Constrained Application Protocol (CoAP)
GOAL: to enable web-based services in constrained wireless networks
- 8 bit micro-controllers
- limited memory
- low-power networks

Problem: WEB solution are hardly applicable

Solution: re-design web-based services for constrained networks -> COAP
How Does the Web Work?

- Resources in the Web are:
  - managed by servers
  - identified by URIs
  - accessed synchronously by clients through request/response paradigms

- In a word, Representational State Transfer (REST)
URL Resolution

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

Resource

HTTP
TCP
IP
Ethernet Link

2001:dead:beef::1

DNS

http://www.example.org:8080/sensors?id=light
Other common HTTP methods: PUT, POST, DELETE
The CoAP Architecture
CoAP Design Requirements

See draft-shelby-core-coap-req
CoAP At a Glance

- Embedded web transfer protocol (coap://)
- Asynchronous transaction model
- UDP binding with reliability and multicast support
- GET, POST, PUT, DELETE methods
- URI support
- 4 byte header
- Subset of MIME types and HTTP response codes
- Built-in discovery
- Optional observation and block transfer
COAP Messaging Basics

- **Transport:**
  - (mainly) UDP binding

- **Message Exchange between Endpoints**
  - Messages with 4 bytes header (shared by request and responses) containing a message ID (16 bits)
  - Reliable exchange through Confirmable Messages which must be acknowledged (through ACK or Reset Messages). Simple Stop-and-Wait retransmission with exponential back-off.
  - Unreliable exchange through Non-Confirmable Message
  - Duplicate detection for both confirmable and non-confirmable messages (through message ID)
COAP Messaging

Client

CON [0x7d34]

ACK [0x7d34]

Message ID

Server

Client

NON [0x01a0]

Server
COAP Message Semantics

- REST Request/Response piggybacked on CoAP Messages

- Method, Response Code and Options (URI, content-type etc.)
COAP Request/Response Examples

Client | Server | Client | Server

```
CON [0xbc90]
GET /temperature
  (Token 0x71)

ACK [0xbc90]
  2.05 Content
  (Token 0x71)
  "22.5 C"

CON [0xbc91]
GET /temperature
  (Token 0x72)

ACK [0xbc91]
  4.04 Not Found
  (Token 0x72)
  "Not found"
```
COAP: Separate Response

Client

CON [0x7a10]
GET /temperature
(Token 0x73)

ACK [0x7a10]

... Time Passes ...

CON [0x23bb]
2.05 Content
(Token 0x73)
"22.5 C"

ACK [0x23bb]

Server
COAP: Non-confirmable Request

Client

NON [0x7a11]
GET /temperature
(Token 0x74)

Server

NON [0x23bc]
2.05 Content
(Token 0x74)
"22.5 C"
Message Header (4 bytes)

![Message Header Diagram]

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 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>x</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></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>(see below)</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>x</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>16</td>
<td></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>x</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>x</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>x</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=Repeatable
Dealing with Packet Loss

- Stop and Wait approach
- Repeat a request after a time-out in case ACK (or RST) is not coming back
Back-Off Details

- Initial time-out set to:
  - Rand [ACK_TIMEOUT, ACK_TIMEOUT * ACK_RANDOM_FACTOR] ([2s, 3s])

- When time-out expires and the transmission counter is less than MAX_RETRANSMIT (4)
  - retransmit
  - Increase transmission counter
  - double the time-out value

- The procedure is repeated until
  - A ACK is received
  - A RST message is received
  - the transmission counter exceeds MAX_RETRANSMIT
  - the total attempt duration exceeds MAX_TRANSMIT_WAIT (93s)
PROBLEM:
- REST paradigm is often “PULL” type, that is, data is obtained by issuing an explicit request
- Information/data in WSN is often periodic/triggered (e.g., get me a temperature sample every 2 seconds or get me a warning if temperature goes below 5°C)

SOLUTION: use Observation on COAP resources
Observation

See draft-ietf-core-observe
COAP Block Transfer

- PROBLEM: avoid segmentation in the lower layers (IPv6)
- SOLUTION: COAP Block Transfer Mode
  - brings up fragmentation at the application layer
Block2 Option added to messages

- nr=incremental block number within original data
- m=more blocks flag
- sz=block size
Resource Discovery

- GOAL: 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

CoAP Client

CoAP Server

\textit{CON [0x6f6]} \textit{GET /well-known/core}

\textit{ACK [0x6f6]} \textit{2.05 Content "<light>..."}

\texttt{</dev/bat>;obs;if="";rt="ipso:dev-bat";ct="0"},
\texttt{</dev/mdl>;if="";rt="ipso:dev-mdl";ct="0"},
\texttt{</dev/mfg>;if="";rt="ipso:dev-mfg";ct="0"},
\texttt{</pwr/0/rel>;obs;if="";rt="ipso:pwr-rel";ct="0"},
\texttt{</pwr/0/w>;obs;if="";rt="ipso:pwr-w";ct="0"},
\texttt{</sen/temp>;obs;if="";rt="ucum:Cel";ct="0"}
Getting Started with CoAP

- Open source implementations:
  - Java CoAP Library [Californium](#)
  - C CoAP Library [Erbium](#)
  - [libCoAP](#) C Library
  - [jCoAP](#) Java Library
  - [OpenCoAP](#) C Library
  - TinyOS and Contiki include CoAP support
- Firefox has a CoAP [plugin called Copper](#)
- Wireshark has CoAP plugin