Internet of Things (IoT)

Lab 2: Z-WAVE

3rd October 2019

1. Introduction

The goal of this Lab is to set up and manage a set of connected sensors with a Z-Wave network. Sensors are used to measure several data such as temperature, humidity, brightness, physical presence and energy consumption. You will also use smart bulbs equipped with light dimmers which you can command to adjust the brightness of the bulb.

Concretely, the goal of this lab is to:

- Install an Z-Wave network composed of a set of devices which includes multi-function sensors and light dimmers. The multi-function sensors measure several data such as temperature, humidity, brightness and presence. Dimmers are a sort of switches that enable the control of bulb's brightness level. Each device (sensor or dimmer) is connected (paired) to a Z-wave controller (see section 2).
- Complete the code of a REST server which collect data received from the devices and command the light dimmers. This REST server will be deployed on a Raspberry (Figure 1).
- Test the REST server with a REST client. The code of this REST client is available in the Smart building project: git clone https://githepia.hesge.ch/lsds/Smart-building.git (post_client.py)

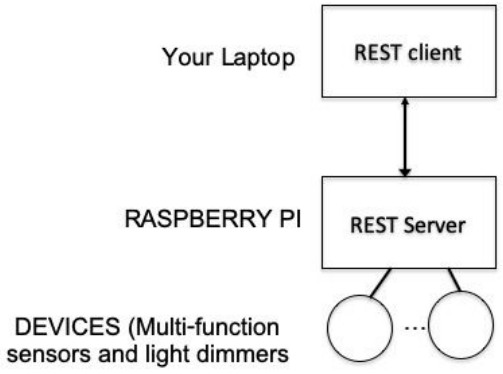

Figure 1 : Three layers :devices layer, Raspberry layer and application layer

2. IoT infrastructure

The lab uses a Z-wave protocol to setup the IoT infrastructure described in Figure 1. A Z-Wave network may include up to 232 nodes, and consists of two sets of nodes: controllers and slave devices. In our case, the controller (Z-stick controller) is a USB dongle with a push

button (Figure 2). The controller controls the Z-wave network. It can include, exclude and/or monitor nodes. It collects data from nodes.

Figure 2: "Z-Stick" Z-Wave controller

2.1. How to connect a device to a Z-wave network?

2.1.1 connecting sensors

The inclusion mode enables nodes to join the Z-Wave network of the controller node. By a simple click on the push button, the controller switches to "inclusion" mode and the light, surrounding the button, starts flashing. During inclusion, the controller must not be plugged to the raspberry.

Once the controller switches to "inclusion" mode, simply click on the button of the node you want to add to the network (Figure 3).

After adding a sensor, the flashing restarts (in case we want to add another sensor). To exit "inclusion" mode, click a second time on the button of the controller (in this case, the flashing stops).

Figure 3 : Multi-sensors Z-Wave.

2.1.2 connecting dimmers

Start by plugging the bulb and putting the manual switch on, you will have the light on. The next step is to turn the light off manually (using the switch).

Now put the controller into "inclusion" mode by simply clicking on its button. Then put the light on manually using the switch. If the inclusion is successful, the bulb will flash twice. If not, repeat the previous procedure.

After adding the dimmer, the controller's flashing restarts (in case we want to add another node). To exit "inclusion" mode, click a second time on the button of the controller (in this case, the flashing stops).

2.2. How to disconnect a device from a Z-wave network?

2.2.1 disconnecting sensors

The "exclusion" mode enables nodes to leave the Z-Wave network of the controller node. To move to "Exclusion" mode, click on the button continuously for 3 seconds, a yellow light surrounding the button will start flashing quickly.

When the controller is in "exclusion" mode, click on the button of the sensor you want to remove from the network. After removing a sensor from the network, the quick flashing

restarts in case we want to remove another sensor. To exit "exclusion" mode, click a second time on the button of the controller (the flashing stops).

2.2.1 disconnecting dimmers

Start by turning the light off manually (using the switch).

Now put the controller into "exclusion" mode by clicking on its button continuously for 3 seconds. Then put the light on manually using the switch. If the exclusion is successful, the bulb will flash twice. If not, repeat the previous procedure.

After adding the dimmer, the controller's flashing restarts (in case we want to remove another node). To exit "exclusion" mode, click a second time on the button of the controller (in this case, the flashing stops).

2.3. Building your own IoT infrastructure

The IoT infrastructure is composed of a set of devices (sensors and dimmers), a Z-stick controller, and a Raspberry Pi.

This section explains how to connect the controller to the Raspberry Pi and how to connect the Raspberry Pi to your laptop.

2.3.1 How to connect the controller to your Raspberry Pi?

Connect your "Z-Stick" controller to your Raspberry Pi as shown in Figure 4.

Figure 4: "Z-Stick" Controller plugged to a Raspberry Pi

2.3.2 How to connect your Raspberry Pi to your machine?

Connect the Raspberry Pi to your machine using the Ethernet cable. You should change your machine's IP configuration so that you can connect to the Raspberry Pi.

The IP address of your raspberry Pi is 192.168.1.2. See Annex 2 for the login and password of your raspberry. The wired connection between the raspberry and your laptop is used to have internet access. You have to configure your laptop as a gateway: your laptop must have the static address 192.168.1.1 and must allow internet access to the raspberry (internet sharing option must be activated on your laptop, see Annex 3).

The wifi connection is used to have multiple access to the raspberry.

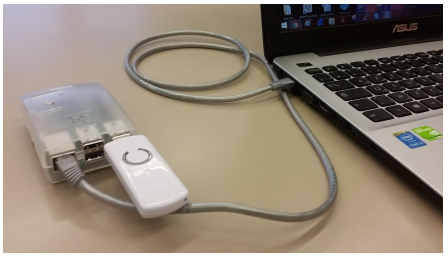

Figure 5: The Raspberry Pi connected to the PC via Internet

3. Configuring and programming your IoT platform

3.1. Using python-openzwave to configure a Z-wave network

Python-openzwave is an open source library, which enables the interaction with the controller connected to the Raspberry Pi so that we can retrieve data or send commands to the nodes (sensors and dimmers) connected to the network. The python-openzwave library builds a software representation of the Z-Wave network by mapping the components of the Z-wave network to a software objects (Figure 6).

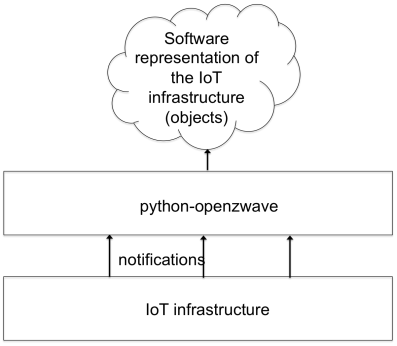

Figure 6: python-openzwave: Software (virtual) representation of the IoT infrastructure.

A detailed documentation of python-openzwave is available on this URL:

http://pyozw.readthedocs.io/en/latest/openzwave.html

This section briefly explains how to use python-openzwave to configure a Z-wave network.

The update of the software representation (objects) is done through notifications which inform the "python-openzwave" layer of the occurrence of "events" such as adding nodes, removing nodes, changing a node's measurements, etc. (Figure 6). For each notification, python-openzwave updates the software representation of the Z-wave network and sends signals to the "end user" application (Figure 7).

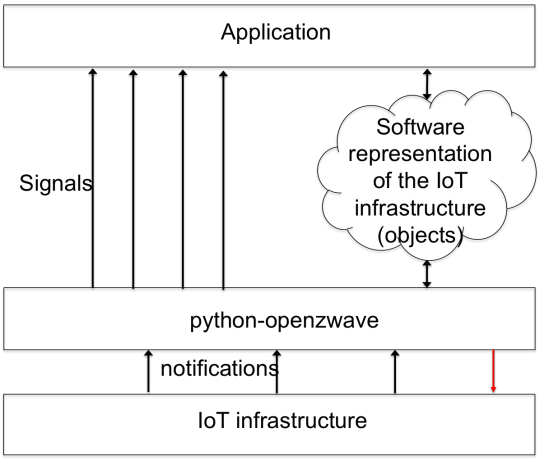

Figure 7: Signals are handled by the end user application

Here is the list of the most important classes, which represents the sensor's network:

"Network" class

This class represents the Z-Wave network. Here are some methods:

- home_id_str (type:string): unique identifier for the Z-Wave network.

- is_ready (type: Boolean): indicates if the network is ready or not. As soon as the network is ready, a *SIGNAL_NETWORK_READY* event is generated by python-openzwave.
- nodes_count (type:integer): number of network nodes (including the controller)
- controller (type:Controller): See Controller class.
- nodes (type: dict() of "node"): a dictionary of "node" objects representing the nodes connected to the network (including the controller).
- start () (return type:integer): This method generates the software representation of the Z-Wave network managed by the controller.
- stop () (return type:none): this method deletes the software representation

"Controller" class

This class models the network's controller. Here are few methods:

- name (type: string): controller's name.
- device (type: string): path of the device's driver (e.g. /dev/USB0)
- begin_command_add_device() (return type: Boolean): gets the controller in inclusion mode. The method returns "True" if the command is accepted and started.
- begin_command_remove_device() (return type: Boolean): gets the controller in exclusion mode. The method returns "True" if the command is accepted and started.
- cancel_command() (return type:none): gets the controller out of the inclusion or exclusion mode.

"Node" class

This class is used to represent a node: sensor or controller. A "node" object has the following methods:

- node_id (type: integer): identifying the node in the network (note: the node with the id "1" is always the controller).
- product_name (type:string): name of the product. (Sensor's product name is "Multisensor 6" and dimmer's product name is "ZE27")
- name (type:string): name of the node.
- location (type:string) : location of the node.
- isReady (type: Boolean): Indicates whether the node is ready or not (this method is not in the documentation)
- getNodeQueryStage (type:String): Indicates the State of the Node object (Once the Query Stage is at "Complete", this Node object is ready to be used).
- neighbors (type: set()) : a set of the neighbouring nodes. Neighbouring nodes are nodes that can be reached by the current node
- values (type:dict()): This is a dictionary containing pairs of (ID, value_object). A "value" object represents a measure or a configuration parameter of this Node.
- set_config_param(param, value, size) : Set the value of a configurable parameter in a device. Some devices have various parameters that can be configured to control the device behaviour. These are not reported by the device over the Z-Wave network but can usually be found in the devices user manual. This method returns immediately, without waiting for confirmation from the device that the change has been made. The "param" argument is the index of the parameter, while "size" is the size of the value.

- get_battery_level()(return type:integer): retrieves the battery level of the node.
- get_values (class_id, genre, type, readonly, writeonly) (return type:set()): retrieves measures and/or parameters of a node from the dictionary "values" mentioned earlier:
    - The "class_id" parameter describes the class of the value (temperature, presence, battery level, etc.). The classes and their IDs are fixed by the Z-Wave specifications. For example, you should use "0x31" as class_id to retrieve temperature, luminance and humidity and "0x30" for motion. To keep things simple, you can put this argument at "All" to retrieve values from all classes.
    - "genre" specifies whether the values are appropriate for the user as the temperature measurements (in this case genre = "User") or setup for the administrator such as sending frequency measures (genre = "Config").
    - "type" is the type of value ("String", "Int"...).
    - "readonly" and "writeonly" are boolean indicating whether the required values are respectively “read only” or “write only” values.

  Note: One can use "All" with the five parameters to recover all kinds of values.

Example1: get_values ("All", "User", "All", True, False) is called to retrieve all measurements.

Example2: get_values ("All","All","All",False,"All") is called to retrieve all Configuration and System parameters.

For further details regarding the methods of this class, see these URLs (in case of contradiction between the URLs and this document, this document takes precedence):

http://pyozw.readthedocs.io/en/latest/node.html

“value” class

This class represents a value retrieved from a node. Here are few methods:

- label (type:string): the label of the value (eg. "Temperature" in the case of temperature value recovered by the sensor node or "Level" in case of brightness level).
- data (type:depends on the type of the value) : value of the measurement.
- units (type:string) : unit of the value
- index (type:integer) : a unique ID of the value (useful to identify a parameter value by its index)
- genre (type:string) : genre of the value (Basic, User, Config, System)
- type (type:string) : type of the value
- command_class (type:integer) : class_id of the value.

For further details regarding methods of this class, see this URL:

http://pyozw.readthedocs.io/en/latest/value.html

3.2. Using python-openzwave to program your IoT application

The IP address of your raspberry Pi is 192.168.1.2.

python-openzwave library is installed in `/home/pi/IoTLab/python-openzwave`. You don’t need to access this directory. The installed version is `0.3.0b8` -- ***please, do not use newer releases as they might not work with our template code***.

In the home directory (/home/pi/), download the source codes of the Smart building project:

git clone https://githepia.hesge.ch/lsds/Smart-building.git

A directory called *Smart-building* is created. It contains the application source codes.

The application to write is composed of three components:

- Source code of the REST server: *flask-main.py* file
- "Backend" class, which implements the methods called by the REST server (Figure 8): backend.py file. This class uses python-openzwave library.
- Python-openzwave library.

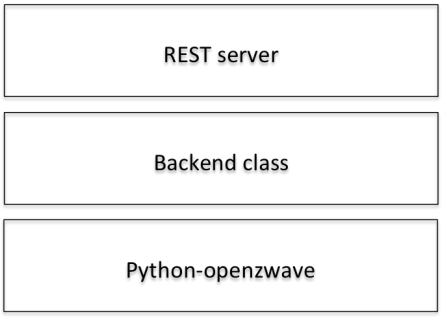

Figure 8: Architecture of the application

flask-main.py and *backend.py* are stored in the *Smart-building* directory.

To run the server, go to the *Smart-building* directory and type:

sudo python flask-main.py

flask-main.py is a RESTful web service deployed on port 5000: 192.168.1.2:5000

This web service (*flask-main.py* and *backend.py*) is assumed to implement a set of methods for the Z-wave network management. Only four methods are implemented (in the *backend.py* file): *get_temperature, get_humidity, start* and *stop*. The other methods are missing.

A complete documentation of the routes of the methods supported by the "Backend" class is available on this link : https://githepia.hesge.ch/lsds/Smart-building/tree/master/doc

Upload the directory and click on the *index.html* file.

The goal of this lab is to implement the missing methods and test them via the REST client: https://githepia.hesge.ch/lsds/Smart-building/blob/master/post_client.py

Work to do:

1. Install the Z-Wave network as explained in section 2
2. Implement the methods listed in annex 1 according to the specifications given in the documentation (see the cyberlearn web site). These methods are used by the *flask_main* REST Web server.
3. Test the REST server with the REST client.

Deliverable: Your backend.py file with your implemented methods,

Annex 1: List of methods to implement

1. get_nodes_list
2. get_sensors_list
3. get_temperature (already implemented)
4. get_humidity (already implemented)
5. get_luminance
6. get_motion
7. get_battery
8. get_all_Measures
9. get_dimmers
10. get_dimmer_level
11. set_dimmer_level
12. set_node_location
13. set_node_name
14. get_node_location
15. get_node_name
16. set_node_config_parameter
17. get_node_config_parameter
18. get_nodes_configuration
19. set_basic_sensor_nodes_configuration
20. get_nodes_Configuration
21. network_info
22. set_node_location
23. get_node_location
24. get_neighbours_list
25. removeNode
26. addNode

Annex 2: login and password of raspberries.

Here is the login (username is '`pi`' for all) and password of your Raspberry Pi. Please do not forget to update it.

ID	Password	SSID
1	owez0ljf	Pi1-AP
2	42cnvsnf	Pi2-AP
3	veoz8pcj	Pi3-AP
4	qcs5qz4z	Pi4-AP
5	5yfu6ki4	Pi5-AP
6	t265s0x0	Pi6-AP
7	kc483hza	Pi7-AP
8	7c3xgbyu	Pi8-AP
9		
	4ej577uh	Pi10-AP
11	m4pppt3j	Pi11-AP
12	breb21ef	Pi12-AP
13	plop3br2	Pi13-AP
14	8l3w9asb	Pi14-AP
15	kram46wj	Pi15-AP

Ethernet IP = 192.168.1.2

WiFi IP = 192.168.2.1

Login and WiFi passwords are the same.

Annex 3: internet sharing with Linux

In order to share a network card as an internet gateway in Linux, IP forwarding and masquerading must be set up [1]. The following `internet-sharing` script does the trick:

```
#!/bin/bash
gwint=${1:-'net0'}  # internet gateway interface -- all outbound
                    # traffic
clint=${2:-'net1'}  # client interface -- input traffic

sysctl net.ipv4.ip_forward=1 \
  net.ipv6.conf.default.forwarding=1 \
  net.ipv6.conf.all.forwarding=1

iptables -t nat -A POSTROUTING -o $gwint -j MASQUERADE
iptables -A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED \
  -j ACCEPT
iptables -A FORWARD -i $clint -o $gwint -j ACCEPT
```

By default, the script expects two interfaces named `net0` and `net1` for, respectively, the internet gateway and the client inbound interface. So, if you want to reroute all incoming traffic from, say, `eth0` (where your Raspberry is connected to) through your wireless connection `wlan0`, you'd call the script like this (as *superuser*):

```
$ sudo internet-sharing wlan0 eth0
```

To make the changes persistent, please consult your Linux distribution's documentation.

[1] https://wiki.archlinux.org/index.php/Internet_sharing