Contact Tracing with Body-worn IoT Devices

Middleware Technologies for Distributed Systems

Chiara Marzano
Massimiliano Nigro
Daniele Paletti

29/09/2021

Contents

1 Overview **2**

2 Technology choices **2**

3 Assumptions **3**

4 Structure **3**

5 Design choices **3**
5.1 Topic Structure 3
5.2 Actor modelization 4

1 Overview

This project simulates a contact tracing application for body-worn IoT devices. These devices move and use the radio to detect if any other is close. If they are one hop away from each other, therefore reachable by broadcast, the wearers are considered in contact. The backend receives data about contacts and devices raise events of interest, which are also reported to the backend and to devices that were in contact with it.

2 Technology choices

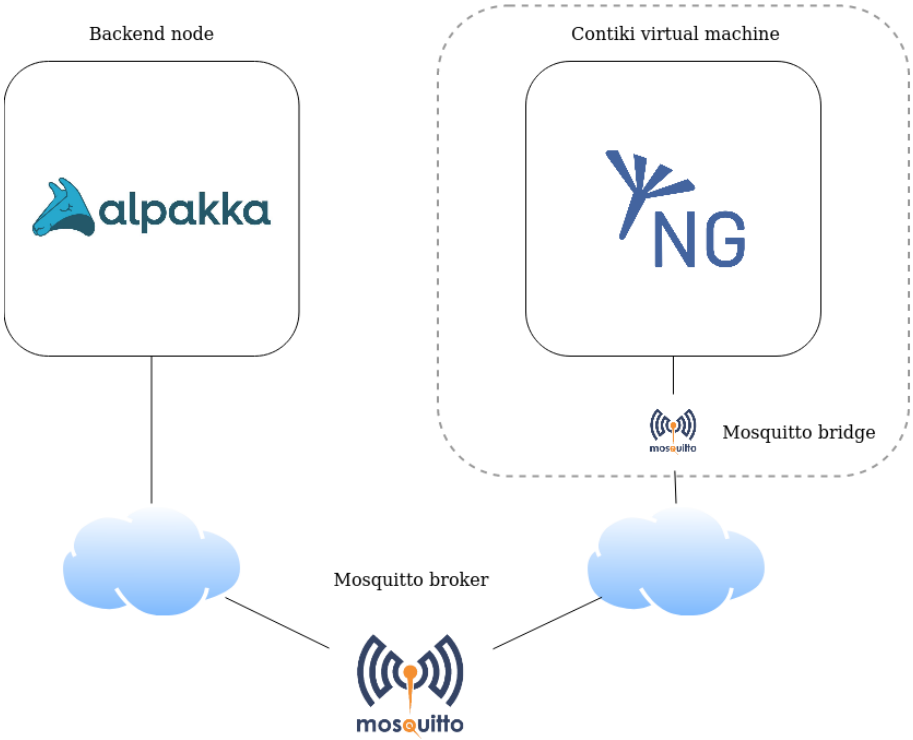

Figure 1: Project setup

For backend processing we chose AKKA to leverage the actor model which allowed us to model the COOJA motes as independent actors.
On the other hand, Kafka didn't provide an accurate model for the indpendent behaviour of each mote.
On top of this, AKKA through Alpakka provided an MQTT connector that facilitated the communication between the motes and the backend.

As a connector we chose Alpakka MQTT over Alpakka MQTT Streaming because it provided a more convenient interface for our use case and is more mature as a project.
We used Mosquitto as an MQTT broker, because it is the most adopted and documented open source broker. We decided to bridge the Mosquitto server inside the virtual machine with one outside of it and expose it through ngrok.

3 Assumptions

We assumed:

- IoT devices to be constantly reachable, possibly across multiple hops, from a single static IoT device that acts as a IPv6 border router
- the address of the Mosquitto broker to be known both to the Mosquitto bridge and to the backend
- simulations run only on COOJA motes

4 Structure

The contact tracing node behaviour can be summarized as:

- Neighbour discovery: neighbors are fetched periodically through the rpl-dag.
- Publish of contacts and events of interest to the backend: leveraging the MQTT protocol. Events of interest publication is randomic.
- Notification receipt from backend: taking advantage of the MQTT protocol and signaling the receipt through logging on the console of the COOJA simulator.

5 Design choices

5.1 Topic Structure

We exploited the hierarchical topic structure of the MQTT protocol to lessen the computational burden on the motes. The topics we used are:

- /mote_id/notifications: to receive event of interest notifications from the backend.
- /mote_id/contacts: to publish the contacts of the mote.
- /mote_id/events: to publish the events of interest.

Using this structure, the nodes only needed to keep track of three topics, while still allowing a coherent and simple routing between the actors in the backend.

5.2 Actor modelization

We modeled each mote as an independent actor. Each actor is named with its correspondent mote_id. It keeps track of the list of its contacts(mote_ids) and updates it by receiving publish on its own "/mote_id/contacts" topic. Upon receipt of an event of interest on its own "/mote_id/events" topic, it performs a publish on the notification topics correspondent to each contact.