Constrained Application Protocol (CoAP)

Vilen Looga, M.Sc.
Doctoral Student @dcs.aalto
Outline

• Introduction
• CoAP at a glance
• Messages
• Observe
• Hardware
• Demo
• MAMMOTH
• Conclusions
• References
50 billion connected devices

First wave
Networked consumer electronics

Second wave
Networked industries

Third wave
Networked everything
Networked society

Improved reach
Improved value - consumer lifestyle
Improved process efficiency
Improved human efficiency
Machine to machine (M2M)
Machine to machine (M2M)
Node hardware

Zolertia Z1 (Contiki OS)  Arduino
Node hardware

• Limited resources: 8KB RAM, 92KB ROM, 16-bit CPU
• Wireless communication: 6lowPAN over 802.15.4, GPRS, 4G
• Long-lasting battery
Event-driven networking
Constrained Application Protocol (CoAP)

• Application level protocol over UDP
• Designed to be used with constrained nodes and lossy networks
• Designed for M2M applications, such as home and infrastructure monitoring
• Built-in resource discovery and observation
• RESTful for easy interfacing with HTTP
• Multicast support
• Low overhead and simple
Request/Response model

Client

CON [0x11aa]
GET /temp
(Token 0x66)

ACK [0x11aa]

CON [0xc123]
2.05 Content
(Token 0x66)
“23.9 C”

ACK [0xc123]

Server

Client

NON [0xabc1]
GET /temp
(Token 0x76)

NON [0xbbbaa]
2.05 Content
(Token 0x76)
“23.9 C”

Server
## CoAP message

A UDP packet

<table>
<thead>
<tr>
<th>SRC port</th>
<th>DST port</th>
<th>Length</th>
<th>CRC</th>
<th>CoaP MSG</th>
</tr>
</thead>
</table>

## CoAP message

<table>
<thead>
<tr>
<th>Ver</th>
<th>T</th>
<th>OC</th>
<th>Code</th>
<th>Message ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option Delta</td>
<td>Length</td>
<td>Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Payload
### CoAP message

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>0010</td>
<td>00000001</td>
<td>0001</td>
<td>1011</td>
<td>0000</td>
<td>0100</td>
</tr>
<tr>
<td>Ver. 1</td>
<td>Type 0</td>
<td>Options 2</td>
<td>Request</td>
<td>Message ID</td>
<td>0+9=9: Uri-Path</td>
<td>Length 11</td>
<td>“.well-known”</td>
</tr>
</tbody>
</table>
Message types

- Confirmable (0)
- Non-Confirmable (1)
- Acknowledgement (2)
- Reset (3)
Message Code

• Request (1-31)
• Response (64-191)
• Empty (0)
# Options

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>If-Match</td>
</tr>
<tr>
<td>3</td>
<td>Uri-Host</td>
</tr>
<tr>
<td>4</td>
<td>ETag</td>
</tr>
<tr>
<td>5</td>
<td>If-None-Match</td>
</tr>
<tr>
<td>7</td>
<td>Uri-Port</td>
</tr>
<tr>
<td>8</td>
<td>Location-Path</td>
</tr>
<tr>
<td>11</td>
<td>Uri-Path</td>
</tr>
<tr>
<td>12</td>
<td>Content-Format</td>
</tr>
<tr>
<td>14</td>
<td>Max-Age</td>
</tr>
<tr>
<td>15</td>
<td>Uri-Query</td>
</tr>
<tr>
<td>16</td>
<td>Accept</td>
</tr>
<tr>
<td>19</td>
<td>Token</td>
</tr>
<tr>
<td>20</td>
<td>Location-Query</td>
</tr>
<tr>
<td>35</td>
<td>Proxy-Uri</td>
</tr>
</tbody>
</table>
RESTful protocol

GET /node-xxx/a/relay
-> 1

GET /node-xxx/s/temp
-> 22

POST /node-xxx/a/relay/1
-> OK

DELETE /node-xxx/a/relay
-> OK

/node-xxx/a/relay: 1
/node-xxx/s/temp: 22
CoAP to HTTP proxy

/node-xxx/a/relay: 1
/node-xxx/s/temp: 22

CoAP
UDP

HTTP
TCP
Resource discovery

GET /.well-known/core
->

&lt;/a/relay&gt;;rt="Relay";ct=0,
&lt;/s/temp&gt;;rt="Temperature";ct=0
Observe functionality

/.well-known/core
/a/relay
/s/temp

Observe /s/temp
Notification 1, 22.5 C
Notification 2, 22.9 C
Notification 3, 22.6 C
Multicast

- One-to-many has obvious application in IoT networks:
  - Data and code dissemination
  - Delivering event data to a set of interested servers
  - Measurement data to more than one subscriber
- Current multicast proposals have drawbacks:
  - SMRF: one-way directional flooding; only useful for data flowing from border router to nodes
  - PIM-WSN: too many forwarding states inherited from PIM-SSM alleviated by Bloom filters
Multicast

• Our proposal (Deng Y., Doctoral student):
  – Any node can be the sender
  – Interoperability with outside domains
  – Link layer broadcast
  – One network interface per node

• RPL (Routing protocol for Low-power and Lossy networks):
  – Uses tree structure
  – Multicast messages can be sent in RPL control messages
Proxy

Proxy-URI: “coap://domain.fi:60001/node-00x/s/temp/”

Uri-Host: domain.fi
Uri-Port: 60001
Uri-Path: s
Uri-Path: temp
Uri-Query: //
Security

• Similarly to TLS used in HTTP (-> HTTPS), CoAP is secured using Datagram TLS (DTLS)
• DTLS = TLS + features to deal with unreliability of TLS:
  – DTLS records are independent: if record N is lost N+1 can still be decrypted, while N is retransmitted
  – TLS handshake breaks if the packets are out of order; DTLS queues handshake messages until the correct one
  – Application is responsible for dealing with packet reordering, loss, data re-assembly etc.
• Minimal implementation of MUST configurations:
  – Not all cipher suits supported
  – DTLS does not work for multicast communication
• Devices should keep connection open as long as possible to avoid mutual authentication setup overhead
Resource directory

- /domain.fi/node-001
  - /a/relay
  - /s/temp
- /domain.fi/node-002
  - /a/relay
  - /s/temp
- /domain.fi/node-003
  - /a/relay
  - /s/temp
NanoService Platform© Demo
Hardware

Tiny CoAP sensor by Ericsson Research

- Ethernet, IPv6, UDP
- 48 lines of assembler code
- Does not support Observe
### Implementation

<table>
<thead>
<tr>
<th>Software</th>
<th>Type</th>
<th>Supported Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>libcoap</td>
<td>Client, Server</td>
<td>x86, Contiki, tinyOS</td>
</tr>
<tr>
<td>Californium</td>
<td>Client, Server</td>
<td>Java</td>
</tr>
<tr>
<td>Copper</td>
<td>Client</td>
<td>Javascript</td>
</tr>
<tr>
<td>Erbium</td>
<td>Client, Server</td>
<td>Contiki</td>
</tr>
</tbody>
</table>

... and many more
MAMMOTH: Large-scale IoT emulator

• According to many predictions regarding the future of interconnectivity, we will have large deployments of IoT infrastructures in various areas of our everyday lives
• Deploying such large infrastructure is costly and often does not tolerate radical post-installation changes
• Therefore, it is crucial to test future infrastructures and application running on top of them before deployment
• Current simulators/emulators are not suitable for testing massive IoT networks
• Thus, we started developing MAMMOTH, a large-scale massive IoT network emulator
MAMMOTH Architecture

- 20 million nodes
- Proxy per 2k nodes
- Node traffic going to balancer, which forwards to backend
- Apps and services can connect to the backend (resource directory)
References

- IETF CoAP draft: https://datatracker.ietf.org/doc/draft-ietf-core-coap/
- More than 50 billion connected devices – taking connected devices to mass market and profitability (whitepaper). Ericsson
Master’s Thesis topics

“Modeling sensor node behavior in large-scale network emulation”

“Using traffic prediction for energy efficiency in mobile networks”

Instructor: vilen.looga@aalto.fi

Special Assignment

“Make *libcoap* library safe for multi-threading”

This would allow us to reduce significantly the RAM consumption per node. Consequently, we can stuff more nodes in a VM.
Skills: C programming
Credits: 2 - …
Instructor: vilen.looga@aalto.fi

“Implementing DTLS on libcoap”

Skills: C/C++ programming
Credits: 2 - …
Instructor: vilen.looga@aalto.fi
Thank you!

Questions?