



IoT Project

Smart home automation:

Wireless Sensor network to handle Smart Air Conditioner and Smart Illumination

Nocerino Raffaele
a.a 2019/2020

Main Goal

- ▶ Our proposal was to build a WSN that allowed us to remotely control the Air Conditioning system and smart light system installed in our house
- ▶ In order to receive instantaneous update whenever we want and to perform some other operations on that systems



To do so we had to deploy and create 3 important elements:

- ▶ **WSN** → composed by some sensors and actuators
- ▶ **Cloud Application** → always on application that does all the important computations
- ▶ **Client application** → application that will be used by the user to perform all the needed operations

Wireless Sensor Network

- ▶ The first element that we had to deploy was the WSN

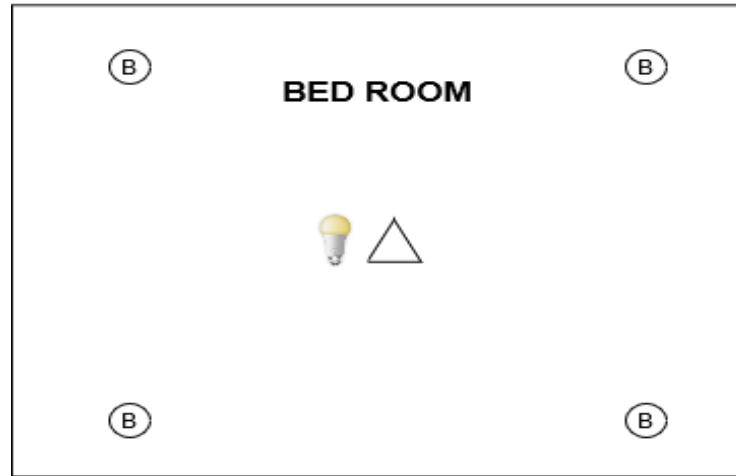
Composed by 4 types of different nodes:

- ▶ Temperature sensors
- ▶ Brightness sensors
- ▶ AirConditioner actuator
- ▶ Smart light actuator

Our main idea was to separate the two smart systems inside 2 different rooms:

- ▶ Living room: temperature sensors and Air Conditioner actuator
- ▶ Bed room : brightness sensors and Smart light actuator



Example of WSN architecture





Legend:

(B) Brightness sensor

(T) Temperature sensor

  Smart Light actuator

  Smart AirC remote control



Some clarifications

- ▶ To be more precise we didn't had the possibility to deploy a real WSN using real temperatures/light sensors and actuators

For this reason we used Cooja in our project:

- ▶ Using Cooja we had the possibility to create a virtual WSN composed by sensors whose behaviour is defined by the developer

Temperature sensors

- ▶ Due to the fact that Cooja doesn't provide/simulate the behaviour of temperature sensors, we had to implement inside the node's code also the sensing logic of the sensor:

Periodically the node will execute a function (COAP trigger event) that using some variables (more clear in the next) is able to know if Air Conditioning system of the room is ON or OFF:

- ▶ If the AIR COND is ON with a certain Temp → it will generate 5 «sensed» temperature value that will stay around Temp with a certain tolerance ($\pm 0.5^{\circ}\text{C}$) and computing after their average value
- ▶ If the AIR COND is OFF → the node will generate 5 «sensed» random temperatures within a Range interval of (15-30) $^{\circ}\text{C}$ and computing after their mean value

The mean value of 5 «sensed» random temperatures is done in order to try to reduce significant differences in temperatures generated by different nodes, obtaining more real data

Temperature actuators

- ▶ In our original system the temperature actuator would have been a remote control for air conditioning .
- ▶ Since we don't have the possibility in cooja to simulate the presence of the air conditioning system to send command to it, we have created an actuator node which used a led and mote output to prove that the «simulated» actuation was done correctly.
- ▶ If Air C. is powered ON → led GREEN powered and the mote outputs AIR ON
- ▶ If Air C is powered OFF → led GREEN powered OFF and mote outputs AIR OFF
- ▶ The temperature actuator behaves as a COAP server which exposes a «temp_act» resource used to receive POST requests.
- ▶ When the cloud application deems it necessary to send a command to the air conditioner, it sends a POST request to the actuator using this format:

```
coap://[node_address]/temp_act?temperature=x -e mode=on|off
```

The temperature variable will be used by the cloud application to specify the temperature on which the air conditioner has to be set (if mode=ON)

The mode variable is instead used to specify if the air cond has to be powered ON or OFF

How we do implements consistency on data sensed by the sensors?

- ▶ As we have explained in the previous slide, there's not a real actuation done on the external environment by the remote air cond control
- ▶ So we need some mechanisms to notify the sensors that an actuation was done, in order to allow them to generate accordingly consistent «sensed» data
- ▶ To do so the we have implemented, inside the resource exposed by the temperature node, a «dummy» POST handler which is used by the node just to set some global variables used periodically by the trigger handler to understand if the Air Cond is ON or OFF
- ▶ So when the cloud Application requests an actuation, sending a POST request to the Air cond remote control(actuator), sends also for each temperature sensor in the room a POST request using same syntax defined for the actuator, notifying them if the Air Cond is ON/OFF and eventually the temperature set

```
coap://[node_address]/temp?temperature=x -e mode=on|off
```


Brightness sensors

- ▶ Due to the fact that Cooja doesn't provide/simulate the behaviour of brightness sensors, we had to implement inside the node's code also the sensing logic of the sensor

Periodically the node will execute a function (COAP trigger event) that using some variables is able to know if the smart light of the room is ON or OFF:

- ▶ If the Smart light is ON with a certain brightness → the sensors will generate a brightness percentage in the room which stays within a C.I of the brightness percentage set for the smart light
- ▶ If the Smart light is OFF → the sensor will generate a brightness percentage in the room which will depend on the natural light coming from outside.

To do so the sensor use the current hour to understand in which part of the day we are, generating corresponding percentage of brightness

For example:

If **current hour > 20:00 and < 08:00** the percentage of brightness will stay inside (0-9)%

If **current hour > 08:00 and < 15:00** the bright. Perc. Will stay inside (70-100)%

Brightness actuators

- ▶ In our original system the light actuator would have been a smart light which can be powered ON with different values of brightness percentage
- ▶ Since we don't have the possibility in cooja to simulate the behaviour of a smart light with different brightness percentage we have used the 3 leds of the mote and its output to emulate the actuations and to verify is everything was done correctly

Everytime the light actuator (COAP server) receives a POST request on its exposed resource:

If the command is Light ON with a certaing brightness, the mote will:

- ▶ Set the GREEN light to ON and outputs High Intensity light if the brightness requested stays between (70-100)%
- ▶ Set the YELLOW light to ON and outputs Medium Intensity light if the brightness requested stays between (40-69)%
- ▶ Set the RED light to ON and outputs Low Intensity light if the brightness requested stays between (1-39)%

If the command is Light OFF will power OFF all the lights

```
coap://[node_address]/light_act?brightness=x -e mode=on|off
```

How we do implements consistency on data sensed by the sensors?

- ▶ As we have explained in the previous slide, there's not a real actuation done on the external environment by the smart light
- ▶ So we need some mechanisms to notify the sensors that an actuation was done, in order to allow them to generate accordingly consistent «sensed» data
- ▶ To do so the we have implemented, inside the resource exposed by the light node, a «dummy» POST handler which is used by the node just to set some global variables used periodically by the trigger handler to understand if the light is ON or OFF
- ▶ So when the cloud Application requests an actuation, sending a POST request to the smart light, sends also for each light sensor in the room a POST request using same syntax defined for the corresponding actuator, notifying them if the smart light is ON/OFF and eventually the brightness percentage set

```
coap://[node_address]/light?brightness=x -e mode=on|off
```

What are the features offered by the Cloud Application?

- ▶ After the registration phase of the nodes , the Cloud Application starts to receive continuous updated data by the sensors that are stored in memory
- ▶ More in details, everytime a new update is received from a sensor, the app will execute a specific function (called CoapHandler) that will be used to manage the received data and to do some checks depending on the user's requests.

Temperature features

The features offered by the Smart Air Conditioning system are:

- ▶ **ObserveTemperature**: typing this command the user sends a request to the cloud app to see all the sensed temperature information of a room of the current day (from the newest to the oldest)
- ▶ **AirCondON**: the user sends a request to the cloud application to set the AirCond ON specifying also the desired temperature
- ▶ **AirCondOFF**: the user sends a request to the app to power OFF the AirCond
- ▶ **AutomaticTempControlON**: the user requests to the cloud app to enable the automatic temperature control with threshold. That request will enable a control inside the handler, used by the cloud app to recognize if the avg sensed temperature in the room exceeds the specified threshold and if this happen the cloud application will activate the Air Cond with the user's requested temp
- ▶ **AutomaticTempControlOFF**: the user disable the automatic temperature control with threshold.
- ▶ **ProgramAirCondON**: the user will request to the cloud appl to activate the air cond on a specific hour with a specific temperature
- ▶ **ProgramAirCondOFF**: the user will remove the last program inserted for the Air Cond

Light Features

The features offered by the Smart light system are:

- ▶ **ObserveBrightness**: typing this command the user sends a request to the cloud app to see all the sensed brightness information of a room of the current day (from the newest to the oldest)
- ▶ **SetLightON**: the user requests to the cloud app to activate the smart light with the maximum brightness percentage(100%)
- ▶ **SetLightONwithBrightness**: the user requests to the cloud app to activate the smart light with the specified brightness percentage
- ▶ **SetLightOFF**: the user requests to the cloud app to power OFF the smart light
- ▶ **progressiveLightON**: the user sends a request to the cloud app to enable the progressiveLight mode specifying the starting hour and the maximum desired brightness percentage. That will enable a control inside the handler of the light sensor. If that check is satisfied, the cloud app will increase progressively the brightness percentage of the room until the desired brightness is reached
- ▶ **progressiveLightOFF**: the user disables the last request of progressiveLight mode