

# IoT Engineering

## 3: Sending Sensor Data to IoT Platforms

CC BY-SA, Thomas Amberg, FHNW  
(unless noted otherwise)  
Slides: [tmb.gr/iot-3](https://tmb.gr/iot-3)



### Overview

These slides introduce *Wi-Fi connectivity*.

How to connect to a Wi-Fi network.

How to send data to a server.

2

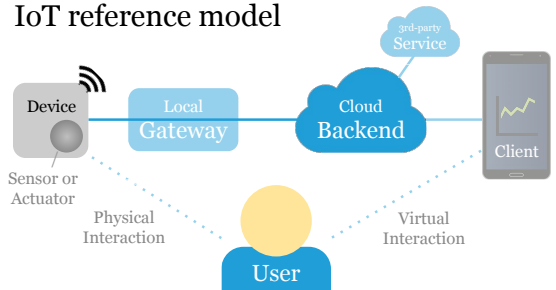
### Prerequisites

Install the Arduino IDE, set up ESP8266, get Wi-Fi:  
Check the Wiki entry on [Installing the Arduino IDE](#).  
[Set up the Feather Huzzah ESP8266](#) for Arduino.  
Get access to a Wi-Fi network\* without a portal.

\*In class, try MY\_SSID with MY\_PASSWORD.

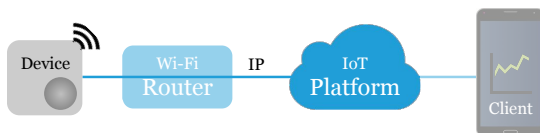
3

### IoT reference model



4

### Wi-Fi connectivity



The device first connects to the local Wi-Fi network, then it uses an **IP**-based protocol to transport data to a backend server, e.g. an existing IoT platform.

5

### Wi-Fi

Wi-Fi is based on [IEEE 802.11/a/b/g/n/...](#) standards.

Uses 2.4 GHz, 5 GHz & 6 GHz radio frequency 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 4/5G.

6

## ESP8266 Wi-Fi setup

.ino

```
#include <ESP8266WiFi.h>

void setup() {
  Serial.begin(115200); // for debug output
  WiFi.mode(WIFI_STA); // _AP|_AP_STA|_OFF
  WiFi.begin("MY_SSID", "MY_PASSWORD"); // TODO
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  Serial.println(WiFi.localIP());
}
```

7

## 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. 8

## ESP8266 Wi-Fi MAC address

.ino

This code reads the ESP8266 Wi-Fi MAC address:

```
#include <ESP8266WiFi.h>

void setup() {
  Serial.begin(115200);
  Serial.print(WiFi.macAddress());
}
```

Some networks grant access based on MAC address. 9

## HTTP Web request

HTTP is a TCP/IP-based protocol to transport data.

To test\* HTTP (Web) requests, we use the **cURL** tool.

```
$ curl -v tmb.gr/hello.json
> GET /hello.json HTTP/1.1\r\n
> Host: tmb.gr\r\n
> \r\n
```

\*On your computer, before writing any code.

10

## HTTP Web 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
}
```

11

## ESP8266 Wi-Fi client

.ino

```
WiFiClient client;
client.connect(host, port);
client.print(
  "GET /hello.json HTTP/1.1\r\n" \
  "Host: tmb.gr\r\n" \
  "\r\n");
while (client.connected() ||
  client.available()) {
  int ch = client.read(); ... }
```

12

## Hands-on, 15': Wi-Fi

Build and run the previous Wi-Fi related examples.

Use the `.ino` link on each page to find the source.

The examples are in the course repository.

Make sure to use the ESP8266 board.

13

## 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

14

## 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.

15

## ESP8266 secure Wi-Fi client

[.ino](#)

```
#include <ESP8266WiFi.h>

const char *host = "www.howsmyssl.com";
const char *path = "/a/check";
const int port = 443;

BearSSL::WiFiClientSecure client;
client.setInsecure(); // no cert validation
if (client.connect(host, port)) {
    // the connection is encrypted
}
```

16

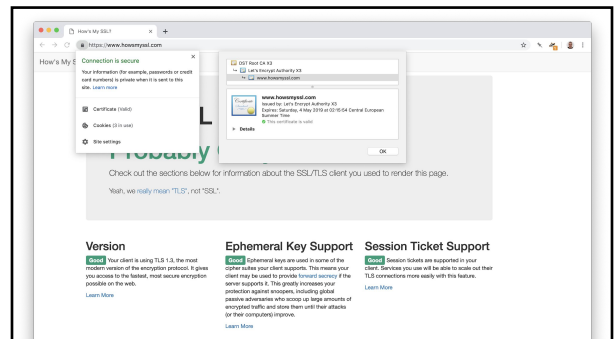
## ESP8266 verify host fingerprint

[.ino](#)

```
// Browser > 🔒 > Certificate > Fingerprint
const char *fingerprint = "CB 4D 0A 4F E5
0A DA 76 68 A4 33 37 BC 9A F9 C5 47 DD 5D
F6"; // SHA-1, not really secure anymore
```

```
BearSSL::WiFiClientSecure client;
client.setFingerprint(fingerprint);
if (client.connect(host, port)) {
    // certificate fingerprint matched
}
```

17



## ESP8266 check CA certificate [.ino](#)

```
#include "certs.h" // generated with cert.py
X509List cert(cert_ISRG_Root_X1);

// make sure time() is set, see NTP client
BearSSL::WiFiClientSecure client;
client.setTrustAnchors(&cert);
if (client.connect(host, port)) {
    // certificate chain verified
    ...
}
```

19

## Hands-on, 15': ESP8266 TLS clients

Build, run and compare the following TLS clients:

[Secure Wi-Fi client](#), with [fingerprint](#), with [CA check](#).

Locate/download the CA certificate in your browser.

Locate the SHA-1 fingerprint of the host certificate.

Bonus: Try to change the host to another Website.

20

## 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.

21

## 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](#).

22

## Hands-on, 15': Dweet.io

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

Use your ESP8266 MAC address as THING\_NAME.

On the ESP8266, read the analog pin Ao, 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)

23

## 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](#).

24

## Hands-on, 15': ThingSpeak

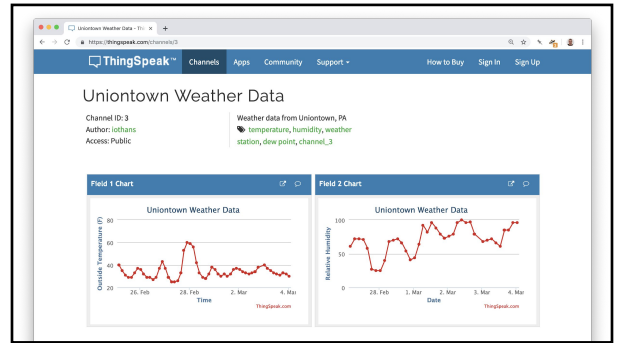
Get an [account](#) to create channels and get API keys.

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.

25



## 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.

27

## 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).

28

## 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
```

29

## 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](#).

30

## 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);
```

31

## 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?

32

## 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.

33

## 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.

34

## Challenge

Implement or finish the temp. sensor you designed.

Post the IoT platform data feed URL\* to the Slack.

Commit the Arduino code to the hands-on repo.

Measure the temperature for at least 24 hours.

\*Ideally public, we'll take a look together.

35

## Feedback or questions?

Write me on <https://fhnw-iot.slack.com/>

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

Thanks for your time.

36