SmartPark

An easy bycicle park

Matteo De Francesco

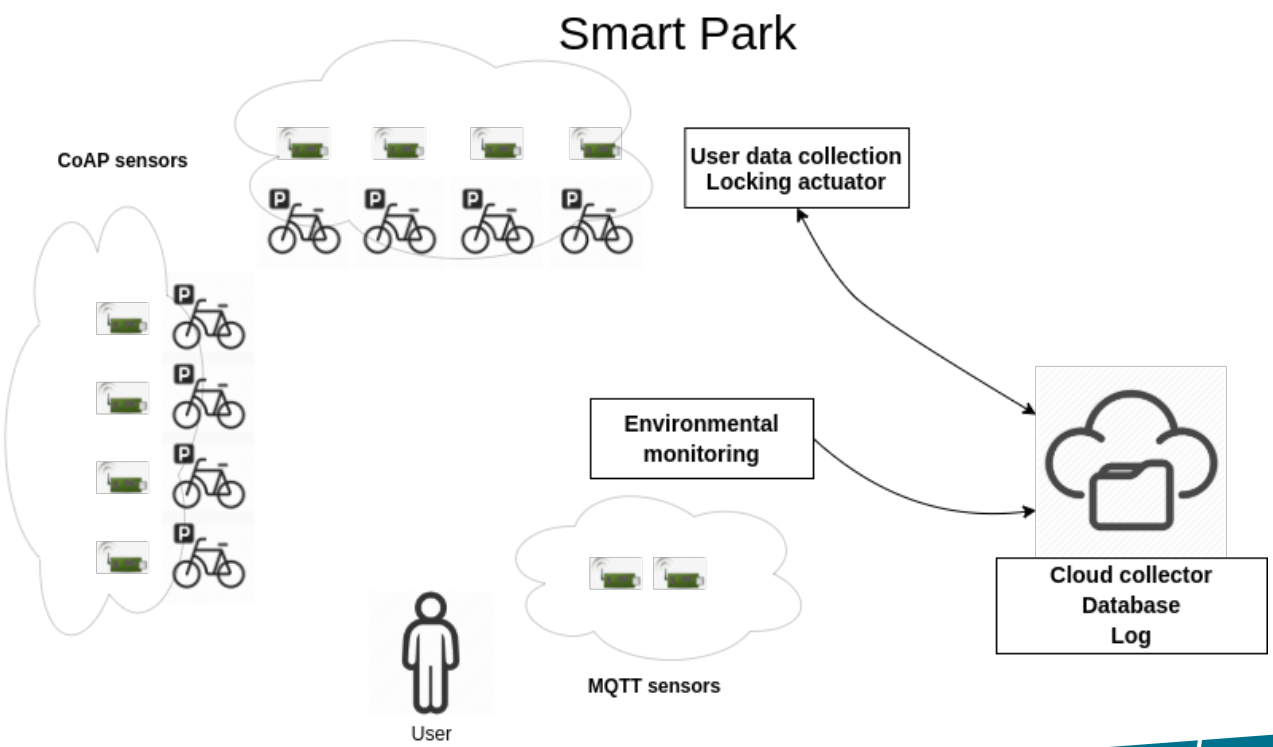
Computer Engineering

June 2, 2021

Università di Pisa

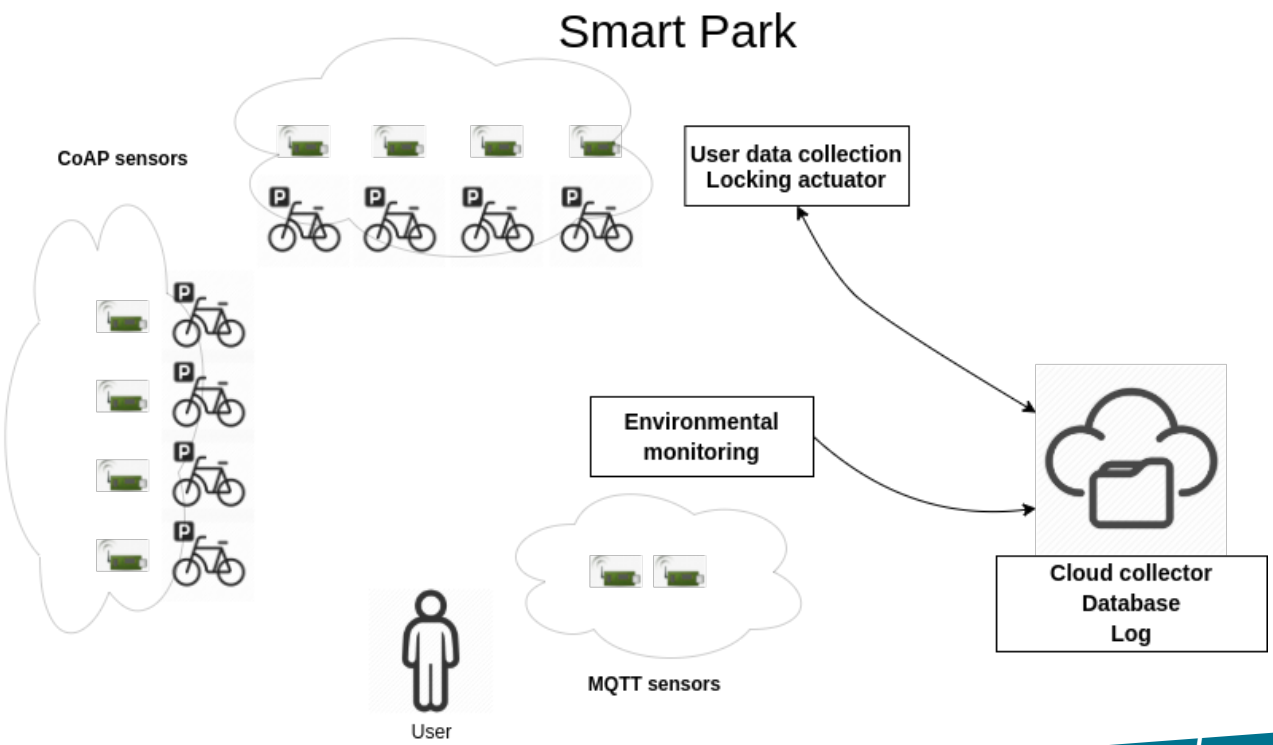
General overview

- SmartPark: a place where bycicle park gets easier



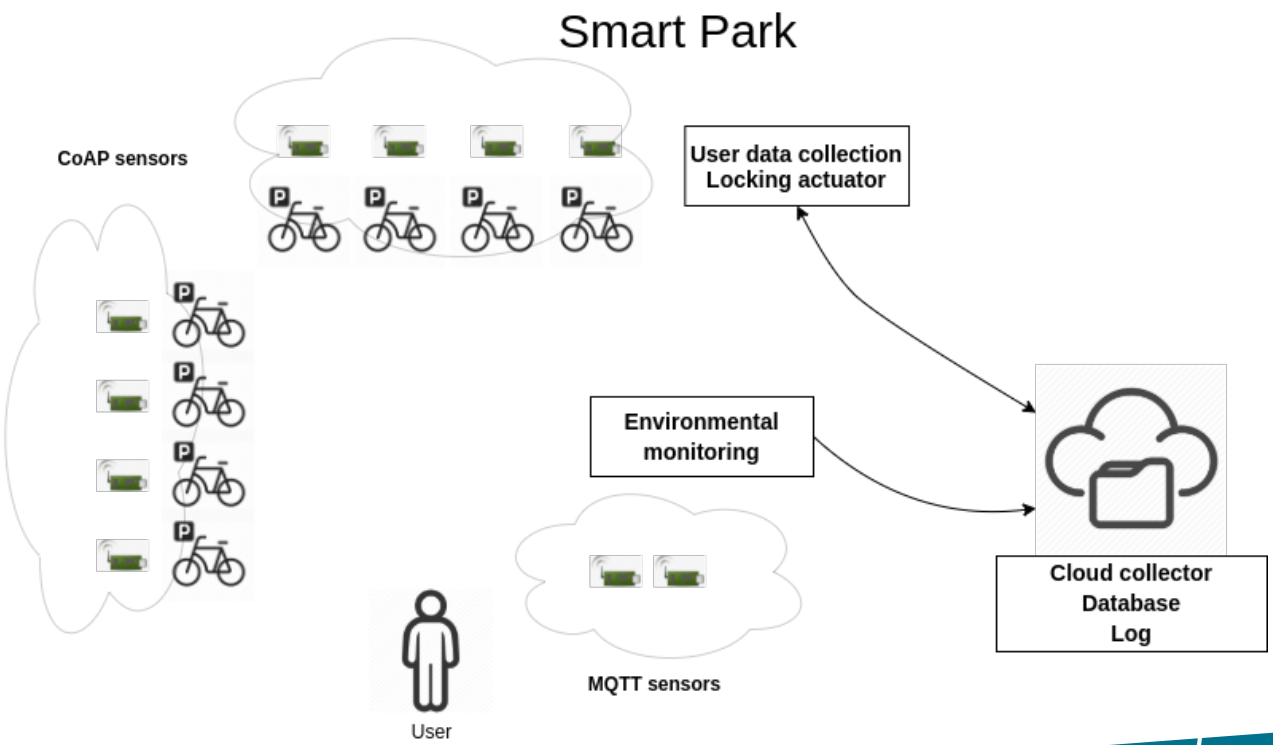
General overview

- SmartPark: a place where bycicle park gets easier
- Why need to bring a lock each time you go to work?



General overview

- SmartPark: a place where bycicle park gets easier
- Why need to bring a lock each time you go to work?
- With SmartPark this is not anymore a problem!



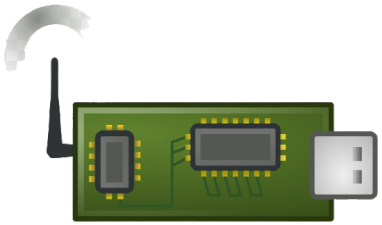
Main Functionalities

- Cloud collector with backend database and log show
- Cloud server send packets to the lock sensors to lock/unlock chain whenever a user is recognized

Main Functionalities

- Cloud collector with backend database and log show
- Cloud server send packets to the lock sensors to lock/unlock chain whenever a user is recognized
- Bycicle spots with sensors enabling user input name and password

Main Functionalities



- Cloud collector with backend database and log show
- Cloud server send packets to the lock sensors to lock/unlock chain whenever a user is recognized
- Bycicle spots with sensors enabling user input name and password
- Central sensors collecting information about the weather
- Both type of sensors collects data which is sent to the cloud collector and stored

Lock sensor

- To enable user authentication, a lock sensor is simulated through leds, buttons and variables

Lock sensor

- To enable user authentication, a lock sensor is simulated through leds, buttons and variables
- First of all, sensor connects through the network to the CoAPServer, with a registration message

Lock sensor

- To enable user authentication, a lock sensor is simulated through leds, buttons and variables
- First of all, sensor connects through the network to the CoAPServer, with a registration message
- Whenever a user approach the spot, it must trigger the button in order to activate the serial line
- User has 30 seconds to input its name followed by password, separated with a whitespace (to avoid overload on sensor, this last buffer size is capped at 10 bytes size)

Lock sensor

- To enable user authentication, a lock sensor is simulated through leds, buttons and variables
- First of all, sensor connects through the network to the CoAPServer, with a registration message
- Whenever a user approach the spot, it must trigger the button in order to activate the serial line
- User has 30 seconds to input its name followed by password, separated with a whitespace (to avoid overload on sensor, this last buffer size is capped at 10 bytes size)
- The input credentials are sent to the server which check the user identity and POST a message to the sensor, enabling the lock or disabling it

Lock sensor

- To enable user authentication, a lock sensor is simulated through leds, buttons and variables
- First of all, sensor connects through the network to the CoAPServer, with a registration message
- Whenever a user approach the spot, it must trigger the button in order to activate the serial line
- User has 30 seconds to input its name followed by password, separated with a whitespace (to avoid overload on sensor, this last buffer size is capped at 10 bytes size)
- The input credentials are sent to the server which check the user identity and POST a message to the sensor, enabling the lock or disabling it
- Upon receipt of the answer, the sensor enable/disable lock with respectively green/red leds

Environmental sensor

- The environmental sensor instead are implemented through the MQTT approach

Environmental sensor

- The environmental sensor instead are implemented through the MQTT approach
- Each of them will follow a registration process through the MQTT broker and a next phase with data reporting
- The status of the MQTT connection is checked very frequently

Environmental sensor

- The environmental sensor instead are implemented through the MQTT approach
- Each of them will follow a registration process through the MQTT broker and a next phase with data reporting
- The status of the MQTT connection is checked very frequently
- Instead the payload with information about temperature, umidity and actual weather is sent through another PROCESS_THREAD every 30 seconds

Collector

- The collector is responsible of accepting connection of CoAP sensors and receiving updates from MQTT broker.
- CoAPServer and MQTT client are executed on different threads

Collector

- The collector is responsible of accepting connection of CoAP sensors and receiving updates from MQTT broker.
- CoAPServer and MQTT client are executed on different threads
- Whenever a new coap sensor connects to the network, a successful registration message is delivered and an observer is started to receive updates on the lock status

Collector

- The collector is responsible of accepting connection of CoAP sensors and receiving updates from MQTT broker.
- CoAPServer and MQTT client are executed on different threads
- Whenever a new coap sensor connects to the network, a successful registration message is delivered and an observer is started to receive updates on the lock status
- The MQTT client instead will register on the MQTT broker for updates about the environmental values reported before

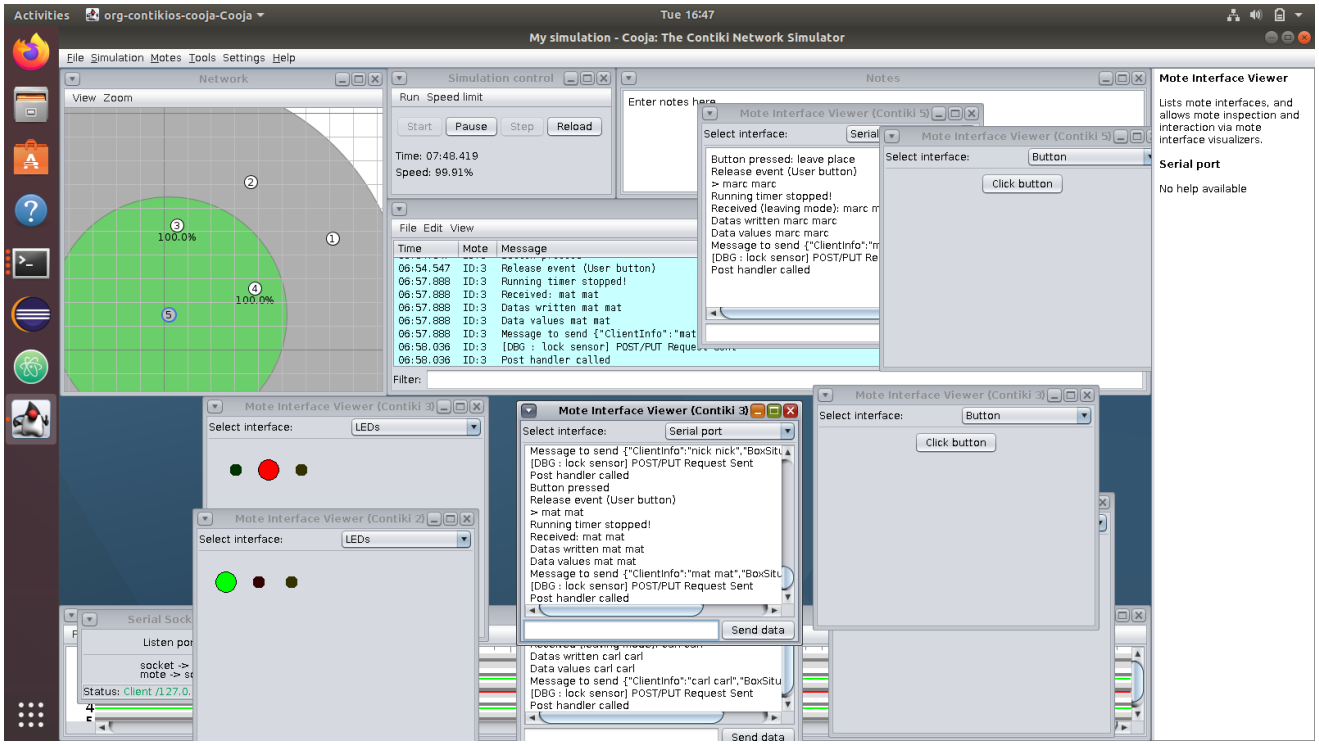
Collector

- The collector is responsible of accepting connection of CoAP sensors and receiving updates from MQTT broker.
- CoAPServer and MQTT client are executed on different threads
- Whenever a new coap sensor connects to the network, a successful registration message is delivered and an observer is started to receive updates on the lock status
- The MQTT client instead will register on the MQTT broker for updates about the environmental values reported before
- Each time an update on the lock status is POST by the coap server or an MQTT update is received, a log of the respective database table is showed

Collector

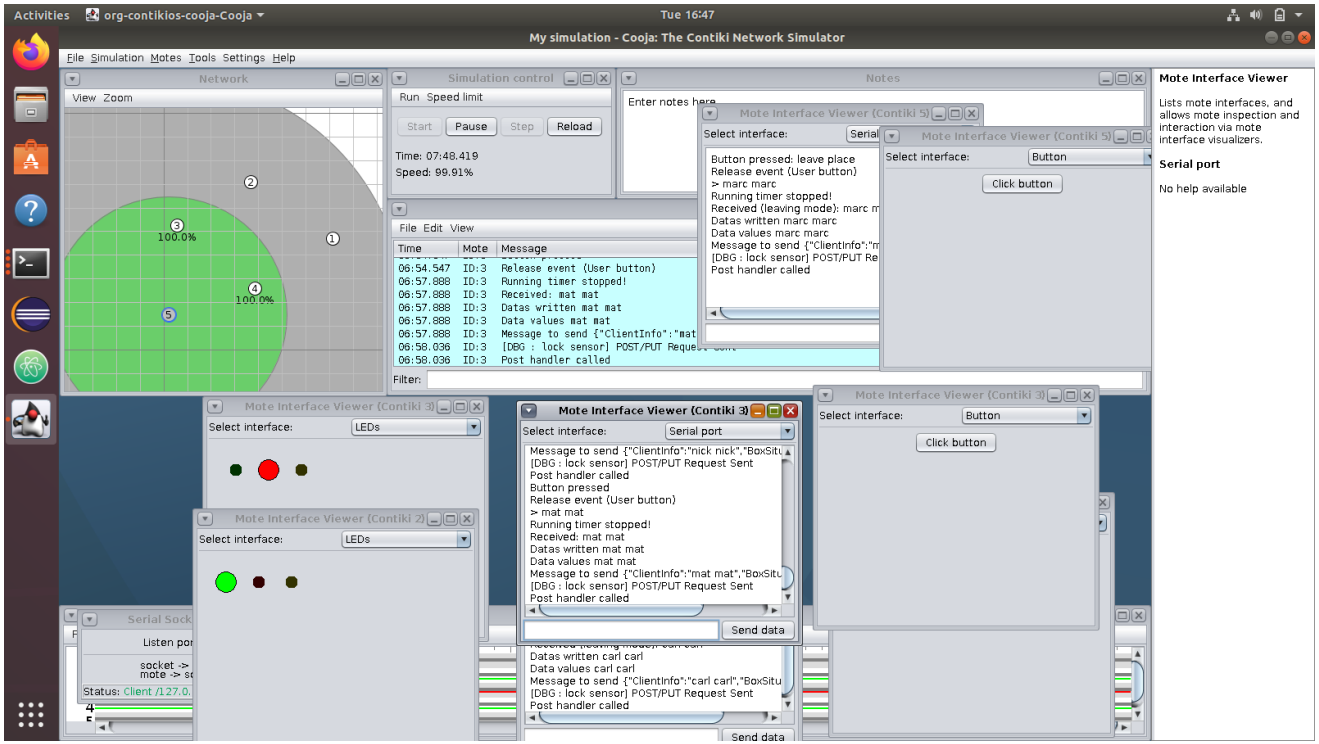
- The collector is responsible of accepting connection of CoAP sensors and receiving updates from MQTT broker.
- CoAPServer and MQTT client are executed on different threads
- Whenever a new coap sensor connects to the network, a successful registration message is delivered and an observer is started to receive updates on the lock status
- The MQTT client instead will register on the MQTT broker for updates about the environmental values reported before
- Each time an update on the lock status is POST by the coap server or an MQTT update is received, a log of the respective database table is showed
- Data is stored inside *datacollector* database, with 2 tables, *coapsensors* and *mqttsensors*

Simulation



- Node 1 is the border router
- Node 3 is locked (red leds)
- Node 5 is available (green leds)

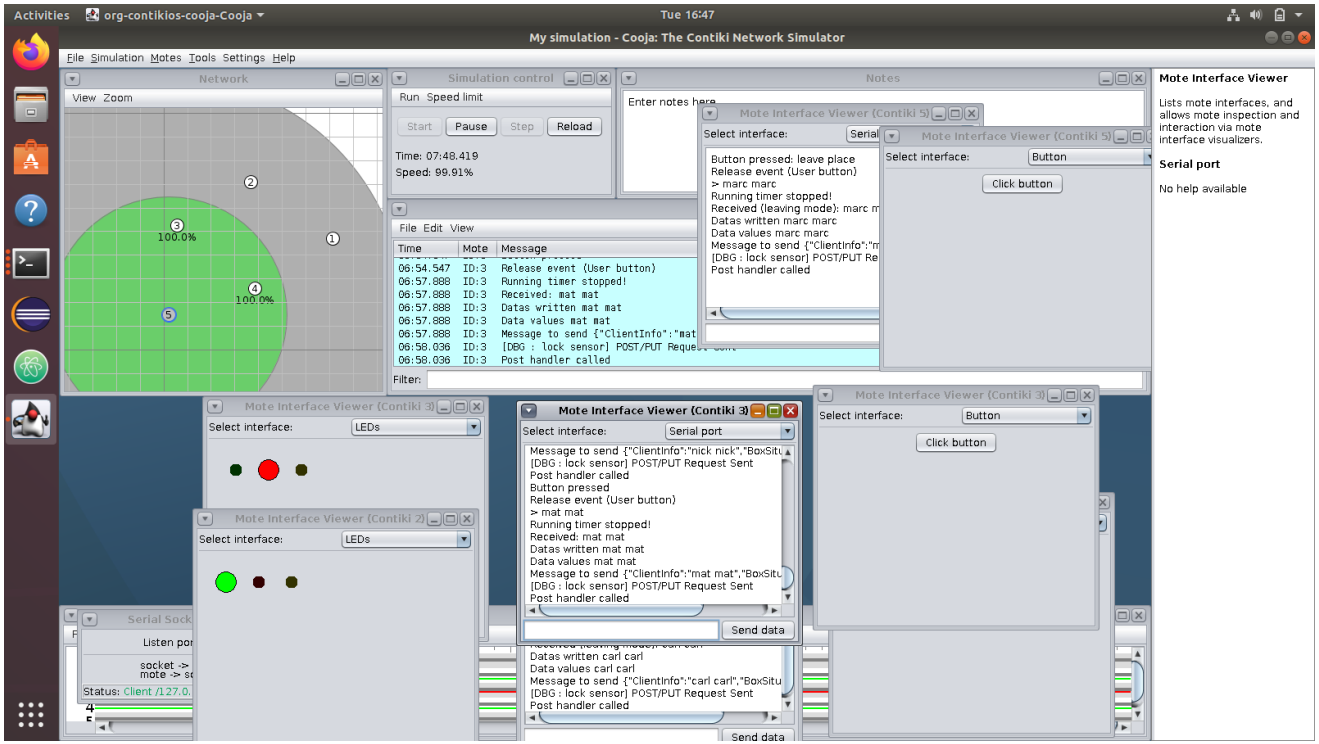
Simulation



- Node 1 is the border router
- Node 3 is locked (red leds)
- Node 5 is available (green leds)

- The border router is the first deployed. After starting the simulation at low speed, the tunslip6 is launched

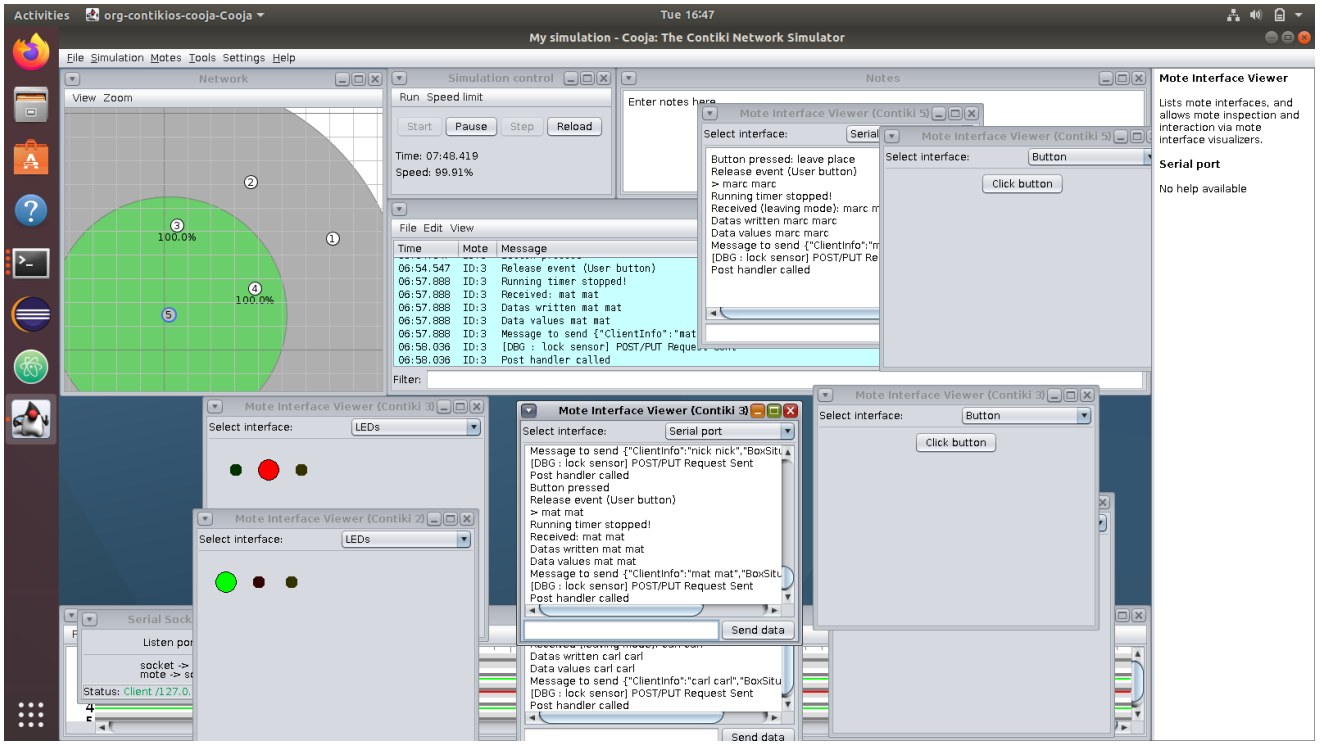
Simulation



- Node 1 is the border router
- Node 3 is locked (red leds)
- Node 5 is available (green leds)

- The border router is the first deployed. After starting the simulation at low speed, the tunslip6 is launched
- From that point on, both the coap sensors and the python coap server are launched to exchange coap messages

Simulation



- Node 1 is the border router
- Node 3 is locked (red leds)
- Node 5 is available (green leds)

- The border router is the first deployed. After starting the simulation at low speed, the tunslip6 is launched
- From that point on, both the coap sensors and the python coap server are launched to exchange coap messages
- In the meantime on the testbed..

Simulation - Testbed

- On the testbed, a border router is deployed on port 67 and tunsplip6 is launched

Simulation - Testbed

- On the testbed, a border router is deployed on port 67 and tunsplip6 is launched
- Then mosquitto is launched from the configuration file, adding the line
  `listener 1883 localhost`
  to allow port forwarding

Simulation - Testbed

- On the testbed, a border router is deployed on port 67 and tunsplip6 is launched
- Then mosquitto is launched from the configuration file, adding the line
  `listener 1883 localhost`
  to allow port forwarding
- Finally, the MQTT sensor is deployed on port 15

Simulation - Testbed

- On the testbed, a border router is deployed on port 67 and tunsplip6 is launched
- Then mosquitto is launched from the configuration file, adding the line `listener 1883 localhost` to allow port forwarding
- Finally, the MQTT sensor is deployed on port 15

```
user@studente15:~/contiki-ng/Project/sensors/mqtt_sensor$ make TARGET=nrf52840 BOARD=dongle login PORT=/dev/ttyACM15
rlwrap ../../../tools/serial-io/serialdump -b115200 /dev/ttyACM15
connecting to /dev/ttyACM15 [OK]
Connecting!
Application has a MQTT connection
Nothing
Nothing
Nothing
Nothing
Message: {"temp":0,"umidity":0,"weather":0}
Nothing
Nothing
Nothing
Nothing
Nothing
Nothing
Nothing
Nothing
Nothing
```

Figure 1: Testbed MQTT node

- After MQTT connection is established, one PROCESS_THREAD checks the connection periodically

Simulation - Testbed

- On the testbed, a border router is deployed on port 67 and tunsplip6 is launched
- Then mosquitto is launched from the configuration file, adding the line `listener 1883 localhost` to allow port forwarding
- Finally, the MQTT sensor is deployed on port 15

```
user@studente15:~/contiki-ng/Project/sensors/mqtt_sensor$ make TARGET=nrf52840 BOARD=dongle login PORT=/dev/ttyACM15
rlwrap ../../../tools/serial-io/serialdump -b115200 /dev/ttyACM15
connecting to /dev/ttyACM15 [OK]
Connecting!
Application has a MQTT connection
Nothing
Nothing
Nothing
Nothing
Message: {"temp":0,"umidity":0,"weather":0}
Nothing
Nothing
Nothing
Nothing
Nothing
Nothing
Nothing
Nothing
```

Figure 1: Testbed MQTT node

- After MQTT connection is established, one PROCESS_THREAD checks the connection periodically

- Another PROCESS_THREAD, every 30 seconds, publish a JSON payload with the environmental information captured

Simulation - Collector

```
osboxes@osboxes:~/contiki-ng/Project/collector$ python3 server_test.py
Initializing server and MQTT client thread
Instantiating!
Mqtt client starting
Connected with result code 0
GET server, received message:

{"NodeType":"Both","MoteResource":"client","NodeID":2}
Instantiating!
GET server, received message:

{"NodeType":"Both","MoteResource":"client","NodeID":4}
Instantiating!
Callback called, resource arrived
{"ClientInfo":"","BoxSituation":"F"}
Credentials are empty: discard message...
Callback called, resource arrived
{"ClientInfo":"","BoxSituation":"F"}
Credentials are empty: discard message...
Callback called, resource arrived
{"ClientInfo":"","BoxSituation":"F"}
Credentials are empty: discard message...
```

Figure 2: Node registration

- Each coap sensor register through the coap server

Simulation - Collector

```
osboxes@osboxes:~/contiki-ng/Project/collector$ python3 server_test.py
Initializing server and MQTT client thread
Instantiating!
Mqtt client starting
Connected with result code 0
GET server, received message:

{"NodeType":"Both","MoteResource":"client","NodeID":2}
Instantiating!
GET server, received message:

{"NodeType":"Both","MoteResource":"client","NodeID":4}
Instantiating!
Callback called, resource arrived
{"ClientInfo":"","BoxSituation":"F"}
Credentials are empty: discard message...
Callback called, resource arrived
{"ClientInfo":"","BoxSituation":"F"}
Credentials are empty: discard message...
Callback called, resource arrived
{"ClientInfo":"","BoxSituation":"F"}
Credentials are empty: discard message...
```

Figure 2: Node registration

```
weatherinfo b'{"temp":11,"umidity":0,"weather":2}'
+------+---------------------+-------------+-----------+-----------+
|  id  |      timestamp      | temperature | umidity   | weather   |
+======+=====================+=============+===========+===========+
|    1 | 2021-06-01 16:40:09 |          30 |        89 | RAINY     |
+------+---------------------+-------------+-----------+-----------+
|    2 | 2021-06-01 16:40:39 |          28 |        65 | RAINY     |
+------+---------------------+-------------+-----------+-----------+
|    3 | 2021-06-01 16:41:09 |           6 |        54 | CLOUDY    |
+------+---------------------+-------------+-----------+-----------+
|    4 | 2021-06-01 16:41:39 |          23 |        51 | CLOUDY    |
+------+---------------------+-------------+-----------+-----------+
|    5 | 2021-06-01 16:42:09 |           9 |         8 | SUNNY     |
+------+---------------------+-------------+-----------+-----------+
|    6 | 2021-06-01 16:42:39 |          25 |        80 | SUNNY     |
+------+---------------------+-------------+-----------+-----------+
|    7 | 2021-06-01 16:43:09 |           0 |        42 | RAINY     |
+------+---------------------+-------------+-----------+-----------+
|    8 | 2021-06-01 16:43:39 |          27 |        42 | SUNNY     |
+------+---------------------+-------------+-----------+-----------+
|    9 | 2021-06-01 16:44:09 |          11 |         0 | RAINY     |
+------+---------------------+-------------+-----------+-----------+
```

```
Callback called, resource arrived
{"ClientInfo":"carl carl","BoxSituation":"B"}
Client credentials are:
['carl', 'carl']
User recognized!
B
<pymysql.connections.Connection object at 0x7f5d98710278>
<pymysql.connections.Connection object at 0x7f5d98710278>
+------+---------------------+--------+------------+
|  id  |      timestamp      | name   | entering   |
+======+=====================+========+============+
|    1 | 2021-06-01 16:40:40 | mat    |          1 |
+------+---------------------+--------+------------+
|    2 | 2021-06-01 16:41:08 | mat    |          0 |
+------+---------------------+--------+------------+
|    3 | 2021-06-01 16:42:06 | carl   |          1 |
+------+---------------------+--------+------------+
|    4 | 2021-06-01 16:42:41 | carl   |          0 |
+------+---------------------+--------+------------+
```

Figure 3: Data log

- Each coap sensor register through the coap server
- Every time an update is published on the MQTT broker or a user acts on the lock sensor, the database is updated and the data log is showed

Implementation choices

- I decided to implement the collector using Python. I exploited the CoAPThon library together with the paho-mqtt. In addition, I used the PyMySQL library to access, insert and query the database, and another module tabulate to show the data log from the database in a nice way
- Regarding the data encoding, I choose JSON, being a lightweight data encoding language
- Very suitable in my case, since the data transmitted is not much and on Python side I have an easy way to manage it
- Only the POST payload from the server to the lock sensors is passed as a POST variable, having only a flag 0/1 meaning unlock/lock of the chain

Reproduce

To reproduce the simulation on remote testbed, login into the remote testbed with 3 different terminals, one of them forwarding the local 1883 port:

- Remember to add `listener 1883 localhost` to *mosquitto.conf* and the `define IEEE802154_CONF_PANID 0x0015` to both router and mqtt sensor

Terminal #1

- `cd contiki-ng/Project/sensors/rpl-border-router`
- `make TARGET=nrf52840 BOARD=dongle border-router.dfu-upload PORT=/dev/ttyACM67`
- `make TARGET=nrf52840 BOARD=dongle connect-router PORT=/dev/ttyACM67`

Terminal #2

- `sudo mosquitto -c /etc/mosquitto/mosquitto.conf`

Terminal #3

- `cd contiki-ng/Project/sensors/mqtt-sensor`
- `make TARGET=nrf52840 BOARD=dongle mqtt-client.dfu-upload PORT=/dev/ttyACM15`
- `make TARGET=nrf52840 BOARD=dongle login PORT=/dev/ttyACM15`