Connectivity 2

Internet refresher …

TCP/IP

IP

Private & public addresses

Routing, NAT and Firewalls

Host names (DNS)

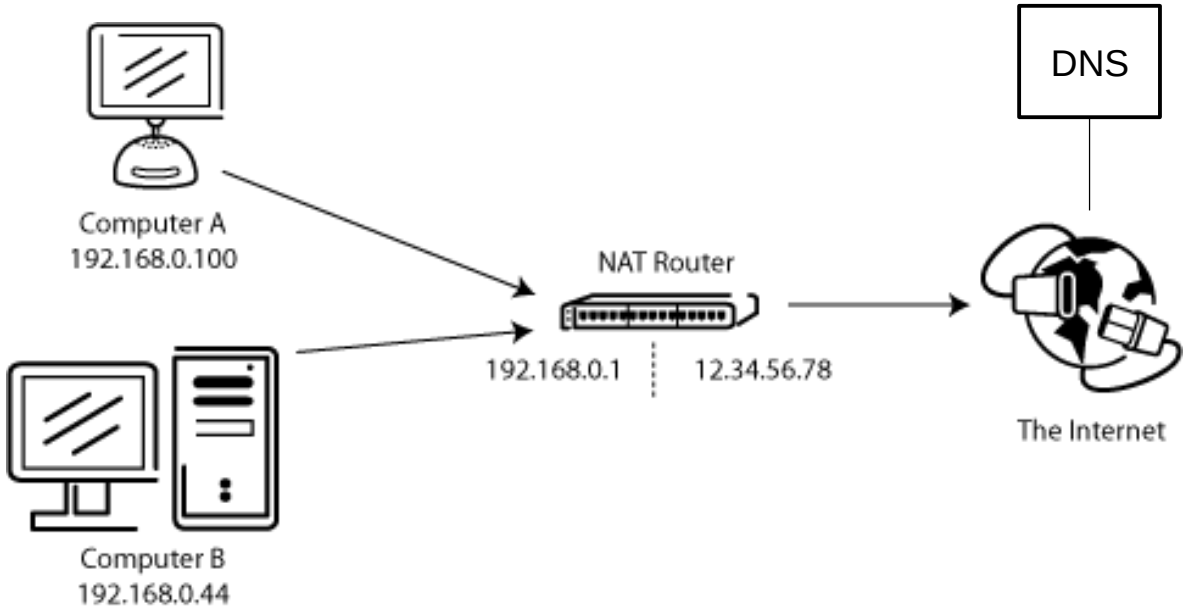
Load balancing

- DNS round robin
- Virtual IP

TCP & UDP

Ports (services)

Error control & ordering



HTTP Request

Method

GET, POST, PUT, DELETE …

Headers

Accept (content type, encoding ….)

Authorization

Cache-Control

Cookies

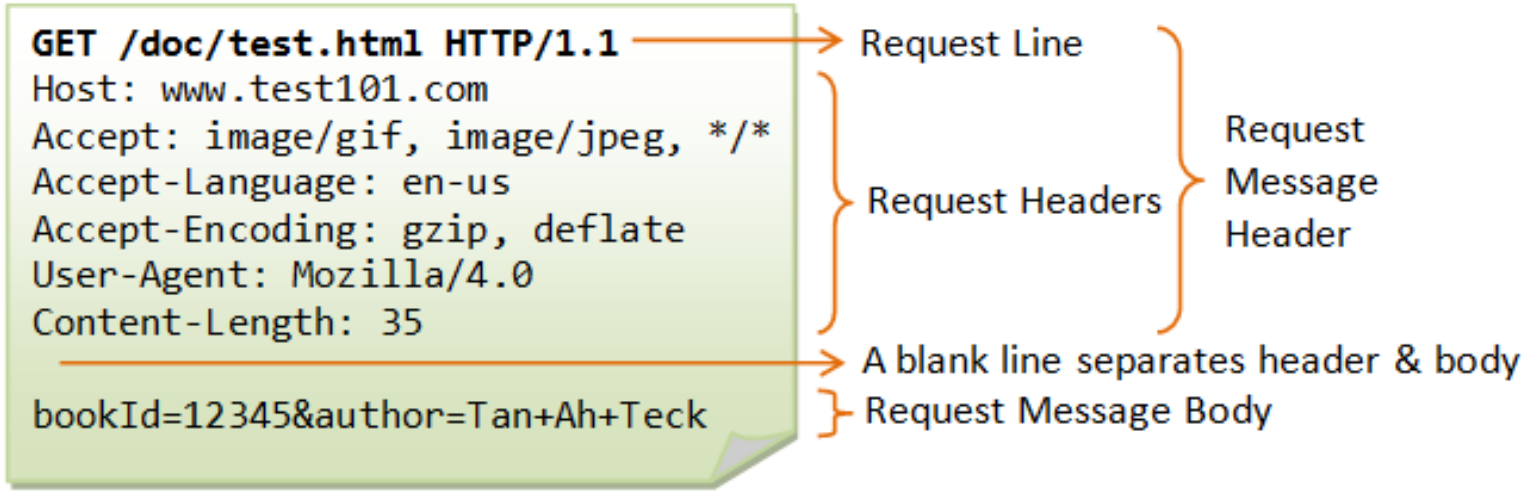
Content-Type

Host

Body

Application specific (e.g. JSON, XML …)

Usually POST and PUT methods only



HTTP Response

Status line

Protocol version

Status

Headers

Access-Control-Allow-Origin

Cache-Control

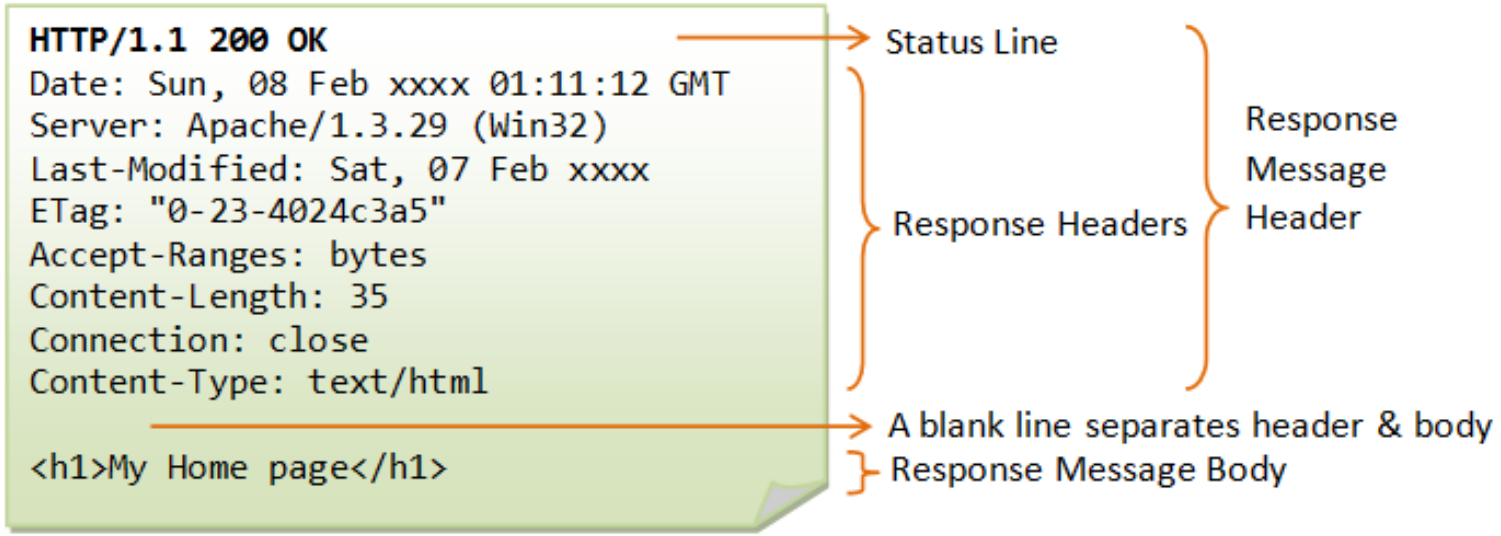
Content-Type

Set-Cookie

…

Body

Application specific (e.g. JSON …)



MQTT (ISO/IEC PRF 20922)

Overview

TCP/IP based

- MQTT-SN (UDP)

Small footprint / low bandwidth

e.g. compared to HTTP

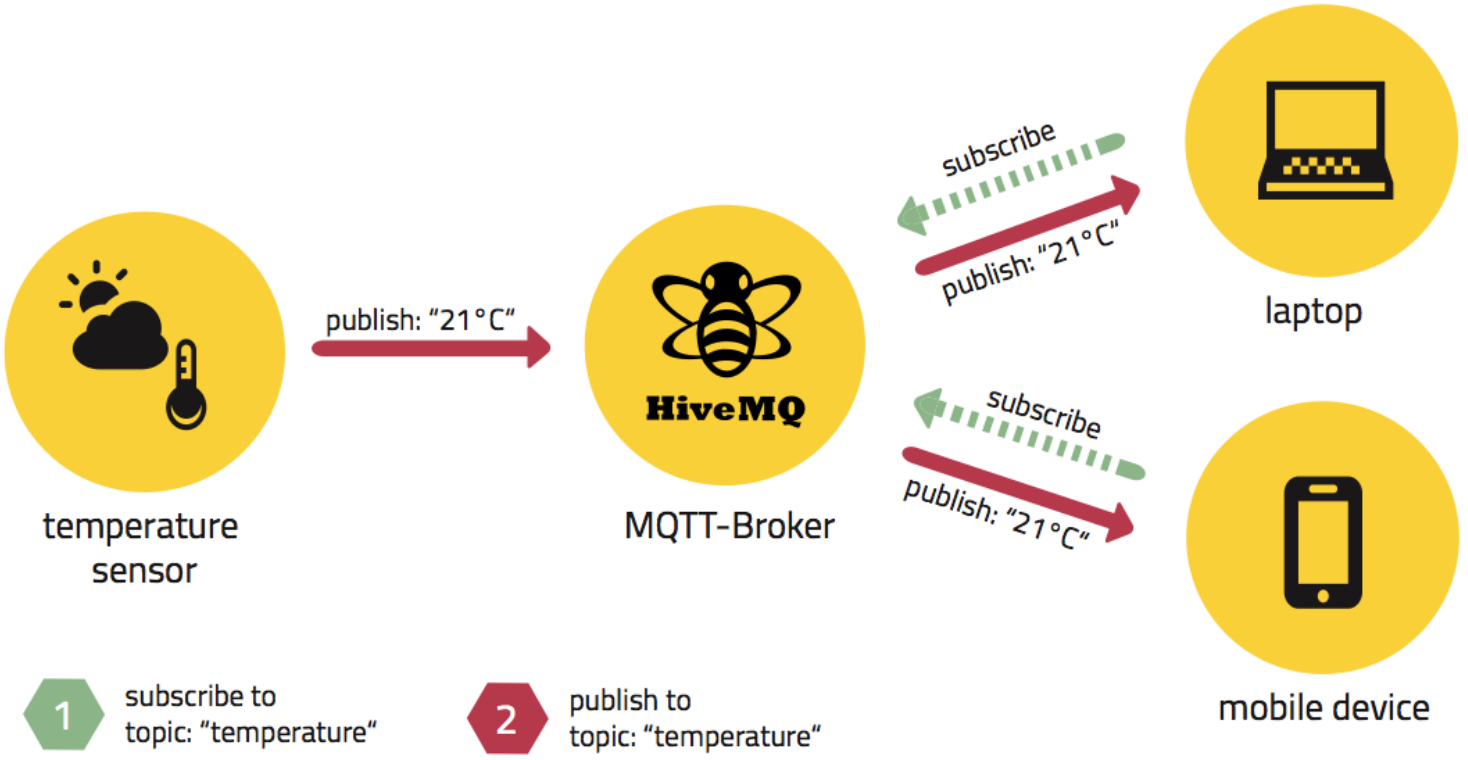
Pub/sub

Client connect to **broker**

And publish to **topics**

Or subscribe to **topics**

- Wildcard subscription



MQTT implementations

Servers

Open source: Eclipse Mosquitto, HiveMQ …

Cloud: AWS, Azure, GCloud …

Clients

ESP32: pubsubclient, Paho

Linux: Paho

…

MQTT QoS

QoS = Quality of service

Does not apply to TCP, only to the client-client connection

Three levels:

- 0: fire and forget (at most once)
- 1: resend until acknowledged (at least once)
- 2: exactly once delivery (exactly once)

WARNING

Cloud providers may deviate from the MQTT QoS specification

Practical tips

Which level to use with ESP32?

- Use 0 when you don’t care too much about lost messages
- Use 1 when you can afford duplicate messages (e.g. deduplicating on the server side)

What to do when the server is not accessible (e.g. connection refused, no route to host)?

- Buffer locally (preferably in-memory)

MQTT security

Encryption

TLS on top of TCP/IP

Authentication

Username / password

Exercises

WiFi modes

Station + HTTP client

ESP Access point + HTTP server -> fire an LED upon HTTP request

MQTT

ESP client to mosquitto communication