# IoT Workshop

Trygve Laugstøl <trygvis@trygvis.io>

## What is IoT

### What is an IoT Device?

- Constrained in (one or more of):
    - Memory
    - CPU
    - Network bandwidth and/or latency
    - Storage
- Has connectivity

Might not have:

- RTC

Extra features:

- IR
- UART
- CAN

Sparkfun and Adafruit etc sell modules with all of these technologies.

### IoT Devices - Example chips

| Protocol | Chip | Specs |
| --- | --- | --- |
| Bluetooth 4/5 | nRF52x | 32-64 MHz, Cortex-M0/M4F, 24-256k RAM, 192-1024 k Flash, $1.88-$3.85 |
| WiFi | ESP8266/ESP32 | 80MHz-160MHz, 1-2 cores, ~80k RAM, < $1 - $4.53 |
| LoRa | Semtech | $3.23 - $4.74 |

BT and Wi-Fi has many, many more chips. Technologies based on open/accessible standards. LoRa is much more closed and driven by a single company.

4.53 is quantity=10

LoRa chips are just trancievers, an MCU with LoRa stack is required.

## ESP8266 Specifications

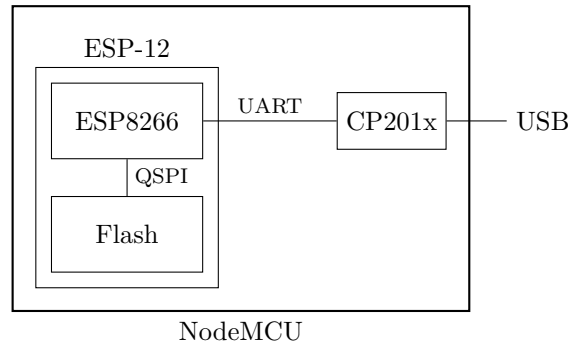| | |
|---|---|
| CPU | Tensilica Xtensa L106 |
| Frequency | 80MHz (160MHz possible |
| RAM | 32 kB instruction RAM 80 kB user RAM 16 kB system RAM |
| Flash | None, integrated SPI driver |
| Peripherals | 16 x GPIO I²C, SPI, I²S, UART 10 bit ADC Wi-Fi |

## ESP8266 Power usage

| State | Current usage |
|---|---|
| Off | 0.5 µA |
| Deep sleep with RTC | 20 µA |
| Light sleep (with Wi-Fi) | 1 mA |
| Sleep with peripherials | 15 mA |
| TX | 170 mA |

Datasheet page 18

# NodeMCU

## NodeMCU hardware

```
ESP-12
  ┌─────────────┐      UART   ┌────────┐
  │   ESP8266   │─────────────│ CP201x │──── USB
  └─────────────┘             └────────┘
       │ QSPI
  ┌─────────────┐
  │    Flash    │
  └─────────────┘
```

NodeMCU

# NodeMCU hardware



| | | | | | | |
|---|---|---|---|---|---|---|
| TOUT | ADC0 | A0 | D0 | GPIO16 | USER | WAKE |
| | RESERVED | RSV | D1 | GPIO5 | | |
| | RESERVED | RSV | D2 | GPIO4 | | |
| SDD3 | GPIO10 | SD3 | D3 | GPIO0 | FLASH | |
| SDD2 | GPIO9 | SD2 | D4 | GPIO2 | TXD1 | |
| SDD1 | MOSI | SD1 | 3V3 | 3.3V | | |
| SDCMD | CS | CMD | GND | GND | | |
| SDD0 | MISO | SD0 | D5 | GPIO14 | | HSCLK |
| SDCLK | SCLK | CLK | D6 | GPIO12 | | HMISO |
| | GND | GND | D7 | GPIO13 | RXD2 | HMOSI |
| | 3.3V | 3V3 | D8 | GPIO15 | TXD2 | HCS |
| | EN | EN | RX | GPIO3 | RXD0 | |
| | RST | RST | TX | GPIO1 | TXD0 | |
| | GND | GND | GND | GND | | |
| | Vin | Vin | 3V3 | 3.3V | | |

# ESP8266 software layers



| Arduino | Generic Arduino | Ethernet | ESP APIs |
|---|---|---|---|
| | ESP interface | | |
| | ESP SDK | | GCC libc |

| ESP8266 Hardware |
|---|
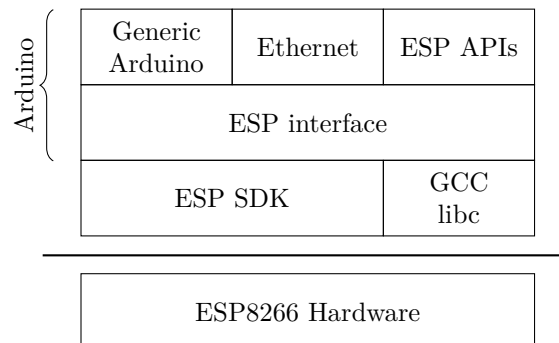
## ESP8266 + Arduino

- Standard Arduino IDE
- ESP8266 Arduino core
    - https://github.com/esp8266/Arduino

## Arduino IDE

```
  sketch_apr25a
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
```

Module), 80 MHz, 4M (1M SPIFFS), v2 Higher Bandwidth, Disabled, None, Only Sketch, 921600 on /dev/ttyUSB0
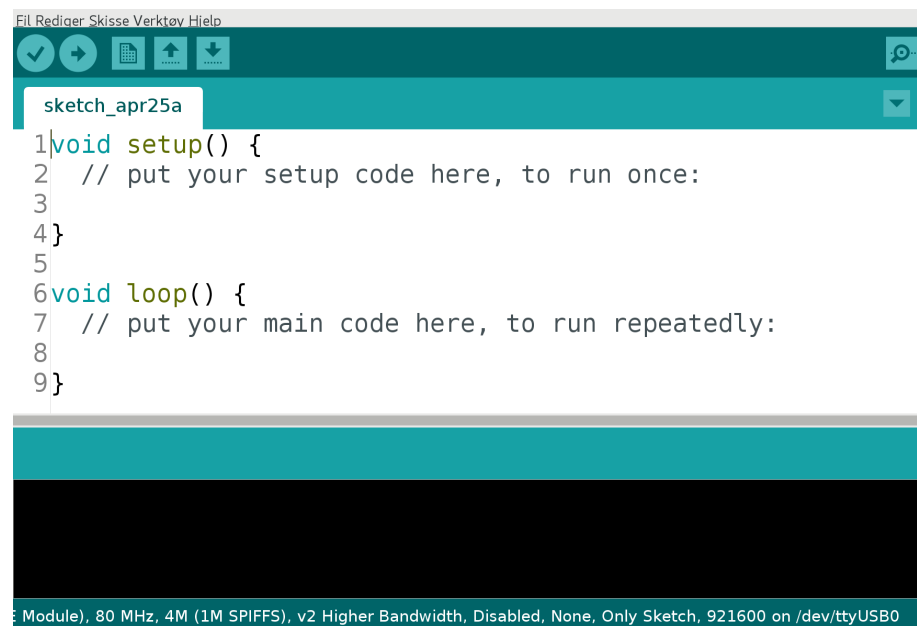
## Generic Arduino APIs

```
// Pin: D0, D1, etc.
// Mode: OUTPUT, INPUT, INPUT_PULLUP
// State: HIGH, LOW, 1/0

void pinMode(pin, mode);
void digitalWrite(pin, state);
int digitalRead(pin);

unsigned long now = millis();
unsigned long now = micros();
```

# Assignment: `blink-a-led`

# Lecture: MQTT

## MQTT

- *Message Queuing Telemetry Transport*
- Wikipedia: MQTT

MQTT is *the* standard for IoT applications (and lots of other useful stuff to). Using HTTP is just silly.

Supports SSL, and requires TCP.

Has UDP-like semantics with "fire and forget" but on a higher level (the message always have to be delivered and ACKed by the broker, not it's final recipient.

Version 3.1.1 er den som gjelder, V 3.1 er rar, de andre finnes ikke (før standardisering).

## Device and application architecture with MQTT
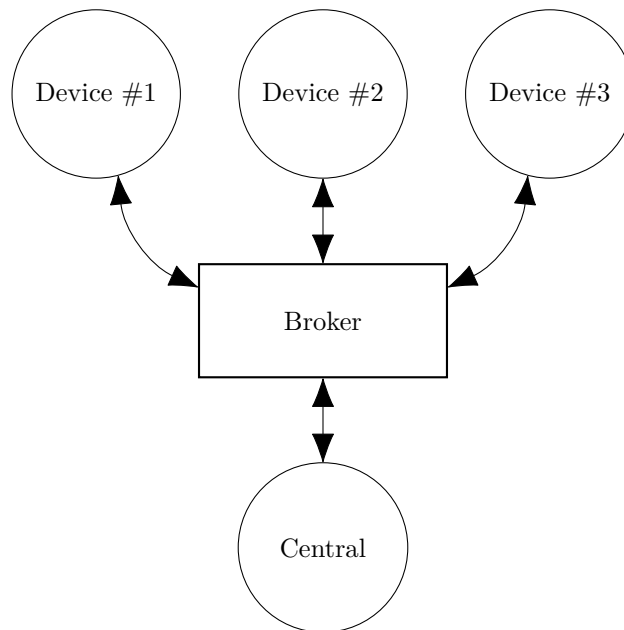
## MQTT Example

The temperature sensor:

- Publishes on:
  - `myapp/$device-id/temperature`
  - `myapp/$device-id/humidity`
  - `myapp/$device-id/altert`
- Subscribes to:
  - `myapp/$device-id/command`

The central application:

- Subscribes to:
  - `myapp/#/temperature`
  - `myapp/#/humidity`
- Publishes on:
  - `myapp/$device-id/command`

Typical first round of implementation.

Commands can be: * load new firmware (maybe an URL and firmware signature). * Set new calibration values * Change reading interval, altert levels (autonomous operation)

## MQTT - The protocol

Agents have one of two roles:

- *Client*
  - Publishes *messages*
  - Subscribes / unsubscribes to *topics*
  - Keep alive
- *Broker* (aka Server)
  - Handles network connections
  - Keeps subscriptions
  - Manages client
    * Timeouts and disconnects
    * *last will*
  - Persistence of *retained* messages

network connections: this includes removing closed sockets, client's that doesn't respons to timeouts and duplicate clients.

http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

Subscriptions are not permanent. The connection is (unlike HTTP) stateful.

Some messages may be persistent, but only one per topic. You will often end up with a "proper" mq on the backend if queuing is needed.

Push vs pull, central applications can push to clients

## MQTT - The protocol - MQTT Topic

- Topic name: `foo/bar/baz`
- Topic filter
  - `foo/bar/?`
  - `foo/#`

## ESP Arduino APIs

```
class {
    void restart();
    uint32_t getFreeHeap();
    uint32_t getChipId();


    ...
} ESP;

// Usage
ESP.restart();
```

## Connecting to a Wi-Fi

```
#include <ESP8266WiFi.h>

void setup() {
    WiFi.mode(WIFI_STA);
    WiFi.begin("NDC2018", NULL);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}
```

## MQTT on Arduino

PubSubClient is our MQTT client implementation.

Preparing to publish messages:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

WiFiClient wifiClient;
PubSubClient mqtt(wifiClient);

String deviceId = "esp-" + String(ESP.getChipId(), HEX);

void setup() {
    // ...
    mqtt.setServer("broker.hivemq.com", 1883);
}
```

## MQTT on Arduino

```
void loop()
{
    if (!mqtt.connected()) {
        reconnect();
    }
    else {
        mqtt.loop();
    }

    // Do work
}
```

## MQTT on Arduino

```
void reconnect()
{
    do {
        Serial.println("Connecting to MQTT");
        delay(1000);
    } while (!mqtt.connect(clientId.c_str()));

    Serial.println("Connected to MQTT server");
}
```

## MQTT on Arduino

```
void sendMessage()
{
    String topic = "ndc/" + deviceId + "/led";
    mqtt.publish(topic.c_str(), "1");
}
```

## MQTT on Arduino

Preparing for subscriptions:

```
void setup() {
    ...
    mqtt.setCallback(callback);
}

void callback(char* topic,
              byte* payload,
              unsigned int length) {
}

void reconnect() {
    ...
    // Subscribe to any topics you need
    mqtt.subscribe(topic_pattern);
}
```

## Assignment: `mqtt-with-button`