UNIVERSITÄT
BERN

b
UNIVERSITÄT
BERN

Internet of Things

X. Application Layer Protocols

Prof. Dr. Torsten Braun, Institut für Informatik

Bern, 03.05.2021 – 10.05.2021

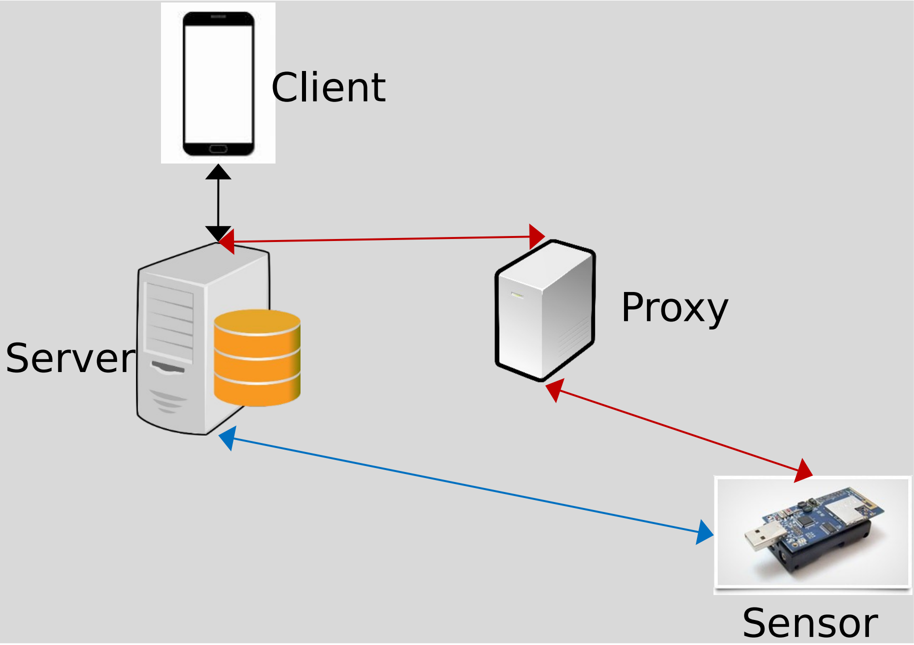
Internet of Things: Application Layer Protocols

Table of Contents

1. Introduction

1. Application Scenarios

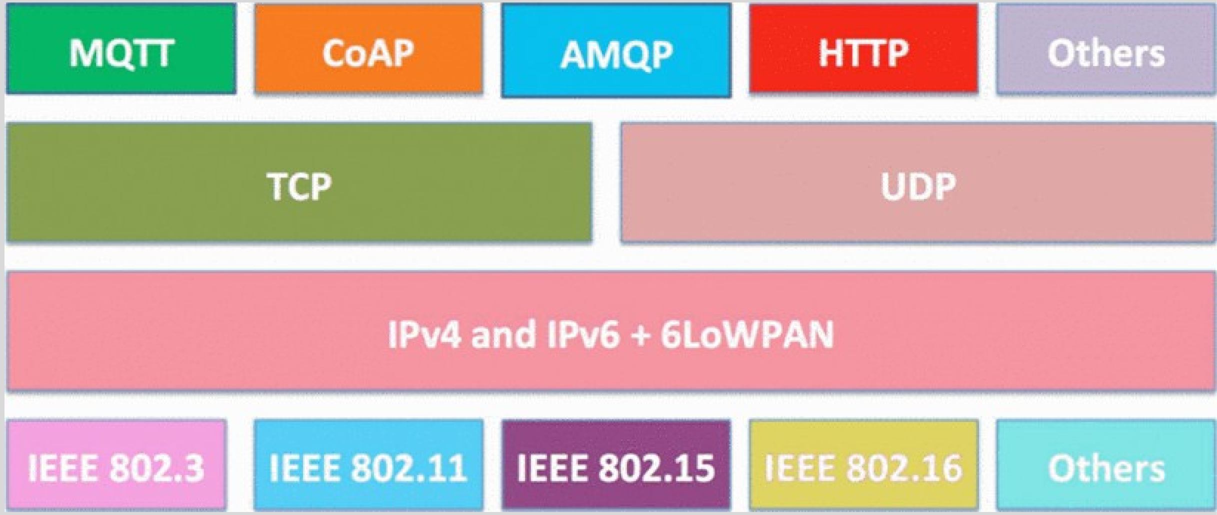
- Goal: End-to-end communication between server and sensors using standard application protocols possibly running on top of TCP/UDP/IP.
- Options
  - Direct communication using HTTP or CoAP
  - Proxy for protocol translation, e.g., between HTTP and CoAP



UNIVERSITÄT BERN

1. Introduction

2. Application Layer Protocols for the IoT



2. Web Services for IoT

By using web service technology for smart object applications, existing

- web-service-oriented systems,
- programming libraries, and
- knowledge

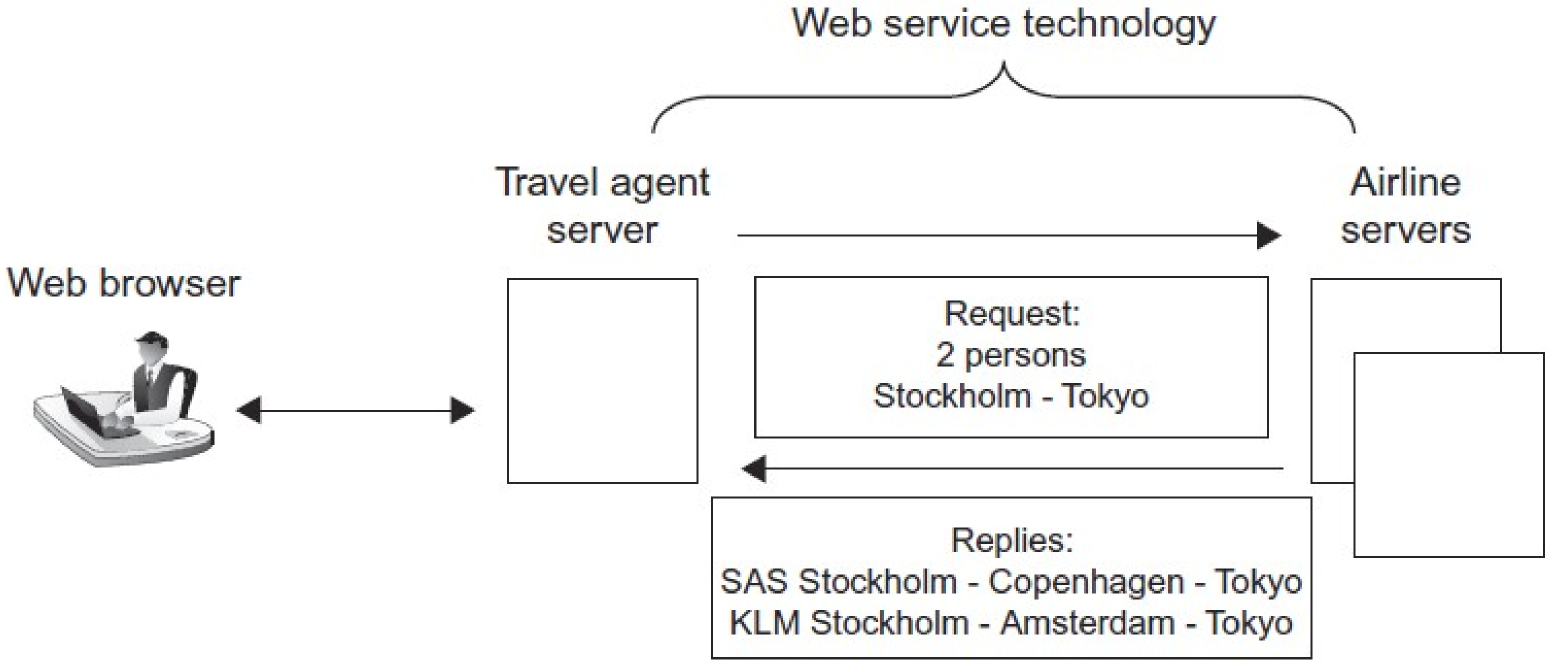
can be directly applied to smart object applications.

Smart object applications

- can be directly integrated with existing business systems.
- use the same interfaces and systems as existing business systems.
- can be integrated into enterprise resource planning systems without any intermediaries, reducing complexity of the system as a whole.

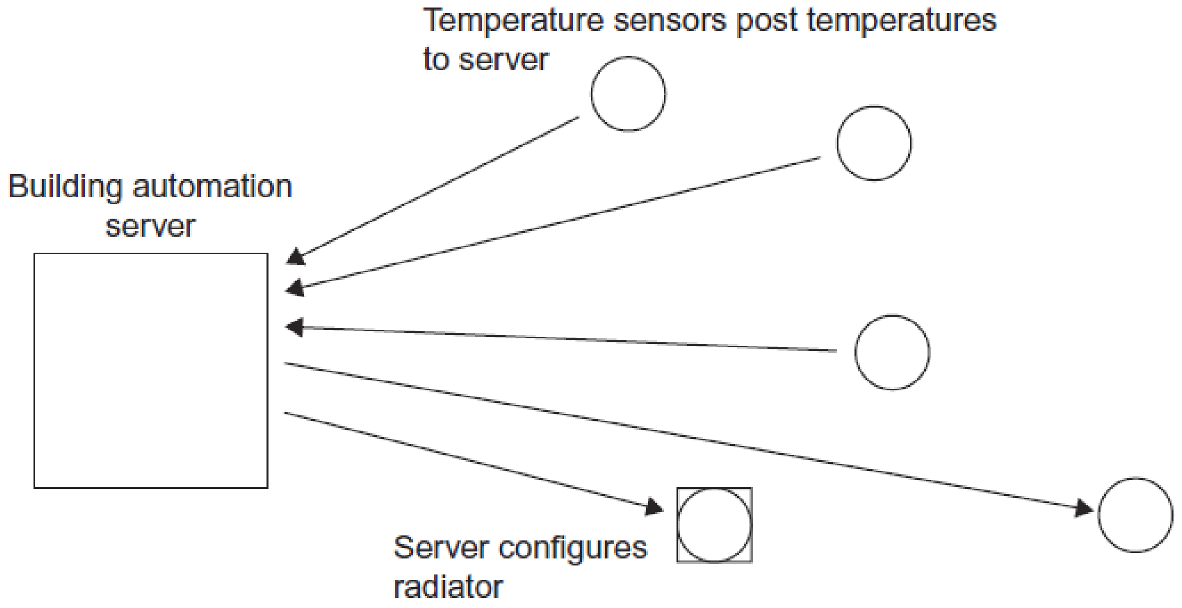
2. Web Services for IoT

1. Web Services Concept



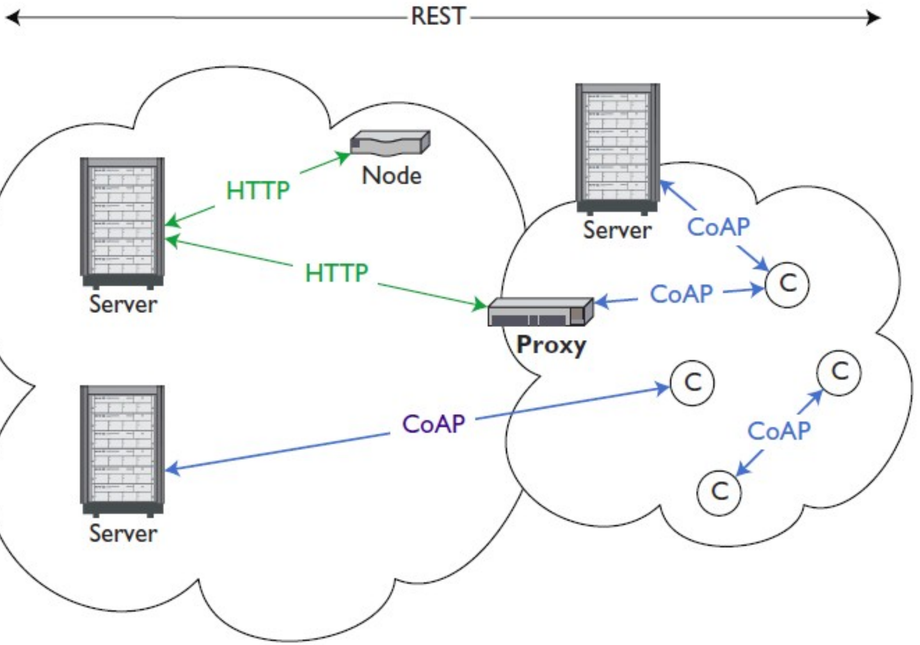
2. Web Services for IoT

2. Web Services in IoT


Temperature sensors post temperatures to server
Building automation server
Server configures radiator

2. Web Services for IoT

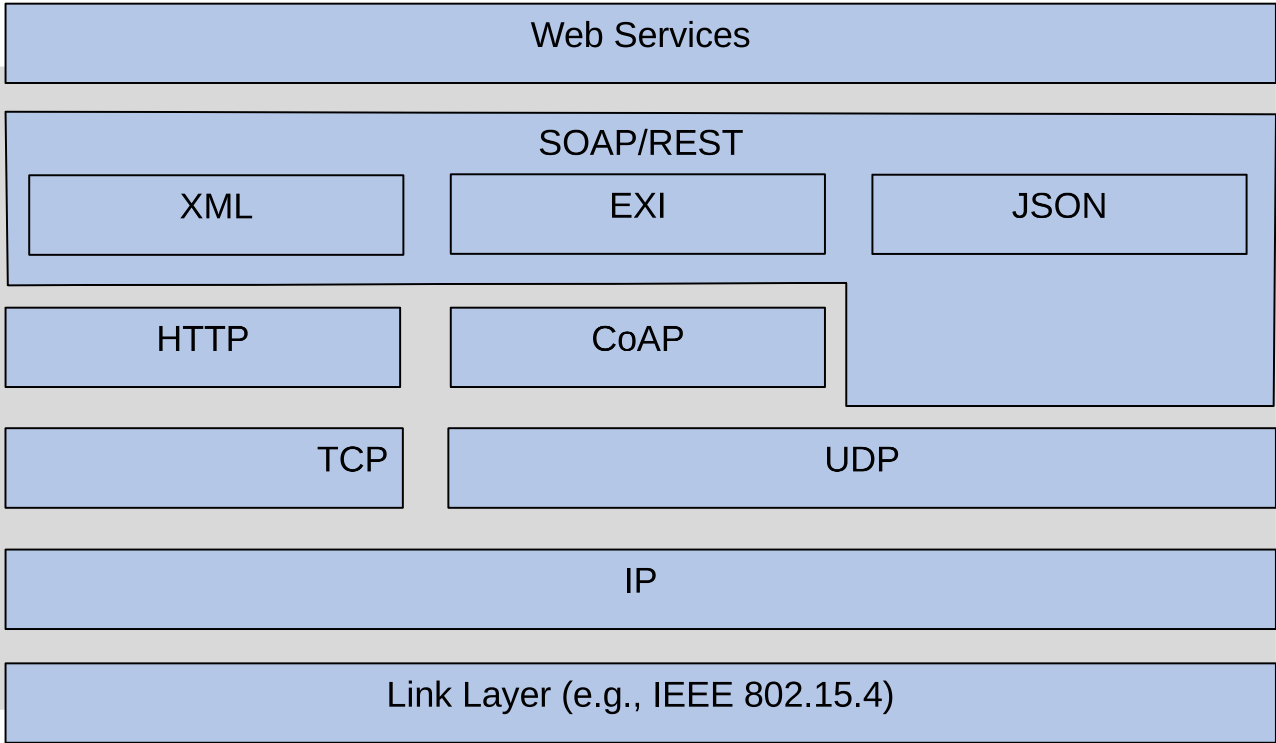
3. IoT with HTTP / CoAP



Payload	Payload
HTTP	CoAP
TCP	UDP
IP	IP
Ethernet link	Constrained link

2. Web Services for IoT

4. IoT Protocol Stack



3. REST for IoT

1. Representational State Transfer

- Representation
  - Data or resources are encoded as representations, e.g.,
    - Resource: temperature
    - Representation: decimal number
- State
  - All necessary state needed to complete a request must be provided with the request.
  - Clients and servers are stateless.
  - A client cannot rely on any state to be stored in the server, and a server cannot rely on any state stored in the client.
  - This does not pertain to the data stored by servers or clients, only to the connection state needed to complete transactions.
- Transfer
  - Representation and state to be transferred between client and server

UNIVERSITÄT BERN

3. REST for IoT

2. Resource Oriented Architecture for IoT

REST is based on the notion of a resource to be used/addressed and
has the following constraints:

- Resource Identification
  - Web relies on Uniform Resource Identifiers (URI) to identify resources.
- Uniform Interface
  - Resources should be available through a uniform interface with
    well-defined interaction semantics.
    → HTTP with 4 main operations:
    GET, PUT, POST, DELETE
  - Most web service applications
    offer RESTful interfaces.
- Self-Describing Messages
  - Web: HTML
  - Machine-oriented services:
    Media types such as
    Extensible Markup Language,
    JavaScript Object Notation have gained widespread
    support across services and client platforms.
    Efficient XML Interchange can be used for
    constrained environments.

3. REST for IoT

3. Data Formats for Web Services: XML/JSON

```
<xml>

    <sensors>

        <sensor>

<name>Temperature</name>

            <value>27.1</value>

        </sensor>

    </sensors>

</xml>
```

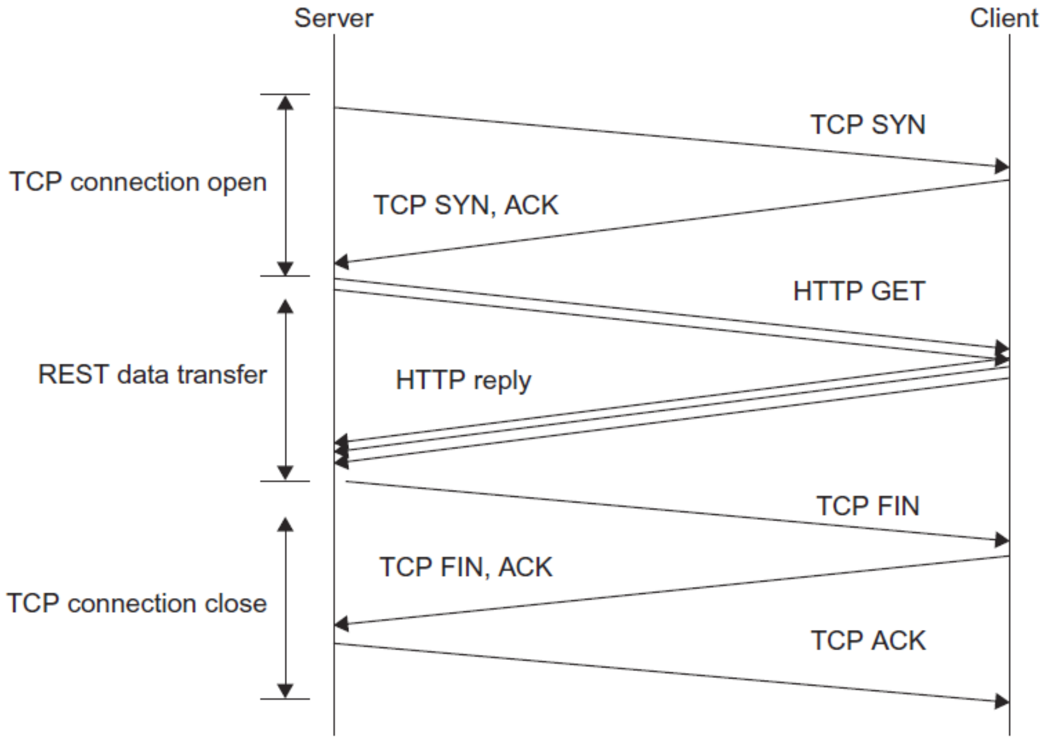
```
{"sensors":

    [{"name": "Temperature",
    "value": 26.1}]

}
```

3. REST for IoT

4. REST Web Service Transfer



3. REST for IoT

5. REST Web Service Transfer Example

HTTP Request

```
GET /sensors/temperature
HTTP/1.1

Content-type:
application/json
```

HTTP Response

```
HTTP/1.1 200 OK

Content-type:
application/json

{"sensors":[{"name":
"Temperature", "value":
26.1}]}
```

3. REST for IoT

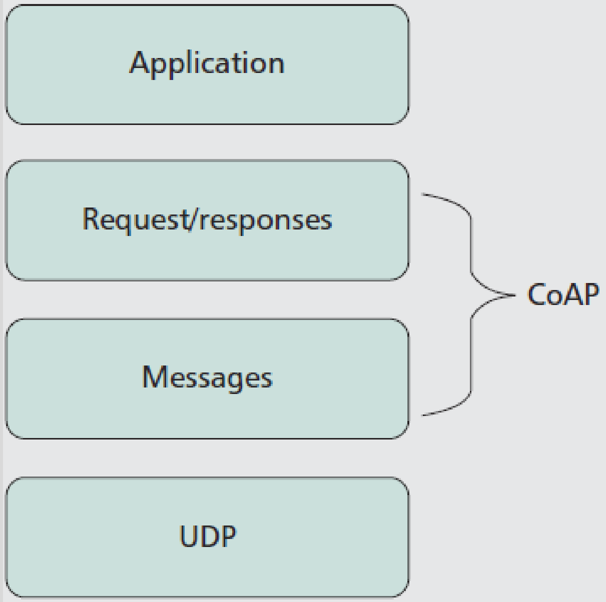
3.6 Operations

- *GET* on http://.../spot1/sensors/temperature
  - returns the temperature observed by *spot1*, i.e., it retrieves the current representation of the temperature resource.
- *PUT* on http://.../sunspots/spot1/actuators/leds/1
  - switches on the first LED of the SunSPOT, i.e., it updates the state of LED resource.
- *POST* on http://.../spot1/sensors/temperature/rules
  - with a JSON representation of the rule as {*"threshold":35*} encapsulated in the HTTP body creates a rule that will notify the caller whenever the temperature is above 35°.
- *DELETE* on http://.../spot
  - is used to shutdown the node.
- *DELETE* on http://.../spot1/sensors/temperature/rules/1
  - is used to remove rule number 1.
- *OPTIONS* to retrieve the operations that are allowed on a resource.
  - allows applications to find out what operations are allowed for any URI.
  - Example: an *OPTIONS* request on http://.../sunspots/spot1/sensors/tilt returns *GET, OPTIONS.*

UNIVERSITÄT BERN

4. Constrained Application Protocol

- realizes a subset of HTTP functions
- is optimized for constrained environments.
- offers features such as built-in resource discovery, multicast support, and asynchronous message exchange.
- adopts datagram-oriented transport protocols such as UDP.
- introduces a two-layer structure to ensure reliable transmission over UDP.
  - Request/response interaction layer to transmit resource operation requests and request/response data.
  - Message layer to deal with asynchronous interactions with UDP



4. CoAP

1. Features

- constrained web protocol fulfilling machine-to-machine (M2M) requirements
- asynchronous message exchange
- low header overhead and parsing complexity
- URI and Content-type support
- simple proxy and caching capabilities
- optional resource discovery
- UDP transport with optional reliability supporting unicast/multicast requests (to query several devices simultaneously)
- stateless HTTP-CoAP mapping, allowing proxy to provide access to CoAP resources via HTTP and vice versa
- security using Datagram Transport Layer Security

UNIVERSITÄT BERN

4. CoAP

2. Message Layer

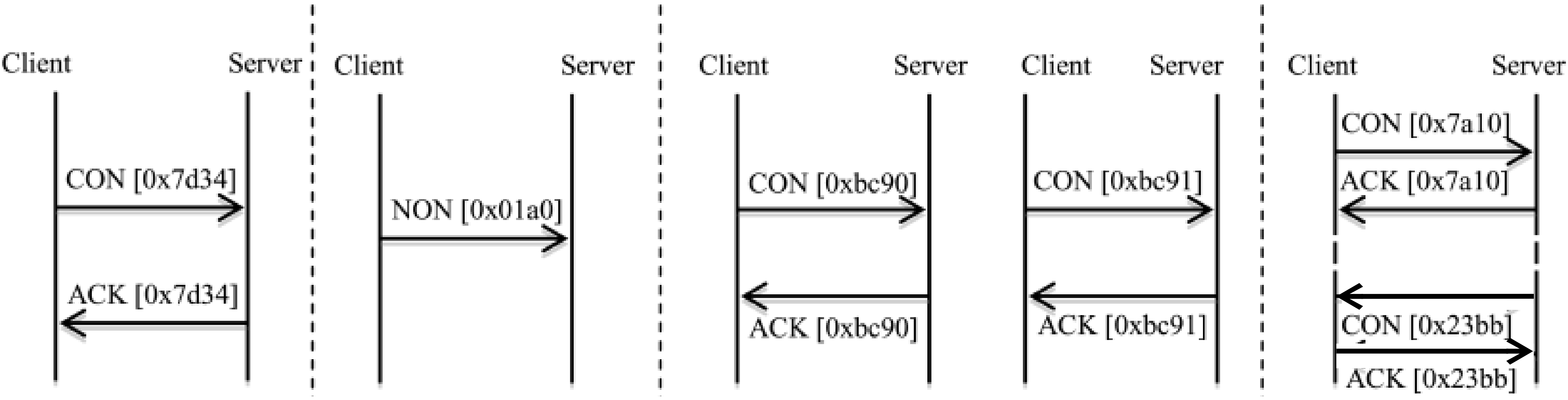
- to control message exchanges over UDP between two endpoints
- Messages are identified by an ID to detect duplicates and to provide reliability.

Message types

- Confirmable
  - requires a response, which can be piggybacked in an acknowledgement or
    sent asynchronously in another message, if the response takes too much time to be computed.
- Non-Confirmable
  - needs to be neither acknowledged nor replied.
- Acknowledgement
  - confirms the reception of a confirmable message and can contain the piggybacked response to the confirmable message.
- Reset
  - if a confirmable message cannot be processed

4. CoAP

3. Message Response Types



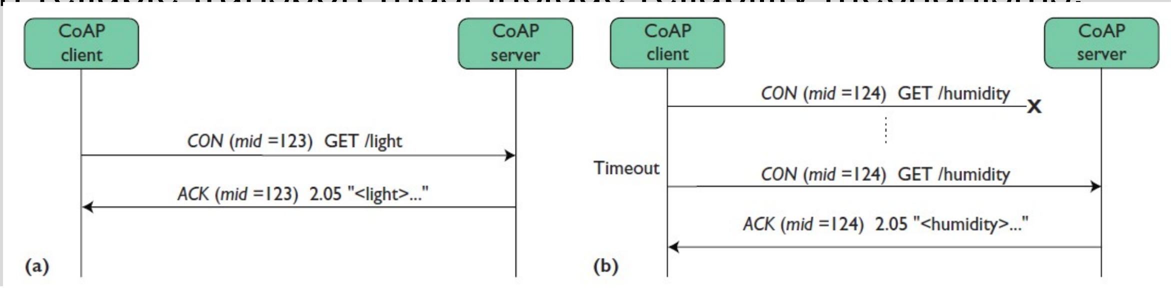
Confirmable Non-confirmable Piggybacked responses

Separate response

4. CoAP

4. Request/Response Interaction Layer

- Request and Response messages include method or response code.
  - A Request consists of the method that should be applied to the resource, the identifier of the resource, a payload, an Internet media type (if any), an optional meta-data about the Request.
  - A Response is identified by the code field in the CoAP header, which indicates the result of the Request.
  - Token option to match Responses to Requests independently from underlying messages.
- CoAP implementation over non-reliable transport must include reliability mechanisms.

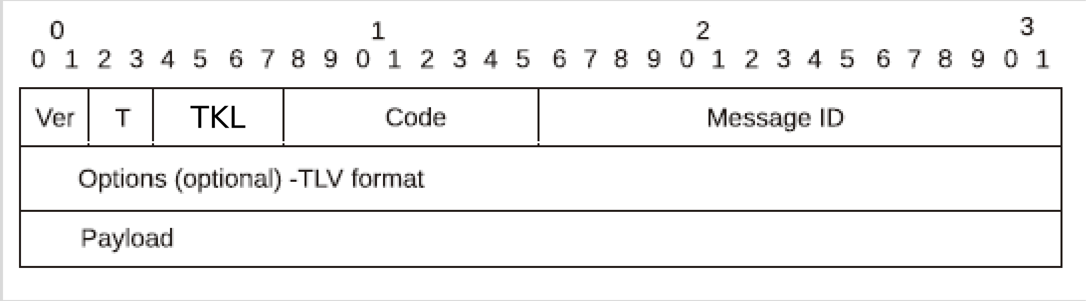

CoAP client
CoAP server
CON (mid =123) GET /light
ACK (mid =123) 2.05 "<light>..."
(a)
CoAP client
CoAP server
CON (mid =124) GET /humidity
X
Timeout
CON (mid =124) GET /humidity
ACK (mid =124) 2.05 "<humidity>..."
(b)

b UNIVERSITÄT BERN

4. CoAP

5. Frame Format

- Version = 1 in the current version.
- Type:
  - (0) Confirmable
  - (1) Non-Confirmable
  - (2) Ack
  - (3) Reset
- Token Length: 4-bit unsigned integer indicating the length of the variable-length Token field (0-8 bytes).



4. CoAP

6. Methods

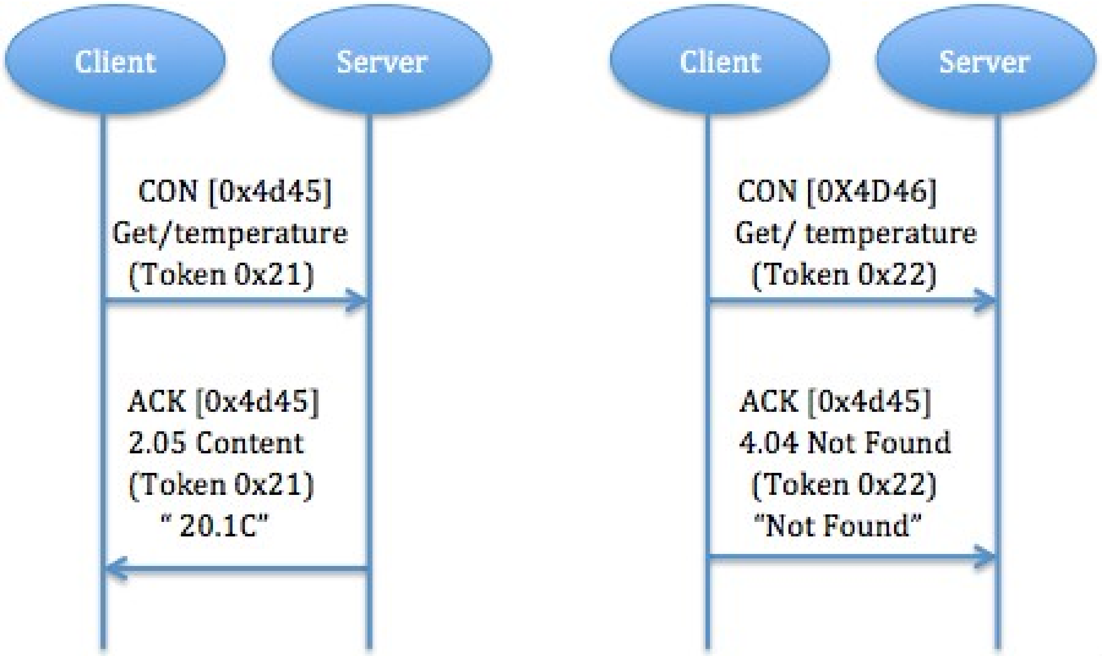
- GET
  - retrieves a representation for the information corresponding to the resource identified by the request URI.
- POST
  - *requests the processing of the representation* enclosed in the resource identified by the request URI.
    Normally, it results in a new resource or the target resource being updated.
- PUT
  - *requests that the resource identified by the request URI be updated or created* with the enclosed representation.
- DELETE
  - requests that the resource identified by the request URI be deleted.

UNIVERSITÄT BERN

4. CoAP

7.1 Requests with Piggybacked Responses

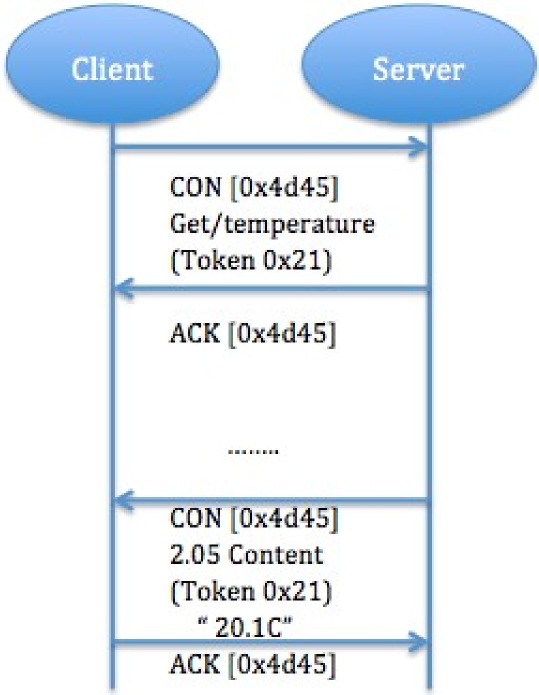
- Client sends request using Confirmable or Non-Confirmable message.
- Acknowledgement contains response message (identified by token) or failure response code for Confirmable message.



4. CoAP

7.2 Requests with Separate Responses

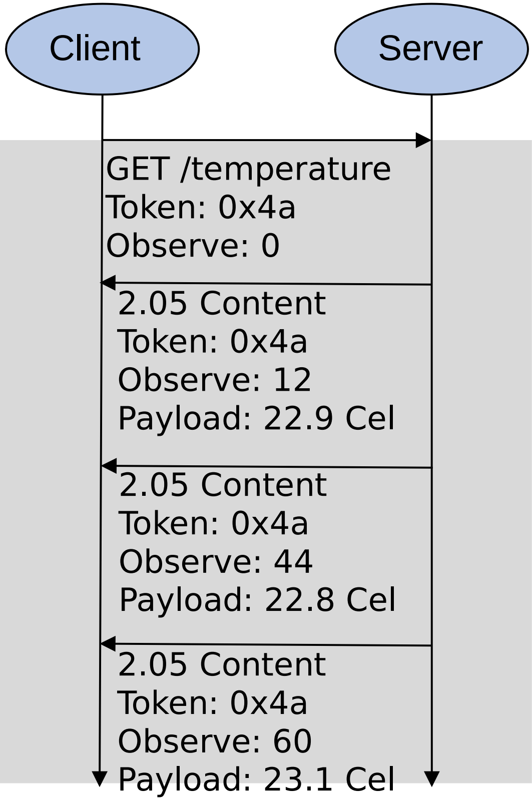
- If a server is not able to respond immediately to a request carried in a Confirmable message, it simply responds with an Empty Acknowledgement message so that the client can stop retransmitting the request.
- When the response is ready, the server sends it in a new Confirmable message (which in turn needs to be acknowledged by the client).



4. CoAP

8. Observe Functionality

- HTTP transactions are client-initiated:
  - A client must perform GET operations again and again (pulling), if it wants to stay up to date about a resource's status.
  - Pull model becomes expensive in a resource-limited environment.
- CoAP uses an asynchronous approach to support pushing information from servers to clients: Observation.
  - In a GET request, a client can indicate its interest in further updates from a resource by specifying the "Observe" option.
  - If the server accepts this option, the client becomes an observer of this resource and receives an asynchronous notification message each time it changes.
  - In notifications, Observe Option provides a sequence number for reordering detection.
  - All notifications carry token specified by client.



4. CoAP

9. URIs

CoAP URIs are very similar to HTTP URIs, providing the means to locate the resources.

```
“coap:” “//” host
[“:” port] path
[“?” query]
```

- Host can be provided either as an IP address, or a name, which should be resolved using a resolution service such as DNS.
- UDP port where the end-point is listening.
- Path defines resource in that host.
- Query in the form of “key=value” pairs enables parameterization of the resource.

UNIVERSITÄT BERN

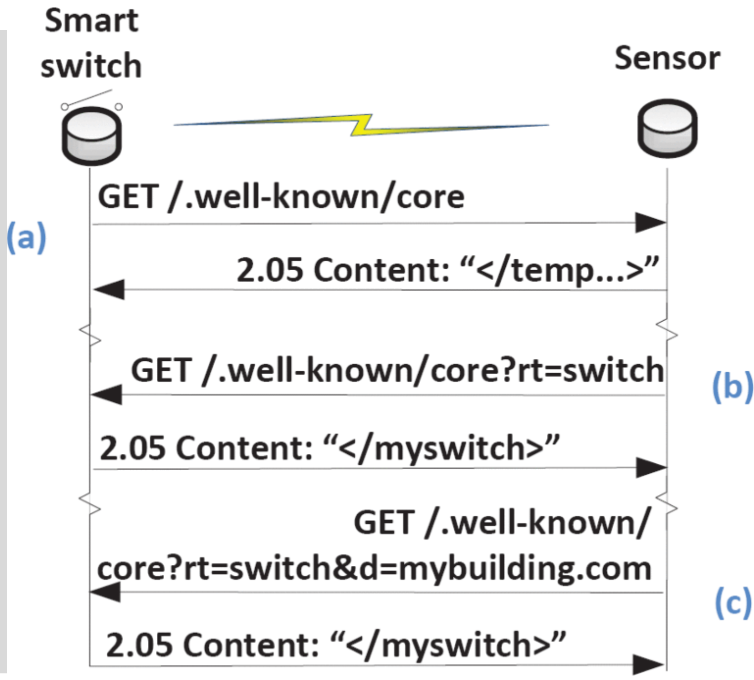
4. CoAP

10. Service Discovery

– CoAP servers can provide a resource description available via the well-known URI /.well-known for resource discovery (unicast or multicast)

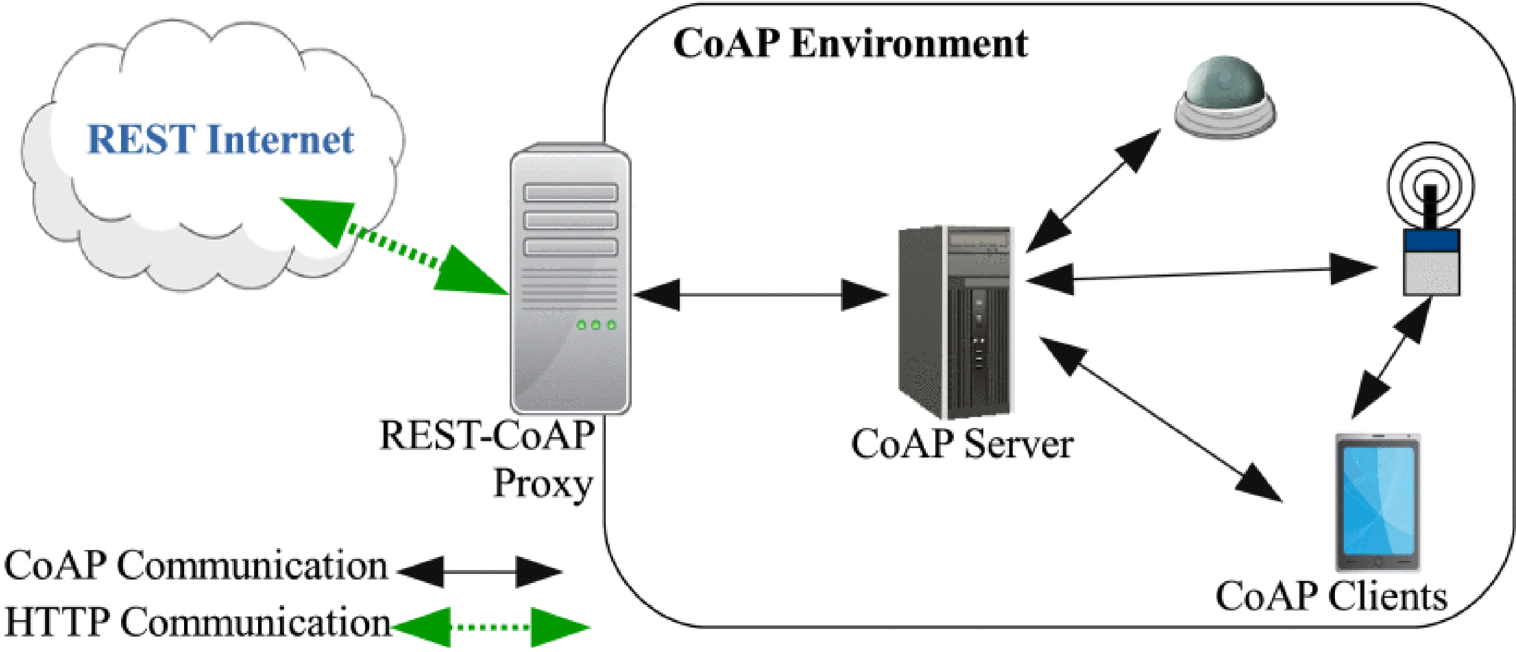
– Clients then access this description with a GET request on that URI.

– The same description could be advertised, or even posted to a description directory.


Smart switch
Sensor
(a)
GET /.well-known/core
2.05 Content: "</temp...>"
GET /.well-known/core?rt=switch
(b)
2.05 Content: "</myswitch>"
GET /.well-known/core?rt=switch&d=mybuilding.com
(c)
2.05 Content: "</myswitch>"

4. CoAP

11. Scenario



4. CoAP

12. HTTP-CoAP Mapping

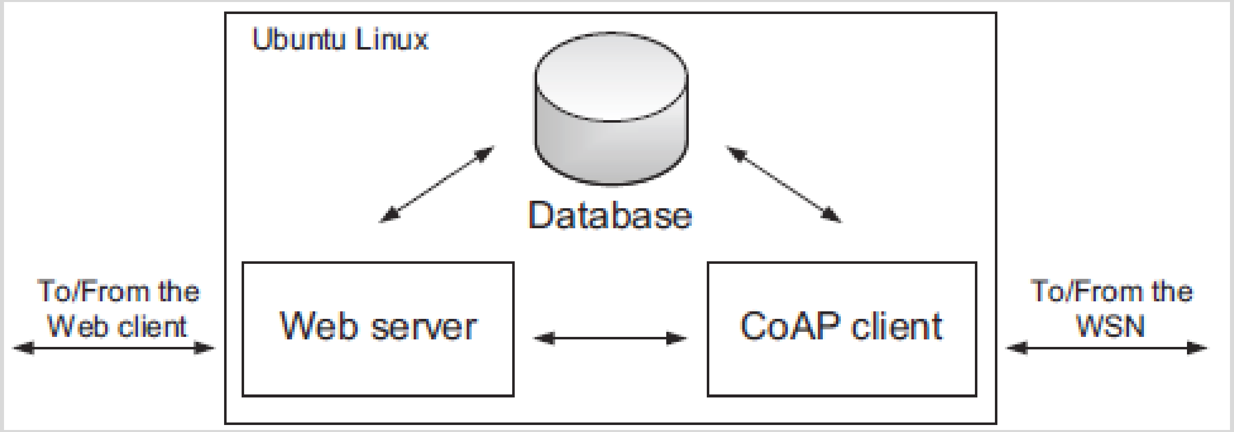
As CoAP implements a subset of the HTTP functionalities, there is a direct mapping between them.

- CoAP-HTTP Mapping
  - enables CoAP clients to access resources on HTTP servers through an intermediary.
  - Mapping is straightforward, requiring the translation of HTTP Status Codes to CoAP Response Codes.
- HTTP-CoAP Mapping
  - enables HTTP clients to access resources on CoAP servers through an intermediary.
  - Mapping requires filtering of those codes, options, and methods that are not supported by CoAP.

UNIVERSITÄT BERN

4. CoAP

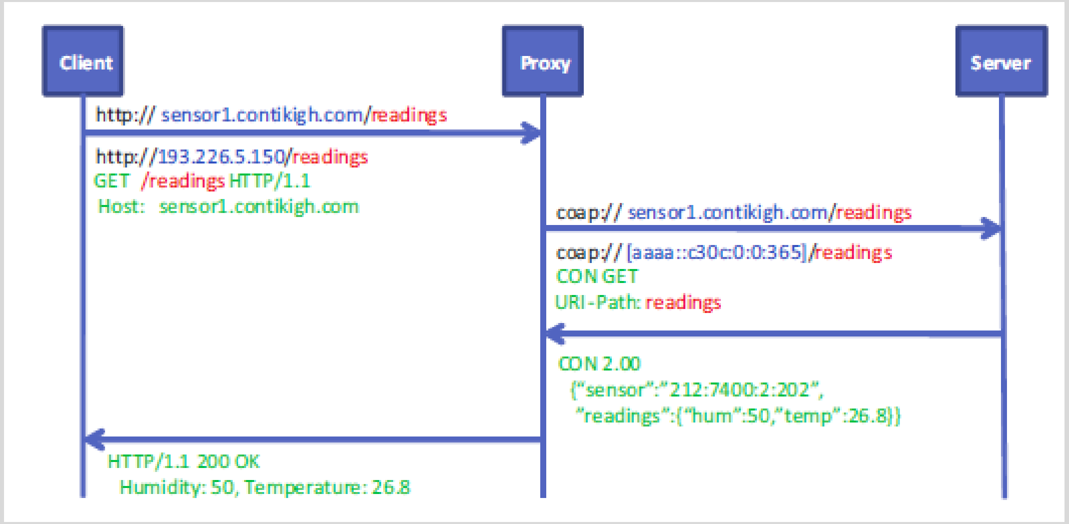
4.13.1 Web/WSN Integration by Proxies/Intermediaries



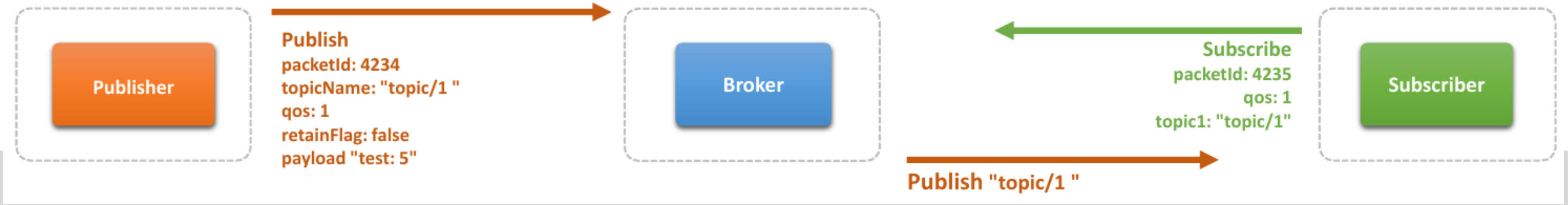
UNIVERSITÄT BERN

4. CoAP

4.13.2 Web/WSN Integration by Proxies/Intermediaries


Client
Proxy
Server
http:// sensor1.contikigh.com/readings
http://193.226.5.150/readings
GET /readings HTTP/1.1
Host: sensor1.contikigh.com
coap:// sensor1.contikigh.com/readings
coap:// [aaaa::c30c:0:0:365]/readings
CON GET
URI-Path: readings
CON 2.00
{"sensor":"212:7400:2:202",
"readings":{"hum":50,"temp":26.8}}
HTTP/1.1 200 OK
Humidity: 50, Temperature: 26.8

5. Message Queuing Telemetry Transport Protocol



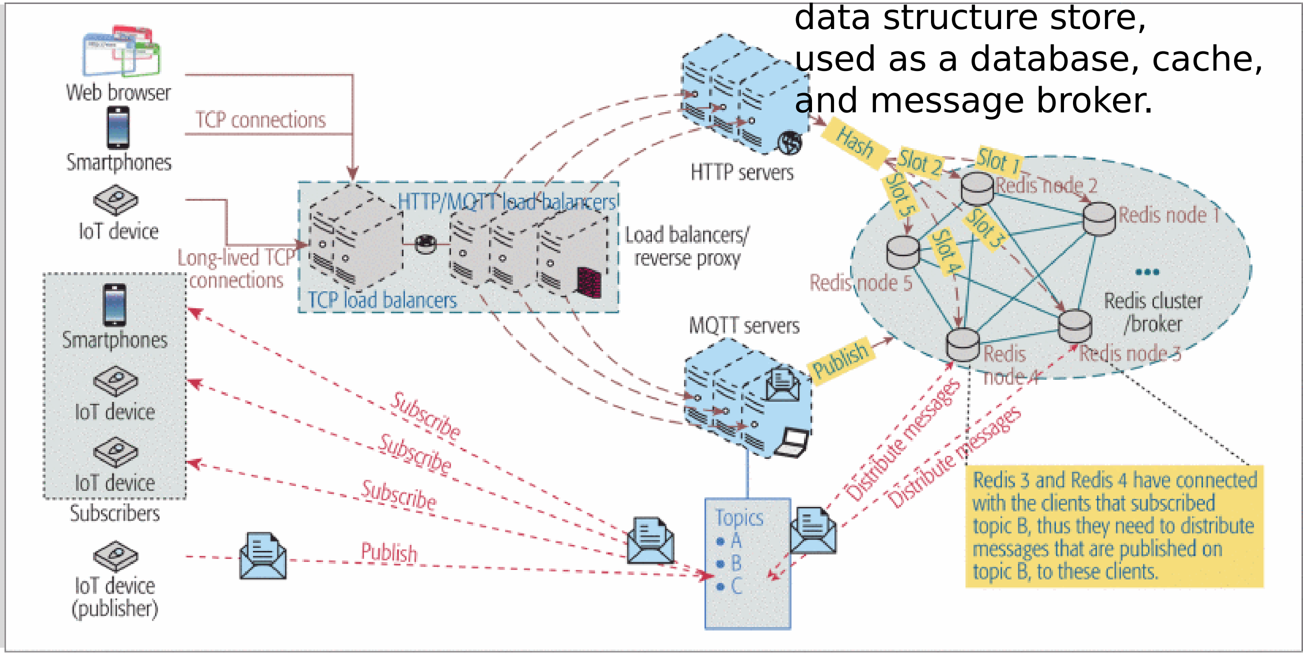
- Use of MQTT in The Things Network
- MQTT client publishes messages to MQTT broker.
- Every message is published to an address, known as a topic.
- Clients can subscribe to multiple topics and receive every message published to each topic.
- Messages may be retained for future subscription.
- MQTT is a binary protocol and requires fixed header of 2-bytes with small message payloads.
- TCP is used as transport protocol and TLS/SSL for security.
- MQTT is neither designed for device-to-device transfer nor for multicast data to many receivers.
- To enhance real-time performance of the MQTT servers, they have to maintain long-lived TCP connections with clients or devices.

UNIVERSITÄT BERN

5. MQTT

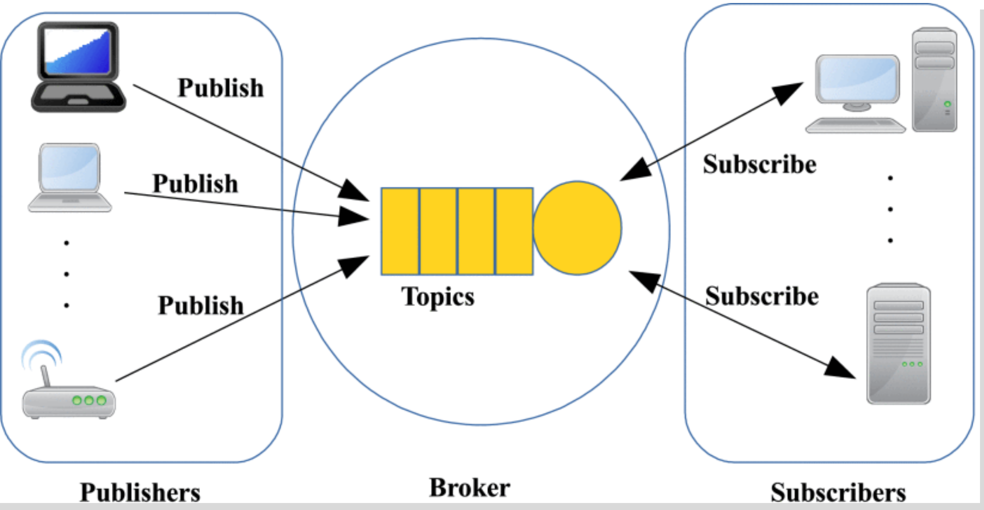
5.1 Scenario

Redis: open source in-memor
data structure store,
used as a database, cache,
and message broker.



5. MQTT

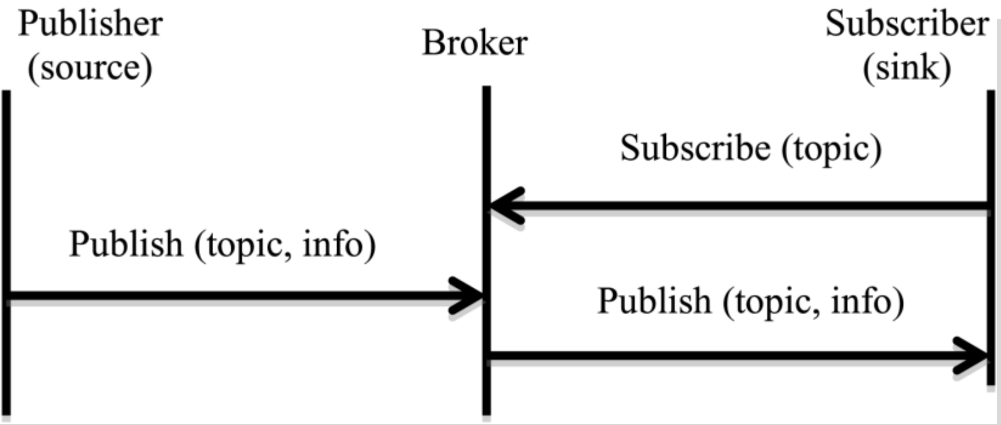
5.2 Publish / Subscribe



MQTT utilizes publish/subscribe pattern to provide transition flexibility and simplicity of implementation.

5. MQTT

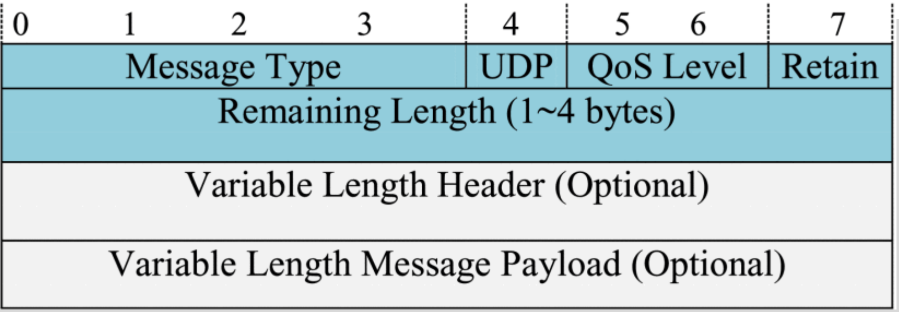
3. Components



- An interested device registers as a **subscriber** for specific topics to be informed by the broker when publishers publish topics of interest.
- **Publisher** acts as generator of interesting data and transmits information to the interested entities (subscribers) through the broker.
- **Broker** achieves security by checking authorization of the publishers and the subscribers.

5. MQTT

4. Message Format



0 1 2 3	4	5 6	7
Message Type	UDP	QoS Level	Retain
Remaining Length (1~4 bytes)			
Variable Length Header (Optional)			
Variable Length Message Payload (Optional)			

- *Message Type* indicates a variety of messages including Connect, ConnAck, Publish, Subscribe etc.
- *UDP* flag indicates that the message is duplicated and that the receiver may have received it before.
- *3 QoS levels* for delivery assurance of Publish messages are identified by the QoS Level field.
- *Retain* informs the server to retain the last received Publish message and submit it to new subscribers as a first message.
- *Remaining Length* shows the remaining length of the message, i.e., length of optional parts.

6. Advanced Message Queuing Protocol

1. Overview

AMQP

– is an open standard application layer protocol for message-oriented environments.

– provides reliable communication via message delivery guarantee primitives including at-most-once, at-least-once and exactly-once delivery.

– supports request/response and publish/subscribe.

– is a binary protocol and requires fixed header of 8-bytes with small message payloads.

– uses TCP as a default transport protocol and TLS/SSL for security.

b UNIVERSITÄT BERN

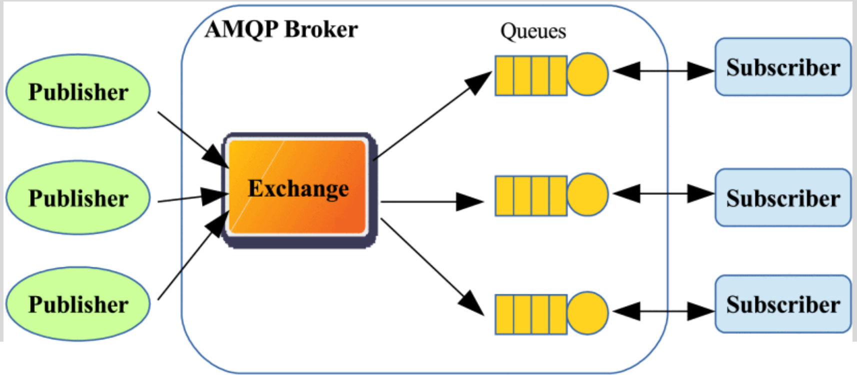
6. Advanced Message Queuing Protocol

2. Components and Communication Architecture

Components

– ***Exchanges*** are used to route the messages to appropriate queues.

– Messages can be stored in ***message queues*** and then be sent to receivers.



Communication Architecture

– Routing between exchanges and message queues is based on pre-defined rules and conditions.

– Publisher or consumer(subscribers) creates an “exchange” with a given name and then broadcasts that name.

– Publishers and consumers use the name of this exchange to discover each other.

– Subsequently, a consumer creates a “queue” and attaches it to the exchange at the same time.

– Messages received by the exchange have to be matched to the queue via a process called “binding”.

7. Data Distribution Service

1. Overview

DDS

– is a publish-subscribe protocol for real-time M2M communications developed by Object Management Group

– is a data-centric approach

– relies on a broker-less architecture, is decentralized and based on peer-to-peer communication.

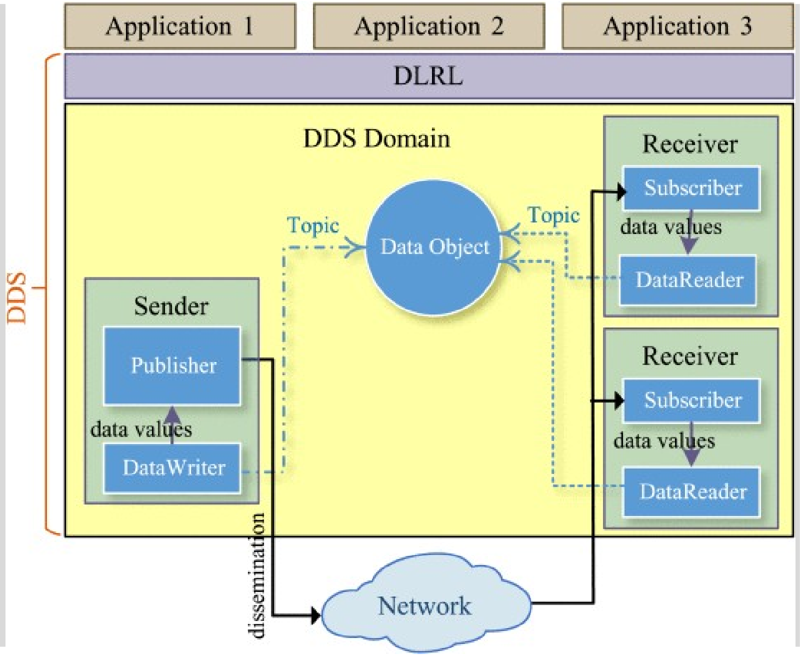
– supports 23 QoS policies by which a variety of communication criteria like security, urgency, priority, durability, reliability, etc. can be addressed by developer.

Discovery process allows subscribers to find out which publishers are present in Global Data Space.

7. Data Distribution Service

2. Architecture

- Layers
  - Data-Centric Publish-Subscribe
    - delivers information to subscribers.
  - Data-Local Reconstruction Layer (optional)
    - serves as interface to DCPS functionalities
    - facilitates sharing of distributed data among distributed objects
- Entities involved with data flow in DCPS layer
  1. Publisher disseminates data.
  2. DataWriter is used by the application to interact with the publisher about the values and changes of data.
  3. Subscriber receives published data and delivers them to the application.
  4. DataReader is employed by the Subscriber to access to the received data.
  5. A Topic is identified by a data type and a name. Topics relate DataWriters to DataReaders.



Thanks

for Your Attention

Prof. Dr. Torsten Braun, Institut für Informatik

Bern, 03.05.2021 – 10.05.2021