CoAP

The Web-based Application-layer Protocol for the Internet of Things

Matthias Kovatsch
Follow the Slides

http://goo.gl/anfy5w
About the Speaker

Matthias Kovatsch

Researcher at ETH Zurich, Switzerland with focus on Web technology for the IoT

IETF contributor in CoRE and LWIG

Author of Californium (Cf), Erbium (Er), and Copper (Cu)

http://people.inf.ethz.ch/mkovatsc
Agenda

The Web of Things
The Constrained Application Protocol
Building RESTful IoT Applications
Getting Started
   Erbium (Er) REST Engine
   Californium (Cf) CoAP framework
   mjCoAP
The Web of Things (WoT)
The Application Layer for the IoT

Well-known patterns

Web mashups

Cloud services
Interoperability and Usability

HTTP libraries exist for most platforms
HTTP is the basis for many of our services
Web patterns are well-known
Scripting increases productivity

“Kids” can program Web applications

The Web fosters innovation!
Tiny Resource-constrained Devices

Class 1 devices
~100KiB Flash
~10KiB RAM

Low-power networks
Tiny Resource-constrained Devices

Target
of less than $1

TCP and HTTP
are not a good fit
Constrained Application Protocol
Constrained Application Protocol

RESTful protocol designed from scratch
Transparent mapping to HTTP
Additional features for IoT applications

CoAP

Request-Response Sub-layer
RESTful interaction

Message Sub-layer
Reliability

GET, POST, PUT, DELETE
URIs and Internet Media Types

Deduplication
Optional retransmissions

UDP
DTLS
...
Constrained Application Protocol

Binary protocol
- Low parsing complexity
- Small message size

Options
- Numbers in IANA registry
- Type-Length-Value
- Special option header marks payload if present

4-byte CoAP Base Header
Version | Type | T-len | Code | ID

0 - 8 Bytes Token
Exchange handle for client

Options
Location, Max-Age, ETag, ...

Marker
0xFF

Payload Representation
CoAP Option Encoding

Delta encoding
- Option number calculated by summing up the deltas
- Compact encoding
- Enforces correct order

Extended header
- Jumps to high opt. numbers
- No limitation on opt. length
- Possible values
  - +0 bytes: 0 – 12
  - +1 byte: 13 – 268
  - +2 bytes: 269 – 65,804

Option Value
- empty, opaque, uint, or string

1-byte Option Header
- 4-bit Delta
- 4-bit Length

Extended Option Delta
- +1 byte for Delta=13
- +2 bytes for Delta=14

Extended Option Length
- +1 byte for Length=13
- +2 bytes for Length=14

max. UDP length is 65k
Option Metadata

- **Critical (C)**
  - Message must be rejected if unknown
  - Elective options be be silently dropped

- **UnSafe (U)**
  - Proxies may forward messages with unknown options
  - Unless they are marked unsafe

- **NoCacheKey (N)**
  - Option is not part of the cache key when all three bits are 1 and U=0
# Registered Options (RFC 7252)

<table>
<thead>
<tr>
<th>#</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>If-Match</td>
<td>opaque</td>
<td>0-8</td>
<td>(none)</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>Uri-Host</td>
<td>string</td>
<td>1-255</td>
<td>IP literal</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>ETag</td>
<td>opaque</td>
<td>1-8</td>
<td>(none)</td>
</tr>
<tr>
<td>5</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>If-None-Match</td>
<td>empty</td>
<td>0</td>
<td>(none)</td>
</tr>
<tr>
<td>7</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>Uri-Port</td>
<td>uint</td>
<td>0-2</td>
<td>UDP port</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>Location-Path</td>
<td>string</td>
<td>0-255</td>
<td>(none)</td>
</tr>
<tr>
<td>11</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>Uri-Path</td>
<td>string</td>
<td>0-255</td>
<td>(none)</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Content-Format</td>
<td>uint</td>
<td>0-2</td>
<td>(none)</td>
</tr>
<tr>
<td>14</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>Max-Age</td>
<td>uint</td>
<td>0-4</td>
<td>60</td>
</tr>
<tr>
<td>15</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>Uri-Query</td>
<td>string</td>
<td>0-255</td>
<td>(none)</td>
</tr>
<tr>
<td>17</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>Accept</td>
<td>uint</td>
<td>0-2</td>
<td>(none)</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>Location-Query</td>
<td>string</td>
<td>0-255</td>
<td>(none)</td>
</tr>
<tr>
<td>35</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>Proxy-Uri</td>
<td>string</td>
<td>1-1034</td>
<td>(none)</td>
</tr>
<tr>
<td>39</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>Proxy-Scheme</td>
<td>string</td>
<td>1-255</td>
<td>(none)</td>
</tr>
<tr>
<td>60</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>Size1</td>
<td>uint</td>
<td>0-4</td>
<td>(none)</td>
</tr>
</tbody>
</table>
Example

Request
- POST
- Resource /examples/postbox
- Content-Format text/plain
- 1000 bytes payload

Encoding
- Uri-Path is Option 11
- Uri-Path is repeatable
- Content-Format is 12
- text/plain is 0
- Size1 is Option 60
- Option Number = Current number + Delta + Extended Delta = 12 + 13 + 35
Retransmission of Confirmables

Retransmission after 2–3 s
Randomized timeout to avoid synchronization effects

Binary Exponential Back-Off
Timeout is doubled after each retransmission

Retransmissions stop when CON is acknowledged or 4 retransmissions failed (here 3rd one is successful)

Receiver must send an Acknowledgement (ACK) for CONs

Non-Confirmables are best-effort messages without retransmissions
Requests and Responses

ACK is matched to CON through MID

Response is matched to request through Token

why?
Separate Responses

Client

Server

ACKs are empty when the code is zero

Token: 0xBEEF00
Content-Format: text/plain

hello separate

Separate Response has different MID
Implicit Acknowledgements

Client

01 CON 3 GET 0x4713
Token: 0x00FEED
Uri-Path: separate

Response implicitly ack's CON
Retransmissions MAY be stopped

Server

01 ACK 0 0.00 0x4712

01 CON 3 2.05 0x0816
Token: 0x00FEED
Content-Format: text/plain
0xFF hello separate
Mixed Separate Responses

NON requests always elicit separate responses (can be NON or CON)

Token: 0x0DECAF
Content-Format: text/plain

hello surprise
Reset Messages

Cannot match token and must reject CON

NONs can be rejected or silently dropped

Token: 0xDEAD

Content-Format: text/plain

hello separate
Deduplication

Worst case transmission

Store sender+MID to filter duplicates

Can be relaxed! (idempotent requests)
Features for the IoT
Observing Resources

Resource state at origin server

Server

Client

Replicated state at client

Observe illustration courtesy of Klaus Hartke

GET

Observe

Notification

Response with Observe opt.

Notification

Notification

Notification lost

Max-Age
Observing Resources - CON Mode

Mode depends on application
- random events: CON
- periodic samples: NON
Group Communication

GET /status/power

PUT /control/onoff

PUT /control/color
    #00FF00

What exactly is RESTful group communication?
Resource Discovery

Based on **Web Linking** (RFC5988)
Extended to **Core Link Format** (RFC6690)

GET /.well-known/core

```xml
</config/groups>;rt="core.gp";ct=39,
</sensors/temp>;rt="ucum.Cel";ct="0 50";obs,
</large>;rt="block";sz=1280,
</device>;title="Device management"
```

Decentralized discovery
Infrastructure-based

**Multicast Discovery**
**Resource Directories**
Alternative Transports

e.g.,
Short Message Service (SMS)

Addressable through URIs

coap+sms://+123456789/bananas/temp*

Could power up subsystems for IP connectivity after SMS signal

* illustration only, +123456789 unfortunately not allowed by URI
Security

Based on **DTLS** (TLS/SSL for Datagrams)
Focus on Elliptic Curve Cryptography (**ECC**)  
Pre-shared secrets, certificates, or raw public keys

Hardware acceleration in IoT devices e.g.,

IETF is currently working on
- Authentication/authorization (ACE)
- DTLS profiles (DICE)
Status of CoAP

“Proposed Standard” since 15 Jul 2013

RFC 7252

Next working group documents in the queue

- Observing Resources
- Group Communication
- Blockwise Transfers

- Resource Directory
- HTTP Mapping Guidelines
Status of CoAP

In use by

- OMA Lightweight M2M
- IPSO Alliance
- ETSI M2M / OneM2M

- Device management for network operators
- Lighting systems for smart cities
- Innovative products, e.g., Spark.io
Building RESTful IoT Applications
What is REST?

Representational State Transfer is the architectural style that powers the World Wide Web

(i.e., a set of constraints applied to the elements within the architecture)
Client-Server Constraint

How to connect the components?

Separation of concerns

- Origin servers provide the data through a server connector
- User-Agents provide the user/application interface and initiate interaction through a client connector

⇒ components can evolve independently
Stateless Constraint

How to do request-response?

REST is all about state in a distributed system!

Requests are constrained by “Stateless”

Each request must contain all the information to understand the request so that servers can process it without context (the state of the client)

⇒ visibility, reliability, and scalability
Application as Finite State Machine

Servers store data that is independent from the individual client states (resource state).

Only the clients keep application state (session/client state)
Cache Constraint

Responses to requests must have implicit or explicit cache-control metadata.

Clients and intermediaries can store responses and re-use them to locally answer future requests.

⇒ efficiency, scalability, and user-perceived performance.
Uniform Interface Constraint

All RESTful Web services use the same interfaces that are defined by:

- **URIs** to identify resources
- **Representations** to manipulate resources
  - State transfer
  - No RPC-like service calls
- **Self-descriptive messages** (also see Stateless)
  - Well-defined media types
  - Standard set of methods
  - Independent from transport protocol
- **HATEOAS**...
Hypermedia As The Engine Of Application State

Clients start from an entry URI or a bookmark.
Response to the GET request has a hypermedia representation.
It contains Web links that define the transitions of the FSM.
Client chooses link to follow and issues the next requests (i.e., triggers a transition).
URIs and possible transitions are never hardcoded into the client: the client “learns” the application on the fly through the media type and link relations.

However, it can also go back.

⇒ loose coupling to evolve independently.
Hypermedia

Media types define

- the **representation format** as well as
- the **processing model**

for the data model of a Web resource

HTML is easy: humans can reason!

What about **machine-to-machine**?
Internet Media Types  (formerly known as MIME)

application/xml or application/json or text/plain
(reusable, meaningless)

application/senml+json
(reusable, standardized)

application/prs.my-actuator-control
(personal, meaningful)

In general

**Reuse** media types as far as possible
(http://www.iana.org/assignments/media-types/media-types.xhtml)

**Standardize** your own if nothing fits internally or globally (RCF 6838)
Layered System Constraint

**Intermediaries** can be placed at various points to modify the system
- Caching proxies
- Load balancers
- Firewalls
- Gateways to connect legacy systems

Fully transparent, as one layer cannot see beyond the next layer

⇒ adaptability, scalability, security
(Code-On-Demand)

Optional, easy to understand, and a constraint that does not constrain, but important for the Web as we know it.

Allows to update client features after deployment, e.g., JavaScript in the browser to improve the user interface.

⇒ improves system extensibility but reduces visibility.
Summary

Elements of a RESTful architecture

- User agents (client connectors)
- Origin servers (server connectors)
- Intermediaries (client and server at once)
- Data (resources, representations, metadata)

REST Constraints

- Client-Server
- Stateless
- Cache
- Uniform Interface (HATEOAS!)
- Layered System
- (Code-On-Demand)
How to Become RESTful
for object- and service-oriented people

Richardson Maturity Model

http://martinfowler.com/articles/richardsonMaturityModel.html
Level 0: The Swamp of POX

(POX = plain old XML)

Happens when HTTP is just used as transport protocol or tunnel because “port 80/443 is safe and always open”

The Web service only has a single URI and clients post RPCs that trigger an action (e.g., WS-* and JSON-RPC)
Level 1: Resources

Expose each service entity as Web resource with individual URI for global addressability

But Level 1 still uses RPCs

- POST /sensors/temperature?method=read
- POST /sensors/temperature?method=configure
  
  ```json
  {"a":3,"b":4}
  ```
Level 2: HTTP CoAP Verbs :)  

...and of course representations to manipulate resources

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>safe and idempotent</td>
</tr>
<tr>
<td>POST</td>
<td>not safe and not idempotent</td>
</tr>
<tr>
<td>PUT</td>
<td>not safe but idempotent</td>
</tr>
<tr>
<td>DELETE</td>
<td>not safe but idempotent</td>
</tr>
</tbody>
</table>

safe: no side-effects on the resource
idempotent: multiple invocations have the same effect as a single invocation
Level 2: HTTP CoAP Verbs :)  

Verbs map to CRUD operations:

**POST** Create a new (sub-)resource
- request body can have initial state
- response body can be an action result
  - **Location-*** options can contain link to new resource

**GET** Read the resource state
- no request body
- response body has representation

**PUT** Update the resource state
- request body has updated representation
- response can have only code or action result in body

**DELETE** Delete the resource
- no request body
- response can have only code or action result in body
Level 2: Still not REST (but helpful)

Level 2 API specifications usually look like this:

- `/config/profile`
  - **GET**
    - **Request**: no parameters
    - **Response**: `application/json`

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int</td>
<td>identifier of the profile</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>name of the profile</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **PUT**
  - **Request**: `application/json`

- `/actuators/pump`

Main problem: **tight coupling**

(hard-coded URIs, non-reusable message descriptions)
Level 3: Hypermedia Controls

HATEOAS

- Define media types for the application
- Embed links to drive application state
- Provide initial URI

“A REST API should spend almost all of its descriptive effort in defining the media type(s) used for representing resources and driving application state, or in defining extended relation names and/or hypertext-enabled mark-up for existing standard media types.”

Level 3 IoT Applications?

Sensors and actuators are rather easy to model

- Resources that provide sensor data
- Resources that provide and accept parameters

All CoAP nodes provide an initial URI

`.well-known/core`

First reusable media types for IoT applications

- `application/link-format` ([RFC 6690](https://tools.ietf.org/html/rfc6690))
- `application/senml` ([draft-jennings-senml](https://datatracker.ietf.org/doc/draft-jennings-senml/))
- `application/coap-group+json` ([draft-ietf-core-groupcomm](https://datatracker.ietf.org/doc/draft-ietf-core-groupcomm/))
Level 3 IoT Applications?

The CoRE Link Format provides **attributes** for links

- Can be more detailed than relation names
- Give meaning to generic media types
- Single values/parameters can be in text/plain

**Bad because of “typed resources”?**

We are just at the beginning!

⇒ **Bottom-up semantics for M2M**
Other REST Mechanisms

Exception Handling with response codes
- 4.xx Client errors
- 5.xx Server errors

Content negotiation
- Accept option

Conditional requests for concurrency
- ETag
- If-Match
- If-None-Match
CoAP and REST

Questions?
CoAP live with Copper!

CoAP protocol handler for Mozilla Firefox

Browsing and bookmarking of CoAP URIs

Interaction with Web resources like RESTClient or Poster

Treat IoT devices like RESTful Web services
CoAP live with Copper!

Available sandboxes

coap://iot.eclipse.org/
coap://vs0.inf.ethz.ch/
coap://coap.me/
Erbium (Er) REST Engine

Contiki OS CoAP implementation

- written in C
- focus on small footprint but also usability

For

- Thin Server Architecture
  (thus, minimal client support)
- RESTful wrapper for sensor/actuator hardware
Two Layers (Contiki apps)

**rest-engine**
- Web resource definitions
- RESTful handling of requests
- users implement resource handlers

**er-coap**
- CoAP implementation
- maps REST functions to protocol
- hides protocol-specific operations
Code Structure

Keep it modular

- er-coap / er-coap-constants
- protocol format & parsing
- er-coap-engine
- control flow (client/server)
- er-coap-transactions
- retransmissions
- er-coap-separate
- er-coap-observe
- er-coap-block
- er-coap-res-well-known...
- er-coap-conf
- tweak for application needs

process (e.g., er-example-server.c)

- start engines
- activate resources
- handle user events

coap_engine (er-coap-engine.c)

- activate /.well-known/core
- open socket
- receive messages
- handle retransmissions

rest_engine_process (rest-engine.c)

- set periodic timers
- call periodic handlers
Resource Handler API

Create a C module for each resource
- choose resource type (see next slide)
- set CoRE Link Format information
- implement resource handlers
  - GET
  - POST
  - PUT
  - DELETE
    or set to NULL for 4.05 Method Not Allowed
- activate resources in main process
Five Resource Macros

- **RESOURCE**
  simple CoAP resource

- **PARENTRESOURCE**
  manages virtual sub-resources (e.g., for URI-Templates)

- **SEPARATERESOURCE**
  long-lasting handler tasks $\Rightarrow$ separate responses

- **EVENTRESOURCE**
  observable resource that is manually triggered

- **PERIODICRESOURCE**
  observable resource that is triggered by timer
Minimal Client API

One call to issue requests

\texttt{COAP\_BLOCKING\_REQUEST()}

Call blocks until response is received

- linear program code for interactions
- also handles blockwise transfers

Working on \texttt{COAP\_ASYNC\_REQUEST()}

- support for observe and separate response
- some projects already have custom solutions
Californinium (Cf) CoAP framework

Unconstrained CoAP implementation

- written in Java
- focus on scalability and usability

For

- IoT cloud services
- Stronger IoT devices
  (Java SE Embedded or special JVMs)
3-stage Architecture

Stages
- Decoupled with message queues
- Independent concurrency models
- Adjusted statically for platform/application
- Stage 1 depends on OS and transport
- Stage 2 usually one thread per core
Stage 3: Server Role

Web resources
- Optional thread pool for each Web resource
- Inherited by parent or transitive ancestor
- Protocol threads used if none defined
Stage 3: Client Role

Clients with response handlers

- Object API called from main or user thread
- Synchronous: Protocol threads unblock API calls
- Asynchronous: Optional thread pools for response handling (e.g., when observing)
Endpoints

Encapsulate stages 1+2

Enable
- multiple channels
- stack variations for different transports

Individual concurrency models, e.g., for DTLS
Endpoints

Implemented in CoapEndpoint

Separation of bookkeeping and processing

Exchanges carry state
mjCoAP
mjCoAP

Java-based CoAP implementation

Focuses on fast and easy development of CoAP-based applications

Lightweight (small footprint) and compatible with Java-enabled devices
- Java SE
- Embedded Java/Java ME
mjCoAP

mjCoAP provides a simple set of APIs for creating server-side and client-side applications

Design principles:
- asynchronous (callback)
- lightness
- easy-to-use/fast-development
- re-usability (can be used to implement other protocols that share the same message syntax - e.g. CoSIP\(^1\))

Paper on mjCoAP

Layered Architecture

mjCoAP is formed by 3 sub-layers

CoAP application

CoAP Transaction
(CoapTransactionClient, CoapTransactionClientListener, CoapTransactionServer, CoapTransactionServerListener)

CoAP Reliable transmission
(CoapReliableTransmission, CoapReliableTransmissionListener, CoapReliableReception)

CoAP Messaging
(CoapProvider, CoapProviderListener)

Transport (UDP or DTLS)
**Messagging layer**

- Responsible for sending and receiving CoAP messages over UDP

- At this layer, all messages are handled without inspection (requests and responses, CON/NON/ACK/RST)

- Main classes:
  - `CoapProvider` (send messages)
  - `CoapProviderListener` (receive messages with `onReceivedMessage()` callback)
Reliable Transmission layer

- Responsible for reliable transmission of CON CoAP messages (both requests and responses)

- This layer takes care of:
  - Retransmitting messages that are not ACKed
  - Notifying failure (timeout) and success (ACK)

- Support for piggybacked and separate responses
Reliable Transmission layer

- Main classes:
  - `CoapReliableTransmission`
  - `CoapReliableTransmissionListener`
  - `CoapReliableReception`
Transaction layer

- Request/response exchange
- Reliable transmissions are handled by the underlying layer
- At the client-side, send a request and receive the corresponding response
- At the server-side, receive a request and send the response
Transaction layer

- Main classes:
  - CoapTransactionClient
  - CoapTransactionClientListener
  - CoapTransactionServer
  - CoapTransactionServerListener
Follow the Slides

http://goo.gl/anfy5w
Let’s get concrete!
Californium (Cf)

Five repositories on GitHub

- [https://github.com/eclipse/californium](https://github.com/eclipse/californium)
  Core library and example projects

- [https://github.com/eclipse/californium.element-connector](https://github.com/eclipse/californium.element-connector)
  Abstraction for modular network stage (Connectors)

- [https://github.com/eclipse/californium.scandium](https://github.com/eclipse/californium.scandium)
  DTLS 1.2 implementation for network stage (DtlsConnector)

- [https://github.com/eclipse/californium.tools](https://github.com/eclipse/californium.tools)
  Stand-alone CoAP tools such as console client or RD

- [https://github.com/eclipse/californium.actinium](https://github.com/eclipse/californium.actinium)
  App server for server-side JavaScript*

*not yet ported to new implementation and using deprecated CoAP draft version
Code structure

https://github.com/eclipse/californium

- Libraries (“californium-” prefix)
  - californium-core: CoAP, client, server
  - californium-osgi: OSGi wrapper
  - californium-proxy: HTTP cross-proxy

- Example code
- Example projects (“cf-” prefix)
Code structure

https://github.com/eclipse/californium

- Libraries
- Example code
  - cf-api-demo API call snippets
- Example projects
Code structure

https://github.com/eclipse/californium

- Libraries
- Example code
- Example projects
  - `cf-helloworld-client` basic GET client
  - `cf-helloworld-server` basic server
  - `cf-plugtest-checker` tests Plugtest servers
  - `cf-plugtest-client` tests client functionality
  - `cf-plugtest-server` tests server functionality
  - `cf-benchmark` performance tests
  - `cf-secure` imports Scandium (DTLS)
  - `cf-proxy` imports californium-proxy
Maven handles dependencies and more

Call

```
mvn clean install
```

in this order (internal dependencies)

- californium.element-connector
- californium.scandium
- californium
- *

to build and install the artifacts
Server API

Important classes (see org.eclipse.californium.core)

- CoapServer
- CoapResource
- CoapExchange

Learn about other classes through auto-complete

Basic steps

- Implement custom resources by extending CoapResource
- Add resources to server
- Start server
import static org.eclipse.californium.core.coap.CoAP.ResponseCode.*; // shortcuts

public class MyResource extends CoapResource {
    @Override
    public void handleGET(CoapExchange exchange) {
        exchange.respond("hello world"); // reply with 2.05 payload (text/plain)
    }
    @Override
    public void handlePOST(CoapExchange exchange) {
        exchange.accept(); // make it a separate response
        if (exchange.getRequestOptions()....) {
            // do something specific to the request options
        }
        exchange.respond(CREATED); // reply with response code only (shortcut)
    }
}
public static void main(String[] args) {

    CoapServer server = new CoapServer();

    server.add(new MyResource("hello"));

    server.start(); // does all the magic

}
Client API

Important classes

- CoapClient
- CoapHandler
- CoapResponse
- CoapObserveRelation

- Instantiate **CoapClient** with target URI
- Use offered methods `get()`, `put()`, `post()`, `delete()`, `observe()`, `validate()`, `discover()`, or `ping()`
- Optionally define **CoapHandler** for asynchronous requests and observe
public static void main(String[] args) {

    CoapClient client1 = new CoapClient("coap://iot.eclipse.org:5683/multi-format");

    String text = client1.get().getResponseText(); // blocking call
    String xml = client1.get(APPLICATION_XML).getResponseText();

    CoapClient client2 = new CoapClient("coap://iot.eclipse.org:5683/test");

    CoapResponse resp = client2.put("payload", TEXT_PLAIN); // for response details
    System.out.println( resp.isSuccess());
    System.out.println( resp.getOptions());

    client2.useNONs(); // use autocomplete to see more methods
    client2.delete();
    client2.useCONs().useEarlyNegotiation(32).get(); // it is a fluent API
}
public static void main(String[] args) {

    CoapClient client = new CoapClient("coap://iot.eclipse.org:5683/separate");

    client.get(new CoapHandler() { // e.g., anonymous inner class

        @Override
        public void onLoad(CoapResponse response) { // also error resp.
            System.out.println(response.getResponseText());
        }

        @Override
        public void onError() { // I/O errors and timeouts
            System.err.println("Failed");
        }
    });
}
Client API - Observe

```java
public static void main(String[] args) {

    CoapClient client = new CoapClient("coap://iot.eclipse.org:5683/obs");

    CoapObserveRelation relation = client.observe(new CoapHandler() {

        @Override public void onLoad(CoapResponse response) {
            System.out.println( response.getResponseText() );
        }

        @Override public void onError() {
            System.err.println( "Failed" );
        }
    });

    relation.proactiveCancel();
}
```
Advanced API

Get access to internal objects with

```advanced() on
CoapClient, CoapResponse, CoapExchange```

Use clients in resource handlers with

```createClient(uri);```

Define your own concurrency models with

```ConcurrentCoapResource` and
CoapClient.useExecutor() / setExecutor(exe)```
Erbium (Er)

Erbium is part of Contiki OS
https://github.com/contiki-os/contiki

You already have it :)

- **Libraries (in ./apps/)**
  - er-coap CoAP
  - rest-engine resources, REST calls

- **Examples (in ./examples/er-rest-example)**
  - er-example-server how to add resources
  - er-example-client how to issue requests
  - er-plugtest-server ETSI Plugtest test cases
  - ./resources/res-* resource modules
Erbium (Er) Project Files

Makefile

# ensure proper IPv6 configuration

# add libraries
APPS += er-coap
APPS += rest-engine

project-conf.h

/* if needed, tweak parameters found in
apps/er-coap/er-coap-conf.h */
Erbium (Er) Server Program

Global

```c
extern resource_t <resources>;
```

In PROCESS

```c
rest_init_engine();
rest_activate_resource(&<resource>, <URI-Path>);
SENSORS_ACTIVATE(<sensor>); /* if used by resource */
```

In PROCESS while(1) loop

```c
PROCESS_WAIT_EVENT();
if (ev==<notification event>) <event resource>.trigger();
if (ev==<response ready>) <separate resource>.resume();
```
Erbium (Er) Resources I

#include "rest-engine.h"

static void res_get_handler(void *request, void *response,
    uint8_t *buffer, uint16_t preferred_size, int32_t *offset);

RESOURCE(res_<name>,
    "title=""<human readable>"";ct=0", /* see CoRE Link Format */
    res_get_handler,
    NULL, /* or res_post_handler */
    NULL, /* or res_put_handler */
    NULL /* or res_delete_handler */
);
Erbium (Er) Resources II

```c
static void res_get_handler(void *request, void *response,
    uint8_t *buffer, uint16_t preferred_size, int32_t *offset)
{
    /* use `REST.get_*` functions to access request */

    /* use `REST.set_*` functions to access response */

    /* use `buffer` to create response body */
    /* do not exceed `preferred_size` */
    /* engine handles block transfers up to `REST_MAX_CHUNK_SIZE` */
    /* use `offset` to fragment manually if larger */
}
```
Global or static in PROCESS

    uip_ipaddr_t server_ip;
    coap_packet_t request[1];

In PROCESS

    coap_init_engine(); /* not rest_ because CoAP-only */
    coap_init_message(request, COAP_TYPE_CON, <coap_method_t>, 0);
    coap_set_header_uri_path(request, <URI-Path>);
    /* if COAP_POST or COAP_PUT only */
    coap_set_payload(request, <string>, <length>);
    coap_set_header_content_format(request, <coap_content_format_t>);
    COAP_BLOCKING_REQUEST(&server_ip, <port>, request,
                           client_response_handler);
void client_response_handler(void *response) {
    /* use coap_get_* functions */

    /* OR */

    coap_packet_t *const coap_res = (coap_packet_t *)response;
    /* client is CoAP-only, no need for indirection */
    /* use coap_res-> to access fields */
}
mjCoAP
CoapProvider

- It is the fundamental class that enables CoAP messaging in an application
- A CoapProvider is bound to a specific UDP port
- Provides a `send()` method
- Forwards incoming messages to registered CoapProviderListeners
import org.zoolu.coap.core.*;
import org.zoolu.coap.message.*;
public class CoapClient implements CoapProviderListener{

    private CoapProvider coapProvider;

    public CoapClient(){
        this.coapProvider = new CoapProvider(CoapProvider.ANY_PORT); // get random port
        this.coapProvider.setListener(CoapMethodId.ANY, this); // receive all messages
    }

    public void send(CoapMessage message) {
        this.coapProvider.send(message);
        System.out.println("SENT: " + message);
    }

    @Override
    public void onReceivedMessage(CoapMessage message) {
        System.out.println("RECV: " + message);
    }
}
public static void main(String[] args) {

    CoapClient client = new CoapClient();

    CoapRequest request = CoapMessageFactory.createCONrequest(
        CoapMethod.GET,
        "coap://localhost/test");

    client.send(request);
}

import java.net.*/;
import org.zoolu.coap.core.*/;
import org.zoolu.coap.message.*/;
import org.zoolu.net.*/;

public class CoapTransactionClient {

    private CoapProvider coapProvider;

    public CoapTransactionClient(){
        this.coapProvider = new CoapProvider(CoapProvider.ANY_PORT);
    }

    public void request(CoapMethod method, String resource, byte[] payload, CoapTransactionClientListener listener) {
        URI uri = new URI(resource);
        CoapRequest req = CoapMessageFactory.createCONrequest(method,uri);
        req.setPayload(payload);
        new CoapTransactionClient(coapProvider, new SocketAddress(uri.getHost(),uri.getPort()), listener).request(req);
    }
}

CoapTransactionClient API

```java
import java.net.*;
import org.zoolu.coap.core.*;
import org.zoolu.coap.message.*;
import org.zoolu.net.*;
public class CoapTransactionClient {

    private CoapProvider coapProvider;

    public CoapTransactionClient(){
        this.coapProvider = new CoapProvider(CoapProvider.ANY_PORT);
    }

    public void request(CoapMethod method, String resource, byte[] payload, CoapTransactionClientListener listener) {
        URI uri = new URI(resource);
        CoapRequest req = CoapMessageFactory.createCONrequest(method,uri);
        req.setPayload(payload);
        new CoapTransactionClient(coapProvider, new SocketAddress(uri.getHost(),uri.getPort()), listener).request(req);
    }
}
```
public static void main(String[] args) {

    CoapTransactionClient client = new CoapTransactionClient();
    client.request(CoapMethod.GET, "coap://localhost/test", null,
                   new CoapTransactionClientListener({
                       @Override
                       public void onTransactionResponse(CoapTransactionClient tc, CoapMessage resp)
                       {
                           System.out.println("RECV: " + resp);
                       }
                   })
                      {
                @Override
                public void onTransactionFailure(CoapTransactionClient tc)
                {
                    System.out.println("FAILED");
                }
            });
}
Questions?
HANDS-ON!