IoT Engineering
6: Raspberry Pi as a Local IoT Gateway

Slides: tmb.gr/iot-6

Overview

These slides introduce the *Pi as a local gateway*.

Connecting to, receiving data from BLE devices.

Providing the data to Web servers or clients.

Prerequisites

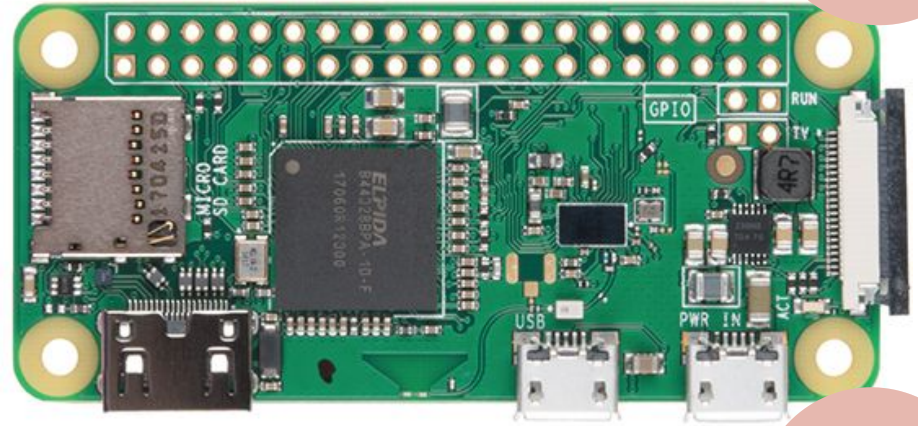
Set up SSH access to the Raspberry Pi, via USB/Wi-Fi:

Check the Wiki entry on Raspberry Pi Zero W Setup.

Submit the Raspberry Pi MAC address via Teams*.

*For Wi-Fi access on campus.

Raspberry Pi



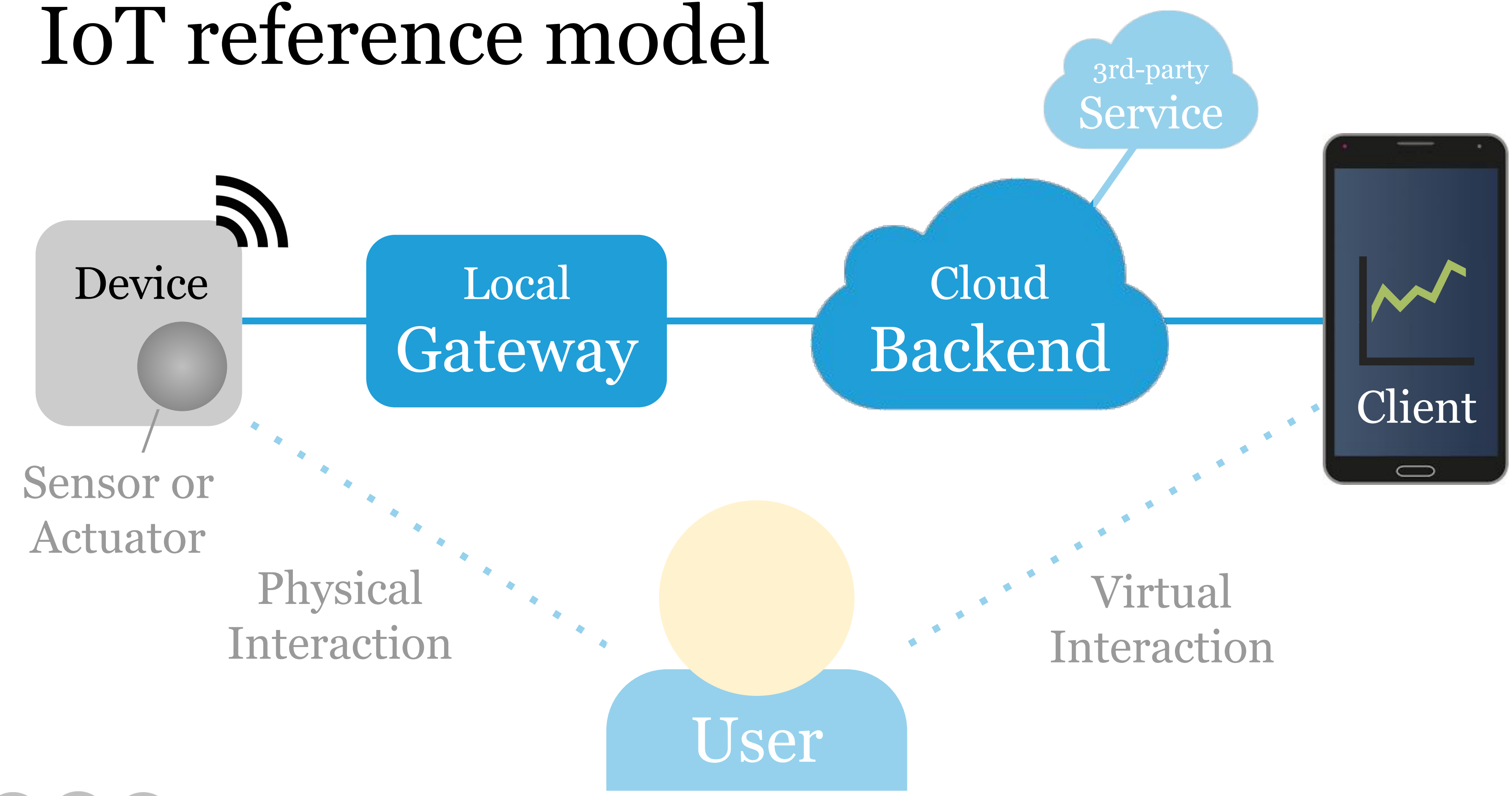
Single-board computer, running a full Linux OS.

https://raspberrypi.org/products/raspberry-pi-zero-w/

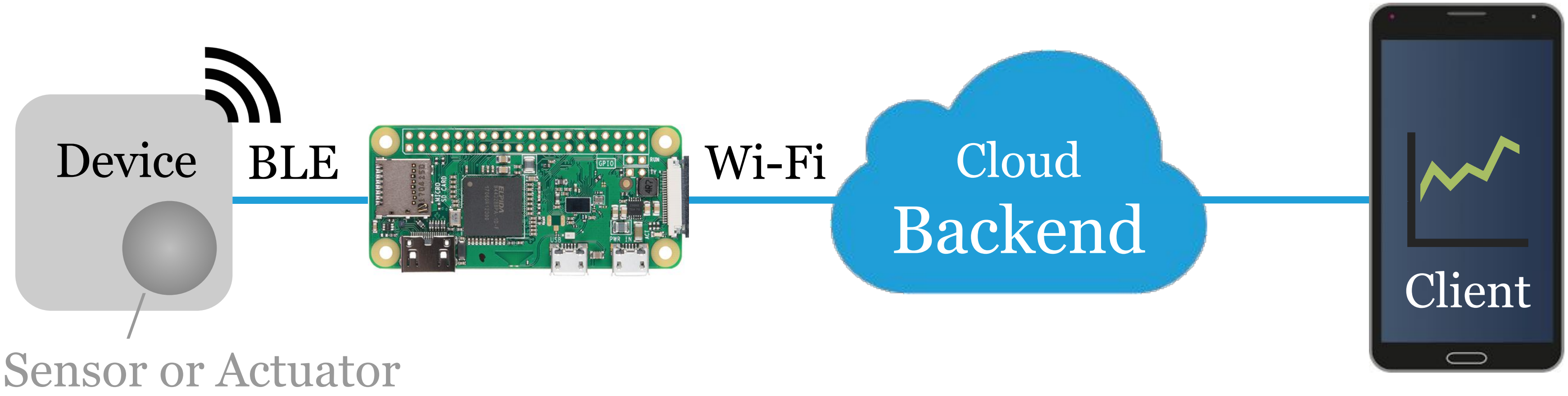
1GHz, single core ARM CPU, 512 MB RAM, Wi-Fi, Bluetooth LE, Mini HDMI, USB OTG, etc.

Hold the board as shown, avoid touching any chips.

IoT reference model



Local gateway



A local gateway connects devices in the local network to the backend, forwards data packets back and forth, translates between protocols and maps identities.

Raspberry Pi BLE to Wi-Fi gateway

As a simple example we build a BLE to Wi-Fi gateway.

Devices are peripherals, the gateway is a BLE central.

The gateway is also either a HTTP client or a service.

It translates HTTP Web requests to BLE requests.

The Web request URL contains the BLE address.

Use cases

Our BLE to Wi-Fi gateway supports these use cases:

Discovery — list the available BLE device addresses*.

Remote sensing — get sensor values from a device.

Remote control — write target values to a device.

*The result of a BLE scan or a preconfigured list.

Discovery

.png

Device ← Gateway (Scan) ← … ← Client (GET)

Client can be local or remote, via backend, e.g.

```
$ curl -v https://LOCAL_IP/devices?uuid=…
{
  "devices": [
    {"bt_addr":"2c-41-a1-14-2e-b1"},
    {"bt_addr":"d7-76-54-22-b4-b1"}
  ]
}
```

Remote sensing

.png

Device ← Gateway (Read) ← ... ← Client (GET)

Client can be local or remote, via backend, e.g.

```
$ curl -v https://LOCAL_IP/devices\
/d7-76-54-22-b4-b1/0x180d/0x2a37/value
{
  "value": 180
}
```

Remote sensing 2.png, 3.png

Gateway is *polling* devices, *pushing* data to backend:

Device ← Gateway (Read, POST) → Backend

Or, device is *pushing* and gateway is *pushing* again:

Device (Notify) → Gateway (POST) → Backend

Remote control .png

Device ← Gateway (Write) ← … ← Client (PUT)

Client can be local or remote, via backend.

```
$ curl -vX PUT https://LOCAL_IP/devices\
/d7-76-54-22-b4-b1/0x180d/0x2a37/value \
--data '{"value": … }'
```

Implementing the use cases

How to implement the above use cases on the Pi?

We'll need a BLE central to scan, read, write, notify.

As well as Web client and Web service functionality.

And the gateway should start up when plugged in.

Let's look at these building blocks, in Node.js.

Node.js

Install Node.js, a runtime for server-side JavaScript:

```
$ wget https://unofficial-builds.nodejs.org\
/download/release/v14.18.1\
/node-v14.18.1-linux-armv6l.tar.gz
$ tar -xzf node-v14.18.1-linux-armv6l.tar.gz
$ cd node-v14.18.1-linux-armv6l
$ sudo cp -R * /usr/local
$ node -v
```

New to JavaScript? Read Eloquent JavaScript.

Node.js BLE with *noble*

Install noble, a Node.js library to build a BLE central:

```
$ sudo apt-get update
$ sudo apt-get install bluetooth bluez \
libbluetooth-dev libudev-dev
$ npm install @abandonware/noble # in local dir
```

To use BLE from the command line, use *bluetoothctl*:

```
$ sudo bluetoothctl
[bluetooth]# scan on | scan off | help | quit
```

Node.js BLE scan .js

Scan for BLE devices advertising e.g. a HRM service:

```
const noble = require("@abandonware/noble");

noble.on("discover", function(peripheral) {
  console.log("found:", peripheral);
});

noble.startScanning(["180d"], true); // HRM
```

Node.js BLE read

.js

```
p.connect((err) => { // peripheral connected
  p.discoverServices(["180d"]], (err, svs) => {
    svs[0].disc…Cha…(["2a37"]], (err, chs) => {
      chs[0].read((error, data) => {
        const value = data.readUInt8(0);
      });
    });
  });
});
```

Node.js BLE write

.js

```
p.connect((err) => { // peripheral connected
  p.discoverServices(["180d"]], (err, svs) => {
    svs[0].disc…Cha…(["2A39"]], (err, chs) => {
      const data = new Buffer(1);
      data.writeUInt8(value, 0);
      chs[0].write(data, noRes, (err) => {…});
    }); // noRes = write without response
  });
});
```

Node.js BLE notify

.js

```
p.connect((err) => { // peripheral connected
  p.discoverServices(["180d"]], (err, svs) => {
    svs[0].disc…Cha…(["2a37"]], (err, chs) => {
      chs[0].subscribe((error, data) => {…});
      chs[0].on("data", (data, isNoti) => {
        const value = data.readUInt8(0); });
    });
  });
});
```

Hands-on, 20’: Bluetooth LE

Run the previous Bluetooth LE examples on the Pi.

Make sure *node*, *npm* and *noble** are all installed.

Use the *.js* link on each page or check the main repo.

To run a Node.js program *my.js*, type `$ node my.js`

Use the nRF5280 HRM BLE Peripheral for testing.

*Install *noble* locally, in `~/fhnw-iot/06/Nodejs`

Node.js Web client

.js

```
const http = require("http");

http.get("http://tmb.gr/hello.json", (rsp) => {
  let data = "";
  rsp.on("data", (chunk) => { data += chunk; });
  rsp.on("end", () => { console.log(data); });
}).on("error", (err) => {
  console.log(err.message);
});
```

Node.js secure Web client .js

```
const https = require("https"),
  qs = require("querystring"); // for POST body

let reqData = qs.stringify({ "value": 42 });
let options = { hostname: "postb.in", path:
  "/MY_POSTBIN_ID", method: "POST", headers: {
    "Content-Type": …, "Content-Length": … }};
let req = https.request(options, (res) => { … });
req.write(reqData); // write request body
req.end(); // sends the request
```

Node.js Web service

.js

```
const http = require("http");
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader("Content-Type", "text/plain");
  res.end("It works!\n");
});
server.listen(8080, "0.0.0.0", () => {
  console.log("Server running ...");
});
```

Node.js secure Web service

.js

```
const fs = require("fs"),
  https = require("https");

const options = {
  key: fs.readFileSync("./key.pem"),
  cert: fs.readFileSync("./cert.pem"),
}
const server = https.createServer(
  options, (req, res) => {…});
server.listen(4443, "0.0.0.0", () => {…});
```

Hands-on, 20’: Web client & service

Run the previous Web client and service examples.

Use the *.js* link on each page or check the main repo.

To display the IP address on the Pi, type: `$ ifconfig`

To run a Node.js program *my.js*, type: `$ node my.js`

Then access `http://IP:8080/` or `https://IP:4443/`

Creating a *systemd* service

```
$ sudo wget -O /lib/systemd/system/my.service \
https://raw.githubusercontent.com/tamberg/ \
fhnw-iot/master/06/Bash/my.service
```

Edit *my.service* to run your Node.js command line:

```
$ sudo nano /lib/systemd/system/my.service
…
WorkingDirectory=/home/pi/fhnw-iot/06/Nodejs/
ExecStart=/usr/bin/node my.js
…
```

Using the service with *systemctl*

To start/stop/remove the systemd service, type:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable my.service
$ sudo systemctl start my.service

$ sudo systemctl stop my.service

$ sudo rm /etc/systemd/system/multi-user.\
target.wants/my.service
$ sudo rm /lib/systemd/system/my.service
```

Hands-on, 5’: Create a systemd service

Create a systemd service as shown on previous slides.

Instead of *my.js* use one of the Web server examples.

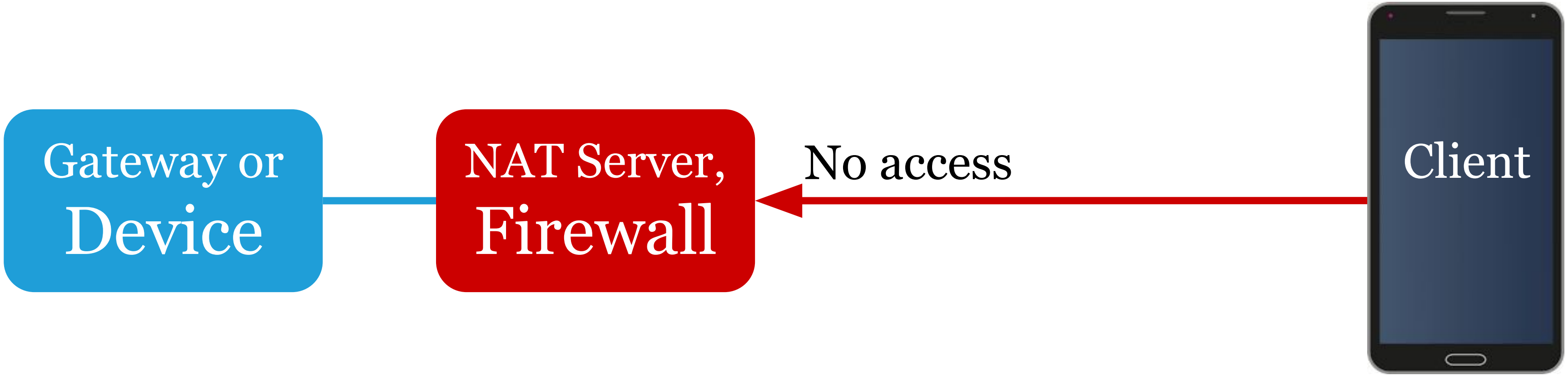
Reboot the Raspberry Pi device with `$ sudo reboot`

Make sure the Web service still runs after the reboot.

Remote access challenges

Devices behind a firewall or NAT are not accessible.

They usually have no public or no static IP address.

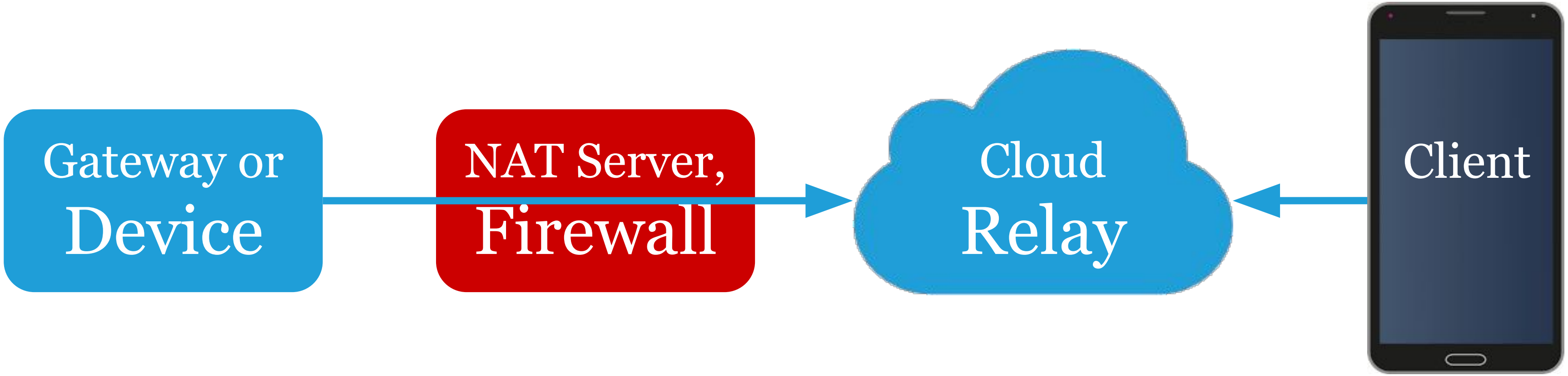


Opening incoming ports is not recommended.

Remote access via relay service

Relay services provide a public endpoint for access.

Based on an outgoing TCP connection to the relay.



E.g. Ngrok, Pagekite or Yaler (I'm a founder).

Why not just use VPN?

VPN extends the "local" network to clients — but also:

One compromised device can expose many devices.

VPN requires substantial resources on the device.

Managing VPNs can be a challenge at scale.

See Is VPN a false friend? by @clemensv.

Hands-on, 10’: Remote access

Install a Ngrok, Pagekite or Yaler relay service dæmon.

Configure it to publish the secure Node.js Web service.

Submit the URL to access your Web service via Teams.

Summary

We used the Raspberry Pi as a BLE to Wi-Fi gateway.

Use-cases are discovery, remote sensing and control.

We looked at architectural patterns & involved roles.

Clients push or pull, services accept or provide data.

We saw how to install a service & access it via relay.

Next: Messaging Protocols and Data Formats.

Challenge: Putting it all together

Choose one of the BLE to Wi-Fi gateway use cases.

Implement it, combining the above building blocks.

If the Pi is a Web Server, expose it via a relay service.

Or*, if the Pi is a client, send data to an IoT platform.

*Depending on the use case you chose.

Project: Join a team and sketch ideas

To join a 3 person team, add yourself to the sheet*.

Sketch three ideas** that your team could work on.
We will review ideas, to prevent too much overlap.
Consider this list of various sensors and actuators.

**Past ideas: Connected Plant, Air Quality Monitor, Intrusion Alert, Smart Washer, IoT Cat Feeder, etc.

*The sheet will be published in Teams during class.

Feedback or questions?

Join us on MSE TSM MobCom in MS Teams

Or email thomas.amberg@fhnw.ch

Thanks for your time.