UNIVERSITY OF PISA

School of Engineering

Master of Science in Computer Engineering

Internet of Things

PROJECT DOCUMENTATION

Smart Intensive Care Unit

WORKGROUP:

Diego Casu

ACADEMIC YEAR 2020/2021

Contents

1 Introduction **3**

2 Design **4**

2.1 Overview 4

2.2 MQTT monitor 6

2.3 CoAP monitor 7

2.4 Collector 8

3 Testing **10**

3.1 Simulated environment 11

3.2 Hybrid environment 14

1. Introduction

Vital signs monitors are devices deployed in intensive care units to continuously track medical conditions of severely ill patients: they are a compact, lightweight and portable solution to assist clinicians in the accurate evaluation of the patient's general health and well being. Such devices are employed in hospitals, clinics and doctor's offices, offering multiple sensors to measure basic vital signs like temperature, pulse rate, respiration and oxygen saturation.

The traditional use of vital signs monitors is mainly based on a reactive approach: in addition to showing information on a screen, the classical devices detect if one or multiple vital signs fall outside healthy levels, issuing a warning or an alarm. If the latter occurs, the system not only plays a loud beeping noise, but reports the possible life-threatening problem to a control room, so that the healthcare staff can properly react. In such cases, the proactive approach consists only in periodic visual checks made by clinicians, whose frequency is determined heuristically based on consolidated best practices.

A more advanced approach exploits the continuous flow of clinical data produced by the available sensors to detect early signs of patient health deterioration by means of artificial intelligence techniques. This approach allows for an accurate and constant monitoring of multiple patients, without involving doctors unnecessarily: the healthcare staff is notified only when the early warning score of a patient crosses a threshold, so that a potential emergency situation can be faced — and possibly avoided — in advance.

The objective of this project is to design and implement an IoT telemetry and control system for vital signs monitors deployed in an intensive care unit, with the capability of detecting and signalling early deterioration of patients' health: sampled data are sent by the monitors, either using MQTT or CoAP as application protocol, to a collector, which saves them in a database for both historical data collection and analysis.

The document is organised as follows: Section 2 discusses the design of the IoT system and its components, with particular attention to the implementation of the MQTT and CoAP communication flows; Section 3 reports the testing results obtained reproducing the behaviour of vital signs monitors in a simulated environment (*Cooja*) and in a hybrid environment (*Cooja* plus a real testbed).

2. Design

2.1 Overview

The telemetry and control system is characterised by the presence of one or multiple networks of vital signs monitors, installed in an intensive care unit. The monitors are IoT devices equipped with the Contiki-NG operating system and exchanging data with a collector, which is executed on a standard apparatus deployed either on-premises or on the cloud. The networks are LLNs exploiting the 802.15.4 and IPv6 protocols, where the multi-hop communication is made possible by RPL and the traffic with the collector is enabled by a dedicated border router for each network.

Vital signs monitors are supplied with multiple sensors and an alarm system: the latter is able to reproduce a warning sound and turn on a set of LEDs when activated, so that a life-threatening situation can be signalled both acoustically and visually. The devices support MQTT and CoAP as application protocols to exchange telemetry and command information, eventually with an MQTT broker placed either on-premises or on the cloud to guarantee the relay of messages.

The collector is responsible for receiving data from the devices, saving them into a database for historical collection and running a periodic analysis to detect early signs of patient health deterioration. The overall system structure is represented in Figure 1.

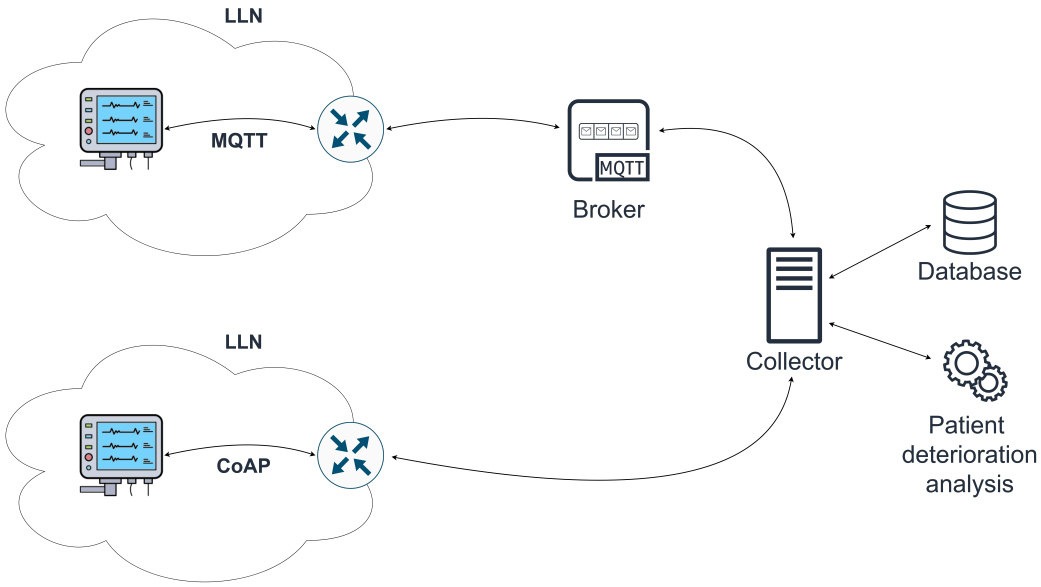

Figure 1: High level organisation of the IoT telemetry and control system.

More in details, a vital signs monitor operates in the following way:

1. once turned on, it waits until it is connected to the network, namely until it has a global IPv6 address and a default route;

2. once connected, it sends a registration message to the collector, using its preferred application protocol. This allows the collector to instantiate local resources dedicated to the remote device and to start listening for its messages;

3. if the registration succeeds, it is ready to be attached to a patient and waits for a patient ID to be provided on the serial line. Once a patient has been prepared and her ID inserted, the monitor is fully operational;

4. during the fully operational state, it periodically acquires health information by means of its sensors. The samples are processed individually, since different sensors could have different sampling periods. Once a new sample is ready, the monitor sends it to the collector and checks if it should be considered as life-threatening by using pre-configured thresholds. If this occurs, it turns on the alarm system to alert the healthcare staff and notifies the collector about the event;

5. during the fully operational state, the collector periodically analyses the received patient data to detect early signs of health deterioration. If the algorithm recognises such a situation and, at the same time, the monitor is not known to be already in an alarm state, the collector notifies the device to turn on the alarm system;

6. in case of alarm, a team of clinicians provides the needed medical care. Even if the acoustic signal stops after a certain time, the monitor stays in the alarm state indefinitely, unless a clinician informs the monitor itself about the end of the intervention. This notification is provided by pressing a button of the device, which triggers the shutdown of the alarm and the sending of a notification to the collector;

7. when the monitor must be detached from the patient and prepared for the next one, it must be reset by pressing another specific button. This brings back the device to the state described at point 3.

At each step, the exchanged application data is encoded in plain JSON using a proprietary format, since the carried information is limited in size, a punctual representation is not needed and interoperability is not required. This choice allows for a more lightweight communication, faster processing times and a simpler representation of information.

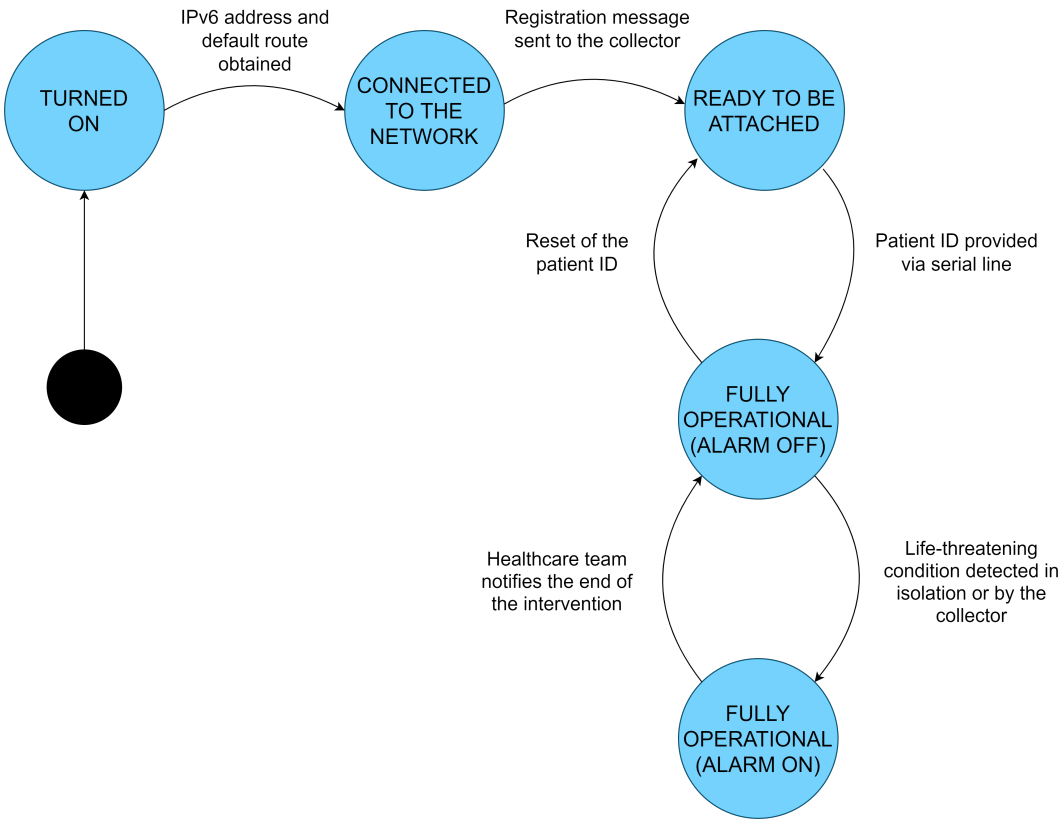

Figure 2: State diagram of a vital signs monitor.

2.2 MQTT monitor

An MQTT vital signs monitor connects as a client to an MQTT broker, publishing and subscribing to dedicated topics to communicate with the collector. Each monitor is characterised by an ID, which is chosen to be equal to its global IPv6 address. It publishes to the following topics:

- `cmd/smartICU/collector/monitor-registration`, to register itself to the collector. The format of the message is `{"monitorID": "MONITOR_ID", "registration": true}`;
- `cmd/smartICU/collector/patient-registration`, to notify the collector about the ID of the patient currently attached to the monitor. A message is sent every time a patient ID is inserted or removed, so that the collector can understand which monitors are effectively operating. The format is `{"monitorID": "MONITOR_ID", "patientID": "PATIENT_ID"}`, where `"patientID": ""` denotes a monitor without a patient;
- `telemetry/smartICU/MONITOR_ID/patient-state/SENSOR_NAME`, to notify the collector about the availability of a new sample from a sensor; the sensor name, if composed by multiple words, is hyphenated. Each monitor has a number of dedicated topics embedding its ID, which makes them unique and allows to omit

the ID itself in every message. The format of the messages is `{"sensorName": SAMPLE_VALUE, "unit": "MEASUREMENT_UNIT", "timestamp": TIMESTAMP}`, where `SAMPLE_VALUE` and `TIMESTAMP` are numbers. For example, supposing that the monitor `fd00::41` has an heart rate sensor, it will publish the sample 100 in `telemetry/smartICU/fd00::41/patient-state/heart-rate` with the message `{"heartRate": 100, "unit": "bpm", "timestamp": 156}`;

- `telemetry/smartICU/MONITOR_ID/patient-state/alarm-state`, to notify the collector about the activation or deactivation of the alarm system. Also in this case, each monitor has a dedicated topic. The format of the message is `{"alarm": true}` for alarm activation, `{"alarm": false}` for alarm deactivation.

For what concerns subscriptions, only one topic is involved:

- `cmd/smartICU/MONITOR_ID/patient-state/alarm-state`, from which the monitor receives commands to turn on the alarm system, when an early degradation of the patient's health is detected. The format of the message is `{"alarm": true}` only, since the deactivation of the alarm is not supported and must be done physically by the clinicians after the intervention has been completed.

Since the Contiki-NG implementation of MQTT does not provide an output queue and allows for only one message at a time to be sent to the broker, the MQTT monitor manages an internal queue to store messages rejected by the OS module. The queue is employed to store overflowing messages every time multiple sensors report a new sample at the same time or a sample triggers an alarm. The queue is emptied periodically, one message at a time, and its size can be configured to minimise or avoid the loss of messages.

2.3 CoAP monitor

A CoAP vital signs monitor acts both as a client and server: it registers to the collector issuing a `POST` to the `/registeredMonitors` endpoint, with payload `{"monitorID": "MONITOR_ID", "registration": true}`; afterwards, it exposes a set of resources to notify the collector about state changes and to receive commands. The telemetry related resources are observable, in order to minimise the number of exchanged messages. More in details, it makes available the following endpoints:

- `/registeredPatient`, holding the ID of the patient currently attached to the monitor, in the format `{"patientID": "PATIENT_ID"}`; if `"patientID": ""`, the monitor has not a patient attached. This endpoint accepts only `GET` requests;

- `/patientState/sensorName`, holding the last value sampled by the corresponding sensor, in the format `{"sensorName": SAMPLE_VALUE, "unit": "MEASUREMENT_UNIT", "timestamp": TIMESTAMP}`, where `SAMPLE_VALUE` and `TIMESTAMP` are numbers. For example, supposing that the monitor has an heart rate sensor, it will make available the last sampled valued, suppose 100, at the endpoint `/patientState/heartRate` as `{"heartRate": 100, "unit": "bpm", "timestamp": 156}`. This endpoint accepts only `GET` requests;

- `/patientState/alarmState`, holding the current status of the alarm system. If the latter is activated, it reports `{"alarm": true}`, otherwise `{"alarm": false}`. This endpoint accepts `GET` and `PUT` requests, where the latter can be used by the collector to turn on the alarm when an early degradation of the patient's health is detected. As underlined for the analogous MQTT command topic, only `{"alarm": true}` payloads for the PUT request are accepted, since the deactivation of the alarm is not supported and must be done physically by the clinicians after the intervention has been completed.

2.4 Collector

The collector is responsible for interacting with both MQTT and CoAP vital signs monitors, managing the connection to the database where patient data are stored and running a periodic analysis to detect a possible early degradation of the health of patients currently under medical care.

Before being fully operational, every device must register itself and the attached patient to the collector: this is required to instantiate local resources keeping track of the current state of the device (alarm activated or deactivated) and to avoid the embedding of the patient ID in each message. Knowing the latter also ensures that the periodic analysis involves only patients effectively attached to vital signs monitors and not dismissed ones.

To receive information from the MQTT monitors, the collector subscribes to the topics:

- `cmd/smartICU/collector/+`, to receive commands concerning monitor and patient registrations;
- `telemetry/smartICU/+/patient-state/+`, to receive telemetry data sent by all the monitors, including alarm states.

To send information to the MQTT monitors, the collector publishes to the topics:

- `cmd/smartICU/MONITOR_ID/patient-state/alarm-state`, to turn on the alarm system of the monitor specified in the topic itself.

The message formats are the ones described in Section 2.2.

For what concerns the CoAP protocol, the collector acts both as a client and server. It exposes a single resource, making it available at the following endpoint:

- `/registeredMonitors`, holding the list of registered vital signs monitors, in the format `{"registeredMonitors": ["MONITOR_ID_1", ... , "MONITOR_ID_N"]}`. This endpoint accepts GET and POST requests, where the latter is issued with payload `{"monitorID": "MONITOR_ID", "registration": true}`. The removal of registrations is not available, since a monitor can just re-register on a new startup to reset its saved state in the collector.

The registration is fundamental when CoAP is used, since it is the only way to let the collector understand that a new monitor has joined the network: the collector is deployed in a traditional network, separate from the LLNs, and so it is not possible to exploit the CoAP discovery capabilities. After the registration, the collector sets

up the observe relations to receive updates about the telemetry resources of the device; the observe relations are instantiated only once and they are not torn down each time the patient ID is reset.

Periodically, the collector analyses the data of patients in the intensive care unit to detect early signs of health deterioration; the analysis can exploit the entire historical telemetry data saved in the database. If an alarming situation is detected, the collector triggers the alarm system of the involved monitor issuing a `PUT` to its `/patientState/alarmState` endpoint, as described in Section 2.3. If the collector is aware that the alarm system of a monitor is already turned on, it avoids to perform the analysis and so to send an unnecessary command.

It should be noted that, given the maximum age mechanism associated to observable resources, spurious telemetry data could be saved in the database, for example not-initialised values obtained in response to the extended GET request used to set up the observer relations, or duplicates. It is supposed that such records are properly filtered out by the analysis algorithm.

3. Testing

The vital signs monitors employed in the system are IoT devices equipped with the Contiki-NG OS. Not having at disposal real monitors, the testing is performed simulating their behaviour with dedicated modules written in C, which can be exploited by the component implementing the MQTT or CoAP-based application. In particular:

- the sensors are simulated instantiating dedicated processes, one for each type. A sensor process can be started (without beginning the sampling) and stopped by a subscriber process, which will be the one receiving notifications about the availability of new values. Once started, a sensor process can receive commands, under the form of event posts, to start or stop the sampling activity, during which it extracts random values in a pre-configured range. In the current implementation, the monitor can interact with heart rate, blood pressure, respiration, temperature and oxygen saturation sensors;

- the alarm system is simulated with a structure holding the states of the system and of the acoustic signal. When activated, both the states are changed to "ON" and a timer to replicate the automatic shutdown of the warning sound is set: after the given period, the acoustic signal state changes to "OFF", while the alarm state stays "ON". The visual signals are simulated by turning on all the available LEDs;

- the buttons used to notify the end of the alarm state and the reset of the patient ID are substituted by a single button, where the action to perform is chosen depending on the amount of time for which it is kept pressed.

The collector is implemented in Java and divided into two main modules acting as MQTT and CoAP collectors. The former is built using the Paho library, while the latter exploits Californium. For testing purposes, the analysis routine is replaced by a stub module performing a random extraction that determines if the patient's health is worsening or not. The MQTT broker used in all the tests is *mosquitto*.

In the following sections, the results of the tests performed in a purely simulated environment by means of *Cooja* and in a hybrid environment, exploiting *Cooja* and a real testbed, are presented.

3.1 Simulated environment

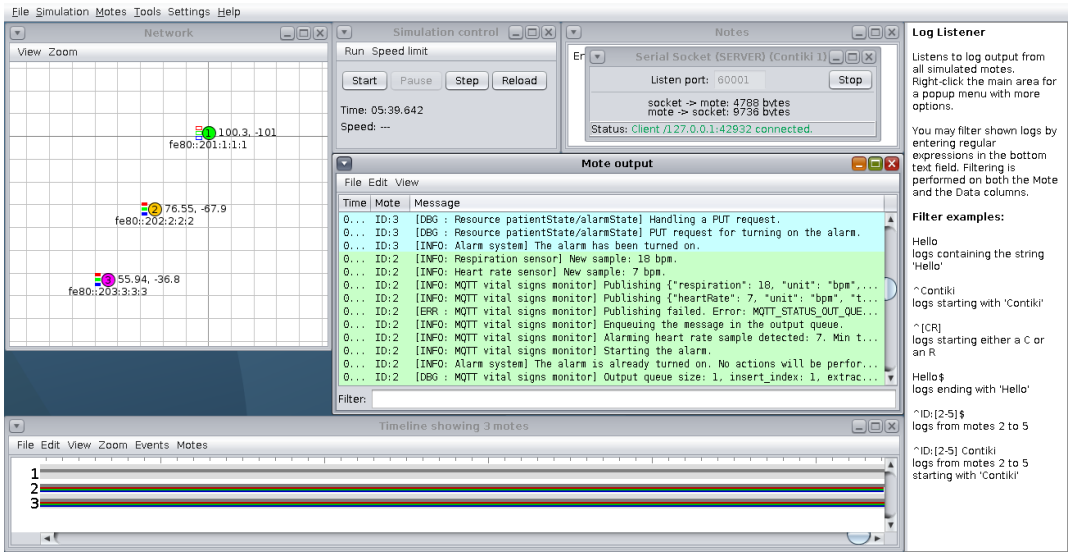

Figure 3: The Cooja simulated network, composed by one border router (node 1), one MQTT monitor (node 2) and one CoAP monitor (node 3).

The test is performed deploying all the components, i.e. Cooja, the MQTT broker, the collector and the database, on the same Ubuntu 18.04 machine. The IoT system is simulated by creating a single LLN in which a border router (node 1), an MQTT vital signs monitor (node 2) and a CoAP vital signs monitor (node 3) are instantiated as Cooja motes; the bridging of packets between the border router and Ubuntu is enabled by *tunslip6*, which creates an ad hoc virtual interface with IPv6 address fd00::1.

The components are configured so that:

- the broker listens for messages on all the interfaces, on port 1883;
- the collector instructs its MQTT module to contact the broker at localhost:1883 and its CoAP module to expose the resources on [fd00::1]:5683;
- the vital signs monitors contact the MQTT broker at [fd00::1]:1883 and the CoAP collector at [fd00::1]:5683;
- the database is reachable at localhost:3306;
- the patient health analysis runs every 120 seconds.

The test is performed as follows:

1. the simulation is started. When the connection to the network is completed, the MQTT monitor (node 2) and the CoAP monitor (node 3) receive the prefix fd00 from the border router, and assign to their interfaces the IPv6 addresses fd00::202:2:2:2 and fd00::203:3:3:3, respectively. The devices start waiting for the patient ID to be provided via the serial line;

2. the patient ID *"patient2"* is given to the MQTT monitor, while the patient ID *"patient3"* is given to the CoAP monitor. Both devices inform the collector and start the sampling activity of their sensors;

3. during the working period, the samples are correctly sent to the collector and saved in the database. If a life-threatening value is sampled or an early degradation of the patient's health is detected, the alarm system of the monitor is turned on, unless already active. The MQTT monitor, when multiple messages are available at the same time, saves the non-deliverable ones in its queue and retries the transmission later;

4. after an alarm has been started, the MQTT monitor is notified to stop it by pressing the mote's button for 5 seconds (pre-configured value). The monitor stops the alarm and informs the collector;

5. the CoAP monitor is reset by holding the mote's button for 10 seconds (pre-configured value). The monitor stops eventual on-going alarms, resets the patient ID and informs the collector;

6. a new patient ID *"patient4"* is given to the CoAP monitor. The monitor informs the collector and restarts its working activity.

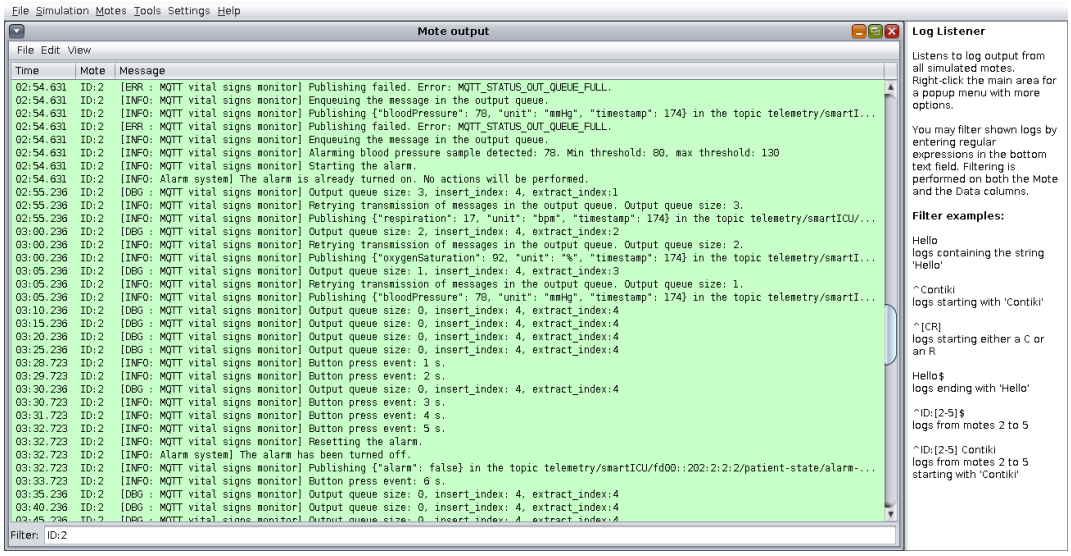

Figure 4: The output of the MQTT monitor when the alarm is stopped.

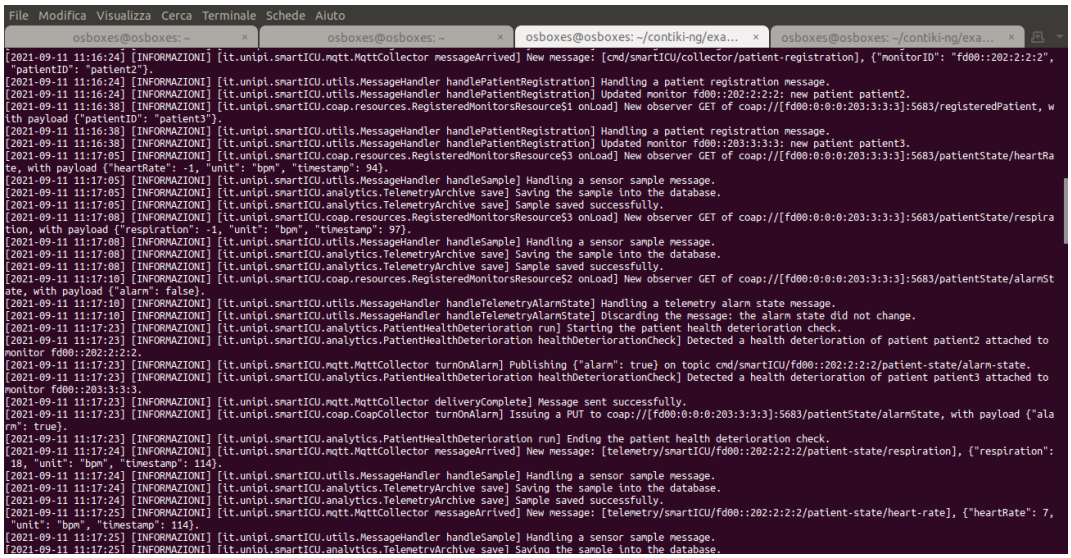

Figure 5: The output of the collector when the worsening conditions of a patient are detected.

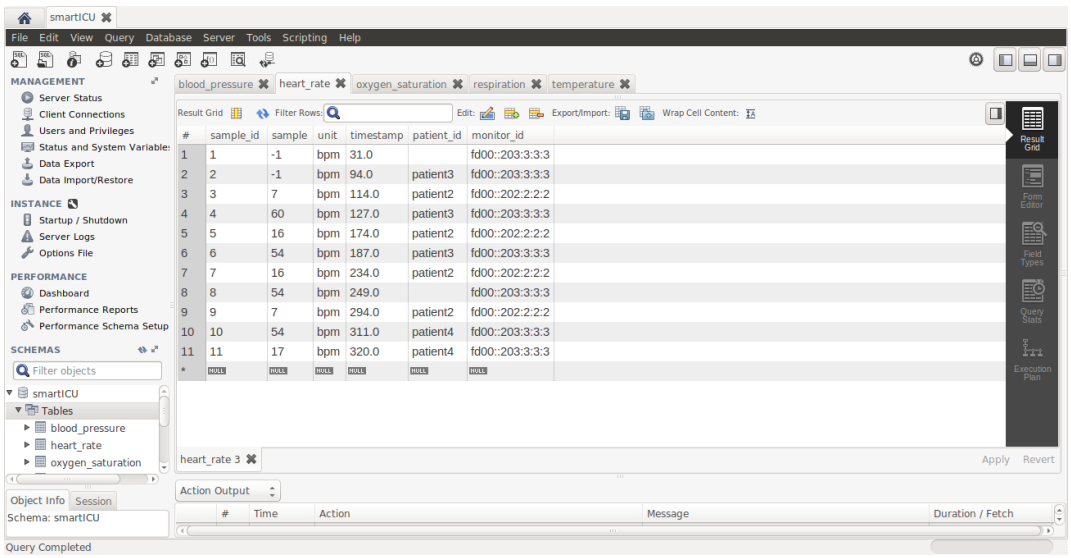

Figure 6: The heart rate samples saved in the database. The records with value -1 and empty patient ID are spurious ones, generated by the observe relation with the CoAP monitor: Californium automatically issues a GET request when the max-age timer associated to the resource expires.

3.2 Hybrid environment

The test is performed using both a Cooja simulated environment and two nRF52840 dongle devices hosted in a remote testbed. The setup is shown in Figure 7: a Cooja network with one CoAP monitor and one border router, the collector and the database are deployed in a local machine running Ubuntu 18.04; a network composed of one MQTT monitor and one border router, both flashed on real devices, and the MQTT broker are deployed on the remote testbed, running Ubuntu 18.04. The bridging of packets between the border routers and Ubuntu is enabled by *tunslip6*. The two machines interact via SSH port tunnelling, so that the remote port exposed by Mosquitto is mapped on the local port of the other machine.

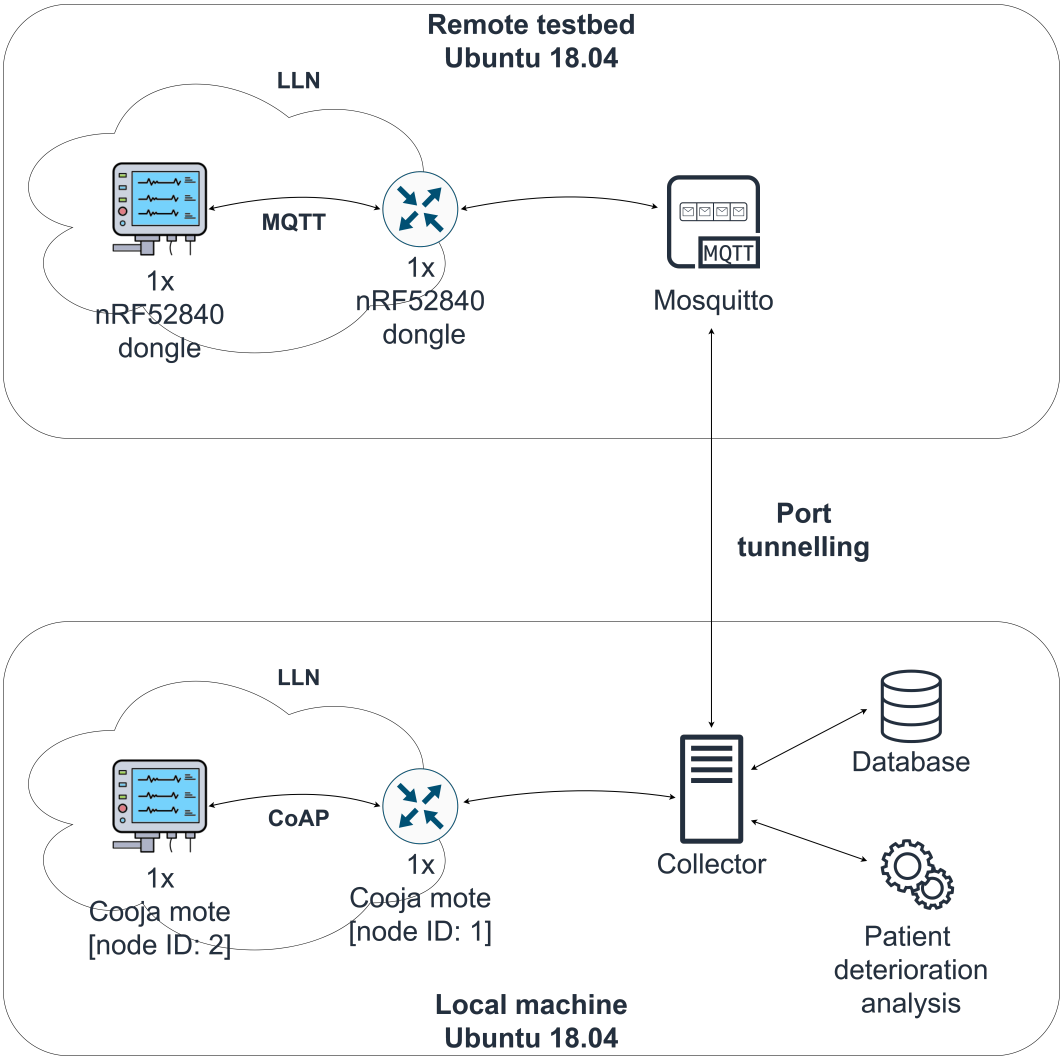

Figure 7: Hybrid environment used in the test.

The components of the local machine are configured so that:

- the collector instructs its MQTT module to contact the remote broker at localhost:1883 and its CoAP module to expose the resources on [fd00::1]:5683;

- the CoAP vital signs monitor attempts to reach the CoAP collector at [fd00::1]:5683;
- the database is reachable at localhost:3306;
- the patient health analysis runs every 120 seconds.

The components of the remote machine are configured so that:

- the broker listens for messages on all the interfaces, on port 1883;
- the MQTT vital signs monitor attempts to reach the MQTT broker at [fd00::1]:1883 and auto-generates the patient ID when it is required, so that the manual interaction via the serial line is not needed.

The test is performed as follows:

1. the remote devices are programmed and the simulation is started. When the connection to the network is completed, the MQTT monitor and the CoAP monitor (node 2 in Cooja) receive the prefix fd00 from the border router, and assign to their interfaces the derived IPv6 addresses (fd00::202:2:2:2 for the CoAP monitor);
2. The CoAP device starts waiting for the patient ID to be provided via the serial line, while the MQTT one auto-configures it and becomes fully operational. The patient ID *"patient2"* is given to the CoAP monitor. Both devices inform the collector and start the sampling activity of their sensors;
3. during the working period, the samples are correctly sent to the collector and saved in the database. If a life-threatening value is sampled or an early degradation of the patient's health is detected, the alarm system of the monitor is turned on, unless already active.

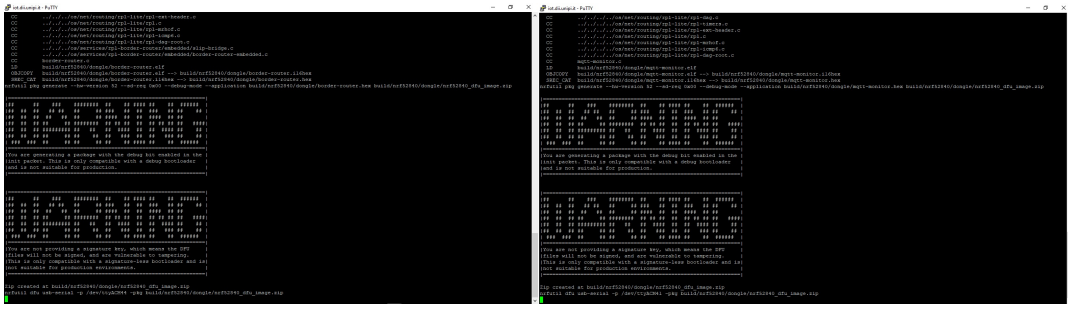

Figure 8: Compilation and flashing of the border router (left) and MQTT monitor (right) on the real devices.

```
File  Modifica  Visualizza  Cerca  Terminale  Schede  Aiuto
osboxes@osboxes: ~/Scrivania   osboxes@osboxes: ~   osboxes@osboxes: ~   osboxes@osboxes: ~/contiki...   osboxes@osboxes: ~/contiki...
osboxes@osboxes:~/contiki-ng/examples/smart-ICU/collector$ java -jar target/smartICU-collector-1.0-SNAPSHOT-jar-with-dependencies.jar -c confi
guration.json
[2021-09-11 13:58:53] [INFORMAZIONI] [it.unipi.smartICU.Collector start] Initializing the connection to the telemetry database.
[2021-09-11 13:58:54] [INFORMAZIONI] [it.unipi.smartICU.analytics.TelemetryArchive init] Connected to the telemetry database.
[2021-09-11 13:58:54] [INFORMAZIONI] [it.unipi.smartICU.Collector start] Instantiating the MQTT collector.
[2021-09-11 13:58:54] [INFORMAZIONI] [it.unipi.smartICU.mqtt.MqttCollector start] Starting the MQTT collector.
[2021-09-11 13:58:54] [INFORMAZIONI] [it.unipi.smartICU.mqtt.MqttCollector start] Attempting to connect to the broker tcp://localhost:1883.
[2021-09-11 13:58:55] [INFORMAZIONI] [it.unipi.smartICU.mqtt.MqttCollector start] Connected to the broker.
[2021-09-11 13:58:55] [INFORMAZIONI] [it.unipi.smartICU.Collector start] Instantiating the CoAP collector.
[2021-09-11 13:58:55] [INFORMAZIONI] [org.eclipse.californium.core.network.config.NetworkConfig load] loading properties from file /home/osbox
es/contiki-ng/examples/smart-ICU/collector/Californium.properties
[2021-09-11 13:58:55] [INFORMAZIONI] [it.unipi.smartICU.coap.CoapCollector start] Starting the CoAP collector.
[2021-09-11 13:58:55] [INFORMAZIONI] [org.eclipse.californium.core.CoapServer start] Starting server
[2021-09-11 13:58:55] [INFORMAZIONI] [org.eclipse.californium.core.network.CoapEndpoint start] Starting endpoint at /fd00:0:0:0:0:0:0:1:5683
[2021-09-11 13:58:55] [INFORMAZIONI] [it.unipi.smartICU.Collector start] Scheduling the execution of the patient health deterioration routine.

[2021-09-11 14:00:55] [INFORMAZIONI] [it.unipi.smartICU.analytics.PatientHealthDeterioration run] Starting the patient health deterioration ch
eck.
[2021-09-11 14:00:55] [INFORMAZIONI] [it.unipi.smartICU.analytics.PatientHealthDeterioration run] Ending the patient health deterioration chec
k.
[2021-09-11 14:01:29] [INFORMAZIONI] [it.unipi.smartICU.mqtt.MqttCollector messageArrived] New message: [cmd/smartICU/collector/monitor-regist
ration], {"monitorID": "fd00::f6ce:3609:93f6:201e", "registration": true}.
[2021-09-11 14:01:29] [INFORMAZIONI] [it.unipi.smartICU.utils.MessageHandler handleMonitorRegistration] Handling a monitor registration messag
e.
[2021-09-11 14:01:29] [INFORMAZIONI] [it.unipi.smartICU.utils.MessageHandler handleMonitorRegistration] Registered a new monitor with ID fd00:
:f6ce:3609:93f6:201e.
[2021-09-11 14:01:34] [INFORMAZIONI] [it.unipi.smartICU.mqtt.MqttCollector messageArrived] New message: [cmd/smartICU/collector/patient-regist
ration], {"monitorID": "fd00::f6ce:3609:93f6:201e", "patientID": "auto_3242"}.
[2021-09-11 14:01:34] [INFORMAZIONI] [it.unipi.smartICU.utils.MessageHandler handlePatientRegistration] Handling a patient registration messag
e.
[2021-09-11 14:01:34] [INFORMAZIONI] [it.unipi.smartICU.utils.MessageHandler handlePatientRegistration] Updated monitor fd00::f6ce:3609:93f6:2
01e: new patient auto_3242.
```

Figure 9: The output of the collector when the remote MQTT monitor sends its registration.

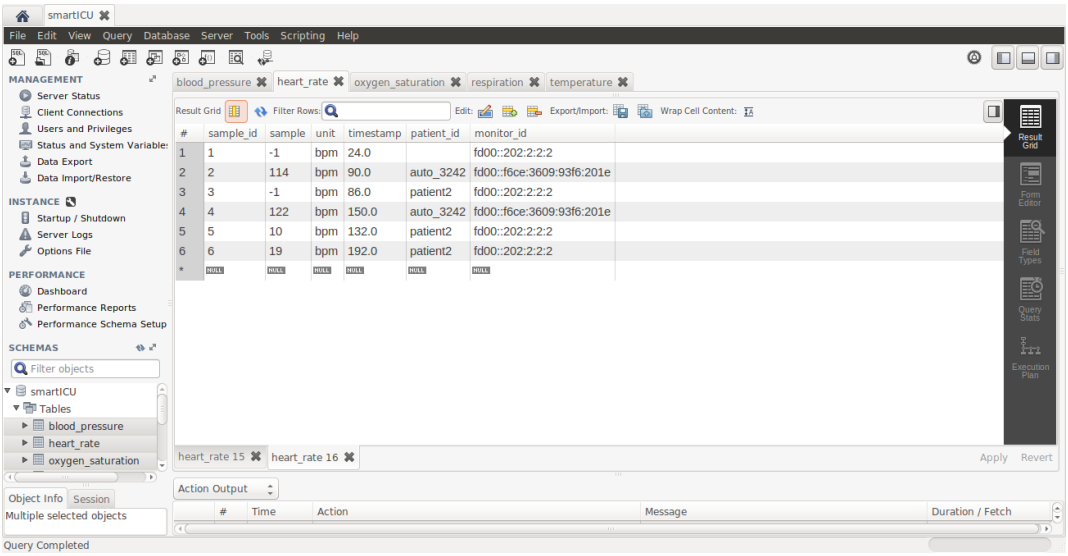

Figure 10: The heart rate samples saved in the database.