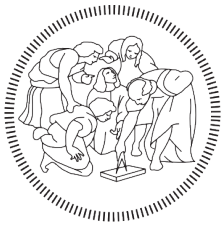
Contact Tracing with Body-worn IoT Devices

Design Document for the Akka & Contiki-NG project of the course "Middleware Technologies for Distributed Systems".

Authors:
Accordi Gianmarco
Buratti Roberto
Motta Dennis



POLITECNICO
MILANO 1863
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Italy
21/03/2021

Contents

1 Introduction

The purpose of this report is to explain the work done by this group on the project "Contact Tracing with Body-worn IoT Devices" for the "Middleware Technologies for Distributed Systems" course of "Politecnico di Milano". The analysis of the task to be solved has been analyzed and taken from the projects' pdf provided by the professors [1].

2 Project Analysis and Assumption

The **goal of this project** is to implement a contact tracing application using IoT devices as sensors. Every time two such devices are within the same broadcast domain, that is, at 1-hop distance from each other, the two people wearing the devices are considered to be in contact. The contacts between people's devices are periodically reported to the backend on the regular Internet. Whenever one event of interest occur to an IoT, every device that was in contact with it is informed.

2.1 Actors in the system

- **Sensors**: The IoT devices used for contact tracing running Contiki-NG.
- **Servers**: The servers used in each region to collect the contacts sent by Sensors.
- **Event of Interest Reporters (EoIRs)**: The devices used to report event of interests. In a real-life scenario these devices would be in the hands of local health care authorities.

2.2 Regions and logical sharding

Every IoT is assigned to a specific region, hardcoded in its implementation. This allow the server to use a logical sharding approach having a single Akka instance that is responsible for each region.

2.3 Assumptions

- The IoT devices are assumed to be constantly reachable, so network partitions are not considered.
- The IPv6 addresses of the IoT devices are assumed to be constant over time.
- An EoIR knows the IPv6 address of the person affected by an event of interest and to which region the device of the person is assigned to.

3 Design

As can be seen in Figure 1 we decided to create a symmetrical design where both Sensors and EoIRs can be created dynamically and they communicate with servers (that instead are more static) through a pub/sub architecture.
EoIRs use the pub/sub Akka facility to communicate with a server in a certain region while sensors use the MQTT protocol to communicate with a server in a certain region.
The main difference between EoIRs and Sensors is that Sensors need also to subscribe to an MQTT topic to receive notifications in case they had been in contact with another sensor that signaled an event of interest.

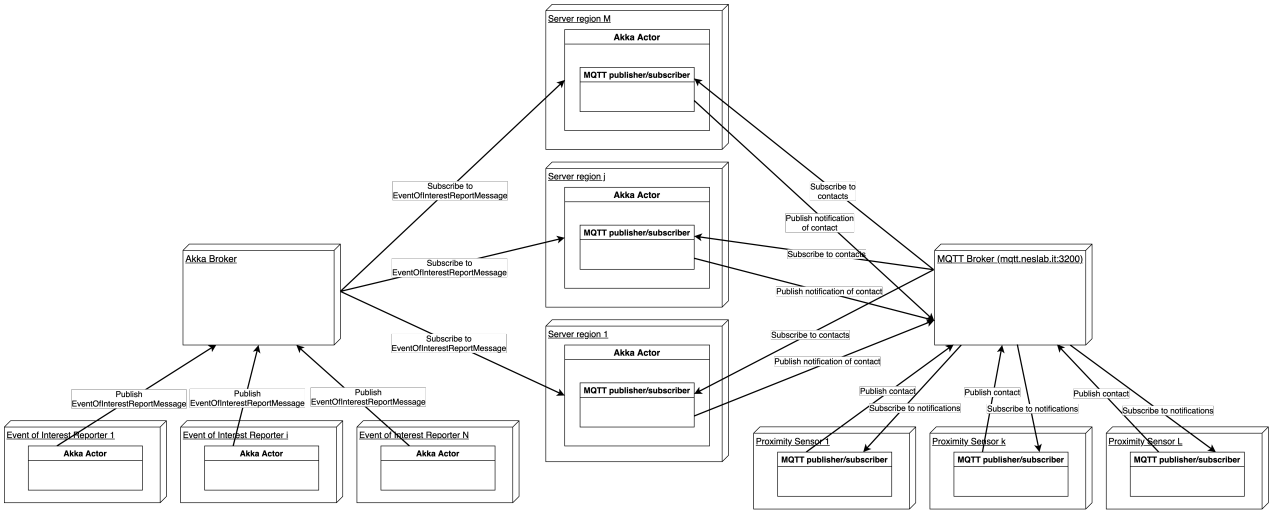


Figure 1: The architecture of the system.

4 Implementation

4.1 Sensors - using Contiki-NG

In order to implement the communication among the various IoT sensors we have used the RPL rooting protocol, which helps us to reconstruct a DAG in each node in order to implement the communication. This also allows us to know which are the nearest node to the IoT sensors: **we consider as "near" another IoT sensor that is at one-hop distance on the DAG we have reconstructed locally**. Than in order to test our system we have placed in the network a Border Router to which each sensors must be able to connect, and this node will provide connectivity to the internet. After the RPL tree is constructed each node publish to an MQTT Broker, on a topic depending on the region X that is assigned to that IoT (AccordiBurattiMotta-Topic/contact/regionX/json), with a payload that contains its IPv6 address and the IPv6 addresses of the nodes near it. This operation is repeated with an interval defined in the code that can be configured in the source code by modifying the macro **STATE_MACHINE_PERIODIC**. When the backend receives an event of interest from an EoIR it notifies all the sensors that have been in contact with the infected sensor, to do this it makes a publish on the topic to which each sensor Y is subscribed (AccordiBurattiMotta-Topic/notif/sensorY/json), with a payload containing the timestamp of the contact.
The IPv6 address is chosen as identifier because it is unique in the same broadcast domain and allow every node to know the IPv6 of its neighbors without contacting them (since it's already stored in the IoT thanks to the RPL implementation). The topics of the sensor is specified in the file *project-conf.h*, with also other configurations settings.

4.2 Publish

In order to fill the payload of the publish performed by each node, we use the function *void publish(void)* in which we take the IPv6 of the local node with the call to *uip_ds6_get_link_local(-1)*, and then we iterate over the neighbors table using *nbr_table_next()*. In this way we collect the address of the neighbors and

we add all this information inside the payload of the publish message. The sensors will send in the payload all the most recent contacts, also if it is no more in contact with the other node.

4.2.1 Finite state machine

The implementation of the contiki-NG process consist on an infinite loop where periodically a timer expires and triggers a transaction on a finite state machine. The FSM has 7 states:

- 0 STATE_INIT where the configuration is loaded,
- 1 STATE_REGISTERED where the IoT is registered and starts to try to connect to the border router,
- 2 STATE_CONNECTING where the IoT remains until it's connected to the border router,
- 3 STATE_CONNECTED is the state in which the IoT enters when it's connected to the border router. Also, in this state the sensor subscribes to notifications of events of interest.
- 4 STATE_PUBLISHING where periodically the IoT publishes on an MQTT topic the list of contacts,
- 5 STATE_DISCONNECTED is the state in which the IoT enters when it's disconnected and retries to connect,
- 6 STATE_NEWCONFIG where the IoT tries to load a new configuration

plus 2 error states :

- 0xFE STATE_CONFIG_ERROR is the state in which the IoT enters if there is an error in the configuration,
- 0xFF STATE_ERROR is the state in which the IoT enters if there is a generic error.

4.3 Servers & EoIRs - using Akka

Each Server in a region is represented by an Akka actor called ServerActor. This actor also contains the MQTT module used to receive contacts sent by sensors (implemented by subscribing to the MQTT topic "AccordiBurattiMotta-Topic/contact/regionX/json") and to send notifications of contacts with devices affected by an event of interest (implemented by publishing to the MQTT topic "AccordiBurattiMotta-Topic/notif/sensorY/json").
EoIRs are also represented by an Akka actor, called EventOfInterestReporterActor. This actor sends a report of an event of interest that affected a certain device to the region associated with the device.

4.3.1 Crash tolerance

The server has been developed in such a way that is crash tolerant, in fact contacts received are saved on disk so that in case a crash happens the server retrieves its previous state.
In practice there is an Akka supervisor actor that manages failures of the ServerActor and restarts the ServerActor whenever an exception occurs.
We also developed a way to simulate server crashes: it is possible to simulate a server crash by sending with an EoIR an event of interest which affect a device with id equal to -1.

4.3.2 Testing

As said previously we tested the crash tolerance of the server by simulating a crash manually thanks to a specific command sent by an EoIR.
But we also developed a total of 7 automatic tests using JUnit to test the data structure that we created and where the server saves the contacts it receives.
We also developed a Java class called ProximitySensor that simulates the behaviour of a Contiki-NG sensor. This allowed us to perform some tests on the whole system architecture by artificially creating random contacts between instances of the ProximitySensor Java class.

5 Notes

The border router is not considered differently from any other node in the contact tracing algorithm so also contacts with the border router are reported to the server and a notification is sent also to the border router in case of an event of interest. This fact does not pose any problem since it doesn't affect the functionality, in fact:

- No extra messages are produced since the report is done periodically by each node and is not event based;
- The fact that also the border router is counted doesn't affect the discovery of other contacts, so all the other contacts are discovered correctly;
- The notifications are received correctly by every node and since the notifications are delivered using the subscription to an MQTT topic the notification to the border router is never sent.

References

[1] Luca Mottola and Alessandro Margara. Middleware technologies for distributed systems - exam projects 2020/2021, 2021.