POLITECNICO
MILANO 1863

Middleware technologies for distributed systems
Project 1

2020-2021

Authors:
Andrea Falanti
Federico Ferri
Abanoub Faltaous

Professors:
Prof. Luca Mottola
Prof. Alessandro Margara

Contents

1 Introduction

The project is about creating a system for managing contact tracing among multiple IoT devices. Contacts happen when two devices are within the same broadcast domain. The contacts among devices are reported periodically to the backend on the regular Internet. Whenever a device signals an event of interest, all the devices that had a contact with it must be notified. The project is based on the fact that all nodes are constantly reachable from a static IoT device that acts like an IPv6 border router.

2 Implementation

2.1 Technologies used

The implementation requires to use the COOJA simulator of ContikiNG for the network simulation of the IoT devices. For the backend we decided to use Akka. We decided to use publish/subscribe architecture style for the communication between the simulator and Akka. The communication happens through MQTT, as a bridge between Akka and COOJA motes.

2.2 ContikiNG

For the simulation of the IoT devices we use nodes to publish and subscribe and a rpl border router to allow nodes to connect to the broker on our machine. We decided also to use the broker Mosquitto, that is already installed on the provided virtual machine, to finalize the publish/subscribe through MQTT. Nodes implement three main functionalities:

- First, nodes use a state machine to connect with MQTT to the message broker and then publish data periodically (these data represent the events of interest).
- Second, is to send in broadcast to all other visible nodes its own id. This is implemented using udp on an IP address set to the link local all-nodes multi-cast address. When a node receives this kind of message, it performs a callback, that allows that node to publish its own id and the id of the sender of the message.
- Third, nodes receive data from their subscription through MQTT.

In our implementation there are three topics: one used for messages about contacts between two people, one for the events of interest and one for the notifications. Our nodes publish on the first two topic, while are subscribed to the third one, vice versa for Akka which is subscribed to the first two topics and publish on the third one. When a node from the simulation published a message on a topic, the broker on our machine takes care to publish it on a broker server, in our case *test.mosquitto.org:1883*. We need a bridge to publish/subscribe from

local broker to remote broker, so we need to configure the local broker as follow:

```
connection bridge-01
address test.mosquitto.org:1883
topic # out 1
topic # in 1
```

The first line is the connection name that we choose ('bridge-01'). The second line says the address of the remote broker. The last two rows allow to bridge all topic since we are using the wildcard '#', 'in' specifies on which topic the broker can receive from the remote broker and 'out' specifies on which topic the broker can publish on the remote broker. '1' term express the QoS (quality of service), which for MQTT is "at-least-once". Moreover to connect correctly to our broker we need to define a unique client-id, username and a password (username and password not need to be unique).

2.3 Akka

The backend server is realized by using the Akka middleware and its composed by three main parts:

- the server main file, that bootstrap the server and the actor hierarchy. At launch, it initialize the server actor, then it creates the MQTT connection with the broker. After the connection is established, the server subscribes to the topics where the IoT devices publish the messages about contacts and events of interest, providing also the functions to deserialize their json content to build actual message instances for the server actors.

- the server actor, that has the references to all device actors, which are instantiated as children of it. It receives all the messages from the server main file and distributes them to the correct device actor. If the IoT device that have sent the message is not present in the actor hierarchy, the respective device actor is created as a child of the server actor. When it receives notification messages from the device actors, it sends a message on the apposite topic of MQTT, so that IoT devices can get the notification. Server actor also perform a simple fault tolerance strategy, restarting the device actors in case of any exception, restoring also the previous state of the actors so that the contacts set is not lost.

- the device actors, that represent a digital abstraction of the physical IoT devices. In our case they simply store the set of ids of devices with which they have contacts. When they receive contact messages they simply add the ids to the set. When they receive an event message, they produce and send notification messages to the server actor, one for each device to contact.