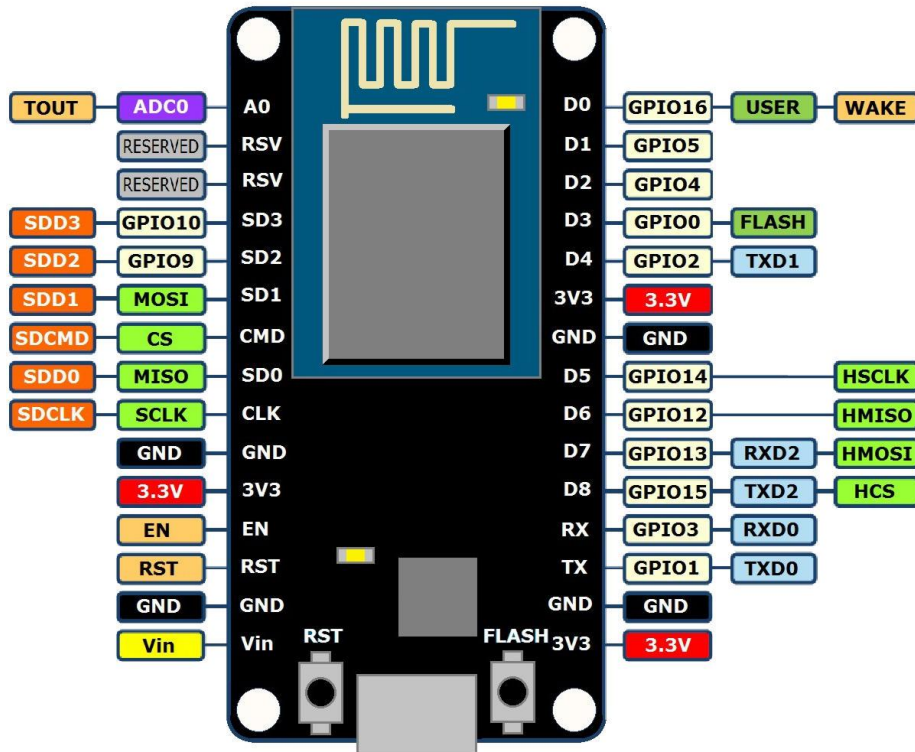


# IoT Workshop

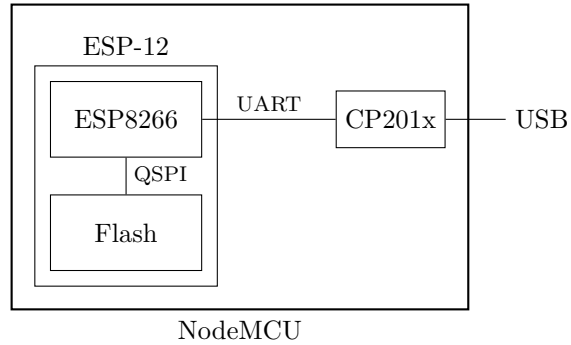
Trygve Laugstøl <trygvis@trygvis.io>

## NodeMCU

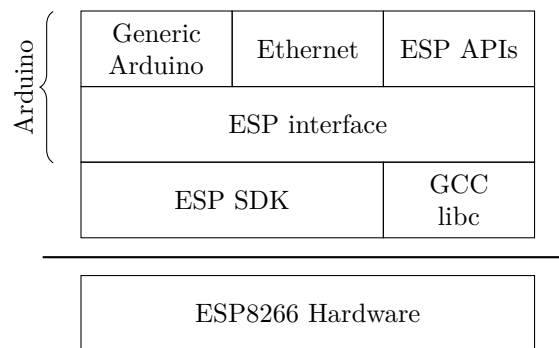
### NodeMCU hardware



## NodeMCU hardware



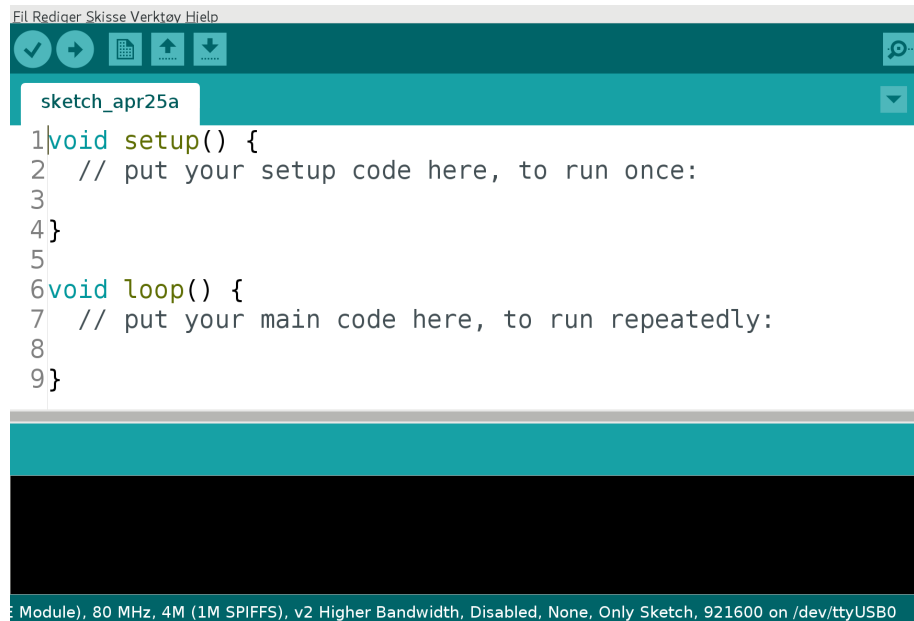
## ESP8266 software layers



## ESP8266 + Arduino

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

## Arduino IDE



```
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 }
```

## Arduino code structure

```
void setup() {
    // Called once
}

void loop() {
    // Called repeatedly
}
```

MCU programming is often structured into:

- Configure
  - CPU
  - GPIO ports
  - MCU's peripherals
  - The rest of the board
  - Configure application and callbacks.
- Sleep

Arduino chooses to run the cpu at 100% instead of the sleep step..

## Arduino file structure

```
foo/  
  foo.ino  
  config.h
```

foo.ino must always be in a foo directory.

config.h is created by “new tab”.

## Generic Arduino APIs

```
// Pin: D0, D1, etc.  
// Mode: OUTPUT, INPUT, INPUT_PULLUP  
void pinMode(uint8_t pin, uint8_t mode);  
  
// State: HIGH, LOW, true/false, 1/0  
void digitalWrite(uint8_t pin, uint8_t state);  
int digitalRead(uint8_t pin);  
  
unsigned long now millis();  
unsigned long now micros();
```

## ESP Arduino APIs

```
class {  
  void restart();  
  uint32_t getFreeHeap();  
  uint32_t getChipId();  
  
  ...  
} ESP;  
  
// Usage  
ESP.restart();
```

## ESP Arduino APIs

```
class {  
  String macAddress();  
  
  wl_status_t status();  
  int32_t RSSI();
```

```

IPAddress localIP();
IPAddress subnetMask();
IPAddress gatewayIP();
IPAddress dnsIP(uint8_t dns_no = 0);

...
} WiFi;

// Usage:

Serial.println(WiFi.localIP().toString());
http://arduino-esp8266.readthedocs.io/en/latest/libraries.html

```

## What is IoT

### What is IoT

- Not “a computer connected to the internet”
  - Then it is really just another computer connected to the internet
- Must be something else
  - It is simply devices that are resource constrained
    - \* Usually in more than one way
- Autonomous operation, the connection might not be permanent

### IoT is just a concept

- *The Internet of Things (IoT) is the network of physical devices, vehicles, home appliances and other items embedded with electronics, software, sensors, actuators, and connectivity which enables these objects to connect and exchange data.*<sup>1</sup>

### What is an IoT Device?

As for their definition.

What differentiates a computer from an IoT device?

### What is an IoT Device?

- Constrained in (one or more of):

---

<sup>1</sup>Wikipedia “Internet of Things”

- Memory
- CPU
- Network bandwidth and/or latency
- Storage
- Has connectivity
  - Bluetooth
  - Wi-Fi
  - NB-IoT
  - LTE Cat-M
  - LoRA
  - Proprietary radio

Might not have:

- RTC

Extra features:

- IR
- UART
- CAN

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

## IoT Devices - Bluetooth 4/5 chips

Chip	CPU	Freq	RAM	Flash	Price
nRF52810	Cortex-M4	64 MHz	24k	192k	\$1.88
nRF52832	Cortex-M4F		32k	256k	\$2.54
			64k	512k	\$2.59
nRF52840	Cortex-M4F		256k	1024k	\$3.85

- nRF52810: High performance, entry-level Bluetooth 4/ANT/2.4GHz SoC
- nRF52832: High performance Bluetooth 4/ANT/2.4GHz SoC
- nRF52840: Advanced multi-protocol System-on-Chip Supporting: Bluetooth 5, ANT/ANT+, 802.15.4 and 2.4GHz proprietary

All quantities are 1000 pieces

nRF51: <https://www.digikey.no/products/en/rf-if-and-rfid/rf-transceiver-ics/879?k=nrf51822>

nRF52832: these have different packagings, not only difference price

<https://www.digikey.no/products/en/rf-if-and-rfid/rf-transceiver-ics/879?FV=1c0001%2Cffe0036f&quantity=>

nRF52810: High performance, entry-level Bluetooth 4/ANT/2.4GHz SoC

nRF52832: High performance Bluetooth 4/ANT/2.4GHz SoC nRF52840: Ad-

vanced multi-protocol System-on-Chip Supporting: Bluetooth 5, ANT/ANT+, 802.15.4 and 2.4GHz proprietary

## IoT Devices - LoRA

### Modules

Module	Data Rate	Price
RN2483A-I/RM104		\$12.05 @ 250
CMWX1ZZABZ-078	SX1276	\$10.74 @ 1000
RF-LORA-868-SO	SX1272	\$16.55 @ 1000

### Chips

Chip	Price
SX1281	\$3.23
SX1272	\$4.25
SX1276	\$4.25
SX1279	\$4.74

These modules require an external MCU, so does the chips.

## IoT Devices - NB-IoT

Module	Price
uBlox SARA-N210	~\$10 @ 100
Sierra Wireless HL7800_1103933	\$15.72

## IoT Devices - Wi-Fi

Chip	CPU	Freq	ROM	RAM	Price
ESP8266	Tensilica L106	160 MHz	N/A	~50 kB	< \$1

ESP32 - dual cpu, Wi-Fi, Bluetooth 4 ESP32-D0WDQ6 2x Xtensa @ 160MHz  
\$ 4.53 @ 10

The ESP8266's RAM depends on which firmware stack is used. Physical is probably 128k or most likely 64k.

### ESP8266 details - Power usage

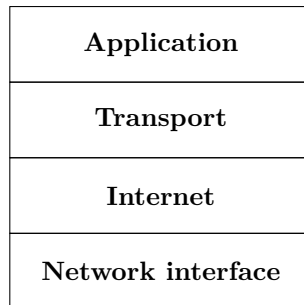
State	Current usage
Off	0.5 $\mu$ A
Deep sleep with RTC	20 $\mu$ A
Light sleep (with Wi-Fi)	1 mA
Sleep with peripherals	15 mA
TX	170 mA

Datasheet page 18

### Going back to basics

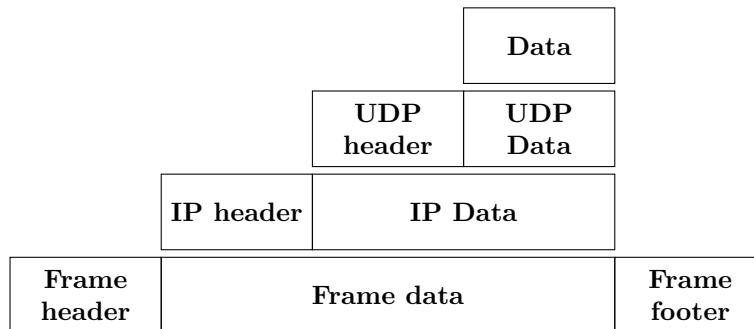
What is the internet again?

TCP/IP Layers





## Packet encapsulation



## Network interface

- Ethernet
  - 10BASE5, 10BASE2, 10BASE-T / 100BASE-TX / 1000BASE-TX
- Wi-Fi
  - 802.11a/b/g/n
- RS-232

Ethernet: Hubs and switches (that act on this level) is not on it's own layer. It is more of a implementation detail in the architecture diagram.

RS-232 signaling is used in *all* MCUs, many have several ports available. It is extremely flexible, both used for implementing applications and debugging. Frequently an easy way to hack embedded devices. “USB dongles”, “USB TTL” all use RS-232 signaling.

Note that this only applies to its logical signals, not voltage levels. The signaling does not specify any max data rate, very high rates ( $\geq 1\text{Mbps}$ ) is often supported.

## Internet

- IP
- ICMP

## Transport

- TCP
- UDP
- SCTP
- QUIC

## Layer 7: Application Layer

- HTTP
- DNS
- MQTT
- CoAP
- (everything else..)

## Details: IP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				Header length				TOS								Packet length															
4	32	Identification																X	D	M	Fragement offset												
8	64	TTL								Protocol								Header checksum															
12	96	Source ip address																															
16	128	Destination ip address																															
20	160	Options (variable length)																															

## Details: UDP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	source port																destination port															
4	32	length																checksum															

## Lecture: MQTT

### MQTT

- *Message Queuing Telemetry Transport*
- [Wikipedia: MQTT](#)

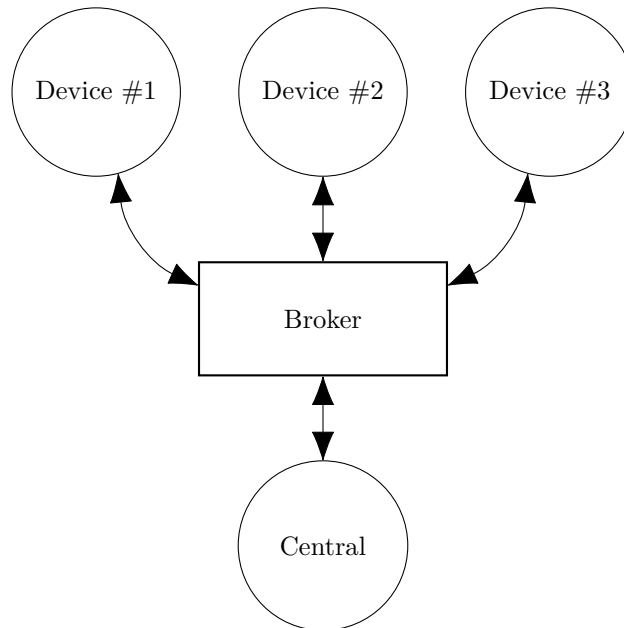
MQTT is *the* standard for IoT applications (and lots of other useful stuff to). Using HTTP is just silly.

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 - Implementations

- Mosquitto
- Eclipse Paho
- RabbitMQ
- ActiveMQ

RabbitMQ has a separate connector that must be installed Not sure about ActiveMQ but it is at least a part of the project so it is releases at the same time.

## MQTT Cloud Connectors

- Cloud
  - Amazon IoT
  - Google Cloud IoT
  - Microsoft Azure IoT
  - CloudMQTT (at Heroku)
- DIY
  - ThingMQ
  - HiveMQ

In between are:

- self hosted
- Generic bridges

## MQTT - The protocol

Agents have one of two roles:

- *Client*
  - Publishes *messages*
  - Subscribes / unsubscribes to *topics*
- *Broker* (aka Server)
  - Handles network connections
  - Keeps subscriptions
  - Manages client
    - \* Disconnects
    - \* *(last) will*
  - Persistence of retained messages

network connections: this includes removing closed sockets, client's that doesn't respond 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 Packet

- Size oriented
- Flags indicate type of remaining bytes
  - Packet type
  - Topic name
  - Payload

Only packet type + flags (1 byte) is required, everything else is optional.

The size field is variable length encoded, 0-127 bytes is 1 byte, 128-16383 use 2 bytes etc, up to 4 bytes for 256MB payload.

## MQTT Connect

- CONNECT

- clientId
- username
- password
- keepAlive
- Keep alive
  - PINGREQ
  - PINGRESP

## MQTT - The protocol - MQTT Topic

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

## MQTT - The protocol - Retained message

Message is kept by the server even after disconnect

- CONNECT
- PUBLISH
  - RETAIN
  - \$app/\$device/temperature
  - 22.3
- DISCONNECT

Later on:

- SUBSCRIBE
  - \$app/#/temperature
- PUBLISH
  - \$app/\$device/temperature
  - 22.3

The last PUBLISH is an incoming message

## MQTT - The protocol - Will message

Message sent when you disconnect

Client #1:

1. CONNECT
  - WILL TOPIC: \$app/\$device/online
  - WILL PAYLOAD: 0

2. PUBLISH
  - \$app/\$device/online
  - 1
3. DISCONNECT

Broker

1. *To all subscribers* PUBLISH
  - \$app/\$device/online
  - 0

## MQTT Topic

The temperature sensor:

- Publishes on:
  - myapp/\$device-id/temperature
  - myapp/\$device-id/humidity
  - myapp/\$device-id/alert
- 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, alert levels (autonomous operation)

## MQTT on Arduino

PubSubClient is our MQTT client implementation.

```
WiFiClient wifiClient;
PubSubClient mqtt(wifiClient);

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

void setup() {
```

```
    // Configure WiFi
    mqtt.setServer(mqtt_server, 1883);
    mqtt.setCallback(callback);
}
```

## MQTT on Arduino

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

void reconnect() {
    while (!mqtt.connect(client_id));

    mqtt.subscribe(topic_pattern);
}
```

This is blocking!

## Assignment

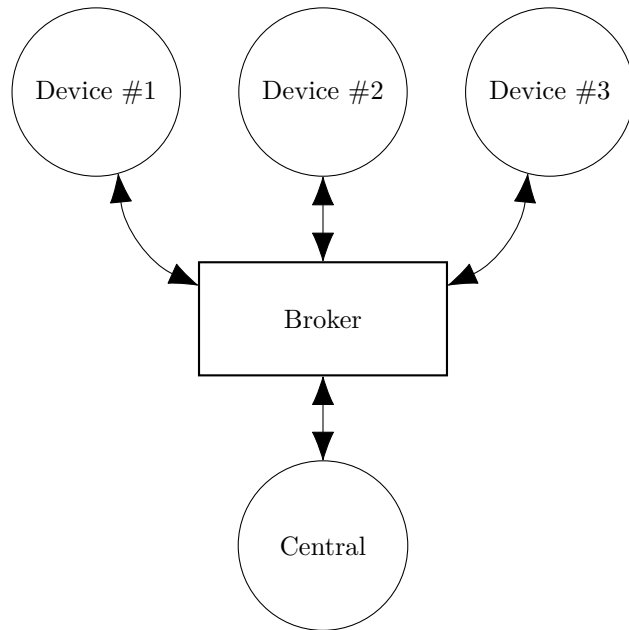
- mqtt

## MQTT topic architecture

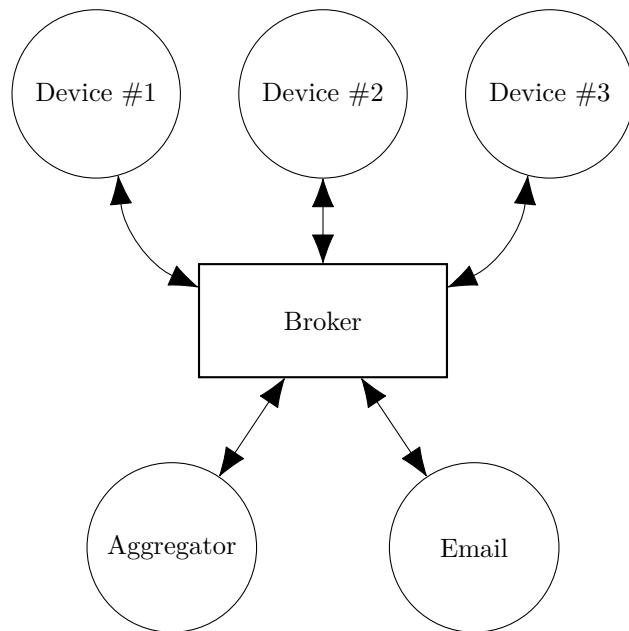
The central application is split:

- An aggregating agent:
  - myapp/#/temperature
  - myapp/#/humidity
- Emailing agent
  - myapp/\$device-id/alert
- Publishes on:
  - myapp/\$device-id/command

### MQTT topic architecture



### MQTT topic architecture





## MQTT - Patterns

- Combining MQTT and HTTP
- Using web sockets transport

## Assignment

- mqtt2

## Assignment

- mqtt3

discussion: how to connect these two devices?