The Constrained Application Protocol (CoAP)

Moosa Yahyazadeh

The University of Iowa

November 5, 2016

What is CoAP?

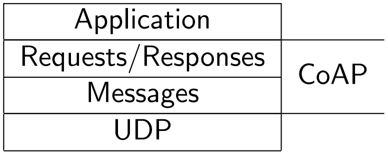
- The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks
- The protocol is designed for machine-to-machine (M2M) applications
  - smart energy
  - building automation
- It provides a request/response interaction model between application endpoints
  - One design goal $\rightarrow$ keep message overhead small
    - Why? limiting the need for fragmentation in constrained environments

Request/response interaction model

- Interaction model of CoAP is similar to the client/server model of HTTP
- Machine-to-Machine interaction result in CoAP implementation acting in both client and server roles
- CoAP request/response
  - a request is sent by a client for an action (using a Method Code) on a resource (identified by a URI) on a server
  - server then sends a response with a Response Code; this response may include a resource representation
- Unlike HTTP, CoAP deals with these interchanges asynchronously over a datagram-oriented transport such as UDP.
  - using messages layer that supports optional reliability

CoAP layers

- Abstract Layering of CoAP



- One could think of CoAP logically as using a two-layer approach, a CoAP messaging layer used to deal with UDP and the asynchronous nature of the interactions, and the request/response interactions using Method and Response Codes
- CoAP is however a single protocol, with messaging and request/response as just features of the CoAP header.

Messaging model

- Message types
  - Confirmable
  - Non-confirmable
  - Acknowledgement
  - Reset
- Method Codes and Response Codes included in some of these messages make them carry requests or responses
- The basic exchanges of the four types of messages are somewhat orthogonal to the request/response interactions; requests can be carried in Confirmable and Non-confirmable messages, and responses can be carried in these as well as piggybacked in Acknowledgement messages
  - Thus, Requests cannot be carried in Ack messages

Messaging model (Cont...)

- Each message contains a Message ID
- Reliability is provided by marking a message as Confirmable (CON)
  - A Confirmable message is retransmitted using a default timeout and exponential back-off between retransmissions, until the recipient sends an Acknowledgement message (ACK) with the same Message ID from the corresponding endpoint
  - When a recipient is not at all able to process a Confirmable message (i.e., not even able to provide a suitable error response), it **replies** with a Reset message (RST) instead of an Acknowledgement (ACK).
- Non-confirmable messages are not acknowledged, but still have a Message ID for duplicate detection
  - When a recipient is not able to process a Non-confirmable message, it **may reply** with a Reset message (RST)

Request/response model

- Request and response semantics are carried in CoAP messages
  - Request → include Method Code
  - Response → include Response Code
  - Optional (or default) request and response information, such as the URI and payload media type are carried as CoAP Options
  - matching responses to requests independently from the underlying messages is done by Token
    - Notice: Token is a concept separate from the Message ID
- Scenario: A request is carried in a Confirmable (CON)
  - If immediately available, the response to a request carried in a Confirmable message is carried in the resulting (ACK) message
    - Called a piggybacked response; same Msg ID, same Token
    - No need for separately acknowledging a piggybacked response, as the client will retransmit the request if the (ACK) message carrying the piggybacked response is lost

Request/response model (Cont...)

- Scenario: A request is carried in a (CON) and the server is not able to respond immediately (with Pigg. Res.)
  - It simply responds with an Empty Ack message so that the client can stop retransmitting the request
  - When the response is ready, the server sends it in a new Confirmable message (which then in turn needs to be acknowledged by the client)
    - called a "separate response"
    - different Msg ID, same Token
- Scenario: A request is carried in a Non-confirmable (NON)
  - the response is sent using a new Non-confirmable message, although the server may instead send a Confirmable message
    - The response $\rightarrow$ different Msg ID, same Token

Request/response model (Cont...)

- Methods in CoAP:
  - GET, PUT, POST, and DELETE
    - Similar manner to HTTP, but there are some differences
  - Methods beyond the basic four can be added to CoAP in separate specifications
    - New methods do not necessarily have to use requests and responses in pairs
    - Even for existing methods, a single request may yield multiple responses, e.g., for a multicast request or with the Observe option

Message format

- CoAP is based on the exchange of compact messages
    - transported over UDP $\rightarrow$ each CoAP message occupies the data section of one UDP datagram
    - may also be used over Datagram Transport Layer Security (DTLS)
    - It could also be used over other transports such as SMS, TCP, or SCTP
    - UDP-lite [RFC3828] and UDP zero checksum [RFC6936] are not supported by CoAP

Message format (Cont...)

- messages are encoded in a simple binary format
- message structure
  - header
  - token (if any)
  - options (if any)
  - payload marker (if there is any payload)
  - payload (if any)

Ver	T	TKL	Code	Message ID
Token (if any, TKL bytes) ...				
Options (if any) ...				
11111111			Payload (if any) ...	

Message format - Header

- header (fixed-size 4-byte)
  - Version (Ver): 2-bit unsigned integer
    - CoAP version number
    - Implementations of this specification **MUST** set this field to 1 (01 binary). Other values are reserved for future versions. Messages with unknown version numbers **MUST** be silently ignored
  - Type (T): 2-bit unsigned integer
    - Indicates if this message is of type Confirmable (0), Non-confirmable (1), Acknowledgement (2), or Reset (3)
  - Token Length (TKL): 4-bit unsigned integer
    - Indicates the length of the variable-length Token field (0-8 bytes)
    - Lengths 9-15 are reserved, MUST NOT be sent, and MUST be processed as a message format error

Message format - Header (Cont...)

- header (fixed-size 4-byte) (Cont...)
  - Code: 8-bit unsigned integer
    - split into a 3-bit class (most significant bits) and a 5-bit detail (least significant bits)
    - documented as "c.dd". "c" is a digit from 0 to 7 for the 3-bit subfield and "dd" are two digits from 00 to 31 for the 5-bit subfield.
    - The class can indicate a request (0), a success response (2), a client error response (4), or a server error response (5). (All other class values are reserved.)
    - As a special case, Code 0.00 indicates an Empty message.
    - In case of a request, the Code field indicates the Request Method; in case of a response, a Response Code.

Message format - Header (Cont...)

- header (fixed-size 4-byte) (Cont...)
  - Message ID: 16-bit unsigned integer
    - Used to detect message duplication
    - Used to to match messages of type Acknowledgement/Reset to messages of type Confirmable/Non-confirmable.
    - These rules will be described later
    - Its 16-bit size enables up to about 250 messages per second from one endpoint to another with default protocol parameters

Message format - Token

Rules for generating a Message ID and matching messages

Code registries

Request/Response Semantics

Multicast CoAP

Section 8 discusses the proper use of CoAP messages with multicast addresses and precautions for avoiding response congestion