Modul
- Internet of Things (IoT) -

03-Vorlesung

Prof. Dr. Marcel Tilly

Fakultät Informatik, Cloud Computing

Überblick

21. März	Einführung in das Internet der Dinge
28. März	IoT Architekturen
4. April	Things und Sensoren
11. April	From Device to Cloud
18. April	Vorlesungsfrei – Ostern
25. April	IoT Analytics
02. Mai	Big Data in IoT
9. Mai	Data Exploration
16. Mai	IoT Platformen
23. Mai	Entwicklung einer IoT Lösung
30. Mai	Vorlesungsfrei; Christi Himmelfahrt
05. Juni	opt. Gastvortrag – Digitalisierung
13. Juni	Data Science in IoT
20. Juni	Vorlesungsfrei – Fronleichnam
27. Juni	Intelligente Cloud und intelligente Edge
04. Juli	PStA Abschlusspraesentationen

PStA

Connectivity

Technische Hochschule Rosenheim
Technical University of Applied Sciences

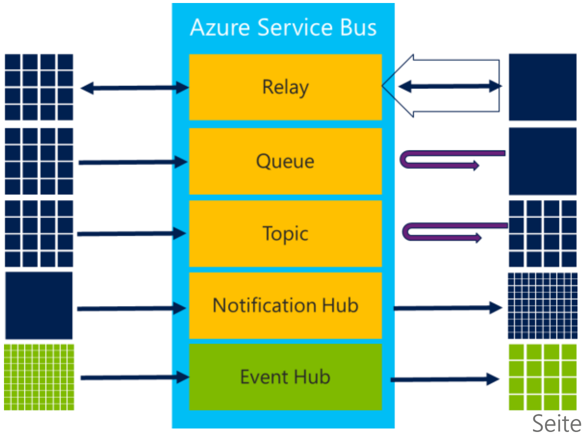
Challenges

Amount of connected things
Variety of protocols
Variety of communication pattern
Security
Variety of topologies
Discovery
Coverage
Partial connectivity
Stability
Bandwidth
Interference
Internationalisation

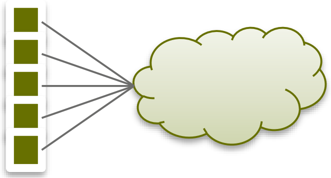
Technology

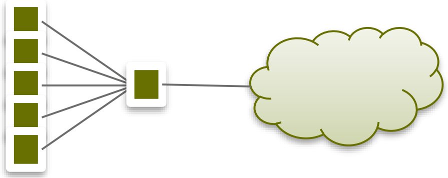
Device-/Configuration Management
HTTPs, MQTT, AMQP, CoAP
CAN, Modbus,
Bluetooth Low Energy (BLE), Zigbee
6LowPan
OPC-UA
LORA, MIOTY, …

Communication Pattern



Topology


directly connected


gateway connected

NON-IP based communication

Non-IP based communication is optimized for cost and energy usage

Samples:

- Bluetooth
- Zigbee
- Z-Wave

Bluetooth (www.bluetooth.org)

Mainly used in phones, peripherals (keyboards, mouse, ..)

Invented around 1994 at Ericsson with the intent to get rid of cables
Intel and Nokia joined with focus to wirelessly link cell phones

2010 Nordic Semiconductor and Nokia developed Ultra Low Power Bluetooth (Bluetooth Low Energy BLE)
-> Created a new market with devise powered by a cell battery

BT is comprised by two wireless technology systems: Basic Rate (BR) and Low Energy (BLE)

Frequency: 2.4GHz ISM band
Range: – 100m

Supports beaconing: Beacons uses BLE and advertises presences (without connecting)

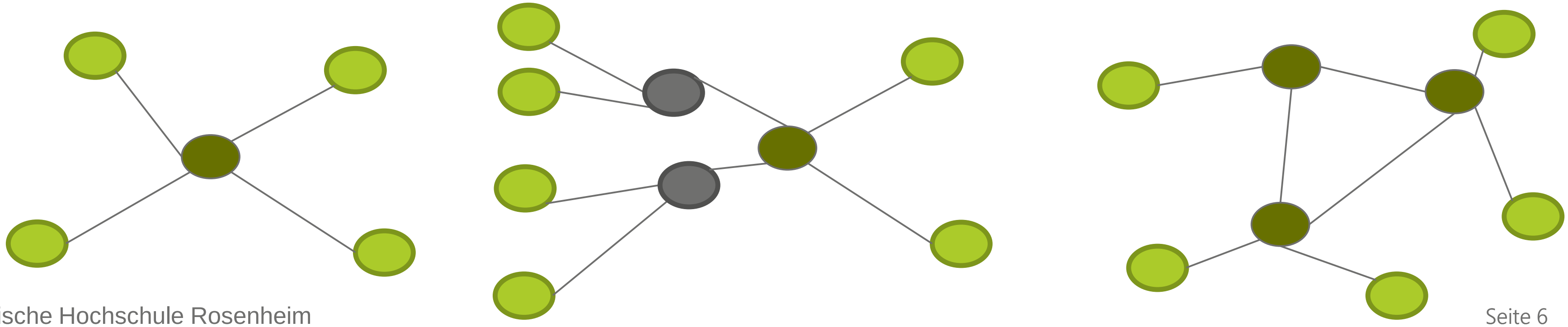
Zigbee

Based on IEEE 802.15.4 with target to enable IOT with constrained cost, power and space

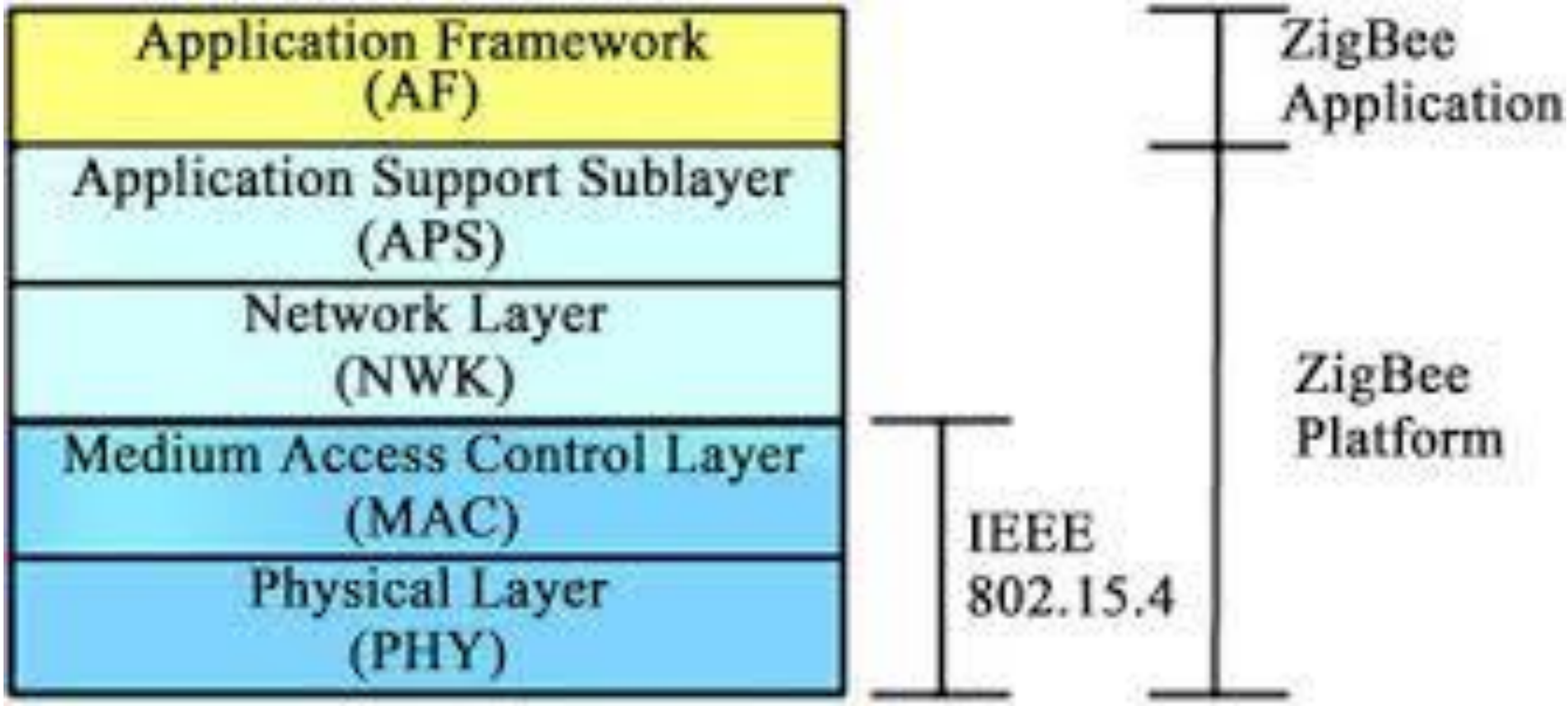
Zigbee alliance was formed 2002, protocol was available in 2004

3 main components:

1. Zigbee controller (ZC): Highly capable device that is used to form the network functions
2. Zigbee router (ZR): Handles load of mesh network hopping and routing coordination.
3. Zigbee end device (ZED): The is a simple endpoint such as a light switch or a thermostat. It contains enough functionality to communicate with the coordinator



Zigbee Stack



Application Framework (AF)
Application Support Sublayer (APS)
Network Layer (NWK)
Medium Access Control Layer (MAC)
Physical Layer (PHY)
IEEE 802.15.4
ZigBee Application
ZigBee Platform

Zigbee vs Bluetooth

Zigbee aims at automation whereas Bluetooth aims at connectivity of mobile devices in close proximity.

Zigbee uses low data rates, low power consumption on small packet devices while Bluetooth uses higher data rates, higher power consumption on large packet devices.

Zigbee networks support longer range devices and more in number compared to Bluetooth networks whose range is small.

Given Zigbee's almost instant network join times(30 milliseconds) its more suitable for critical applications while Bluetooth's longer join time is detrimental (3 seconds).

Z-Wave

For consumer and home automation and has 2100 products using it

Segment of lightning and HVAC control

Was created in 2001

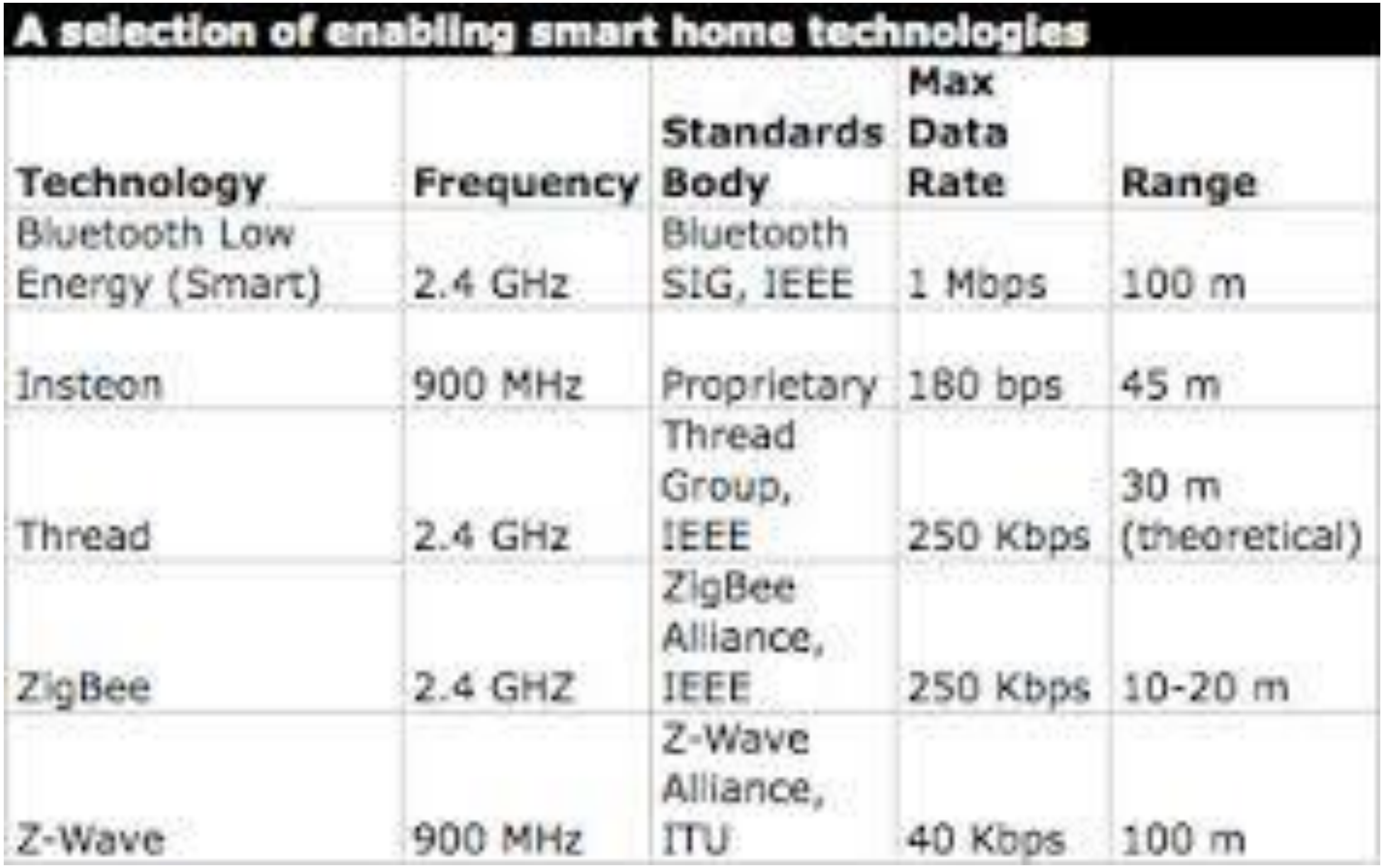
Z-Wave Alliance member are: SmartThings, Honeywell, Belkin, Bosch, Carrier, ADT and LG

Uses the 900 MHz band, It is a closed protocol!

Components:

1. Controller device: This is the top-level device and provides the routing table for mesh networks
2. Slave device/node: These devices perform actions based on commands they receive.

Overview NON-IP based Technologies



A selection of enabling smart home technologies

Technology	Frequency	Standards Body	Max Data Rate	Range
Bluetooth Low Energy (Smart)	2.4 GHz	Bluetooth SIG, IEEE	1 Mbps	100 m
Insteon	900 MHz	Proprietary	180 bps	45 m
Thread	2.4 GHz	Thread Group, IEEE	250 Kbps	30 m (theoretical)
ZigBee	2.4 GHZ	ZigBee Alliance, IEEE	250 Kbps	10-20 m
Z-Wave	900 MHz	Z-Wave Alliance, ITU	40 Kbps	100 m

Low Power Wide Area Network

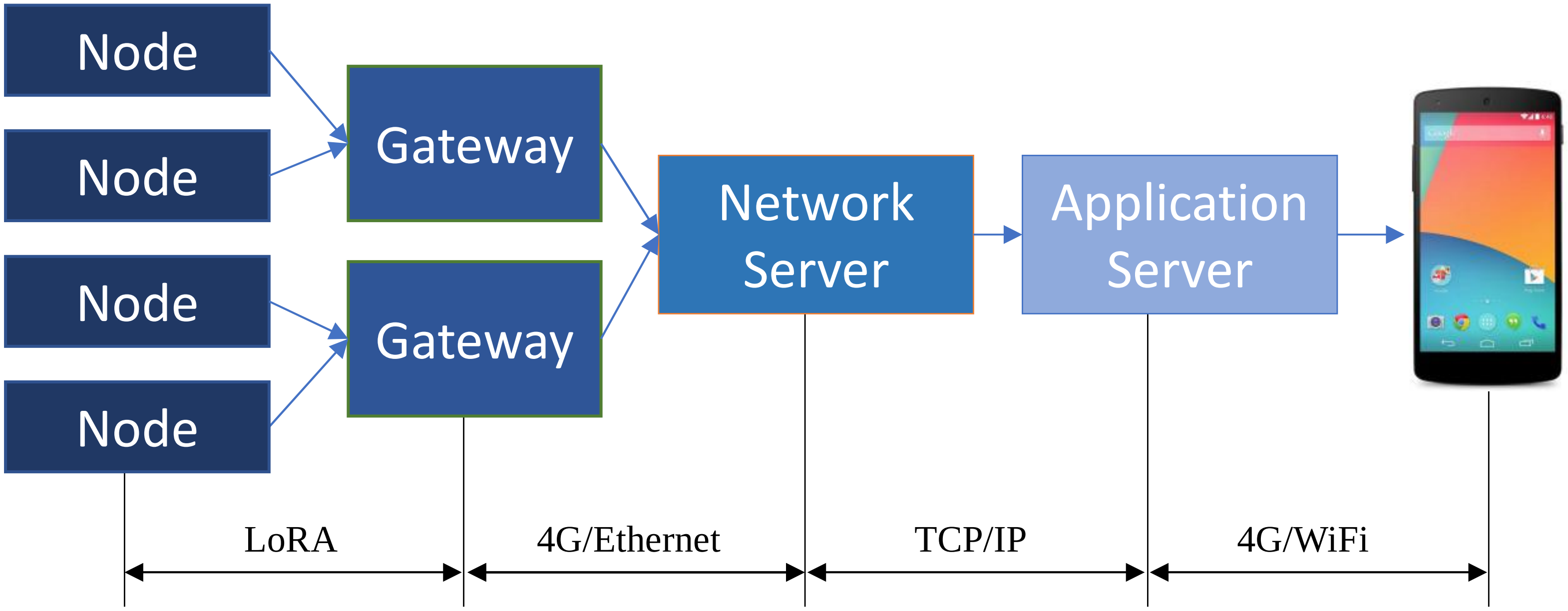
Requirements for LPWAN:

- Long Range (> km)
- Low Power
- Low Data Range

Different concepts:

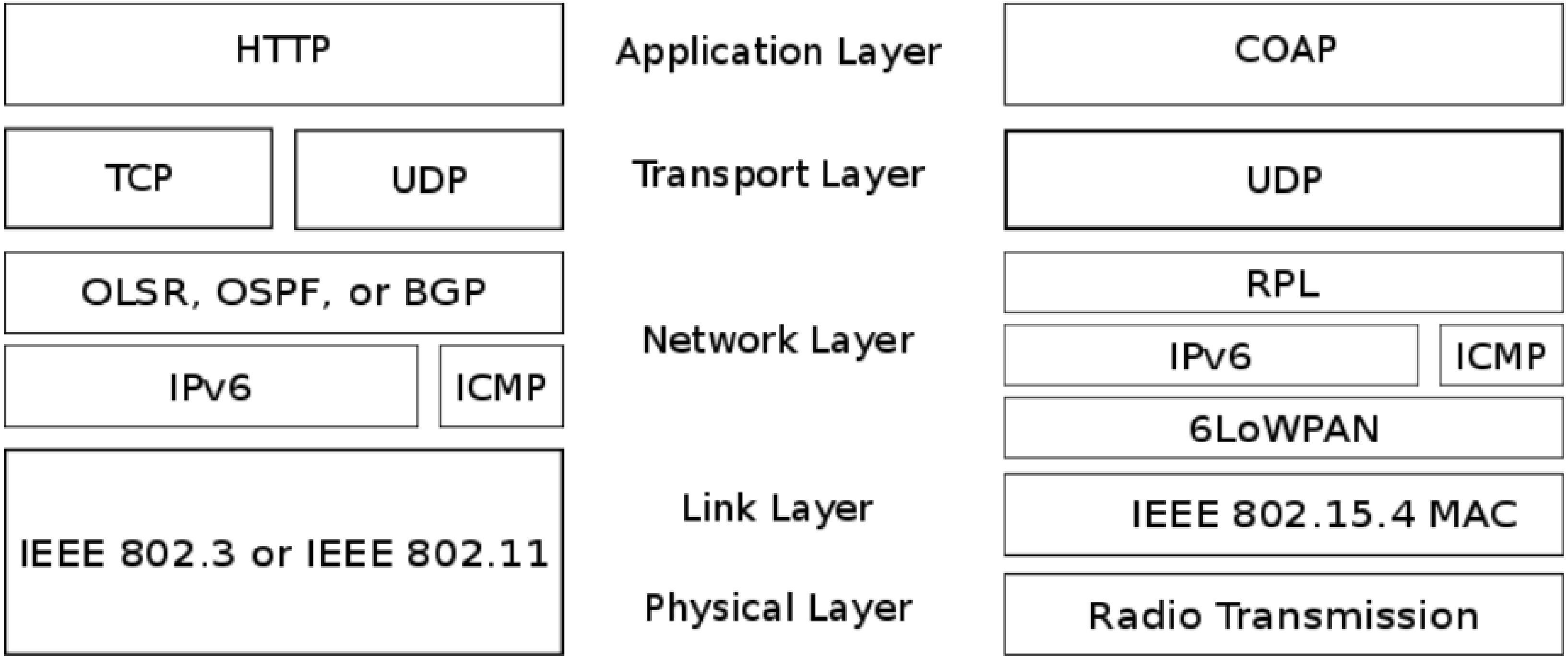
- **LoRa (most popular!)**
  - **~ 9 miles distance**
  - **Rate 50 kbps**
  - **Battery life time 10 years**
- SIGFOX
- MIOTY
- IoT-LPWAN

LoRa Architecture 10.000ft



LoRa Concept

IP-based Protocols



HTTP

Application Layer

COAP

TCP

UDP

Transport Layer

UDP

OLSR, OSPF, or BGP

IPv6

ICMP

Network Layer

RPL

IPv6

ICMP

6LoWPAN

IEEE 802.3 or IEEE 802.11

Link Layer

IEEE 802.15.4 MAC

Physical Layer

Radio Transmission

What is 6LoWPAN?

- 6LoWPAN is an IETF Protocol designed for “Transmission of IPv6 Packets over IEEE 802.15.4 Networks” (rfc4944 , rfc6282)

- 6LoWPAN allows compression of IPv6 header and other headers such as UDP
  - IPv6 overhead reduction from 40bytes to e.g. 2/3 bytes

- The 6Lo Working Group in IETF (https://tools.ietf.org/wg/6lo/) adapts 6LoWPAN protocol to other link layers technologies
  - E.g. 6LoWPAN for Bluetooth Low Energy is being finalized (https://tools.ietf.org/wg/6lo/draft-ietf-6lo-btle/)

Data Protocols

HTTP/ REST (Representational State Transfer)

- IETF standard (RFC 2616 is HTTP/1.1)

CoAP (Constrained Application Protocol)

- IETF standard (RFC 7252)

MQTT (Message Queuing Telemetry Transport)

- soon (end 2014 ?) OASIS standard

AMQP (Advanced Message Queuing Protocol)

- OASIS and ISO 19464 standard (1.0)

REST/ HTTP

The term *representational state transfer* (REST) was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation.

- Communicate statelessly
- RESTful designs use standard HTTP methods: GET, PUT, POST and DELETE
- Give resources an ID: Universal Resource Identifier (URI)
- Link things together

A URI can be broken into a

Scheme : http://
Authority: www.th-rosenheim.de
Port : 8080
Path : /iot
Query :?id=“temperature”

- The message via HTTP for the methods POST and PUT can contain a body
- HTTP Header defines content-type

CoAP

The constrained Application Protocol (CoAP) is the product of the IETF (RFC7228)
First draft was created in 2014 by the IETF Constrained RESTful Environments (CoRE) working group
CoAP is based on concept of mimicking and replacing heavy HTTP with some lightweight protocol

CoAP is HTTP-like
Asynchronous message exchange (vs request-response)
Support for URI and content-types

CoAP has 2 layers:

1. Request/Response: Responsible for sending and receiving RESTful-based queries. REST queries are piggybacked on CON or NON message. A REST response is piggybacked on the corresponding ACK message.
2. Transactional layer: Handles single message exchanges between endpoints using one of the four basic message types.

CoAP

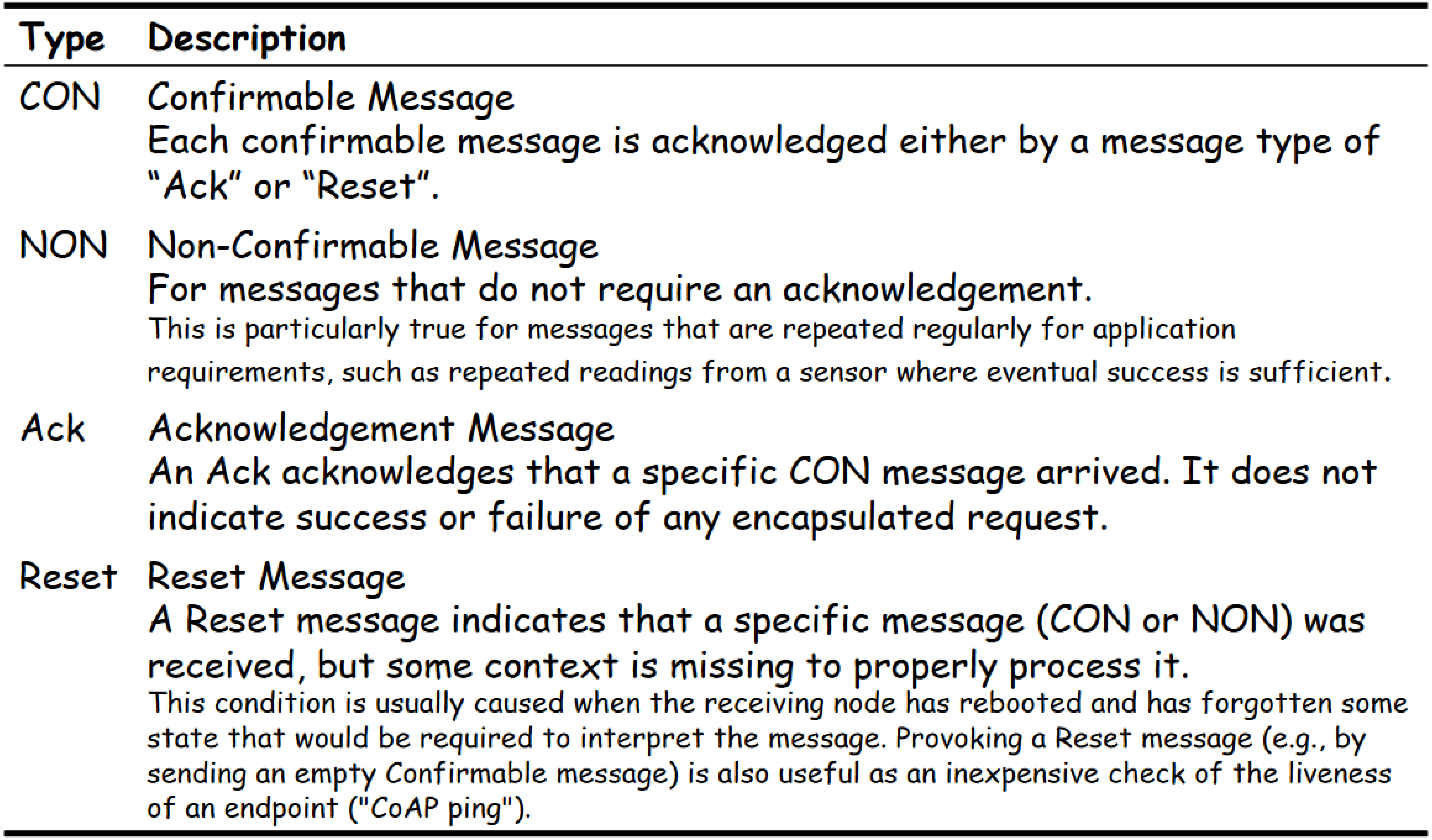
- Constrained web protocol fulfilling M2M requirements
- UDP binding with optional reliability supporting unicast and multicast requests
- Asynchronous message exchanges
- Low header overhead and parsing complexity
- URI and Content-type support
- Simple proxy and caching capabilities
- Stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP
- Security binding to Datagram Transport Layer Security (DT

- Address in CoAP is also styled like HTTP.

coap://host[:port]/path[?query]

- CoAP uses requests such as GET, PUT, POST and DELETE

CoAP: Message Types



Type	Description
CON	Confirmable Message Each confirmable message is acknowledged either by a message type of “Ack” or “Reset”.
NON	Non-Confirmable Message For messages that do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor where eventual success is sufficient.
Ack	Acknowledgement Message An Ack acknowledges that a specific CON message arrived. It does not indicate success or failure of any encapsulated request.
Reset	Reset Message A Reset message indicates that a specific message (CON or NON) was received, but some context is missing to properly process it. This condition is usually caused when the receiving node has rebooted and has forgotten some state that would be required to interpret the message. Provoking a Reset message (e.g., by sending an empty Confirmable message) is also useful as an inexpensive check of the liveness of an endpoint ("CoAP ping").

CoAP Messages

- Reliable message transmission

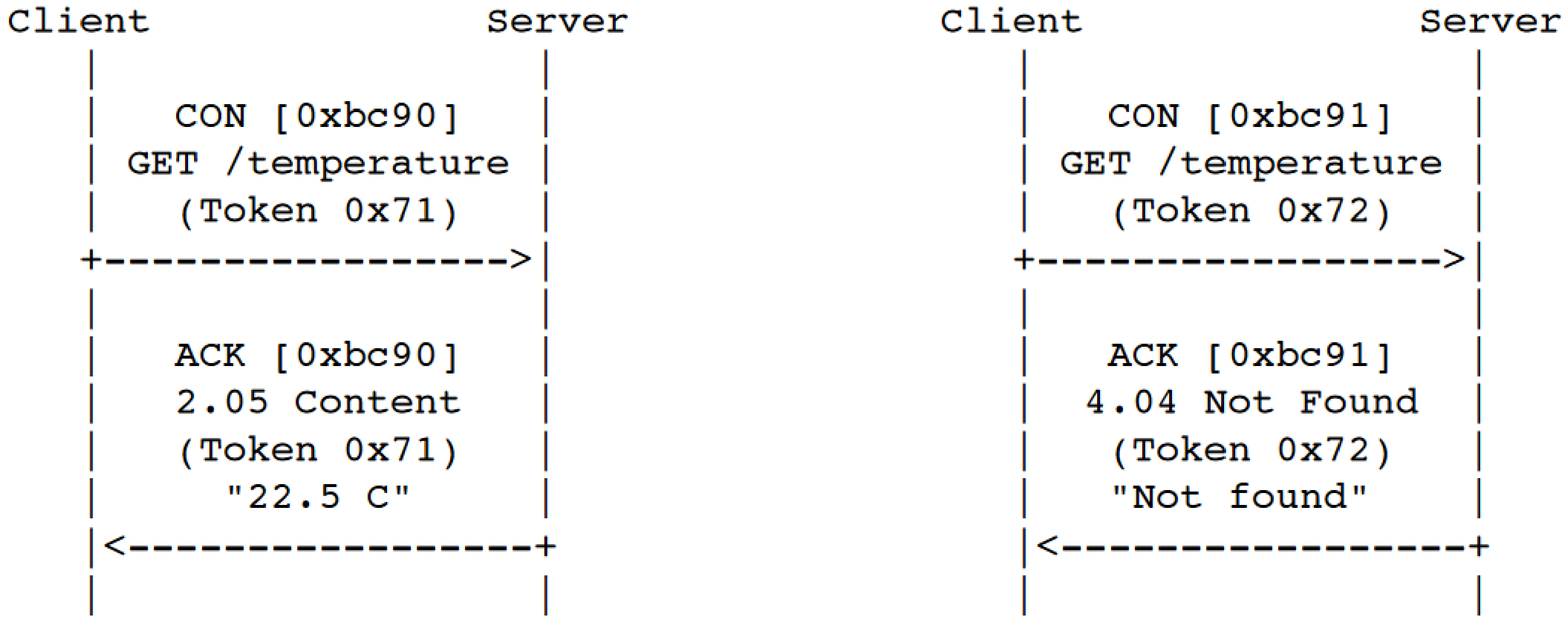
```
Client              Server
   |                   |
   |   CON [0x7d34]    |
   +------------------>|
   |                   |
   |   ACK [0x7d34]    |
   |<------------------+
   |                   |
   |                   |
   |                   |
```

- Unreliable message transmission

```
Client              Server
   |                   |
   |   NON [0x01a0]    |
   +------------------>|
   |                   |
   |                   |
   |                   |
   |                   |
   |                   |
   |                   |
```

CoAP Request/Response

- Request/Response Model



```
Client              Server       Client              Server
  |                  |             |                  |
  |   CON [0xbc90]   |             |   CON [0xbc91]   |
  | GET /temperature |             | GET /temperature |
  |   (Token 0x71)   |             |   (Token 0x72)   |
  +----------------->|             +----------------->|
  |                  |             |                  |
  |   ACK [0xbc90]   |             |   ACK [0xbc91]   |
  |   2.05 Content   |             |  4.04 Not Found  |
  |   (Token 0x71)   |             |   (Token 0x72)   |
  |     "22.5 C"     |             |   "Not found"    |
  |<-----------------+             |<-----------------+
  |                  |             |                  |
```

Figure 4: Two GET requests with piggy-backed responses

CoAP Sample

```
Client  Server
      |      |
      |      |
      +----->|     Header: GET (T=CON, Code=1, MID=0x7d34)
      | GET  |   Uri-Path: "temperature"
      |      |
      |      |
      |<-----+     Header: 2.05 Content (T=ACK, Code=69, MID=0x7d34)
      | 2.05 |    Payload: "22.3 C"
      |      |
```

```
0                   1                   2                   3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | 1 | 0 |   1   |     GET=1     |          MID=0x7d34           |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |  11   |  11   |      "temperature" (11 B) ...
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   0                   1                   2                   3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | 1 | 2 |   0   |    2.05=69    |          MID=0x7d34           |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |      "22.3 C" (6 B) ...
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

MQTT

- MQTT is a lightweight **publish/subscribe** messaging protocol designed for M2M (machine to machine) telemetry in low bandwidth environments.
- It was designed by Andy Stanford-Clark (IBM) and Arlen Nipper in 1999 for connecting Oil Pipeline telemetry systems over satellite.
- Although it started as a proprietary protocol it was released Royalty free in 2010 and became an OASIS standard in 2014.
- **MQTT** stands for **MQ** Telemetry Transport but previously was known as Message Queuing Telemetry Transport.
- **MQTT** is fast becoming one of the main protocols for **IOT** (internet of things) deployments

MQTT

- MQTT is a lightweight **publish/subscribe** messaging protocol designed for M2M (machine to machine) telemetry in low bandwidth environments.

- It was designed by Andy Stanford-Clark (IBM) and Arlen Nipper in 1999 for connecting Oil Pipeline telemetry systems over satellite.

- Although it started as a proprietary protocol it was released Royalty free in 2010 and became an OASIS standard in 2014.

- **MQTT** stands for **MQ** Telemetry Transport but previously was known as Message Queuing Telemetry Transport.

- **MQTT** is fast becoming one of the main protocols for **IOT** (internet of things) deployments

MQTT Versions

There are two versions of MQTT.

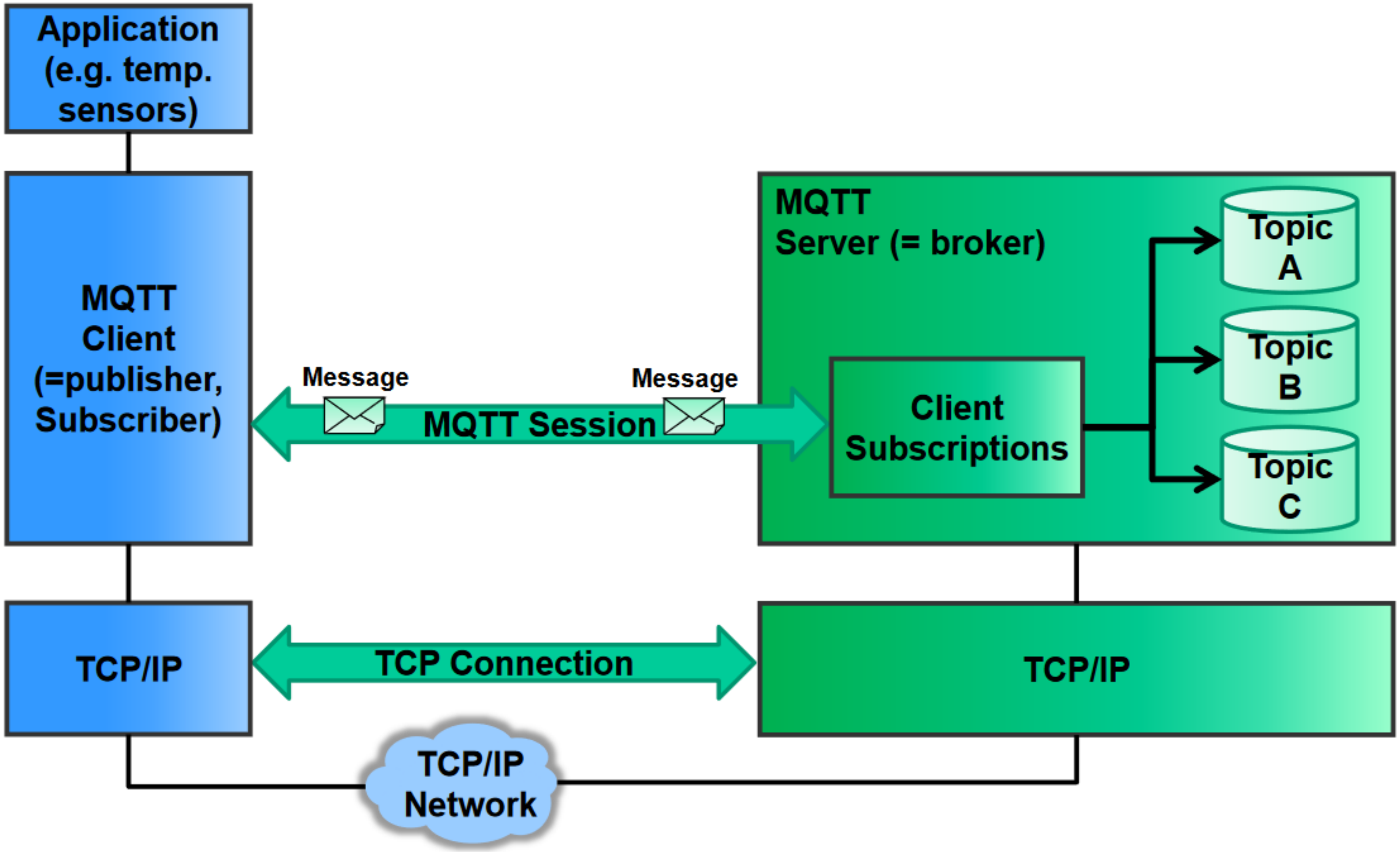
- The original MQTT which was designed in 1999 and has been in use for many years and designed for TCP/IP networks.

- MQTT-SN which was specified in around 2013, and designed to work over UDP, ZigBee and other transports.

- MQTT-SN doesn’t currently appear to be very popular. and the specification hasn’t changed for several years

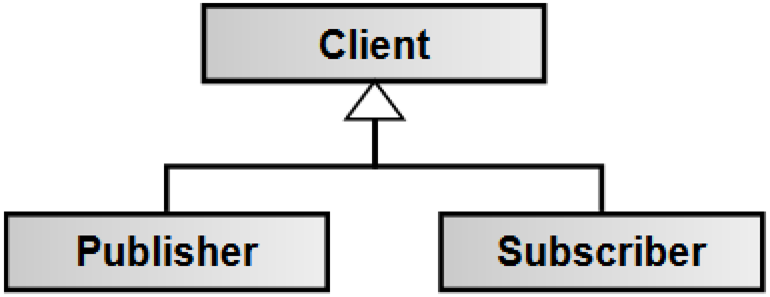
MQTT 1/2

The core elements of MQTT are clients, servers(=brokers), sessions, subscriptions and topics

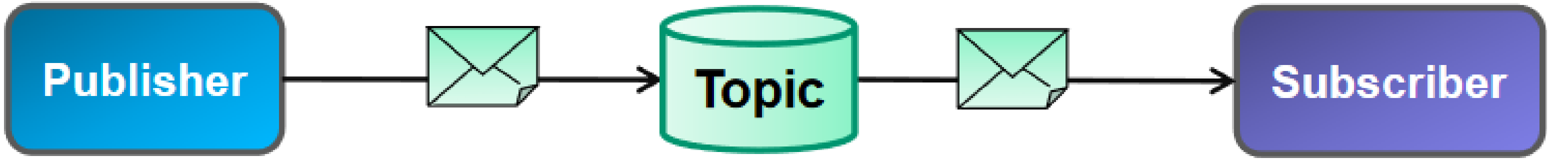


MQTT 2/2

MQTT client(=publisher, subscriber):
Clients subscribe to topics to publish and receive messages. Thus subscriber and publisher are special roles of a client.



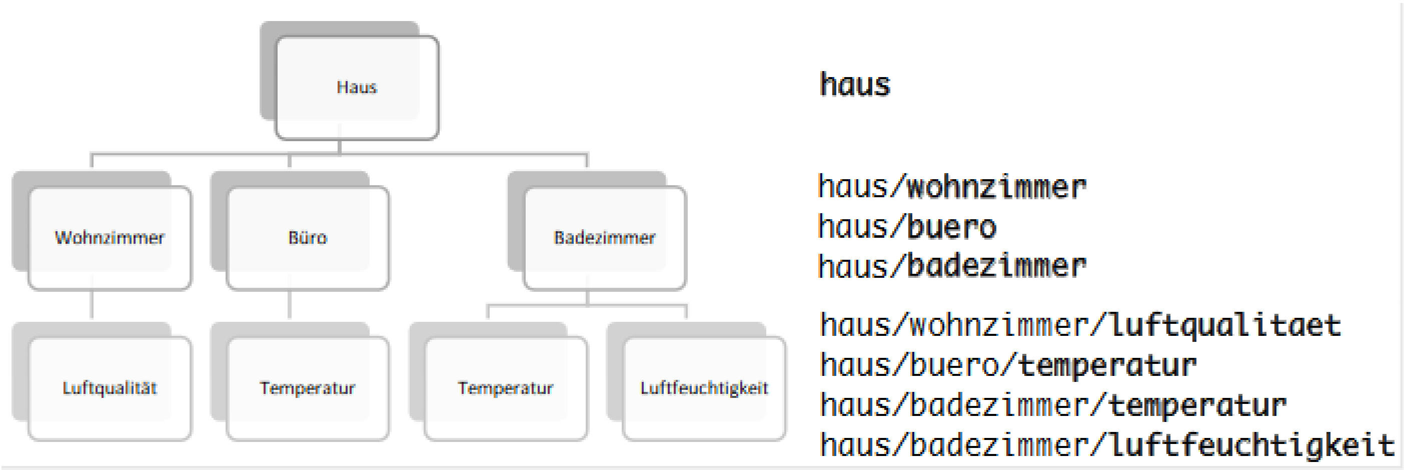
MQTT server(=broker): Servers run topics, i.e. receive subscriptions from clients on topics, receive messages from clients and forward these, based on client's subscriptions, to interested clients.
Topic: Technically, topics are message queues. Topics support the publish/subscribe pattern for clients. Logically, topics allow clients to exchange information with defined semantics. Example topic: Temperature sensor data of a building.



MQTT Topic Names

Topics are defined by a UTF-8-String which has the form of

haus/badezimmer/temperatur

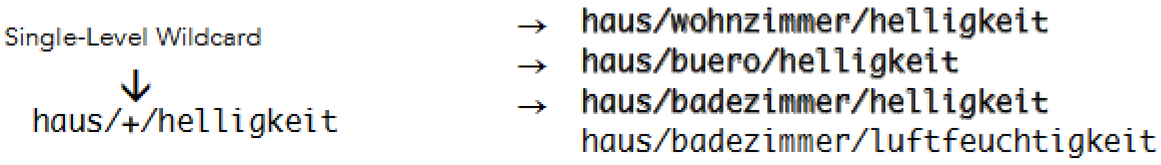


```
haus

haus/wohnzimmer
haus/buero
haus/badezimmer

haus/wohnzimmer/luftqualitaet
haus/buero/temperatur
haus/badezimmer/temperatur
haus/badezimmer/luftfeuchtigkeit
```

https://opus.hs-offenburg.de/frontdoor/deliver/index/docId/2771/file/THESIS_MARIO_SALLAT.pdf

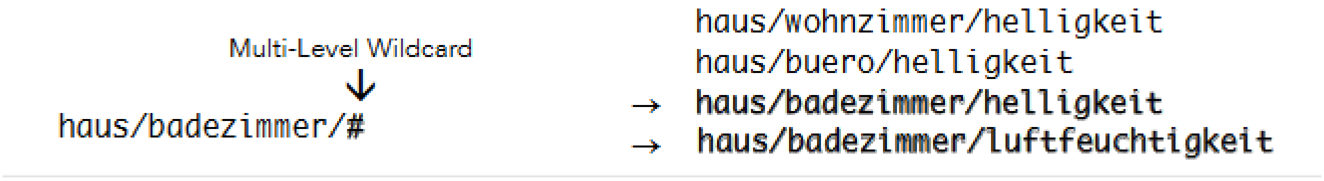
MQTT Topic Filtering

Single-Level Wildcard +



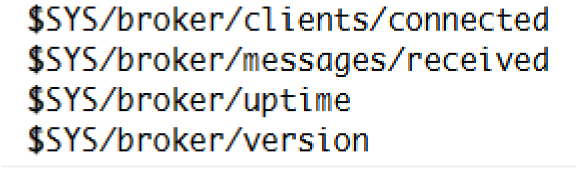
```
Single-Level Wildcard
      ↓
haus/+/helligkeit

→ haus/wohnzimmer/helligkeit
→ haus/buero/helligkeit
→ haus/badezimmer/helligkeit
  haus/badezimmer/luftfeuchtigkeit
```

Multi-Level Wildcard



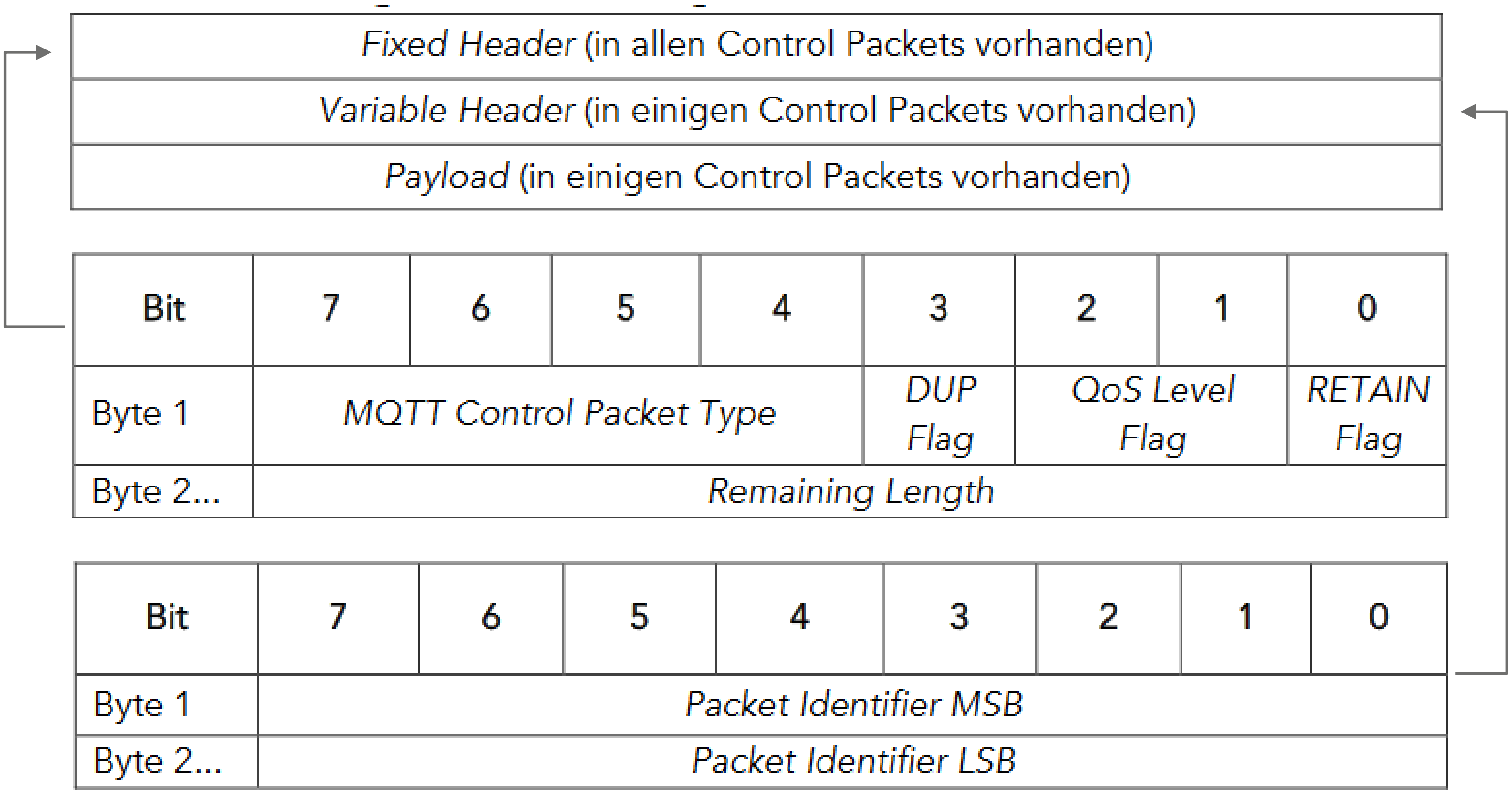
```
Multi-Level Wildcard
             ↓
haus/badezimmer/#

  haus/wohnzimmer/helligkeit
  haus/buero/helligkeit
→ haus/badezimmer/helligkeit
→ haus/badezimmer/luftfeuchtigkeit
```

Topics with $ (internal only, topic is refused!)



```
$SYS/broker/clients/connected
$SYS/broker/messages/received
$SYS/broker/uptime
$SYS/broker/version
```

MQTT Message Format

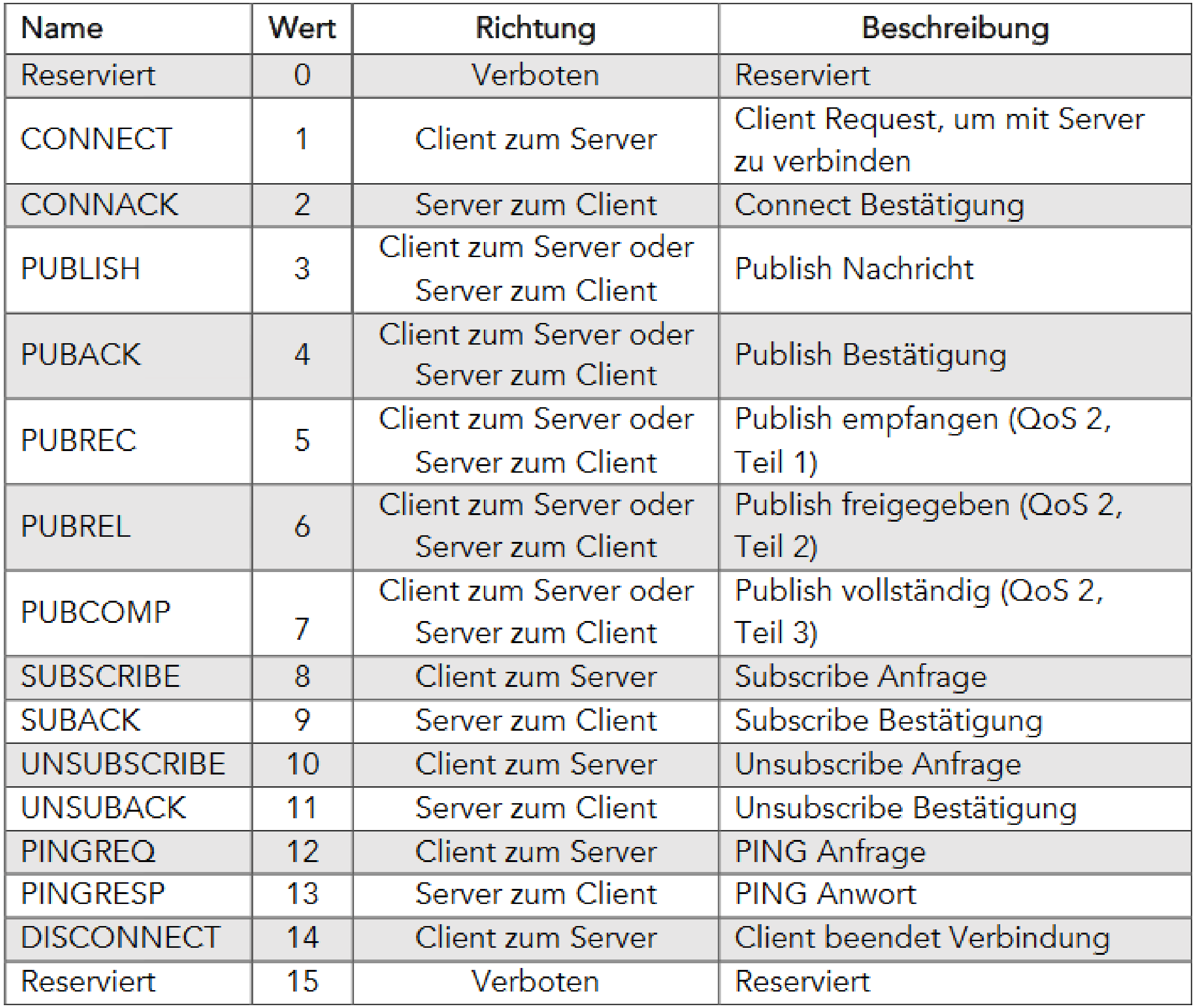


Fixed Header (in allen Control Packets vorhanden)
Variable Header (in einigen Control Packets vorhanden)
Payload (in einigen Control Packets vorhanden)

Bit	7	6	5	4	3	2	1	0
Byte 1	*MQTT Control Packet Type*				*DUP Flag*	*QoS Level Flag*		*RETAIN Flag*
Byte 2…	*Remaining Length*							

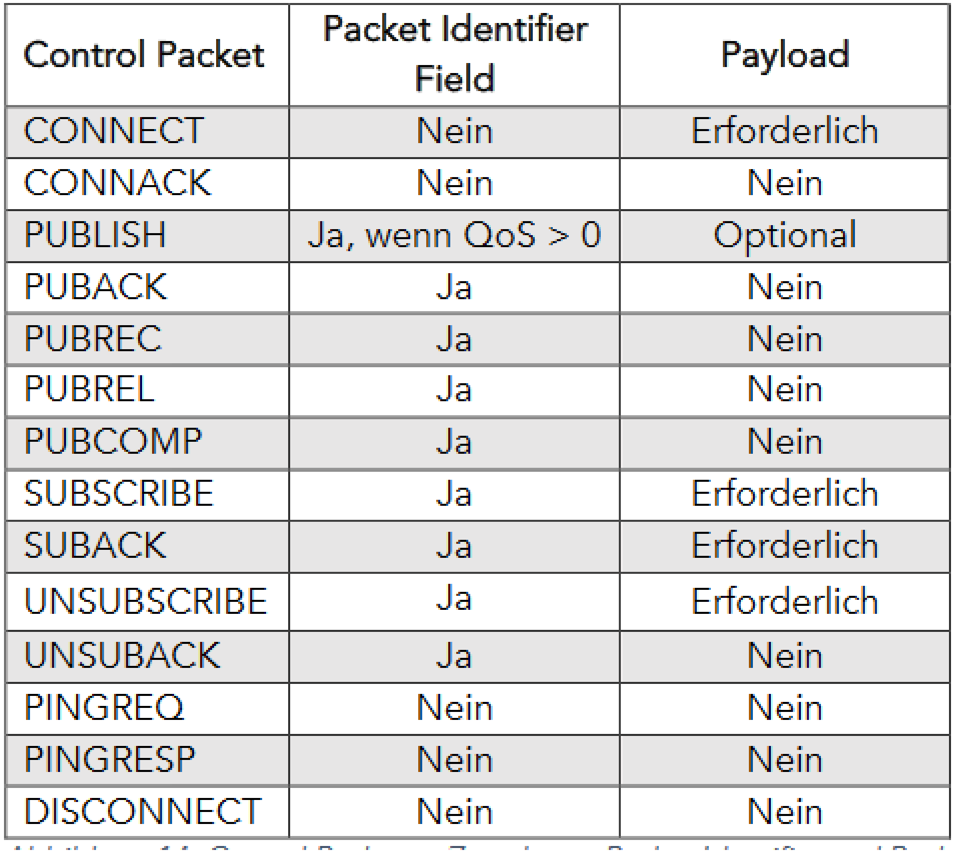
Bit	7	6	5	4	3	2	1	0
Byte 1	*Packet Identifier MSB*							
Byte 2…	*Packet Identifier LSB*							

MQTT Message Format



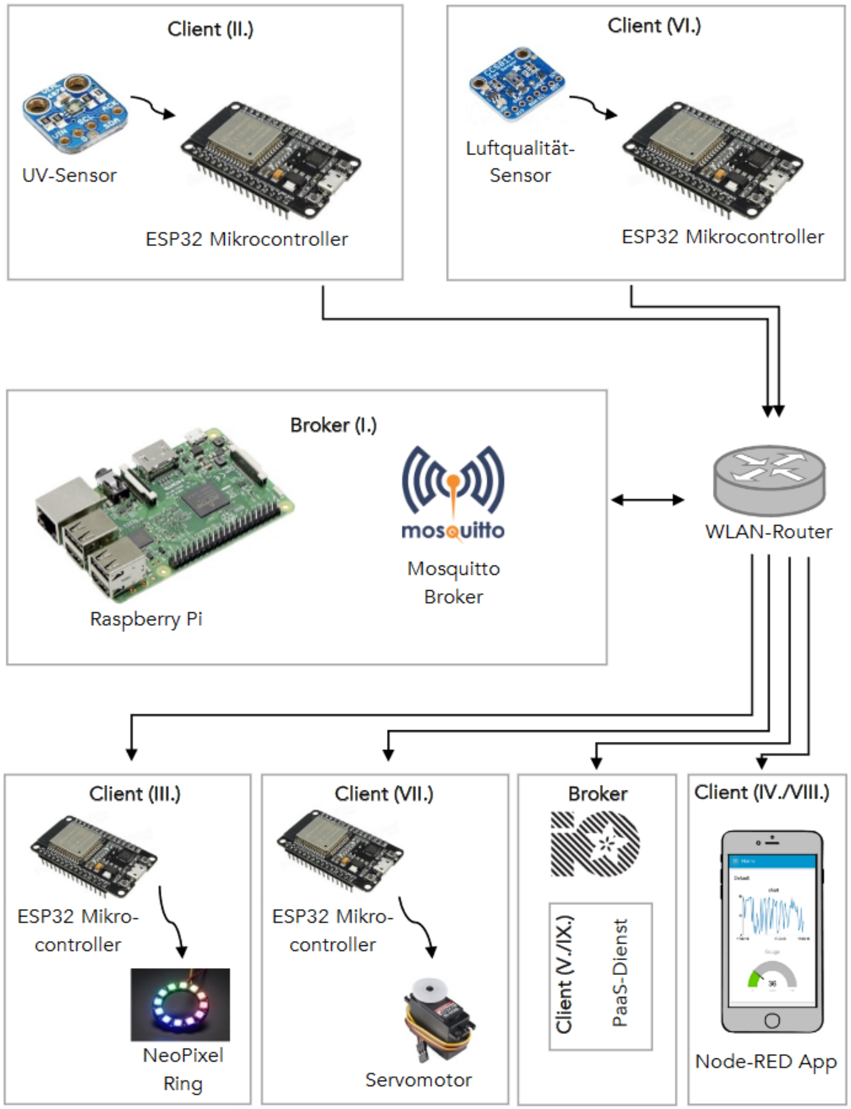
Name	Wert	Richtung	Beschreibung
Reserviert	0	Verboten	Reserviert
CONNECT	1	Client zum Server	Client Request, um mit Server zu verbinden
CONNACK	2	Server zum Client	Connect Bestätigung
PUBLISH	3	Client zum Server oder Server zum Client	Publish Nachricht
PUBACK	4	Client zum Server oder Server zum Client	Publish Bestätigung
PUBREC	5	Client zum Server oder Server zum Client	Publish empfangen (QoS 2, Teil 1)
PUBREL	6	Client zum Server oder Server zum Client	Publish freigegeben (QoS 2, Teil 2)
PUBCOMP	7	Client zum Server oder Server zum Client	Publish vollständig (QoS 2, Teil 3)
SUBSCRIBE	8	Client zum Server	Subscribe Anfrage
SUBACK	9	Server zum Client	Subscribe Bestätigung
UNSUBSCRIBE	10	Client zum Server	Unsubscribe Anfrage
UNSUBACK	11	Server zum Client	Unsubscribe Bestätigung
PINGREQ	12	Client zum Server	PING Anfrage
PINGRESP	13	Server zum Client	PING Anwort
DISCONNECT	14	Client zum Server	Client beendet Verbindung
Reserviert	15	Verboten	Reserviert

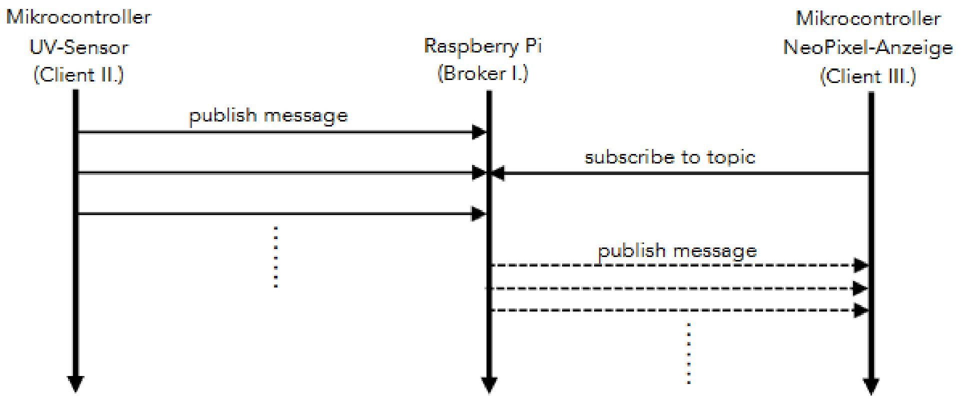
MQTT Message Format

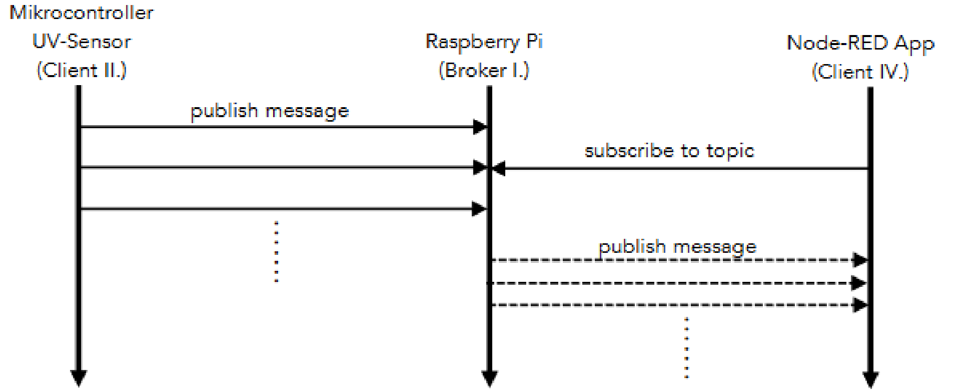


Control Packet	Packet Identifier Field	Payload
CONNECT	Nein	Erforderlich
CONNACK	Nein	Nein
PUBLISH	Ja, wenn QoS > 0	Optional
PUBACK	Ja	Nein
PUBREC	Ja	Nein
PUBREL	Ja	Nein
PUBCOMP	Ja	Nein
SUBSCRIBE	Ja	Erforderlich
SUBACK	Ja	Erforderlich
UNSUBSCRIBE	Ja	Erforderlich
UNSUBACK	Ja	Nein
PINGREQ	Nein	Nein
PINGRESP	Nein	Nein
DISCONNECT	Nein	Nein

MQTT Sample


Client (II.)
UV-Sensor
ESP32 Mikrocontroller
Client (VI.)
Luftqualität-Sensor
ESP32 Mikrocontroller
Broker (I.)
Raspberry Pi
mosquitto
Mosquitto Broker
WLAN-Router
Client (III.)
ESP32 Mikro-controller
NeoPixel Ring
Client (VII.)
ESP32 Mikro-controller
Servomotor
Broker
Client (V./IX.)
PaaS-Dienst
Client (IV./VIII.)
Node-RED App


Mikrocontroller
UV-Sensor
(Client II.)
Raspberry Pi
(Broker I.)
Mikrocontroller
NeoPixel-Anzeige
(Client III.)
publish message
subscribe to topic
publish message


Mikrocontroller
UV-Sensor
(Client II.)
Raspberry Pi
(Broker I.)
Node-RED App
(Client IV.)
publish message
subscribe to topic
publish message