CoAP
Basuke Suzuki - @basuke
Kinoma
Agenda

- What is CoAP?
- DEMO with KinomaJS on Create
  - Behind the scene
- Dive into CoAP internals
- Possibility, Deficiency, Other Protocols, etc.
What is CoAP?

- **Constraint Application Protocol**
- Protocol for embedded devices, IoT, M2M.
- Defined in RFC 7252
- **CoAP = Lightweight Fast HTTP**
CoAP is **Lightweight**

- Message is simple binary format. Easy to parse. Space efficient.

- Client and server share the internal structures, so easy to implement both client and server compare to HTTP.

- RFC Document is just 112 pages. Seriously it’s short. With the full example use case and the default values for timeout or retry count, which is very valuable.
CoAP is Fast

- Implemented on top of UDP

- Stateless. No connection. Less overhead.

- Faster by omitting the protocol level reliability.

- Choices are yours
CoAP is **HTTP** (almost)

- Request and Response model
- RESTful design, GET/PUT/POST/DELETE
- Addressable by URL with query string
- Content-Type can be specified, but limited to pre-defined types, text, json, xml, etc…
HTTP is Great, but…

- Reliability is based on the TCP.
- HTTP Headers are just text with structure, so easy to read for human, but not for machine.
- 20+ years history and so many extensions to take care of. Headers are always huge. 700-800 bytes average today (Google SPDY report)
- Not the best choice for IoT world.
CoAP is **NOT** HTTP

- UDP instead of TCP
- No overhead of headers
- Limited Content-Types
- Headers are limited to pre-defined one
- Content size is limited to packet size. Depend on the lower level stack. On UNIX, about 1K.
UDP is fast, simple

- Connection less, no handshake
- Used for Video-streaming, DNS, DHCP, RIP
- 1:N communication - Broadcast and Multicast
- Small overhead of packet
  - UDP: 8 bytes < TCP: 20+ bytes
but UDP is **Unreliable**!

- Packet may deliver to the destination more than once, or may not deliver at all
- Packets may not arrive in order of delivery
- Sender does not know if or when the packet is delivered
- Application-layer must implement these features if they need those reliabilities.
TCP is Reliable

- Sender knows the result of the delivery. If not delivered, retransmit until it fails by timeout.

- If sender send A then B, receiver receive A and B in that order.

- TCP is streaming transmission so that there are no limit of data size.

- With its congestion control, bandwidth won’t filled by the re-transmitted packets.
CoAP is reliable by itself

- Duplicate detection by 16bit Message ID
- Acknowledgement response with data
- Each request contains a token. The token is binary data defined by the application. That same token must be included in the response. The token is what allows the requestor to match a response to a request.
CoAP is reliable by itself (cont.)

- Retransmission of message with exponentially incrementing interval, to prevent congestion of the network by transmission.

- These features are **optional**. If you really need speed, you can choose to disable them.
HTTP is Infrastructure

- HTTP freed up developers from application
- For many years, protocols are only for specific purpose. (SMTP, FTP, DNS, etc)
- HTTP was originally for delivery of information to human, but now it is basic infrastructure of the conversation of programs.
- So does CoAP, in IoT field.
“CoAP is the first UDP protocol which has the freedom of application level usage.”

– Basuke Suzuki (2015)
DEMO

- CoAP Server and Client
- Share color information
Kinoma Create

- Kinoma Create is the **JavaScript**-powered construction kit for Internet of Things devices and other connected electronics.
A lot of Pins
CoAP on KinomaJS

```javascript
var client = new CoAP.Client();
client.onResponse = function(response) {
    // success
};
client.onError(err) {
    // failure
};
client.request("coap://10.0.1.109/color", 'GET');
```
CoAP on KinomaJS

```javascript
var client = new CoAP.Client();
client.request({'
    uri: "coap://10.0.1.109/color",
    method: 'GET',
}).then(function(response) {
    // success
}, function(err) {
    // failure
});
```
CoAP in Depth
CoAP Message Format

- Minimum bytes are 4 bytes. (Empty Message)

- Must be fit inside UDP packet

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ver</td>
<td>T</td>
<td>TKL</td>
<td>Code</td>
<td>Message ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Token (if any, TKL bytes) ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (if any) ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xff</td>
<td>Payload (if any) ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fixed Header

<table>
<thead>
<tr>
<th>Ver</th>
<th>T</th>
<th>TKL</th>
<th>Code</th>
<th>Message ID</th>
</tr>
</thead>
</table>

Version. Always 01.

Token Length

Message Type. 2 bits.
Token

- 0 to 8 bytes binary
- length is specified in the header
- Response must have same token value with request.
- The value of the token is defined by the application layer. Protocol doesn’t care about the contents.
- Use case: request kind, sequence number, etc.
Options

- Option types are defined as integer index.
- Option format is defined for each Option type.
  - Empty, Unsigned Integer, String, Binary

<table>
<thead>
<tr>
<th>Option Delta</th>
<th>Option Length</th>
<th>Option Delta (extended)</th>
<th>Option Length (extended)</th>
<th>Option Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0-2 bytes</td>
<td>0-2 bytes</td>
<td>0-n bytes</td>
</tr>
</tbody>
</table>
Payload

- Payload is optional. If it exists, Payload Marker (0xff) must be exist to indicate the beginning of payload.

- Format is specified by Content-Format option.

- Available formats are: Text, JSON, XML, EXI, Binary
Message Code

- 3 bits Class and 5 bits Detail
- Formatted to display like this: 2.05
- Method for Request
  - Class = 0
- Result for Response
  - Class = 2, 4, 5

<table>
<thead>
<tr>
<th>Class</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Request</td>
</tr>
<tr>
<td>2</td>
<td>Success Response</td>
</tr>
<tr>
<td>4</td>
<td>Client Error Response</td>
</tr>
<tr>
<td>5</td>
<td>Server Error Response</td>
</tr>
</tbody>
</table>
Message ID

- Each Endpoint has its own IDs space
- 16 bit Unsigned Integer
- Sender endpoint assign new Message ID and recipient respond with same Message ID
- Only for the retransmission management
Message Types

- Confirmable = 0x0
- Non-Confirmable = 0x1
- Acknowledgement = 0x2
- Reset = 0x3
Non-Confirmable

Client → Server:
- NON [0x1234]
- NON [0x9854]
- 2.05 “Hello”

Server → Client:
- NON [0x1235]
- CON [0x8765]
- ACK [0x8765]
Retransmission

- 4 retransmissions.
- Interval is increase on every re-transmit.
- Max time is 247 secs
Retransmission (cont.)

- Client doesn’t know if request reached to server.

- Server logic will be executed more than once for same request. It should be cached.
Retransmission (cont.)

- ACK may be delivered after invocation of timeout.
- Server must send same response, including Piggybacked or not.
What is **Reset**?

- Endpoint has no context for the request.

```
Client

Crash!

CON [0x1235]
ACK [0x1235]

Server

CON [0x8765]
2.05 "Hello"
RST [0x8765]
```
or CoAP Ping

- Detecting the aliveness of the other endpoint
- Minimum packet is used. 4 bytes each.
- Make up for the unreliability of the Non-Confirmable request
CoAP URI

- `coap://example.com/hello?a=b&c=d`
- “coap” for non-secure CoAP
- “coaps” for secure CoAP
- Encoded into several Options
Observe

- Initial request is called Registration.
- Following responses are called Notification.
- Has sequence number.
- Notification can be Confirmable or Non-Confirmable.
- Draft extension.
Block-wise Transfer

- Big response can be sent as a small blocks with sequence number.
- Client who receive blocks will regenerate the original response.
- Preferred size will be sent from the client.
- Draft extension
Resource Discovery

/.well-known/core

CoRE Link Format

Copper(Cu) Firefox Add-on
http://mzl.la/1JNaB7v
Service Discovery

- UDP support Multicast.
- The multicast address for “All-CoAP-Server” is defined.
- Request must be Non-Confirmable
- Server must not send Rest as error response.
Network Errors

- **HTTP**: Errors are handled in TCP layer. If error happen, you can gave up very soon.

- **CoAP Confirmable**: Errors are handled in CoAP protocol layer, usually library takes care of them. Ordering problem may exist though.

- **CoAP Non-Confiramable**: Errors happens and ignored. You just cannot get response.
6LoWPANs

- IPv6 over Low-Power Wireless Personal Area Networks
- Slow, high error rate, small packet size
  - less than 80 bytes
HTTP and CoAP Proxy

- Route the HTTP request to CoAP Server and send back the CoAP response to HTTP Client
  - Or vice versa
- Because of the similarity with HTTP
- Cache
- Well defined in RFC
Security

- DTLS = Datagram TLS
  - Shared Key
  - Public Key and Private Key pair
- Supported by major library: OpenSSL, NSS, ...
- Both for **encryption** and **authentication**
Other IoT protocols

- MQTT
- WebSocket
MQTT

- Publish/Subscribe model
- Broker is required on Internet
- Require long-living TCP connection.
WebSocket

- Connection phase is on HTTP protocol
- Once it accepted, the socket is took over by WebSocket client and server and both endpoint are free to send packets.
- Disconnect detection,
- Protocol is defined in RFC.
- JavaScript APIs are well defined by W3C.
  - Only for client side.
Summary

- Good: CoAP is fast, lightweight HTTP.
- Bad: Small data size, unreliability, less information.
- Apps take care of errors more than http.
- Limited implementations. Not enough use cases for actual field.
- Extensions are still draft.