CoAP

REST

- The REST principle was developed by Roy Fielding in his PhD thesis and used for the specification of HTTP/1.1
- Some of the main ideas:
  - Clients can access resources on a server through a set of standard operations (read, write, delete,...)
  - Client-Server connections are stateless. The requests sent by the client contain all necessary information
  - Caching of resources on the client are possible.
- Many web services provide an API following the REST principles over HTTP.
- Example: Google Docs
  - Create a blank document: `POST /v1/documents`
  - Get a document: `GET /v1/documents/{documentId}`

Refresher: HTTP

- HTTP client sends HTTP query of the form

```
GET /path/to/resource HTTP/1.0
Header1: Value1
Header2: Value2
…
<empty line>
Body
```

- HTTP server responds with

```
1.0 200 Success
HeaderA: ValueA
HeaderB: ValueB
…
<empty line>
Body
```

Refresher: HTTP (2)

- HTTP methods GET, HEAD, OPTIONS PUT, POST, DELETE, TRACE, CONNECT
  - GET: retrieve resource
  - POST: send data to an URI (e.g. a service)
  - PUT: store data on the server at the specified URI

- Semantic is important for example for caching ("should I update the cache?") and for fault-tolerant implementations ("is it okay to resend the query if it was lost?")
  - GET, HEAD, OPTIONS, TRACE are "safe": they do not change any resources.
  - GET, HEAD, OPTIONS, TRACE and PUT are "idem-potent": re-execution does not change the outcome.

The REST principle for IoT and WSN

- The REST principle is also useful for WSN and IoT devices
- A device with a temperature sensor could for example provide an API to remotely read the sensor value:

```
GET /sensors/temp
```

or switch on the LED of the device:

```
POST /sensors/led?state=on
```

- Unfortunately, HTTP is not very suitable for this task: HTTP headers are often 700 bytes or more!

CoAP

- Constrained Application Protocol
- RFC 7252
- HTTP-inspired protocol designed for constrained devices and networks
- Typically used for RESTful access to resources

  ```
  GET coap://example.com:5683/sensors/temp
  ```

- UDP default port 5673 (`coap`) resp. 5684 (`coaps`)
- Key features:
  - Low overhead
  - Easy to implement
  - Simple mapping to HTTP for CoAP-HTTP proxies

CoAP

- Like HTTP, it is mainly a pull protocol: Clients send requests to servers to get the latest data
  - Server = the IoT device providing sensor data etc.
  - Client = some computer in the Internet reading this data
- Advantage: Communication only done when sensor data is needed by the client
- Disadvantage:
  - Wasting bandwidth if data is needed periodically (e.g. every 5 min).
  - Wasting bandwidth if several clients want the same data
- We will later see MQTT as an example for a push protocol
- Note: CoAP is an example for a *Machine-to-Machine* protocol. Typically the client is collecting the data for further processing

CoAP Layers

```
+----------------------+
|      Application     |
+----------------------+
+----------------------+  \
|  Requests/Responses  |  |
|----------------------|  | CoAP
|       Messages       |  |
+----------------------+  /
+----------------------+
|         UDP          |
+----------------------+
```

RFC 7252

- CoAP messages provide reliable messaging over UDP
- Note: CoAP is *one* protocol although it has two layers

Messages

- Message format shared by requests and responses

```
  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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |Ver| T |  TKL  |      Code     |          Message ID           |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   Token (if any) ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   Options (if any) ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |1 1 1 1 1 1 1 1|    Payload (if any) ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Header and options are in binary format
- Very compact. Goal is to avoid IP fragmentation and allow for easy parsing.

Header

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Ver = version number (01). Unknown versions are silently ignored.
  - Allows to mix different versions in a network without getting too many error messages

Methods and Responses

```
 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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |Ver| T |  TKL  |      Code     |          Message ID           |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Methods:
  - GET (code 0.01): retrieve information from URI
    POST (0.02): send data for processing at the URI
  - PUT (0.03): store (create/update) data at the URI
  - DELETE (0.04): delete resource at URI

- Response codes similar to HTTP response codes
  - 2.05: “ok” (as response to GET method)
  - 4.04: “not found”
  - …

Options

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Zero or more options
- Begin of payload (=end of options) marked by 11111111

Options

```
   0   1   2   3   4   5   6   7
+---------------+---------------+
|  Option Delta | Option Length |   1 byte
+---------------+---------------+
/         Option Delta          /   0-2 bytes
\          (extended)           \
+-------------------------------+
/         Option Length         /   0-2 bytes
\          (extended)           \
+-------------------------------+
/         Option Value          /   0 or more bytes
+-------------------------------+
```

- Options are identified by their option code
- Option code = Delta + previous option’s code
- Option length = length of option in bytes
- Special values indicate the presence of extension fields:
  - Delta=13 means “1 byte delta value follows”
  - Delta=14 means “2 bytes delta value follows”
  - etc.

URIs in Options

- A CoAP URI like

  ```
  coap://example.com:5683/sensors/temp?x=1&y=2
  ```

  is not stored as one big string in a request.
- Instead:
  - Host name stored in *Uri-Host* option (option code 3)
  - Port in *Uri-Port* option (option code 7)
  - Path segments in one or more *Uri-Path* options (without /)
  - Query arguments in one or more *Uri-Query* options (without ? and &)

- Simplifies parsing! Perfect for devices with weak CPU and small memory

Token

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Token is a 0 to 8 bytes value (length defined in TKL field)
- Generated by the client: *Concurrent* requests to the same server should have a unique token value
- Response will use same token as the request
- Responses with unexpected token number are rejected by the client

Security and Tokens

- CoAP connections can use DTLS (URI “coaps://…”)
  - DTLS = Datagram Transport Layer Security = TLS-based security protocol for datagram-oriented applications (UDP,…)
- Clients not using DTLS are strongly advised to use long randomized token values
  - prevents response-spoofing attacks (attacker chooses a random token value and sends fake response to client)

Reliable Messaging?

- So far, we have only talked about the *content* of the messages
- How is reliable messaging over UDP achieved?

```
+----------------------+
|      Application     |
+----------------------+
+----------------------+  \
|  Requests/Responses  |  |
|----------------------|  | CoAP
|       Messages       |  |
+----------------------+  /
+----------------------+
|          UDP         |
+----------------------+
```

Message Types

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- T field in header:
  - Confirmable (0)
  - Non-confirmable (1)
  - Acknowledgment (2)
  - Reset (3)

Transmission WITHOUT Reliability

- Lightweight, useful for repeated operations, e.g. repeated reading of a sensor value
  - Type = "Non-confirmable"
- Message might be sent multiple times by sender or network
  - Unique message ID to identify copies:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Message is ignored by recipient if unexpected or ill-formatted (unknown codes, wrong token value, etc.)
  - Recipient *might* send a "Reset" message (with same message ID) in that case

Transmission WITH Reliability

- Message type is “Confirmable”
- Message ID crucial! Not only for de-duplication!
  - Recipient *must* either reply with
    - Acknowledgment (with same msg ID), or
    - Reset (with same msg ID) and ignore the message
  - Sender retransmits message until ACK or Reset received (or runs out of attempts)

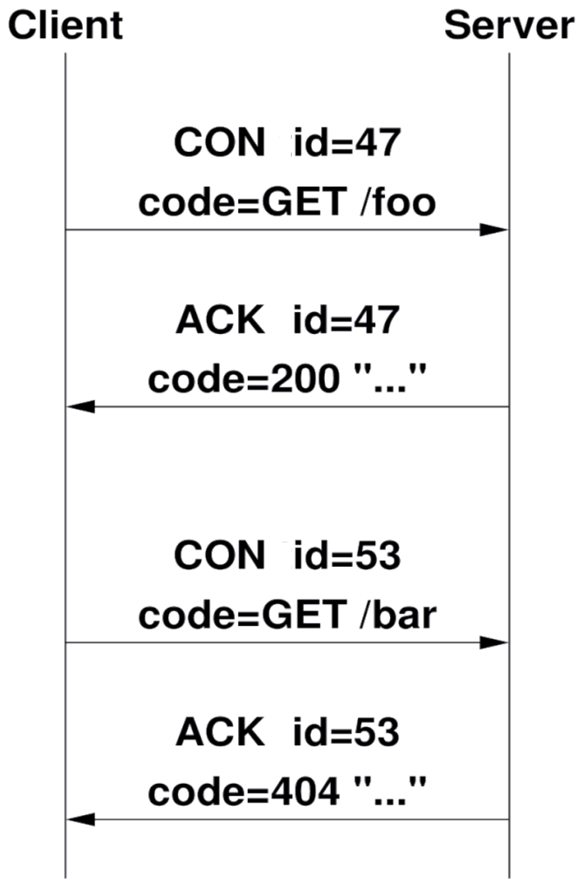
Retransmissions

- Exponential back-off
  1. Initial timeout := random duration in
     [ACK_TIMEOUT, ACK_TIMEOUT*ACK_RANDOM_FACTOR]
  2. retransmission-counter := 0
  3. Send CON message
  4. Wait until timeout or ACK or Reset received
  5. If no reply and
     retransmission-counter<MAX_RETRANSMIT then
     - timeout := timeout*2
     - retransmission-counter++
     - Goto step 3
- Default values:

  TIMEOUT=2s, FACTOR=1.5, MAX_RETRANSMIT=4

Synchronous Message Exchange

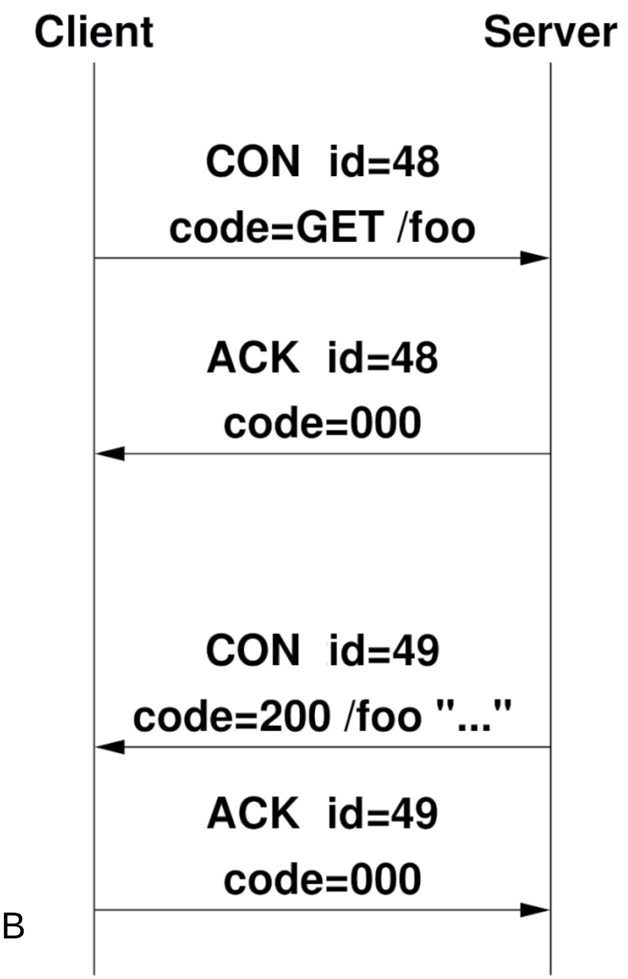
- Request is sent as CON message
- Response is sent as ACK message (piggybacking)
- Efficient



J. Schönwälder, JUB

Asynchronous Message Exchange

- Request sent as CON message
- Recipient replies immediately with empty ACK message (Empty = code 0.00)
- Recipient replies later with actual response in CON message
  - Tokens are used to match the request and the response
- More overhead than synchronous messages
- CoAP does *not* define how long requester should wait for response



Remarks on Messaging Mechanism

- CON and NON-CON messages can be mixed, e.g., request as NON-CON, response as CON
- Empty CON (code 0.00) can be used as “ping” message: Recipient replies with Reset message.

Multicasting and Discovery

- CoAP supports requests to IP multicast groups
- Different behavior if request is multicast:
  - Servers *must not* reply Reset to unwanted NON-CON messages (avoid too many error responses)
  - Servers should wait random time before responding to request (avoid congestion)
- Multicasting useful for automatic service discovery in M2M scenarios

- Servers *should* also support resource discovery (RFC 6690)
  - Client sends `GET /.well-known/core` to server
  - Server returns list of resources

Observing resources (RFC 7641)

- CoAP is mainly a pull protocol: If a client wants the latest sensor value, it has to send a request to the device
- With the “Observe” option, a client can register itself as an observer for a specific resource
  1. Client sends GET request with Observe=0
  2. Server sends response (with same token) whenever the requested resource has changed
  3. Client sends GET request with Observe=1 to deregister

Block-wise transfer (RFC 7959)

- Remember that UDP does not provide segmentation, and IP fragmentation is not recommended on IEEE 802.15.4
- How to access large resources where the response does not fit into a IEEE 802.15.4 frame?
  - Block option: Send multiple GET requests ("I want bytes 0..1023 of the resource", "I want bytes 1023..2047",…)

Summary

- CoAP is a HTTP-inspired protocol designed for constrained environments
- Features:
  - Uses techniques that we have already seen in 6LoWPAN: very compact header, compression
  - Simplified parsing saves resources
  - Reliable message transmissions over UDP: Alternative to “expensive” TCP
  - Support for multicast and discovery