Mittuniversitetet
MID SWEDEN UNIVERSITY

CoAP

IoT Protocols

Stefan Forsström
Institution of Information Systems and Technology

Mittuniversitetet
MID SWEDEN UNIVERSITY

Overview

- History
- CoAP
- Communication Details
- Tips for the Lab Project

Mittuniversitetet
MID SWEDEN UNIVERSITY

Constrained Application Protocol (CoAP)

Mittuniversitetet
MID SWEDEN UNIVERSITY

History

- Devices on the IoT include sensors and actuators
  - Exposing resources (i.e., measurable data, sensors)
  - or enabling interaction with the environment (actuators)
- They have to find ways to
  - Register to services
  - Find each other
  - and interact without much human intervention
- Ideally we would be talking about a decentralized scenario
  - Much like how the web is built up of decentralized web servers

Mittuniversitetet
MID SWEDEN UNIVERSITY

History

- The Constrained Application Protocol (CoAP)
  - Specialized protocol for constrained device communication
  - Defined in RFC 7252. June 2014
  - Expanded in RFC 7959, 7959, 8323, and 8613
- Efficiency is very important.
  - Is intended for use in resource-constrained IP capable devices
  - Such as low end IoT devices or IoT devices in general
- Works on low throughput networks and devices that run on battery
  - For example Class 1 IoT devices with below:
    - 100KB of Flash
    - 10KB of RAM

Mittuniversitetet
MID SWEDEN UNIVERSITY

History

- CoAP is a REST-based protocol largely inspired by HTTP
  - However it brings the Web Server concept to the very constrained space where IoT devices are the ones exposing their resources
- CoAP devices are intended to come from multiple manufacturers, much like the World Wide Web enabled anyone to have an HTTP server.
- Like HTTP, CoAP also uses request/response communication
- CoAP is designed to easily translate to HTTP to fit with the web
  - While also meeting specialized requirements
    - Including multicast support
    - Very low overhead and simplicity

Mittuniversitetet
MID SWEDEN UNIVERSITY

COAP

Mittuniversitetet
MID SWEDEN UNIVERSITY

CoAP Definitions

- Defined in RFC 7252. June 2014
  - The Constrained Application Protocol (CoAP)
    - All the basic functionality
- Expanded in
  - RFC 7959
    - Block-Wise Transfers in the Constrained Application Protocol
      - For handling large and multiple messages
  - RFC 8323 and RFC 8613
    - CoAP over TCP, TLS, and WebSockets
    - Object Security for Constrained RESTful Environments (OSCORE)
      - For managing and supporting security

Mittuniversitetet
MID SWEDEN UNIVERSITY

CoAP Terms

- Before going deeper into the CoAP protocol, structure is useful to define some terms that we will use later:
  - **Endpoint**: An entity that participates in the CoAP protocol. Usually, an Endpoint is identified with a host
  - **Sender**: The entity that sends a message
  - **Recipient**: The destination of a message
  - **Client**: The entity that sends a request and the destination of the response (usually an application or platform)
  - **Server**: The entity that receives a request from a client and sends back a response to the client (usually the sensor)

Mittuniversitetet
MID SWEDEN UNIVERSITY

CoAP Functionality

- Constrained Application Protocol
  - REST-based web transfer protocol
  - Manipulates Web resources using the same methods as HTTP
    - GET, PUT, POST, and DELETE
  - Subset of HTTP functionality re-designed for low power embedded devices such as sensors (for IoT and M2M)
- CoAP provides reliability without using TCP as transport protocol
  - Basically a reliable UDP variant
  - TCP overhead is too high and its flow control is not appropriate for short-lived transactions
- CoAP can also handle asynchronous communication
  - For example, it can first ACKs the reception of the message and then send the response later in an off-line fashion

Mittuniversitetet
MID SWEDEN UNIVERSITY

Communication Details

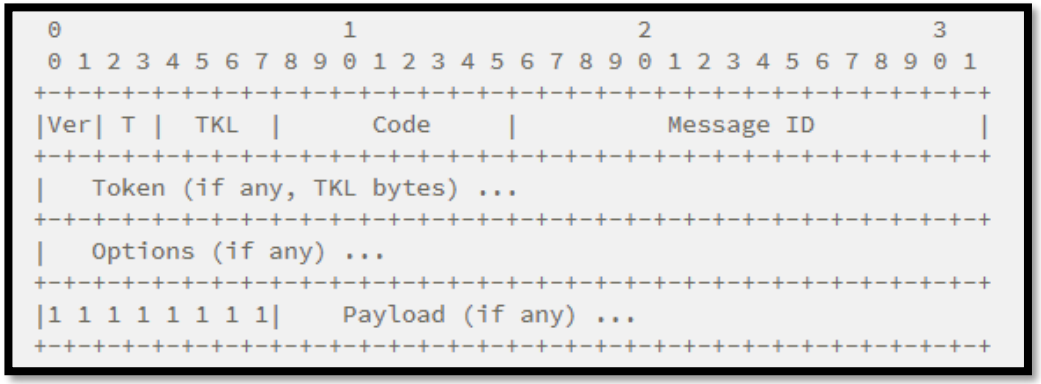
Mittuniversitetet
MID SWEDEN UNIVERSITY

Messsage types

- Four message types:
  - Confirmable – requires an ACK
  - Non-confirmable – no ACK needed
  - Acknowledgement – ACKs a Confirmable
  - Reset - indicates a Confirmable message has been received but context is missing for processing
- Most common methods are:
  - GET
  - POST
  - PUT
  - DELETE
  - Discover (a GET to .well-known/core to get the tree structure)
  - Observe (a GET variant to add a subscription)

COAP message definition

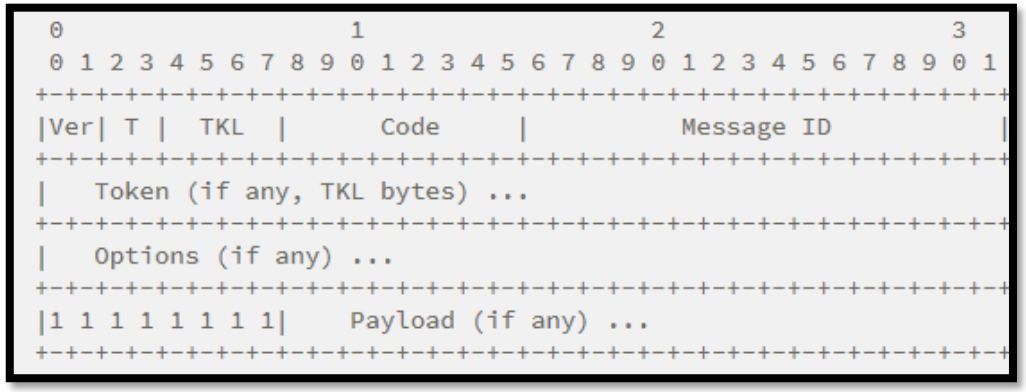
- CoAP messages are very compact and transported over UDP
  - Messages are encoded in a binary format with a header of 4 bytes



CoAP Header

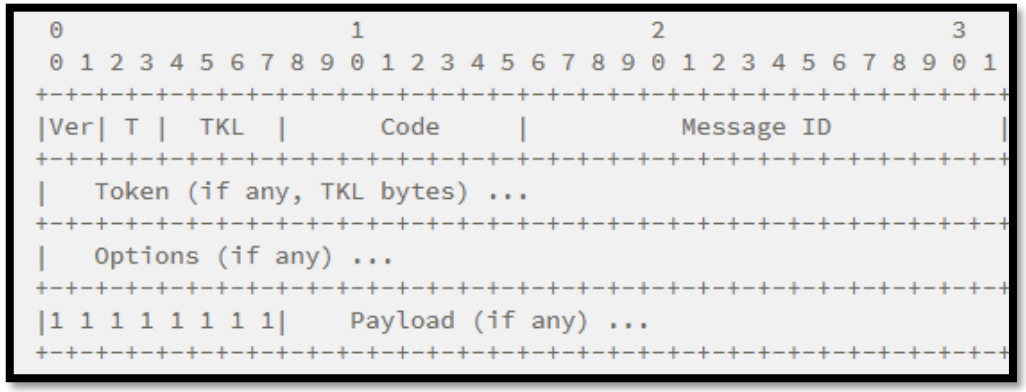
Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
4	32	VER		Type		Token Length				Request/Response Code								Message ID															
8	64	Token (0 - 8 bytes)																															
12	96																																
16	128	Options (If Available)																															
20	160	1	1	1	1	1	1	1	1	Payload (If Available)																							

Message details



- The version field Ver indicates the CoAP version
  - (01) is version 1
- The message type T can be
  - CON (00), NON (01), ACK (10) and RST (11)
    - Use NON (01) for simplicity
- TKL is token length, which specified the length of the token
  - Not needed, (you can it set to 0000) tokens are used to map req/resp
- This is followed by a method- or response-code
  - Method code: Empty (0000 0000), GET (0000 0001), POST (0000 0010), PUT (0000 0011), DELETE (0000 0100)
  - Response code 3+5 bits, Example: 2.00 OK (010 00000) 2.05 Content (010 00101), 4.04 Not Found (100 00100)

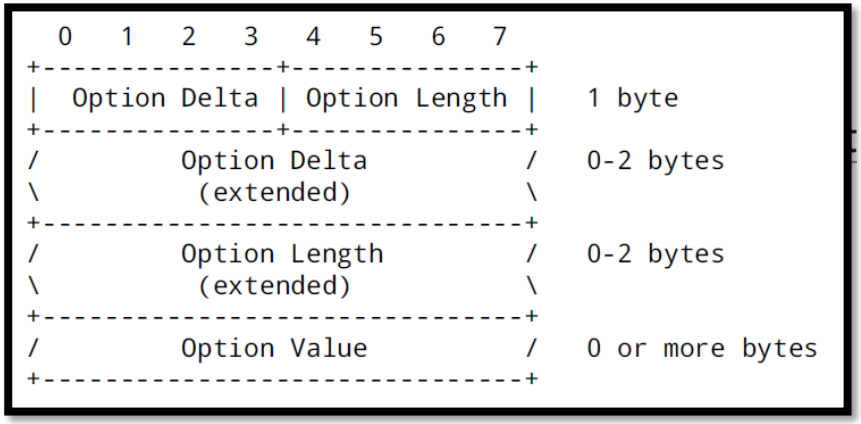
Message details



- Message ID, which is a 16 bit field to detect message duplication
  - Start at random number and then increment
- After the first row, there comes a token which can be ignored if you set the length to 0 before
  - Tokens are used to map a responses to a originating request in asynchronous or bulk responses
- After this, there comes an options field for extensibility
  - Optional, but practically mandatory
  - Since it is what specified the URI path and content format
- The header ends with (1111 1111) 255 in binary to mark the end of the options and beginning of the payload

Message details

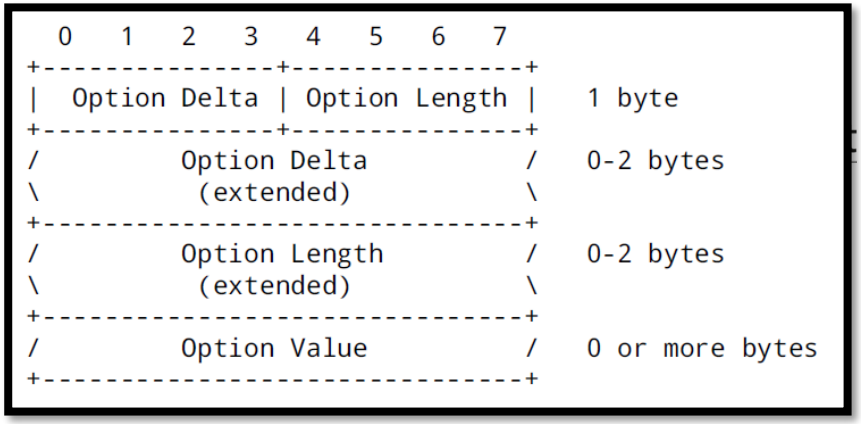
- Options header is 1 byte
  - First four bits indicate which option type it is
  - The remainder four bits are the length of the options value
- The options type are specified as a delta from the previous option (if many)
- Useful example:
  - Header: Uri Path 11 with length 4 (1011 0100)
  - Options value: **sink** (0111 0011, 0110 1001, ..., etc.)
  - Header: text/plain 12 i.e. delta 1 (0001 0000)



No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see note 1)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see note 1)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x				Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
28			x		Size2	uint	0-4	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)
60			x		Size1	uint	0-4	(none)

Media type	Id.
text/plain;charset=utf-8	0
application/link-format	40
application/xml	41
application/octet-stream	42
application/exi	47
application/json	50
application/cbor	60

Extended options



- Sometimes 4 bits are not enough for the options delta or for the option lengths
  - Ex. When there are long path names
- For long option deltas:
  - **For delta between 0 to 12:** set option delta to the real value: **0 to 12**
    - Represents the exact delta value, with no option delta extended value
  - **For delta between 13 to 268:** set options delta to **13**
    - Option delta becomes extended as an 8-bit value that represents the option delta value minus 13
  - **For delta from 269 to 65,804**: set options delta to **14**
    - Option delta extended becomes a 16-bit value that represents the option delta value minus 269
- For long option lengths:
  - **For lengths between 0 to 12:** set length to the real value: **0 to 12**
    - Represents the exact length value, with no option length extended value
  - **For option length from 13 to 268**: set option length to **13**
    - Option length extended is then aa 8-bit value that represents the option length value minus 13
  - **For option length from 269 to 65,804**: set option length to **14**
    - Option length extended is then a 16-bit value that represents the option length value minus 269

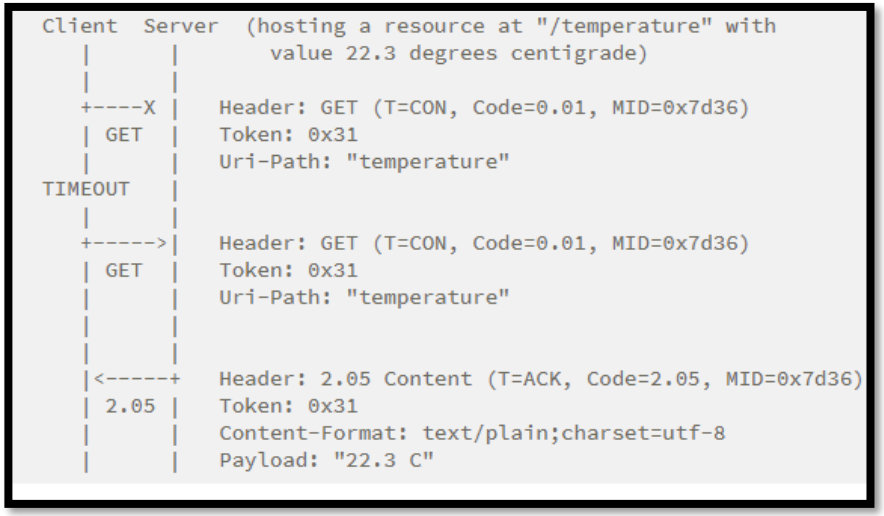
Mittuniversitetet
MID SWEDEN UNIVERSITY

COAP Reliable Transmission

- CoAP was intended for UDP transmission, which is unreliable
  - This means that CoAP request and response messages may arrive out of order, appear duplicated, or go missing without notice.
- CoAP implements a lightweight reliability mechanism
  - Including "confirmable" and "non-confirmable" messages
  - If the messages is confirmable (CON), that means that either the request or the response require an acknowledgement (ACK).
- To ensure retransmission in case of loss
  - Endpoints sending a CON message keeps track of the timeout and number of resends for each message

COAP message definition

- This figure shows how a simple CoAP Request
  - With timeout and resends
- I.E. the first message gets lost due to the unreliable nature of UDP
  - So the client needs to retransmit the message after waiting for an acknowledgement until timeout
- Client sends a **confirmable** request
  - GET /temperature
  - This requires an acknowledgment from the server
  - The message ID is 0x7d36, which will be returned
  - Without the ID, a client could receive duplicated



Observing Resources

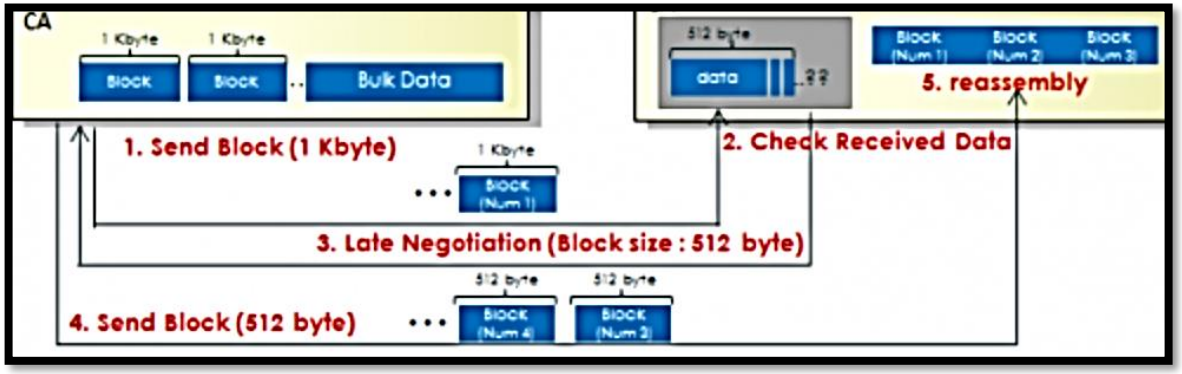
- Basic CoAP functions as a simple resource retrieve system (GET)
  - However, in CoAP one might want to observe a resource and get events upon changes instead. (Pub/Sub)
- In CoAP, this has been solved by using the Observe option.
  - Basically, we add the Observe functionality to the GET as an option (Nr. 6)

No.	C	U	N	R	Name	Format	Length	Default
6		x	-		Observe	uint	0-3 B	(none)

- If observe is added to the GET method
  - The server will add/remove a client from a list of observers of a resource.
  - The value of the Observe option is either 0 for register or 1 for deregister.
- If the servers returns a successful response (2.xx)
  - With the observe option included, that means that the client has been added
  - You can also add more options to the observe, for example Max age

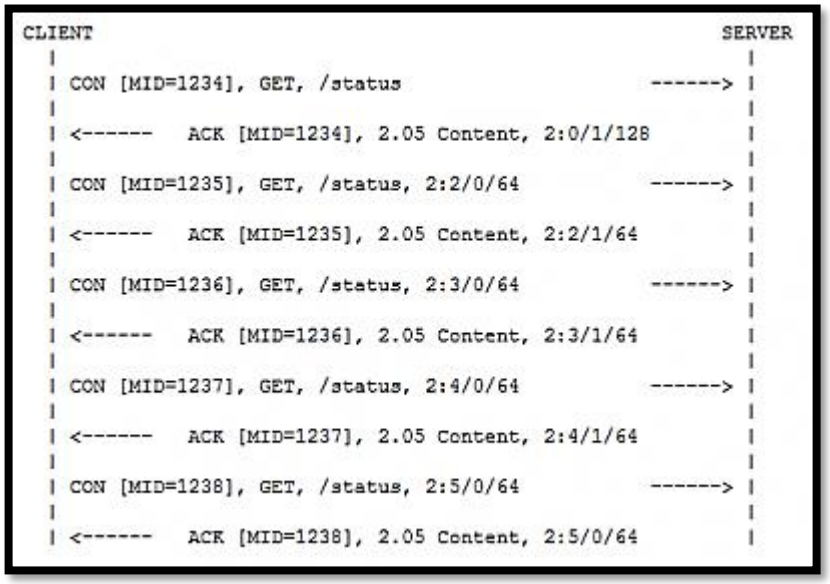
Block Transfer

- There is a hard limit on the size of an UDP packet
  - Which will be problematic when sending or retrieving large data
- Bulk transfer was made in RFC 7959 to solve this
  - Two variants but quite similar (Block1 and Block2)
  - Block1 is useful with the payload-bearing POST and PUT requests
  - Block2 is useful with payload-bearing GET, POST, PUT responses
- General Example



Block Transfer

- Three items of information needs to be communicated
  - The block number(NUM) within the sequence
  - The size of the block (SZX)
  - Whether more blocks are following (M)
- Example: **NUM / M / SZX**
  - With renegotiated size
    - From 128 to 64
  - Meaning, first block is both 1 and 2 (128 bytes)



Mittuniversitetet
MID SWEDEN UNIVERSITY

Tips for the Lab Project

Mittuniversitetet
MID SWEDEN UNIVERSITY

Tips for the Lab Project

- Start by using a normal Coap Client, connect to coap.me and observe the messages in wireshark (udp.port == 5683)
- Then create your own UDP socket connecting to coap.me on port 5683
  - And listening to incoming packets back
- Try to send a very simple GET message
  - Byte 1: (0101 0000) Coap version 1, NON, no token
  - Byte 2: (0000 0001) GET
  - Byte 3: (1010 1010) Random msg id number part 1
  - Byte 4: (0101 0101) Random msg id number part 2
- Upon receiving incoming packets
  - Split on (1111 1111) 255/ff
  - Print header and payload

Mittuniversitetet
MID SWEDEN UNIVERSITY

Tips for the Lab Project

- Remember that you can use bitwise operators
  - For example:
    - **And &** to mask out bits and find if a bit is set
      - 1010 1111 & 0010 0000 = 0010 0000 = true
      - 1010 1111 & 0001 0000 = 0000 0000 = false
    - **Or |** to join bytes, for example two half bytes
      - 1010 0000 | 0000 1111 = 1010 1111
  - You also have:
    - ~ (bitwise NOT)
    - ^ (bitwise XOR)
    - << (bitwise left shift)
    - >> (bitwise right shift)

Mittuniversitetet
MID SWEDEN UNIVERSITY

Tips for the Lab Project

- The actual client program does not need to be fancy
  - Just make a simple console application with multiple choices and that reads keyboard inputs for host, path, etc.
- Remember that the options are specified as delta values
  - Meaning that they need to come in numerical order
- The content length is very important in CoAP
  - The URI length needs to be correctly specified in the packet/header
    - Before it is written out. As there is only a delimiter for the payload (ff)
- Wireshark is your friend
  - To see how other CoAP clients messages looks like is very good for you to learn, debug and compare to yourself to them
- Discover requires you to implement block transfer and handle long path names, so it is not mandatory for the lab

Mittuniversitetet
MID SWEDEN UNIVERSITY

Contact Information

STEFAN FORSSTRÖM
PhD in Computer and System Sciences

MID SWEDEN UNIVERSITY
Institution of Information Systems and Technology (IST)
Campus Sundsvall, Room L426
Email: stefan.forsstrom@miun.se