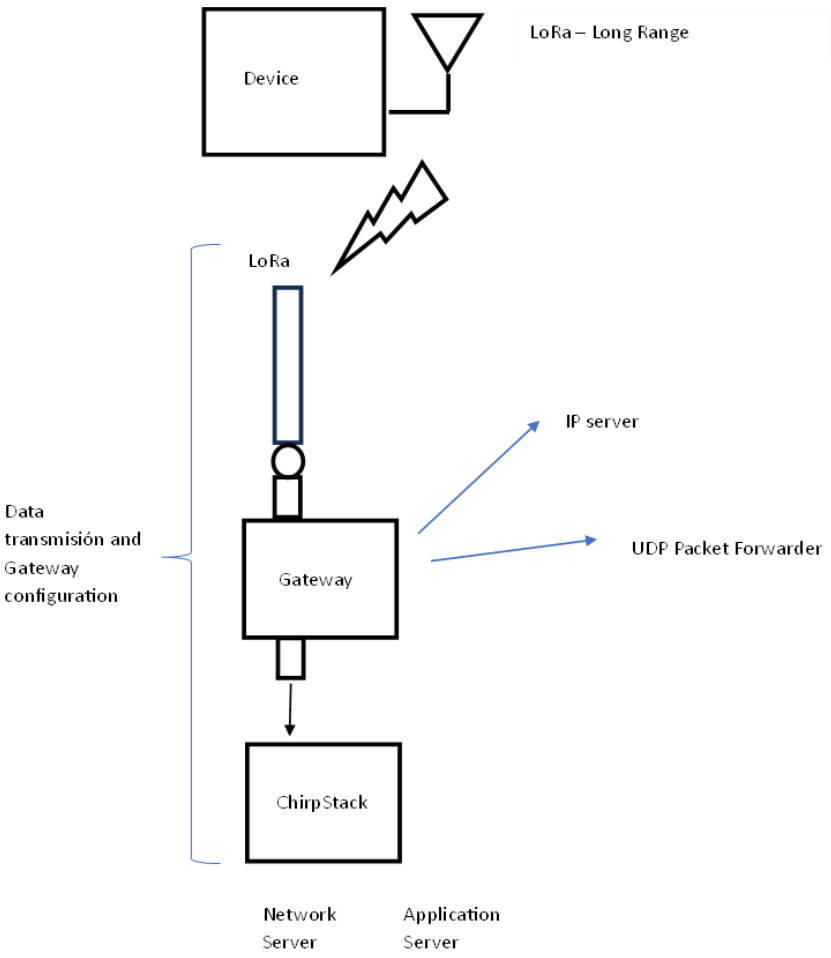
PRACTICE 2: DATA TRANSMISSION – LORA NETWORK SERVER CONFIGURATION

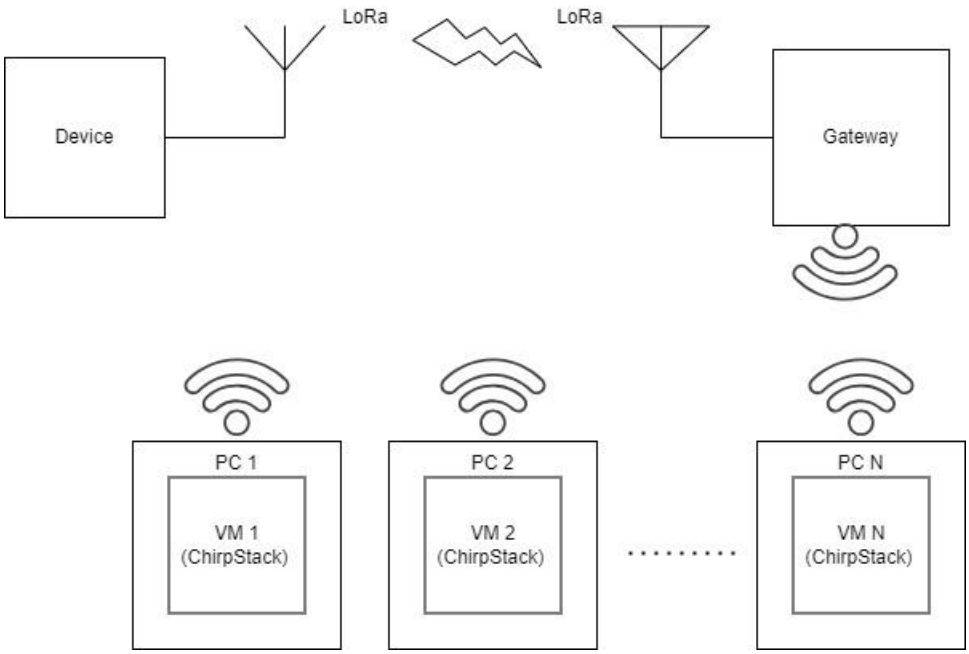
Authors: Alejandro Fernández Garrido

Francisco Jesús Sánchez Rubio

Following the above practice, the data sent by the sensors has to be received by a LoRa Gateway. This Gateway would be the appoximate LoRaWAN protocol equivalent of a TCP/IP router. Its function is essentially to receive LoRa packets and forward them (packet forwarding) with another protocol by backhauling them on Ethernet, WiFi or another medium.



In this practice, the network and application server will be configured. For this purpose, a Gateway has been previously configured to point to the address of each server as the destination for the forwarded packets.



1. Network server and application installation

The network server and application to be used will be Chirpstack. To install it, open a new terminal window and follow the steps in the following link: https://www.chirpstack.io/docs/getting-started/debian-ubuntu.html

In a usual server setup, you would subscribe multiple LoRa gateways to a single server. However, in this assignment we only have one gateway that must be shared between multiple servers. For this reason, one of the computers in the lab has been set up to act as a "relay" to distribute the data packets to every other server. For this setup to work, we need to make one extra configuration step in the installation process.

At the end of the installation, navigate to the folder "/etc/chirpstack" and edit the **region_eu868.toml** file, changing the *server* field under the *[mqtt]* label to the address shown on the classroom whiteboard. Restart the Chirpstack server by running the command "sudo systemctl restart chirpstack" to apply the change.

Lastly, we should get the following results when running the command "sudo systemctl status chirpstack" and "sudo systemctl status chirpstack-gateway-bridge":

```
ubuntuiot@ubuntuiot-VirtualBox:~$ sudo systemctl status chirpstack
[sudo] password for ubuntuiot:
● chirpstack.service - ChirpStack open-source LoRaWAN Network Server
     Loaded: loaded (/lib/systemd/system/chirpstack.service; enabled; vendor preset: enabled)
     Active: active (running) since Sun 2023-10-15 20:00:24 CEST; 1min 29s ago
       Docs: https://www.chirpstack.io/
   Main PID: 719 (chirpstack)
      Tasks: 12 (limit: 4600)
     Memory: 23.6M
        CPU: 200ms
     CGroup: /system.slice/chirpstack.service
             └─719 /usr/bin/chirpstack -c /etc/chirpstack/

oct 15 20:00:42 ubuntuiot-VirtualBox chirpstack[719]: 2023-10-15T18:00:42.081971Z  INFO chirpstack::gateway::backend: Setting up ga>
oct 15 20:00:42 ubuntuiot-VirtualBox chirpstack[719]: 2023-10-15T18:00:42.083164Z  INFO chirpstack::gateway::backend::mqtt: Connect>
oct 15 20:00:44 ubuntuiot-VirtualBox chirpstack[719]: 2023-10-15T18:00:44.184677Z  INFO chirpstack::gateway::backend::mqtt: Connect>
oct 15 20:00:44 ubuntuiot-VirtualBox chirpstack[719]: 2023-10-15T18:00:44.184768Z  INFO chirpstack::gateway::backend::mqtt: Startin>
oct 15 20:00:44 ubuntuiot-VirtualBox chirpstack[719]: 2023-10-15T18:00:44.184788Z  INFO chirpstack::gateway::backend::mqtt: Subscri>
oct 15 20:00:44 ubuntuiot-VirtualBox chirpstack[719]: 2023-10-15T18:00:44.184920Z  WARN chirpstack::gateway::backend: Config exists>
oct 15 20:00:44 ubuntuiot-VirtualBox chirpstack[719]: 2023-10-15T18:00:44.184927Z  INFO chirpstack::downlink: Setting up Class-B/C >
oct 15 20:00:44 ubuntuiot-VirtualBox chirpstack[719]: 2023-10-15T18:00:44.185003Z  INFO chirpstack::downlink: Setting up multicast >
oct 15 20:00:44 ubuntuiot-VirtualBox chirpstack[719]: 2023-10-15T18:00:44.185070Z  INFO chirpstack::api: Setting up API interface b>
oct 15 20:00:44 ubuntuiot-VirtualBox chirpstack[719]: 2023-10-15T18:00:44.185839Z  WARN chirpstack::api::backend: Backend interface>
lines 1-21/21 (END)
```

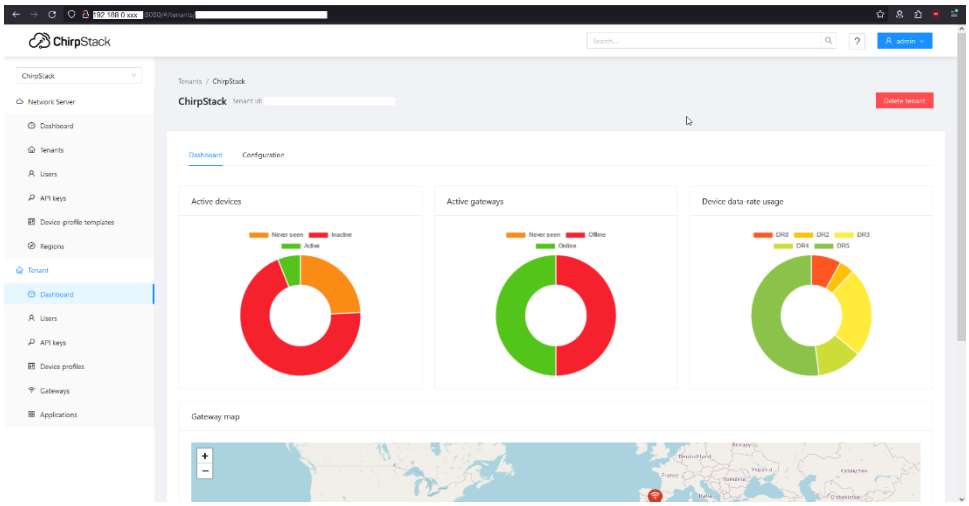
```
ubuntuiot@ubuntuiot-VirtualBox:~$ sudo systemctl status chirpstack-gateway-bridge
● chirpstack-gateway-bridge.service - ChirpStack Gateway Bridge
     Loaded: loaded (/lib/systemd/system/chirpstack-gateway-bridge.service; enabled; vendor preset: enabled)
     Active: active (running) since Sun 2023-10-15 20:00:24 CEST; 2min 42s ago
       Docs: https://www.chirpstack.io/
   Main PID: 716 (chirpstack-gate)
      Tasks: 5 (limit: 4600)
     Memory: 13.9M
        CPU: 249ms
     CGroup: /system.slice/chirpstack-gateway-bridge.service
             └─716 /usr/bin/chirpstack-gateway-bridge

oct 15 20:00:24 ubuntuiot-VirtualBox systemd[1]: Started ChirpStack Gateway Bridge.
oct 15 20:00:24 ubuntuiot-VirtualBox chirpstack-gateway-bridge[716]: time="2023-10-15T20:00:24.68600096+02:00" level=info msg="star>
oct 15 20:00:24 ubuntuiot-VirtualBox chirpstack-gateway-bridge[716]: time="2023-10-15T20:00:24.692048238+02:00" level=info msg="bac>
oct 15 20:00:24 ubuntuiot-VirtualBox chirpstack-gateway-bridge[716]: time="2023-10-15T20:00:24.69928267+02:00" level=warning msg="[>
oct 15 20:00:24 ubuntuiot-VirtualBox chirpstack-gateway-bridge[716]: time="2023-10-15T20:00:24.699372739+02:00" level=info msg="int>
lines 1-16/16 (END)
```

2. Registering the device and Gateway with the application server

Once Chirpstack is installed and we are sure it is working, we will connect from its web application. To do this we will first check the IP address of our machine and enter it in any web browser, followed by the port of the ChirpStack web application. By default this is 8080, so the address to enter will be 192.168.0.xxx:8080.

Once inside the application you will be asked for a username and password. By default these are "admin" for both fields. Entering the credentials will take you to the following dashboard:



The gateway will register in the "Gateway" tab, and its EUI will be entered. If this is done correctly we will start to see diagnostic messages coming from the gateway, indicating a successful connection.

Regarding the device, we first go to "Device profiles", where we will create a profile.

This is important for the activation of the device, as we configured it in the previous practice as OTAA. In our case, we will leave all the default options, checking that the device will indeed be configured as OTAA.

On the other hand, we configure the decoder for incoming data.

We go to the "Codec" tab and select the option "JavaScript functions". Once there, we will create a decoder inside the function "decodeUplink (input)".

It is important to note that when we programmed the Arduino, the output measurements were floats, but the information we received in the packet (input.bytes) are just plain bytes. The decoder must have a function that reconstructs the original variable type working at bit level and obtain the measured values in the device.

The return of the decoder function is a JSON in the format {data: {…}}.

Once the decoder is finished, go to the applications tab and create a new application. Here the devices of our network will be registered. Click on the "Add device" button and enter a name and the EUI of your device. Finally, we generate the OTAA application key for the device and click on "Submit".

After having done all this, we connect the device to the PC and open the Arduino IDE, and in the OTAA_KEY definition we introduce the key that we have just generated and we flash the program again.

If everything has been done correctly, in the "Events" tab on the device in Chirpstack you should start to see incoming data packets. Clicking on the "Up" button should show the correctly decoded data.

Annex

Here are some tips on how to install the server:

- Execute the command lines one by one and pay attention to the responses in the terminal.
- The commands "sudo systemctl status <service>" and "sudo journalctl -f -n 100 -u <service>" are useful for debugging possible problems during Chirpstack installation.

Device profile decoder :

```
function decodeUplink(input) {

  bytes = input.bytes;

  function bytesToFloat(bytes) {
        var bits = bytes[3]<<24 | bytes[2]<<16 | bytes[1]<<8 | bytes[0];
```

```
            …
            var f = sign * mantissa * Math.pow(2, e - 150);
            return f;
}

  function bytesToInt16(bytes) {
            var result = ...
            return result;
  }

  return {
    data: {
         temperature: …
         …
    }
  };
}
```