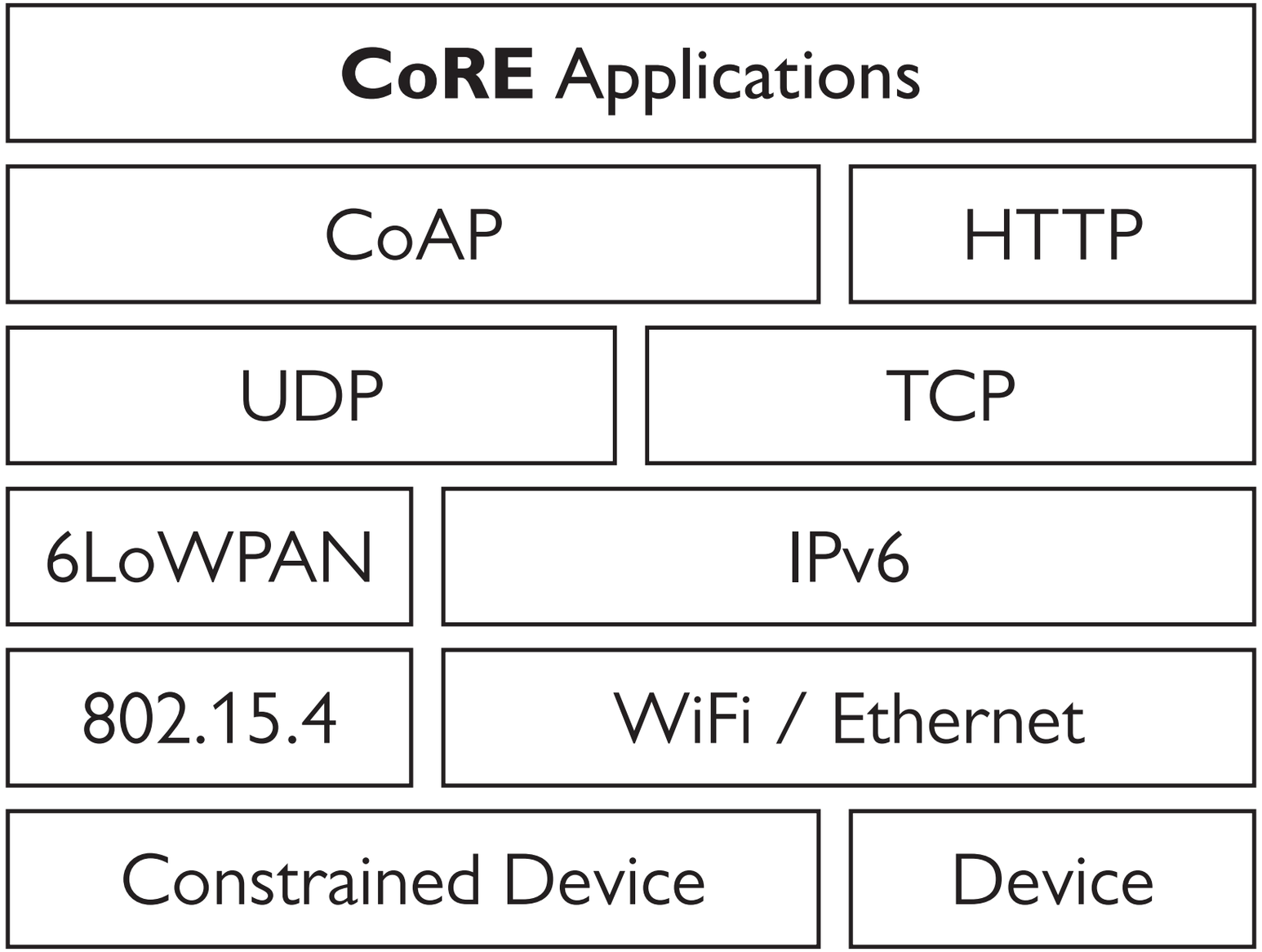
CoRE Applications

Klaus Hartke


CoRE Applications
CoAP
HTTP
UDP
TCP
6LoWPAN
IPv6
802.15.4
WiFi / Ethernet
Constrained Device
Device

Constrained **REST**ful Environments

REST is an *architectural style* for building distributed systems.

An *architectural style* is a coordinated set of architectural **constraints** that restricts the roles/features of architectural elements within an architecture.

REST underlies the most successful distributed system in history – the *World Wide Web* – and was defined by Roy Fielding in his doctoral thesis.

Main Ideas of **REST**

- stateless, request/response-style communication between *clients* and *servers*
- *resources* with a uniform interface, and uniform resource *identifiers*
- hierarchy of *caches* and *intermediaries* for efficiency, scalability and encapsulation
- the exchange of *representations* that capture the current or intended state of a resource
- most importantly, the use of *hypermedia* to drive the application state

REST is the *de-facto standard* for designing and building Internet-based applications.

RESTful design leads to systems that are

open

scalable

extensible

and easy to understand

Nobody uses **REST** properly.

In particular, the '*hypermedia*' constraint is mostly ignored. Instead, most applications define their interface (API) in terms of fixed resource names and the operations supported on them.

(e.g.: Twitter API, Github API, Facebook Graph API, Google+ API, …)

And that's fine.

If there is only *one server* controlled by you and it is easy to upgrade *all clients* every time you make a change, it is perfectly valid to couple server and clients in this way.

If you have an application that spans millions of servers and clients across **multiple organizations** and that needs to run for **decades** without breaking existing implementations and deployments, you need a strategy for *managing change*.

Hypertext-driven **REST** is such a strategy.

Hypertext-driven **REST** manages change …

- by *decoupling* clients and servers as much as possible
- by preferring *forward and backward compatibility* over breaking changes
- by preserving certain *freedoms* in the application architecture

For example, it preserves the freedom …

- to *restructure* the resource name space
- to *offload* resources to other servers
- to *evolve* representation formats
  either by extending formats in a backwards-compatible way
  or by offering new formats in addition to existing ones
- to *add* new functionality
  by introducing new hyperlinks
- to *merge* two applications

RESTful, hypertext-driven applications have a higher upfront cost and require some design effort, but have the benefit of *long-term stability and evolvability.*

Services for a Changing World

RESTful Web APIs

O'REILLY®

Leonard Richardson,
Mike Amundsen & Sam Ruby

CoRE Application Descriptions

How to document the interface of a **REST**ful application?

WADL, Swagger, RAML, API Blueprint, etc. are great for **REST**ful applications with fixed resource names, but do not help designers build *hypertext-driven* applications.

CoRE Application Descriptions provide a simple, consistent, standard format for documenting the interface of **REST**ful, hypertext-driven applications.

What is part of the interface?

Communication Protocol

Representation Formats

media type

text/html

representation

```
<!DOCTYPE html>
<html>
  …
  <body>
    <p>21.6 °C</p>
  </body>
</html>
```

Representation Formats

media type

application/senml+json

representation

```
{ "e": [
    { "v": 21.6, "u": "Cel" }
]}
```

Representation Formats

- almost all of the descriptive effort should be on defining the *representation formats*
- representation formats should allow clients with *different goals*, so they can do different things with the same data
- representation formats evolve over time: a new version of a format should provide both *forward* and *backward compatibility*

Hypermedia: Links

```
<link rel="terms-of-service"
      href="coap://example/tos"
      type="application/tos+xml"/>
```

- a link is the primary means for a client to *discover resources* and *change application state*
- links can be annotated with a *link relation type* that identifies the semantics of the link so that machines know what it means to follow the link

Hypermedia: Forms

```
<form class="change"
      action="coap://example/led"
      method="put"
      accept="application/lighting+xml"/>
```

- a form is the primary means for a client to *change resource state* at the server
- forms can be annotated with a *form relation type* that identifies the semantics of the form for machines

Summary

Communication Protocols *identified by URI schemes*

Representation Formats *identified by media types*

Link Semantics *identified by link relation types*

Form Semantics *identified by form relation types*

https://tools.ietf.org/html/draft-hartke-core-apps

Next steps

- Explore some of the less-understood aspects
  e.g.: forms
- Find a good example for demonstration
  “evaluation/reference framework”?

Thank you!

Photos by John Clare
https://flic.kr/p/okFdc7
https://flic.kr/p/okFu2w
https://flic.kr/p/okiHP4