messages are saved and, once they reach the head of the queue are sent. The queue is also managed by the communicator considering QoS constraints, since whether the packet sent has not been acknowledged by `PacketAcknowledgements`, the head of the queue will not change, so it will be re-transmitted as next packet. At the same time, the communicator is also responsible for packet receptions, which are promptly signaled to the node, in order to be managed based on the packet type.

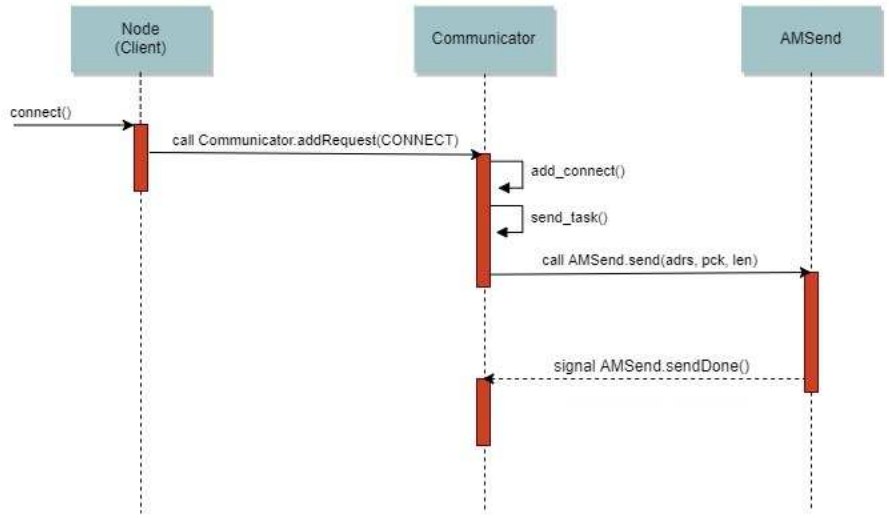

Figure 2: Detail of `connect` phase: Client node requests connection to the Communicator component, which is in charge of sending the relative message.

3.3 Connection and subscription

The connection is the first task triggered after communicator start up (only in Client nodes). Every node, in fact, tries to send a first CONNECT message in order to contact the Broker node. The Broker saves the occurred connection and answers to the initiator with a CONNACK message, which once received in the initiator node, triggers the node subscription. At this point, the node is capable of publishing messages, while it is not capable of receiving publications yet.

In the SUBSCRIBE message, the node reports all the needed information for the subscription, such as topics and relative QoS to be applied. The Broker receiving this message saves the subscriptions and answers with another acknowledgment message, in this case of type SUBACK. At this point the node is fully operative, being also capable of receiving publications to subscribed topics.

In both cases possible packet losses are managed by means of retransmission timers, that once fired trigger the specific action to request a re-transmission.

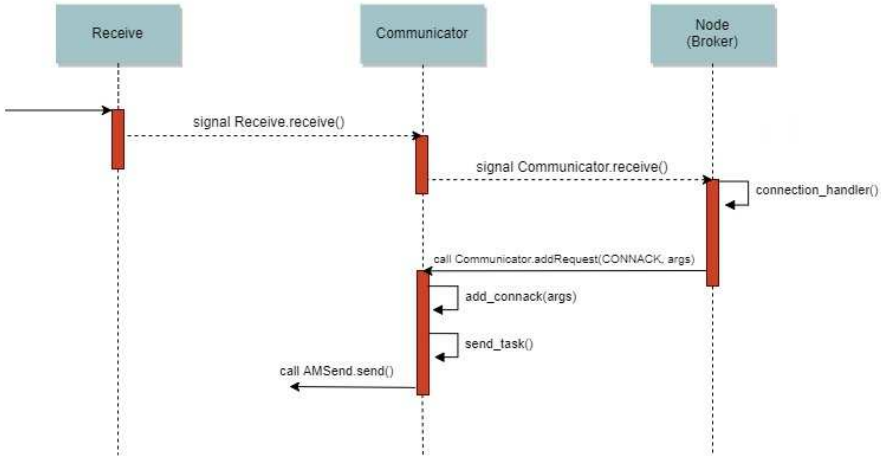

Figure 3: Handling of a new connection at Broker side: the new message flows from the Communicator component to the main one, which takes care of it and requests acknowledgment sending.

3.4 Publishing

The publication of new data starts from the sensor component, which periodically signals to the node the presence of a new value; the node reacts to this event by requesting to the communicator the publication of that value to the Broker.

As a side note, we decided to set a unique QoS requirement for all the messages published by a Client; the choice was done by considering that a node would request a QoS constraint not basing on the single messagge but on the topic itself; this also reflected in a simpler implementation.

The biggest part of the publishing task is then performed by the Broker: it is in charge of sending the incoming payload to all the subscribed Clients, respecting their QoS requirement; the Broker requests to its communicator a new publish request for every Client involved; then, as soon as the messages arrive in the head of the queue, they are actually sent.

4 Simulation

We simulated our implementation using TOSSIM. Regarding subscriptions and publications configuration, we complied the requirement of having at least three nodes subscribed to more than one topic setting five of them in such way, and in particular setting two of them subscribed to all the three topics, with different QoS constraints. Complete configuration is accessible in `NODE_CONFIG.h`. We used the Meyer Heavy model to include noise into the simulation.