RIOT

… in the Internet of Things

Bachelor Project (PO)
Introduction to CoAP

Hamburg 04.04.2022

José Álamos	jose.alamos@haw-hamburg.de
Peter Kietzmann	peter.kietzmann@haw-hamburg.de
Leandro Lanzieri	leandro.lanzieri@haw-hamburg.de

Prof. Dr. Thomas C. Schmidt INET AG, Dept. Informatik HAW Hamburg t.schmidt@haw-hamburg.de

CoAP: Constrained Application Protocol

RESTful APIs for the IoT

Why do we need a web protocol for IoT?

- Web services on the Internet nowadays expose RESTful APIs
- Avoid fragmentation of IoT by
  - Using and extending existing standard Web technologies
  - Providing standardized metadata
  - Integrating platforms, underlying protocols and application domains

Why do we need another web protocol?

- HTTP does not fit the constrained devices commonly found in the IoT:
    - Many 8-bit microcontrollers
    - Limited RAM and ROM
    - Battery-powered or severely energy constrained
    - Lossy wireless networks (e.g., 6LoWPAN)
    - Unreliable transports
    - Small link-layer frames

CoAP: Features

- Low header overhead and parsing complexity
- Supports URIs and Content-type
- Optional reliability
- Unicast and multicast requests
- Defined over multiple transports (including DTLS for security)

- For detailed information:
  - RFC 7252
  - https://coap.technology

REST model interactions

- **Servers** expose resources under URLs:

  `coap://node1.example.com/temperature`

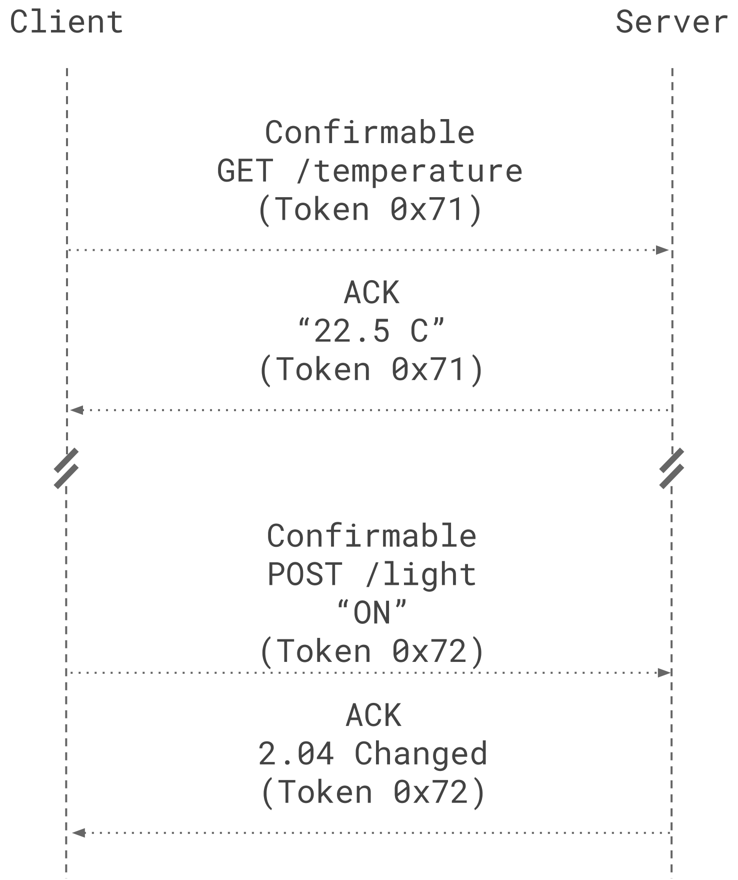
- **Clients** operate on the resources utilizing methods:
  - GET
  - POST
  - PUT
  - DELETE
- The semantics of each method will depend on the specific application

REST model interactions

- **Servers** expose resources under URLs:

  `coap://node1.example.com/temperature`

- **Clients** operate on the resources utilizing methods:
  - GET
  - POST
  - PUT
  - DELETE
- The semantics of each method will depend on the specific application



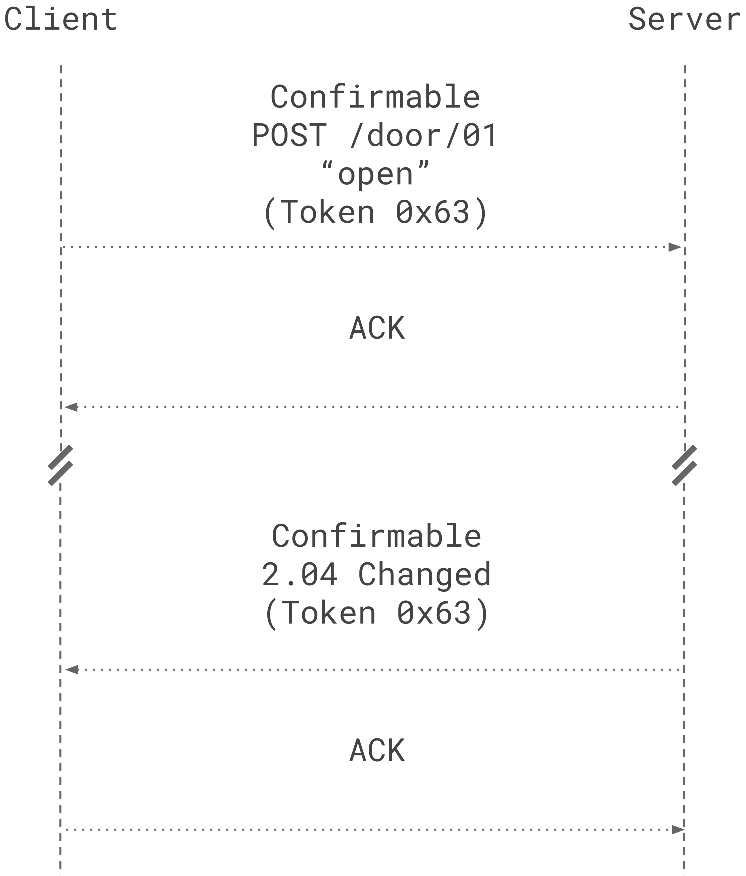
REST model interactions: separate response

- Server responses may be separate due to:
  - Long response processing time.
  - “Real-world” actions (e.g. switching a lock).

- Servers confirm requests by sending an ACK, and send responses at a later time, with a matching token.

REST model interactions: separate response

- Server responses may be separate due to:
  - Long response processing time.
  - “Real-world” actions (e.g. switching a lock).
- Servers confirm requests by sending an ACK, and send responses at a later time, with a matching token.

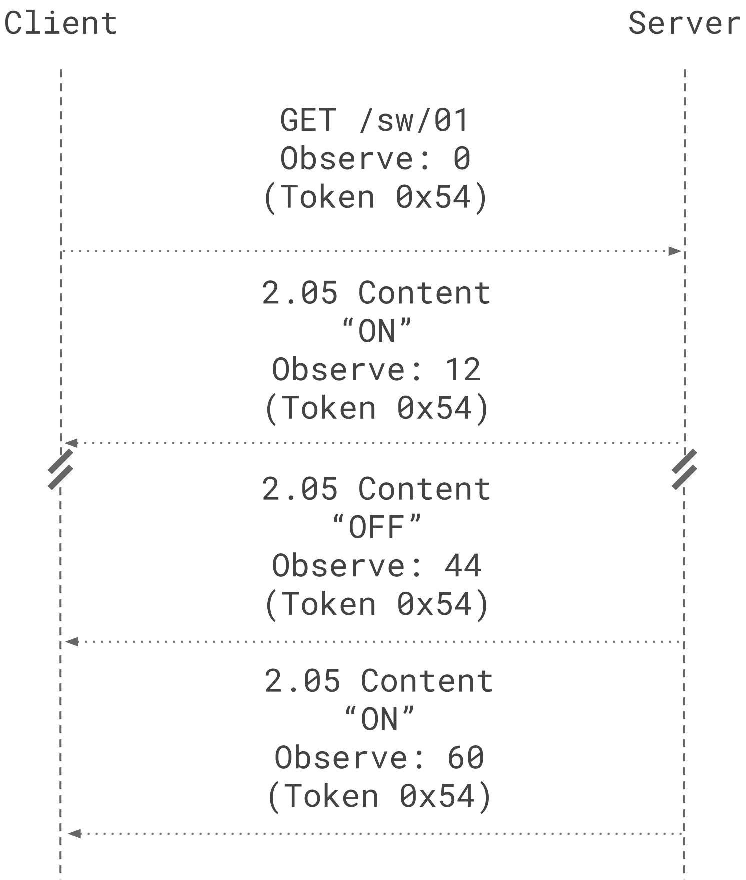


REST model interactions: observation

- Resources may change over time (e.g. the value of a light switch).
- Periodically polling resources consumes a lot of energy and bandwidth.
- The **observe** extension allows clients to request for notifications whenever the resource has changed (this is up to the server to determine).

REST model interactions: observation

- Resources may change over time (e.g. the value of a light switch).
- Periodically polling resources consumes a lot of energy and bandwidth.
- The **observe** extension allows clients to request for notifications whenever the resource has changed (this is up to the server to determine).

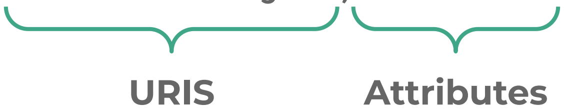


Resource discovery: /.well-known/core

- Clients can discover which resources a given server provides
- The interface accepts GET requests, and returns a list of resources in LinkFormat:

```
Client Request:
   GET /.well-known/core

Server Response:
   2.05 Content
   </sensors/temp>;if="sensor",
   </sensors/light>;if="sensor"
```



Resource discovery: resource directory

- In some scenarios direct discovery of resources may not be possible
  - Long-sleeping nodes
  - Multicasting not efficient
- Resource Directories (RD) contain information about resources in other servers
- A Resource Directory has two interfaces
  - Registration interface: servers register their resources
  - Lookup interface: clients look for resources exposed by servers

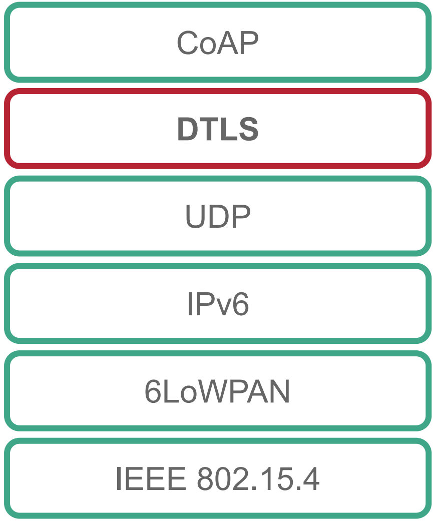
Resource discovery: resource directory

Operation flow

1. The server finds the RD
    - Statically configured
    - Discovery procedure (e.g. multicast)
2. The server **registers** itself on the RD by sending information about its resources
    - The server may periodically update the registration
3. The client performs a **lookup** on the RD, to find a resource with specific characteristics
    - It may use the observe mechanism to be notified about new resources

Securing CoAP: DTLS

- Datagram Transport Layer Security
  - Four different modes
    - NoSec: no protocol-level security
    - PreSharedKey: Symmetric keys
    - RawPublicKey: Asymmetric keys
    - Certificate: Asymmetric keys with X.509 certs.
  - Nodes establish a point-to-point DTLS session
    - Provides authentication, integrity, and confidentiality
    - Intermediate nodes (e.g., gateways) need to decrypt and re-encrypt
      - Difficult to cache
      - Difficult to proxy



CoAP

DTLS

UDP

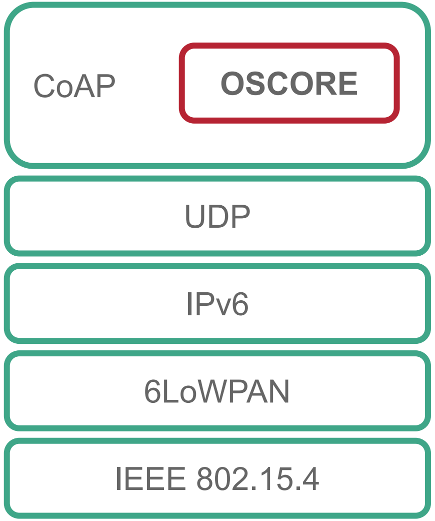
IPv6

6LoWPAN

IEEE 802.15.4

Securing CoAP: OSCORE

- Object Security for Constrained RESTful Environments
    - Uses pre-shared keys
    - Security at object level (no point-to-point session)
        - The original CoAP message is encrypted and encapsulated as a COSE object (CBOR Object Signing and Encryption)
        - The encapsulated message is nested in an outer CoAP message
        - Provides integrity, authenticity, and confidentiality **at CoAP level**
        - Allows protecting multicast messages
        - Allows caching and proxies



Questions?