Introduction
Evolution from M2M to IoT

M2M

Big Data
Internet of Things

Services
The Web
Things

Little Data
CoAP: The Web of Things Protocol

- Open IETF Standard
- Compact 4-byte Header
- UDP, SMS, (TCP) Support
- Strong DTLS Security
- Asynchronous Subscription
- Built-in Discovery
From Web Applications to IoT Nodes

1000s of bytes

Web Object

HTTP

TLS / TCP

IP

Web Application

100s bytes

Proxy

Binary Web Object

CoAP

DTLS / UDP

IP

IoT Backhaul

10s of bytes

Router

Binary Web Object

CoAP

DTLS / UDP

6LoWPAN

IoT Node Network
CoAP is One Key IoT Standard

Application
- CoRE
- Lightweight M2M
- Application Objects
- ZigBee Web Objects

Protocols, Security
- Web: CoAP & HTTP
- TLS Based Security

Local Network
- 6LoWPAN
- ZigBee IP
- BT Smart IP
- ZigBee NAN
- ZigBee IP 1.X
The Web and REST
The Web Architecture
Web Naming

Universal Resource Identifier (URI)

Universal Resource Name (URN)
urn:Sensei:sensinode.com:NanoSensor:N740:3a-43-ff-12-01-01

Universal Resource Locator (URL)
http://www.example.org:8080/sensors?id=light

- Scheme
- Authority
- Port
- Path
- Query
URL Resolution

Resource

HTTP
TCP
IP
Ethernet Link

http://www.example.org:8080/sensors?id=light

2001:dead:beef::1

DNS
An HTTP Request

See RFC2616 - Hypertext Transfer Protocol v1.1
Web Paradigms
A REST Request

GET /temperature

200 OK
application/text
22.5 °C

Server

Client

22.5 °C
CoAP: Constrained Application Protocol
CoAP Design Requirements
The CoAP Architecture
What CoAP is (and is not)

- Sure, CoAP is
  - A very efficient RESTful protocol
  - Ideal for constrained devices and networks
  - Specialized for M2M applications
  - Easy to proxy to/from HTTP

- But hey, CoAP is not
  - A general replacement for HTTP
  - HTTP compression
  - Restricted to isolated “automation” networks
CoAP Features

- Embedded web transfer protocol (coap://)
- Asynchronous transaction model
- UDP binding with reliability and multicast support
- GET, POST, PUT, DELETE methods
- URI support
- Small, simple 4 byte header
- DTLS based PSK, RPK and Certificate security
- Subset of MIME types and HTTP response codes
- Built-in discovery
- Optional observation and block transfer
Transaction Model

- **Transport**
  - CoAP currently defines:
    - UDP binding with DTLS security
    - CoAP over SMS or TCP possible

- **Base Messaging**
  - Simple message exchange between endpoints
  - Confirmable or Non-Confirmable Message answered by Acknowledgement or Reset Message

- **REST Semantics**
  - REST Request/Response piggybacked on CoAP Messages
  - Method, Response Code and Options (URI, content-type etc.)
Message Header (4 bytes)

Ver - Version (1)
T - Message Type (Confirmable, Non-Confirmable, Acknowledgement, Reset)
TKL - Token Length, if any, the number of Token bytes after this header
Code - Request Method (1-10) or Response Code (40-255)
Message ID - 16-bit identifier for matching responses
Token - Optional response matching token
Option Format

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option Delta</td>
<td>Option Length</td>
<td>1 byte</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option Length (extended)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option Delta (extended)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option Length (extended)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option Value</td>
<td>0 or more bytes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Option Delta** – Difference between this option type and the previous
**Length** – Length of the option value
**Value** – The value of Length bytes immediately follows Length
# Base Specification Options

<table>
<thead>
<tr>
<th>No.</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>(see below)</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>x</td>
<td></td>
<td></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>(see below)</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>x</td>
<td></td>
<td></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>x</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>16</td>
<td></td>
<td>x</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>x</td>
<td></td>
<td></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>
</tbody>
</table>

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeateable
Request Example

CoAP Client

CoAP Server

Con [0xaf5] GET /light

ACK [0xaf5] 2.05 Content "<light>..."

Confirmable Request

Piggy-backed Response
Dealing with Packet Loss
Separate Response

CoAP Client

CON [0x1b] GET /light Token: 0x31
ACK [0x1b]

CON [0x823] 2.05 Content /light Token: 0x31 "<light>..."
ACK [0x823]

CoAP Server

takes too much time
/light ready
Bits and bytes...
Caching

- CoAP includes a simple caching model
  - Cacheability determined by response code
  - An option number mask determines if it is a cache key
- Freshness model
  - Max-Age option indicates cache lifetime
- Validation model
  - Validity checked using the Etag Option
- A proxy often supports caching
  - Usually on behalf of a constrained node,
  - a sleeping node,
  - or to reduce network load
Proxying and caching

CoAP Server -> Proxy -> HTTP Client

- CON GET /light
- ACK max-age=30s 2.05 Content "<light>..."
- cache /light

HTTP Client

- HTTP GET /light
- 200 OK "<light>..."
- cache fresh

- HTTP GET /light
- 200 OK "<light>..."
Observation

See draft-ietf-core-observe
Block transfer

CON GET /light

ACK block2(nr=0, m=1, sz=1024) 2.05 "</light>..."

CON block2(nr=1, m=0, sz=1024) GET /light

ACK block2(nr=1, m=1, sz=1024) 2.05 "</light>..."

CON block2(nr=2, m=0, sz=1024) GET /light

ACK block2(nr=2, m=1, sz=1024) 2.05 "</light>..."

CON block2(nr=3, m=0, sz=1024) GET /light

ACK block2(nr=3, m=0, sz=1024) 2.05 "</light>..."

See draft-ietf-core-block
Getting Started with CoAP

- There are many open source implementations available
  - mbed includes CoAP support
  - Java CoAP Library Californium
  - C CoAP Library Erbium
  - libCoAP C Library
  - jCoAP Java Library
  - OpenCoAP C Library
  - TinyOS and Contiki include CoAP support
- CoAP is already part of many commercial products/systems
  - ARM Sensinode NanoService
  - RTX 4100 WiFi Module
- Firefox has a CoAP plugin called Copper
- Wireshark has CoAP dissector support
- Implement CoAP yourself, it is not that hard!
Discovery & Semantics
What is Web Linking?

- Links have been around a long time
- Web Linking formalizes links with defined relations, **typed links**
  - HTML and Atom have allow links
- **RFC5988** defines a framework for Web Linking
  - Combines and expands the Atom and HTML relation types
  - Defines a unified typed link concept
- A link can be serialized in any number of formats
  - RFC5988 revives the HTTP Link Header and defines its format
  - Atom and HTML are equivalent serializations
What is Web Linking?

- A type link consists of:
  - Context URI – What the link is from
  - Relation Type – Indicates the semantics of the link
  - Target URI – What the link is too
  - Attributes – Key value pairs describing the link or its target

- Relations include e.g. copyright, author, chapter, service etc.

- Attributes include e.g. language, media type, title etc.

- Example in HTTP Link Header format:

  Link: <http://example.com/TheBook/chapter2>; rel="previous"; title="previous chapter"
Resource Discovery

- **Service Discovery**
  - What services are available in the first place?
  - Goal of finding the IP address, port and protocol
  - Usually performed by e.g. DNS-SD when DNS is available

- **Resource Discovery**
  - What are the Web resources I am interested in?
  - Goal of finding URIs
  - Performed using Web Linking or some REST interface
  - CoRE Link Format is designed to enable resource discovery
CoRE Link Format

- RFC6690 is aimed at Resource Discovery for M2M
  - Defines a link serialization suitable for M2M
  - Defines a well-known resource where links are stored
  - Enables query string parameters for filtered GETs
  - Can be used with unicast or multicast (CoAP)

- Resource Discovery with RFC6690
  - Discovering the links hosted by CoAP (or HTTP) servers
  - GET /.well-known/core?optional_query_string
  - Returns a link-header style format
  - URL, relation, type, interface, content-type etc.
CoRE Resource Discovery

```
<dev/bat>;obs;rt="ipso:dev-bat";ct="0",
<dev/mdl>;rt="ipso:dev-mdl";ct="0",
<dev/mfg>;rt="ipso:dev-mfg";ct="0",
<pwr/0/rel>;obs;rt="ipso:pwr-rel";ct="0",
<pwr/0/w>;obs;rt="ipso:pwr-w";ct="0",
<sen/temp>;obs;rt="ucum:Cel";ct="0"
```
Resource Directory

- CoRE Link Format only defines
  - The link format
  - Peer-to-peer discovery
- A directory approach is also useful
  - Supports sleeping nodes
  - No multicast traffic, longer battery life
  - Remote lookup, hierarchical and federated distribution
- The CoRE Link Format can be used to build Resource Directories
  - Nodes POST (register) their link-format to an RD
  - Nodes PUT (refresh) to the RD periodically
  - Nodes may DELETE (remove) their RD entry
  - Nodes may GET (lookup) the RD or resource of other nodes

See draft-ietf-core-resource-directory
Resource Directory

Device node341

Registration

POST /rd?h=node341
<s/light>, <s/temp>, <a/relay>

Update

PUT /rd/domain/node341

De-register

Delete /rd/domain/node341

NSP

Subscribe to RD events

Notification: End-point created

Web App

CoAP M2M Interface

HTTP Web Interface

See draft-ietf-core-resource-directory
How to get Semantic?

- So how to use CoRE in real applications?
- Resources need meaningful naming (rt=)
- A resource needs an interface (if=)
  - See [draft-vial-core-link-format-wadl] on using WADL for this
- A payload needs a format (EXI, JSON etc.)
  - Deployment or industry specific today
  - oBIX, SensorML, EEML, sMAP etc.
  - SenML is a promising format [draft-jennings-senml]
  - CBOR is a standard for binary JSON [RFC7049]
- Promising data semantics for use with CoAP
  - OMA Lightweight M2M [http://j.mp/lwm2m]
  - IPSO Objects [http://www.ipso-alliance.org/smart-object-committee-charter]
  - CoRE Interfaces [draft-ietf-core-interfaces]
CoRE Link Format Semantics

- RFC6690 = Simple semantics for machines
  - IANA registry for rt= and if= parameters

- Resource Type (rt=)
  - What is this resource and what is it for?
  - e.g. Device Model could be rt="ipso.dev.mdl"

- Interface Description (if=)
  - How do I access this resource?
  - e.g. Sensor resource accessible with GET if="core.s"

- Content Type (ct=)
  - What is the data format of the resource payloads?
  - e.g. text/plain (0)
CoRE Interfaces

- CoRE Interfaces [draft-ietf-core-interfaces]
  - A paradigm for REST profiles made up of function sets
  - Simple interface types

<table>
<thead>
<tr>
<th>Interface</th>
<th>if=</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link List</td>
<td>core.ll</td>
<td>GET</td>
</tr>
<tr>
<td>Batch</td>
<td>core.b</td>
<td>GET, PUT, POST (where applicable)</td>
</tr>
<tr>
<td>Linked Batch</td>
<td>core.lb</td>
<td>GET, PUT, POST, DELETE (where applicable)</td>
</tr>
<tr>
<td>Sensor</td>
<td>core.s</td>
<td>GET</td>
</tr>
<tr>
<td>Parameter</td>
<td>core.p</td>
<td>GET, PUT</td>
</tr>
<tr>
<td>Read-only</td>
<td>core rp</td>
<td>GET</td>
</tr>
<tr>
<td>Parameter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actuator</td>
<td>core.a</td>
<td>GET, PUT, POST</td>
</tr>
<tr>
<td>Binding</td>
<td>core.bnd</td>
<td>GET, POST, DELETE</td>
</tr>
</tbody>
</table>