

# IoT Data Collection (idb)

## Sending Sensor Data to IoT Platforms

CC BY-SA, T. Amberg & J. Luthiger, FHNW  
(Unless noted otherwise)

Slides: [tmb.gr/idb-sd](https://tmb.gr/idb-sd)

# Overview

These slides introduce *Wi-Fi connectivity*.

How to connect to a Wi-Fi network.

How to send data to a server.

# Prerequisites

Follow the steps for [Installing the Mu editor](#).

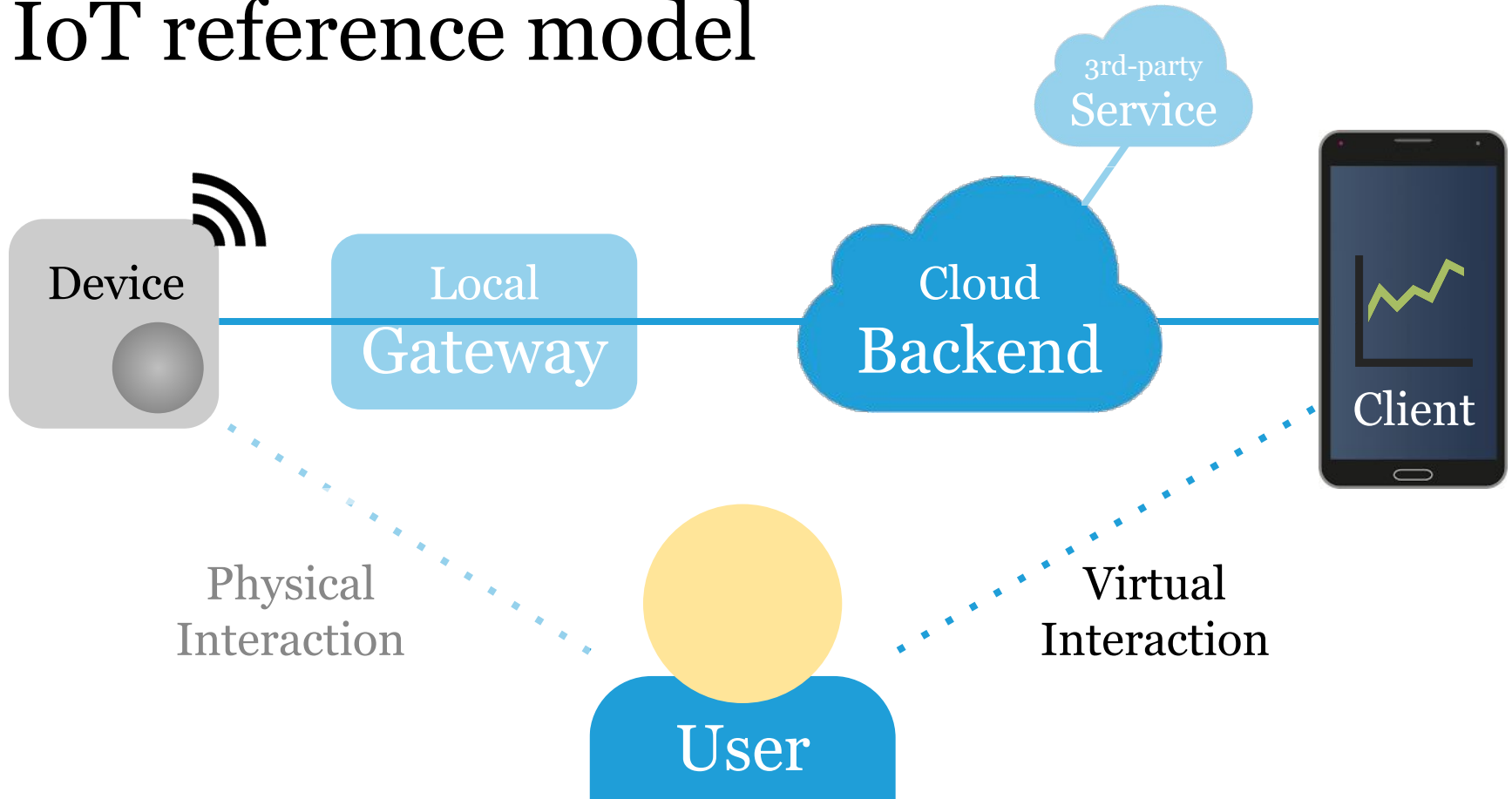
Then [Set up the Feather nRF52840 Express](#).

Add a [FeatherWing ESP32 AirLift](#) module.

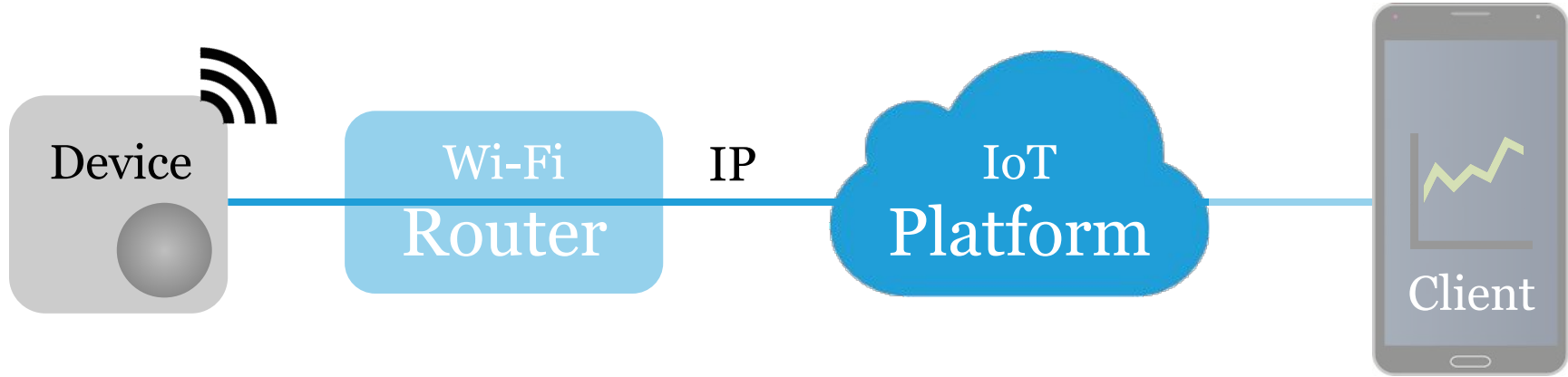
We'll use [CircuitPython](#) on the nRF52840.

You'll need Wi-Fi access, without a portal.

# IoT reference model



# Wi-Fi connectivity



# Wi-Fi

Wi-Fi is based on IEEE 802.11/a/b/g/n/... standards.

Uses 2.4 GHz UHF and 5 GHz SHF ISM radio bands.

100 m line-of-sight, many materials absorb/reflect it<sup>1</sup>.

Throughput depends on version, 11 Mbps up to Gbps.

Uses more energy than Bluetooth LE, less than 3/4G.

# Wi-Fi setup

```
import ... # see wifi\_connect.py

wifi = esp32spi.ESP_SPIcontrol(spi, cs, ...)
while not wifi.is_connected:
    try:
        wifi.connect_AP("MY_SSID", "MY_PASS")
    except RuntimeError as e:
        print("Cannot connect", e)
        continue
print(wifi.pretty_ip(wifi.ip_address))
```

# MAC address

The **MAC address**, e.g. 80:7d:3a:58:8a:ef, is a unique identifier assigned to the network interface controller (NIC) for data link layer communications.

Used as a network address for IEEE 802 technology including Ethernet, Wi-Fi and Bluetooth.

The first six digits **identify the vendor**, e.g. 80:7d:3a.



# Wi-Fi MAC address

This code reads the ESP32 Wi-Fi MAC address:

```
import ... # see wifi\_address.py

spi = busio.SPI(board.SCK, ...MOSI, ...MISO)
wifi = adafruit_esp32spi.ESP_SPIcontrol(...)

print([hex(i) for i in wifi.MAC_address])
```

Some networks grant access based on MAC address.

# Hands-on, 10': Wi-Fi

Build and run the previous Wi-Fi related examples.

Use the *.py* link on each page to find the source.

The examples are in the course repository.

# HTTP request

A simple way to put or get data to/from a backend.

To debug HTTP, the **cURL** client is recommended.

```
$ curl -v tmb.gr/hello.json
```

```
> GET /hello.json HTTP/1.1\r\n
```

```
> Host: tmb.gr\r\n
```

```
> \r\n
```

# HTTP response

```
< HTTP/1.1 200 OK\r\n
< Content-Type: application/json\r\n
< Content-Length: 32\r\n
< \r\n
{\n
  "message": "Hello, World!"\n
}
```

# HTTP client, GET

```
import ... # see http\_get\_client.py
try:
    response = requests.get(
        "http://tmb.gr/hello.json")
    print(response.status_code)
    print(response.json())
    response.close()
except RuntimeError as re: print(re)
except ValueError as ve: print(ve)
```

# HTTP client, POST

```
import ... # see http\_post\_client.py
try:
    # application/x-www-form-urlencoded
    request_data = "temp=23.0&humi=42.0"
    response = requests.post(
        "https://postb.in/b/POSTBIN_ID",
        data=request_data)
    print(response.status_code)
except RuntimeError as e: print(e)
```

# HTTP client, POST data in JSON format

```
import ... # see http\_post\_client\_json.py

request_url = "https://postb.in/b/POSTBIN_ID"
request_headers =
    { "Content-Type: application/json" }
request_data_json =
    "{ \"temp\": 23.0, \"humidity\": 42.0 }"
response = requests.post(request_url,
    data=request_data_json,
    headers=request_headers)
```

# Hands-on, 10': HTTP

Build and run the previous HTTP related examples.

Use the *.py* link on each page to find the source.

The examples are in the course repository.



# Sending sensor data

Here is a simple recipe for "remote sensing".

Repeat the following steps in a loop:

- Ensure WiFi is connected
- Read values from sensors
- Add a timestamp (UTC)
- Send data to backend

# Transport Layer Security

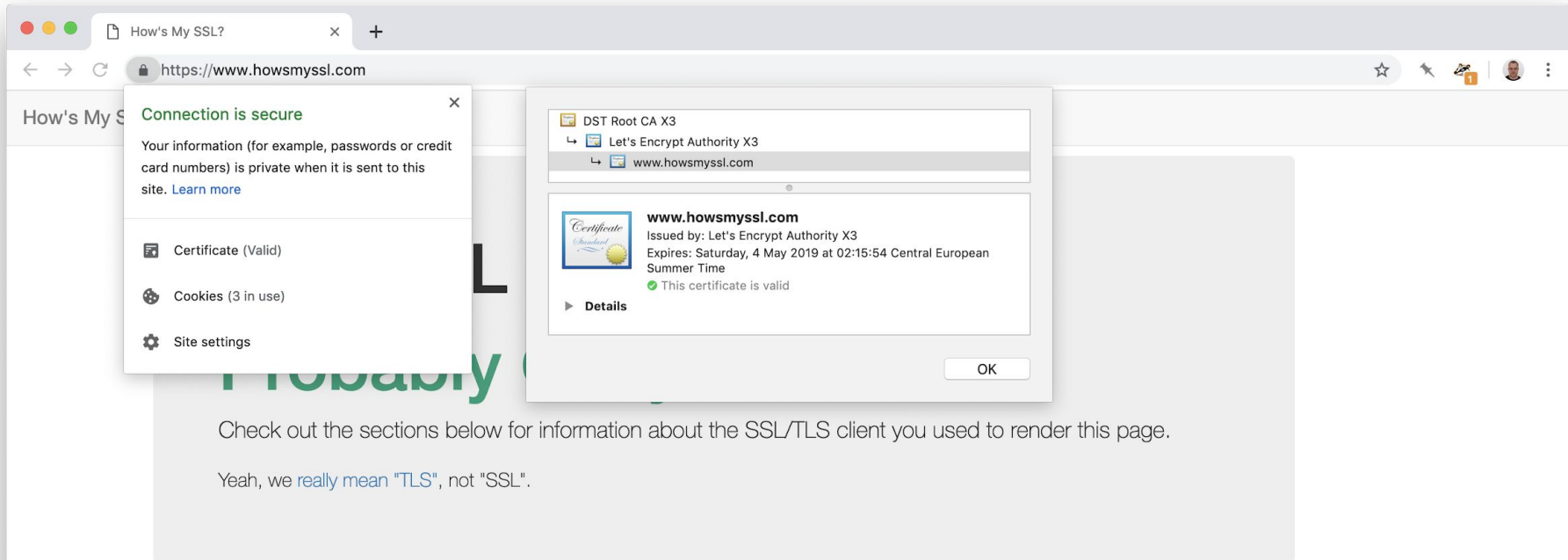
Transport Layer Security (**TLS**) allows a device to:

- Encrypt a communication channel, for privacy.
- Verify that it talks to the right backend server.

Trust is based on certificates issued by **authorities**.

HTTPS relies on TLS to secure HTTP connections.

See **this video** by @spiessa for an introduction.



## Version

**Good** Your client is using TLS 1.3, the most modern version of the encryption protocol. It gives you access to the fastest, most secure encryption possible on the web.

[Learn More](#)

## Ephemeral Key Support

**Good** Ephemeral keys are used in some of the cipher suites your client supports. This means your client may be used to provide [forward secrecy](#) if the server supports it. This greatly increases your protection against snoopers, including global passive adversaries who scoop up large amounts of encrypted traffic and store them until their attacks (or their computers) improve.

[Learn More](#)

## Session Ticket Support

**Good** Session tickets are supported in your client. Services you use will be able to scale out their TLS connections more easily with this feature.

[Learn More](#)

# Check CA certificate

TODO

# IoT platforms

IoT platforms enable storing/displaying sensor data.

There are many examples, we start with these two:

[Dweet.io](#) stores name/value pairs in JSON format.

[ThingSpeak](#) stores sensor data and displays graphs.

Both receive data through HTTP POST requests.

# Dweet.io

[Dweet.io](#) stores name/value pairs in JSON format.

Host: `dweet.io`

Port: `443`

POST `/dweet/for/THING_NAME?name=value`

POST `/dweet/for/THING_NAME?x=23&y=42&t=...`

GET `/get/dweets/for/THING_NAME`

See Wiki for [Dweet.io cURL examples](#).

# Hands-on, 15': Dweet.io

[Dweet.io](#) works without an account, data is public.

Use your Wi-Fi MAC address as `THING_NAME`.

Read the analog pin `A0`, then POST its value to  
`/dweet/for/THING_NAME?a0=value`

Use cURL or your browser to read stored data from  
[https://dweet.io/get/dweets/for/THING\\_NAME](https://dweet.io/get/dweets/for/THING_NAME)

# ThingSpeak

**ThingSpeak** stores sensor data and displays graphs.

Host: `api.thingspeak.com`

Port: 80 or 443

POST `/update?api_key=WRITE_API_KEY&field1=3`

GET `/channels/CHANNEL_ID/feed.json?`

`api_key=READ_API_KEY`

See Wiki for **ThingSpeak cURL examples**.



# Hands-on, 15': ThingSpeak

Get an [account](#) to create channels and get API keys.

Add the Arduino library with *Sketch > Include Library > Manage Libraries... > ThingSpeak > Install*

Try the example code *File > Examples > ThingSpeak > ESP8266 > WriteMultipleFields.ino*

Make sure values arrive in your ThingSpeak channel.

# Uniontown Weather Data

Channel ID: 3

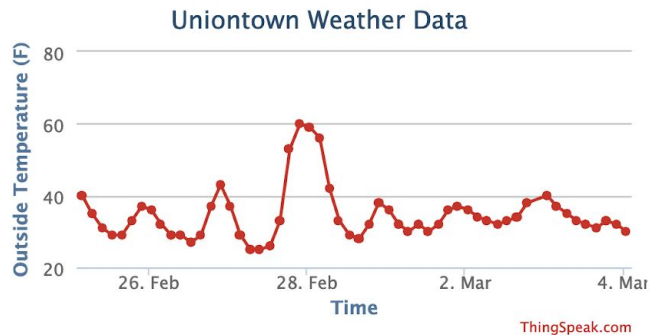
Author: [iothans](#)

Access: Public

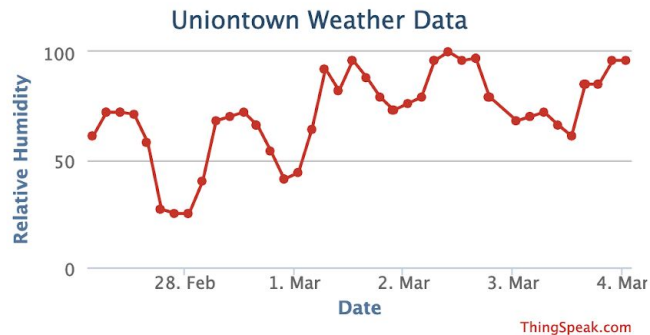
Weather data from Uniontown, PA

🔑 [temperature](#), [humidity](#), [weather station](#), [dew point](#), [channel\\_3](#)

Field 1 Chart



Field 2 Chart



# Timestamps

Adding a timestamp can happen in two places:

- On the backend, when a data packet just arrived.
- On the device, when a sensor value is measured.

The first requires sending immediately or discarding values, the second allows caching of measurements.

Trade-off: simplicity vs. accuracy & completeness.

# Time

The time on a microcontroller is reset to 0 at startup.

Timestamps use Coordinated Universal Time (**UTC**).

There are different ways to get and keep UTC time:

- Get time from a standard Web server, using HTTP.
- Get time from a network time server, using NTP.
- Set and keep time with a real time clock (RTC).

# ESP8266 Web-based time client

.ino

```
> HEAD / HTTP/1.1\r\n
> Host: google.com\r\n
> \r\n
< HTTP/1.1 301 Moved Permanently\r\n
< Location: http://www.google.com/\r\n
< Content-Type: text/html\r\n
< Date: Sat, 02 Mar 2019 17:10:20 GMT\r\n
< \r\n
```

# Network Time Protocol

Network Time Protocol (**NTP**) is a network protocol for clock synchronization between computer systems<sup>1</sup>.

Synchronizes participating computers to within a few milliseconds of Coordinated Universal Time (**UTC**).

Implementations send and receive timestamps using the User Datagram Protocol (**UDP**) on **port 123**.

# ESP8266 built-in NTP client

.ino

```
configTime(timezone * 3600, dst_offset,  
    "pool.ntp.org", "time.nist.gov");  
// wait for time() being adjusted  
while (time(NULL) < 28800 * 2) {  
    delay(500);  
}  
// time() is set  
time_t now = time(NULL);
```

# Hands-on, 15': ESP8266 NTP clients

Build, run and compare the following NTP clients:

The [Web-based time client](#) and [built-in NTP client](#).

[Arduino](#) > Examples > ESP8266WiFi > [NTPClient](#).

Bonus: Read the code of this [low memory version](#).

Which one would you use, and why?



# Hands-on, 15': Temperature sensor

Design a connected temperature sensor as specified:

Gets current time and date in **ISO 8601** UTC format.

Gets temperature & humidity from a **DHT11 sensor**.

Connects\* to `api.thingspeak.com` port 443 with TLS.

Posts sensor values, timestamp every 30 seconds.

\*And robustly reconnects, if disconnected.

# Summary

We learned to connect a device to a Wi-Fi network.

We sent sensor measurements to an IoT platform.

We looked at ways to get UTC time on a controller.

These are the basics of remote sensing.

Next: Internet Protocols, HTTP and CoAP.

# Feedback?

Find us on the [2Da](#), [3Da](#) or [idb](#) MS Teams

Or email [thomas.amberg@fhnw.ch](mailto:thomas.amberg@fhnw.ch)

and [juerg.luthiger@fhnw.ch](mailto:juerg.luthiger@fhnw.ch)