Internet of Things: Services for a Connected World

Laboratory work 1: Sensing and Actuating *things* via IoT gateway
Last updated: October 2021
Version: 1.3

Table of Contents

1 Introduction

In the Internet of Things (IoT) paradigm, *things* are connected to the Internet for sensing and actuating. The objective of such is to collect data from sensors and affect actuators. A thing can be a physical entity (e.g., sensor or actuator) or a virtual entity (e.g., Twitter account, Trafikverket API, etc.).

1.1 Overview

The purpose of this lab is to introduce an IoT application where IoT things such as temperature sensors and actuators (e.g., lighting, heating, etc.) are accessed/controlled via an IoT gateway, e.g., Raspberry Pi. As you may already know that IoT typically follows a three-layer architecture where the first layer consists of things followed by a gateway and other application(s). Gateway collects data from things and forwards it to the application layer (see figure 1). Based on this principle, during this lab, we will collect data from things via a Raspberry Pi which is utilized as an IoT gateway. However, we will limit our focus to things and gateway only during this particular lab. During lab 2, we will try to see how to forward and access collected data from things via a gateway to other application(s).
We will further take advantage of a Tellstick (a transceiver) during the lab, which enables remote wireless communication with the *things*. Therefore, in this lab, we will learn how to connect to a Tellstick and things (sensors and actuators) on a Raspberry Pi for sensing and actuating (see figure 2).

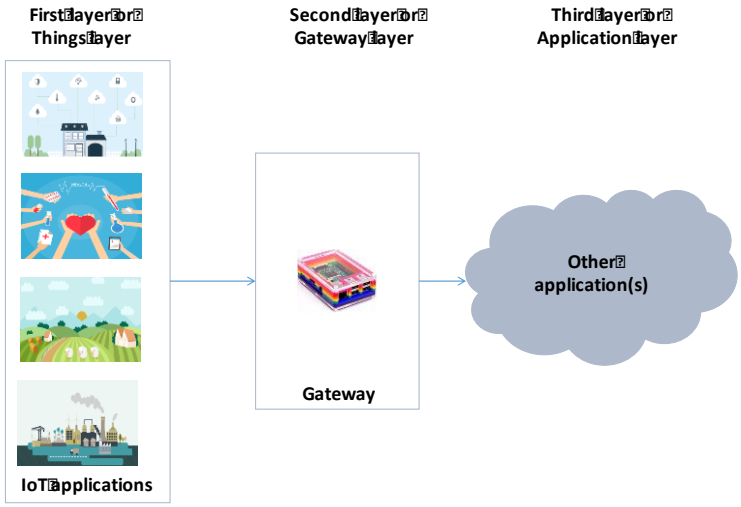


Figure 1: Typical three-layer IoT

1.2 Lab purpose

After the lab, you will:

- ✓ be able to fetch data through GET-requests to an IoT portal
- ✓ be able to configure and control actuators
- ✓ be able to know how to read sensor values

- ✓ be able to know how to utilize Raspberry Pi as an IoT gateway

1.3 Before lab

You should have:

- read through this entire lab manual
- glanced through the lecture notes
- read about Raspberry Pi, Tellstick and Telldus Live! (recommended)

1.4 Tools for the lab

For this lab, you are provided with the following tools and items:

- A Raspberry Pi 4, with static IP, network access and where SSH has been enabled
- Access to temperature and humidity sensors through Telldus Live API
- An actuator, connected to Telldus Live
- A lamp
- PuTTY

In preparation for the lab, the following has been done:

Raspberry Pi

Rasbian has been installed, and SSH has been enabled. To work in DSV, each Raspberry Pi has been given a static IP.

Telldus Live, sensors and actuators

The APIs, which are used for the lab through the Raspberry Pi, are provided by Telldus Live. To use them, an account has been registered on Telldus Live; a Tellstick has been added to the account. To the Tellstick, working as a gateway, several sensors and actuators have been linked to it.

2 Background

2.1 Raspberry Pi

The Raspberry Pi is a credit-card-sized single-board computer developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and developing countries [1, 2]. Raspberry Pi is currently into its 4th generation, with Raspberry Pi 4 released in summer 2019. We will use Raspberry Pi 4 (model B) during the lab. [1] and [2] have more information.

A user can interact with the Raspberry Pi either by connecting peripheral devices such as a mouse, keyboard, and a screen to the board or by connecting to the Raspberry Pi through SSH (Secure shell) [3]. Using SSH, the user can send a command to the Raspberry Pi using Linux commands [4]. In this lab, much of the interaction with the Raspberry Pi is through SSH.

2.2 Tellstick znet lite v2

We will use Tellstick ZNet lite v2 as a wireless transmitter and receiver, i.e., transceiver manufactured by Telldus Technologies [5, 6]. It has two-way communication on Z-Wave, which provides long-range. It also transmits and receives signals at 433MHz, which provides broad compatibility with your existing receivers and sensors.

Range: Up to 30 meters
Frequency: 868.42MHz (EU) (transmit & receive) + 433.92MHz (transmit & receive)
Z-Wave chip: 500 Plus
Power supply: 5V DC / 1A

Telldus products and devices connected to the Tellstick can be monitored, configured, and controlled by their Telldus Live portal or through their mobile applications. For this lab, we will use the API for Telldus Live to retrieve data and send requests.

2.3 Sensors and Actuators

Read more in [8] and [9] about sensors and actuators. In short, sensors are used to measure or sense the environment and provide an output based on the measurement or sensing (e.g., temperature sensor, motion sensor, pressure sensor, etc.). On the other hand, actuators are used to affect a situation, i.e., controlling the environment. Examples of actuators are: light switch, door lock, window blinds, thermostat, etc. As for this lab, we will use sensors and actuators manufactured by Proove [10], specifically temperature sensors and switches to turn on/off lighting.

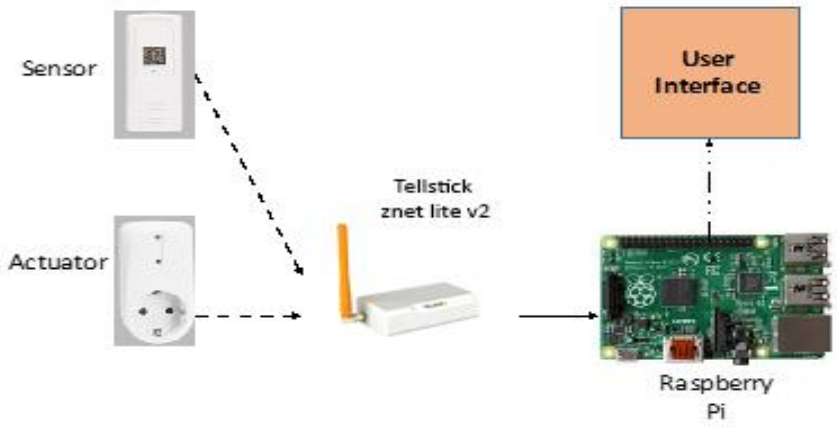


Figure 2: Lab Scenario

2.4 Code template

For this workshop, you have been provided with a python code template. The script communicates with Telldus Live service through GET requests. The requests are authenticated using private keys provided through api.telldus.com/keys.

The code uses uuid[11] to create a nonce[12] which is used in the header of the request.

2.5 Telldus API

The documentation for the API used in this lab is provided by Telldus [13].
The calls should be to:
https://api.telldus.com/{format}/{function}

Where:
{format} it the returned format and should be either JSON or XML
{function} is the function to call. (…)

Calls can be made as either GET or POST.
Example. List of available sensors and return the data as JSON:
https://api.telldus.com/json/sensors/list

TellStick ZNet

Available methods:

device:

- device/bell: Sends bell command to devices supporting this.
  Parameters: id
- device/dim: Sends a dim command to devices supporting this.
  Parameters: id, level (0-255)
- device/down: Sends a "down" command to devices supporting this.
  Parameters: id
- device/info: Returns information about a specific device.
  Parameters: id, (if applicable: supportedMethods)
- device/learn: Sends a special learn command to some devices that need a special learn-command to be used from TellStick
  Parameters: id
- device/stop: Send a "stop" command to the device.
  Parameters: id
- device/turnOff: Turns a device off.
  Parameters: id
- device/turnOn: Turns a device on.
  Parameters: id
- device/up: Send an "up" command to the device.
  Parameters: id

devices:

devices/list: Returns a list of all devices.

sensor:

sensor/info: Returns information about a specific sensor.
Parameters: id, (if applicable: includeUnit)

sensors:
sensors/list: Returns a list of all sensors associated with the current user

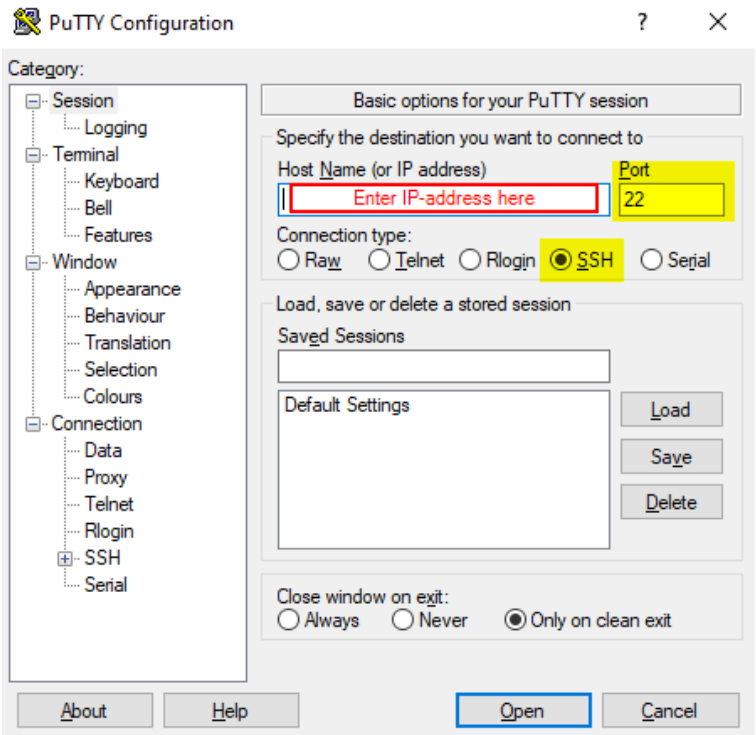
3 Laboratory

As depicted in figure 2, the lab scenario introduces sensing and actuating via a gateway. We will sense the current temperature at the lab and actuate switches. We will use Raspberry Pi as a gateway and PuTTY [14] as the user interface.
During the lab, the first task is to check if an operating system (OS) is installed on the Raspberry pi. If not, we might need to spend some time installing the required OS on raspberry pi. When OS is installed, then we can go ahead and log in to the Raspberry Pi. We will use third-party tools to access via SSH the Raspberry pi (e.g., putty). We need to know the IP address of the Raspberry Pi prior to SSH. Once we access the pi via SSH, we can now login via the following default credentials (unless they are changed, in that case, it will be communicated during the lab):

Username: *pi*
Password: *IoT@2021*

3.1 Connect to Raspberry Pi through PuTTY

To connect to a device through SSH (Secure shell), open PuTTY.
In the field for Host Name, enter the IP address of the Raspberry Pi you want to connect to it. If you are using one of the provided Raspberry pis for the lab, you can find the IP address on the device.
The port number can be left as 22, and make sure “SSH” is selected.
When ready, click Open to initialize the connection.



A new window will open, with a prompt for username and password.

3.2 Setup and create the first script

When successfully connected to the Raspberry Pi, you can now interact with it through Linux commands. See below for a list of useful commands for this lab, or visit the Raspberry Pi documentation for Linux commands [4].

Command	Description	Example
`mkdir foldername`	Create a folder named *foldername*	`mkdir my-folder`
`ls`	Lists content of current directory	`ls`
`cd directory`	Changes the current directory to *directory*	`cd my-folder` `cd /home/pi/my-folder`
`touch filename.type`	Creates a file with *filename* and *type*, if no such file exists	`touch hello.py`
`nano filename`	Opens *filename* for editing in the editor nano	`nano hello.py`
`sudo`	Run a command as superuser	`sudo nano hello.py`
`rm filename`	Removes the file *filename*	`rm hello.py`
`rm -rf foldername`	Attempts to remove the folder *foldername*, and all its content	`rm -rf my-folder`

Following the login, we will now create the Python script on the Raspberry pi.

Create a folder
mkdir iot-lab-ht21

You can see that a new folder has been created with the following command:
ls
or to view a prettier list:
ls -l

Change to the new folder
cd iot-lab-ht21

Now, you need to create a new file where you can add the code. You can create a new file using the *touch* command and the name of the file. Use the following to create a file named ListSensors.py:
touch ListSensors.py

Again, you can check the content of the directory with *ls* or *ls -l*

To edit the file, open the text editor (nano) by using the command `sudo nano filename.filetype`.
To open the file ListSensors.py, type the following:
sudo nano ListSensors.py

In nano, paste the provided code to get a list of the sensors.
(Please note that indentation is important in python, and copying code from a document might change the formatting!)

```
# -*- coding: utf-8 -*-

#!/usr/bin/env python


# This is where to insert your generated API keys (http://api.telldus.com/keys)

pubkey = "ABCDEFGHIJKL123456789" # Public Key

privkey = "ABCDEFGHIJKL123456789" # Private Key

token = "ABCDEFGHIJKL123456789" # Token

secret = "ABCDEFGHIJKL123456789" # Token Secret


import requests, json, hashlib, uuid, time

localtime = time.localtime(time.time())

timestamp = str(time.mktime(localtime))

nonce = uuid.uuid4().hex

oauthSignature = (privkey + "%26" + secret)


# GET-request
response = requests.get(
                url="https://pa-api.telldus.com/json/sensors/list",
                params={
                                "includeValues": "1",
                },
                headers={
                                "Authorization": 'OAuth oauth_consumer_key="{pubkey}",
oauth_nonce="{nonce}", oauth_signature="{oauthSignature}", oauth_signature_method="PLAINTEXT",
oauth_timestamp="{timestamp}", oauth_token="{token}", oauth_version="1.0"'.format(pubkey=pubkey,
nonce=nonce, oauthSignature=oauthSignature, timestamp=timestamp, token=token),
                                },
                )
# Output/response from GET-request
responseData = response.json()

print(responseData)

#print(json.dumps(responseData, indent=4, sort_keys=True))
```

Note that in the template code provided, the values for the strings pubkey, privkey, token and secret are all placeholders. Saving the file and running it now would return an error. **Replace** the placeholder values for the four variables with the values provided by the workshop leader.

To save the file and exit nano:

control + O (write/save)
press Enter (select name of file and save)
control + X (exit nano)

3.3 Run and modify the List sensor script

The first thing we want to do is check the sensors by listSensors.py. Run the program:
python ListSensors.py

What data was retrieved? Can you find any useful values from the sensors?

To make it a bit more readable, we can edit the file. This can be done by using nano, a command-line text editor in Linux. The following command will open nano, where you can edit the file:

```
sudo nano ListSensors.py
```

In nano, find the row that prints the data. Remove it, or use # to mark it as a comment. Then, remove the # in the comment below.
Save the file with the same name:

```
control + O (write/save)
press Enter (select name of file and save)
control + X (exit nano)
```

Run the same program again:

```
python ListSensors.py
```

The uncommented line prettified the JSON response from the GET request, making it easier to read.

Can you find the values from the sensors?

The client name in the response is the given name to the Tellstick. Each sensor also has a name and an ID.

3.4 Modify the script to retrieve a list of devices

To retrieve the list of devices, which includes actuators, we need to modify the request. As noted, the request follows the format https://api.telldus.com/{format}/{function}.
The examples in the workshop work with JSON format, but the function needs to be modified to match the new request.

Have a look at the API documentation. Find a method for listing all devices.

To create a new script in Raspbian, one can use *sudo nano nameOfFile.filetype*.
For example, to create a python file with the name ListDevices:

```
sudo nano ListDevices.py
```

This opens up nano, where you can edit the new file with the chosen name.
Here, write the code or paste it to the terminal window and modify the request.

Save and close nano:

```
control + O (write/save)
press Enter (select name of file and save)
```

```
control + X (exit nano)
```

Run the new script by *python nameOfFile.py*. If needed, open the file again to prettify the output.

Did the values listed for each device differ from the sensors?

For the lab, the goal is to create a script to turn on and/or off an actuator. For this, you need to identify the ID of the actuator device you want to control.

In the output with a list of devices, identify the ID of the actuator you want to control. When found, scribble it down or save it for later.
Hint: the ID should be a larger number, consisting of 5+ characters.

3.5 Create the script to control an actuator

To turn on or off a device, yet another call is needed. Again, refer to the API documentation and identify a suitable method. Note the required parameter(s).

Create another python file and open this in nano: *sudo nano nameOfFile.filetype*.

In the new file, paste the edited code or code template and edit it in nano. Edit the url value to match the new GET request.

Parameters are added or edited in the request:

```
params={
        "parameterName": "parameterValue",
}
```

To add the parameter id with the value 123456789:

```
params={
        "id": "123456789",
},
```

Add the ID you got from the previous step as the value.
When done editing, save the file and close nano.

Run the script to make sure it works, then show the workshop leader.

Hint: for testing purposes, it might be easier to create two scripts: one for turning an actuator on, and one for turning it off.
Another option is to create a script that turns the actuator on and off again, using a loop, *time.sleep(n)* or Timer by importing threading.

4 Lab report

After the end of the lab work, you should hand over a lab report and submit it on the iLearn2 course page. In the report, you should write the following (Numerical values in parenthesis indicate maximum obtainable points for answering each question):

1. What kind of data did you get from ListSensors? Were there any values you were surprised to see? Was there any information or value you expected to see but did not? (1)
2. What data did you get from ListDevices? Did it differ from ListSensors? (1)
3. Can you think of any reason and/or scenario when one would prefer to have the data from sensors and devices go through a Raspberry Pi (or other gateways) rather than directly to the end-users device? Motivate your answer in a few sentences. (1.5)
4. Briefly describe five different IoT applications that you think can be used by the same approach (sensor/actuators – gateway - application) as in this lab. Feel free to include additional sensors that were not used in this lab. (1.5)

To pass the lab, you must obtain ***3.5 points*** out of possible **5 points**.
All questions are **mandatory** to answer.
No submission will be entertained unless you perform the actual lab during scheduled lab time.

References

[1] Raspberry Pi, Wikipedia, https://en.wikipedia.org/wiki/Raspberry_Pi
[2] https://www.raspberrypi.org/
[3] Secure shell, Wikipedia, https://en.wikipedia.org/wiki/Secure_Shell
[4] Linux commands, Raspberry Pi Foundation
https://www.raspberrypi.org/documentation/linux/usage/commands.md
[5] https://telldus.com/produkt/z-wave-gateway-tellstick-znet-lite-ver-2/
[6] http://telldus.se/
[7] http://developer.telldus.com/wiki/TellStick_conf
[8] Sensor Wikipedia, https://en.wikipedia.org/wiki/Sensor
[9] Actuator Wikipedia, https://en.wikipedia.org/wiki/Actuator
[10] Proove: http://proove.se/about-proove/
[11] Python UUID: https://docs.python.org/3/library/uuid.html
[12] Wikipedia Cryptographic nonce: https://en.wikipedia.org/wiki/Cryptographic_nonce
[13] Telldus Live! API https://api.telldus.com/documentation/live
[14] PuTTY, https://en.wikipedia.org/wiki/PuTTY