The Constrained Application Protocol (CoAP)

Moosa Yahyazadeh

The University of Iowa

November 6, 2016

What is CoAP?

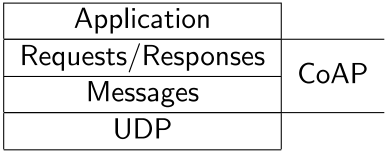
- The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks
- The protocol is designed for machine-to-machine (M2M) applications
  - smart energy
  - building automation
- It provides a request/response 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

Request/Response Semantics

- CoAP operates under a similar request/response model as HTTP
  - a CoAP endpoint in the role of a "client" sends one or more CoAP requests to a "server", which services the requests by sending CoAP responses
- Unlike HTTP, requests and responses are not sent over a previously established connection but are exchanged **asynchronously** over CoAP messages

Request/Response Semantics (Cont...)

- Request
  - Contains
    - a method to be applied to the resource
    - the identifier of the resource
    - a payload and Internet media type (if any)
    - and optional metadata about the request
- Response
  - After receiving and interpreting a request, a server responds with a CoAP response that is matched to the request by means of a client-generated token; note that this is different from the Message ID that matches a Confirmable message to its Acknowledgement.
  - can be sent as Piggybacked, Separate

Request/Response Semantics (Cont...)

- Piggybacked response
  - Response (whether success or failure) is carried directly in the Ack message (request was carried in a Confirmable message)
  - The protocol leaves the decision whether to piggyback a response or not to the server
  - The client MUST be prepared to receive either
  - There is a strong expectation that servers will use piggyback whenever possible
    - Saving resources in the network

Request/Response Semantics (Cont...)

- Separate response
  - Whenever that is not possible to return a piggybacked response
    - Server might need longer to obtain the representation of the resource requested than it can wait to send back the Ack message (Using a Ack timer), without risking the client repeatedly retransmitting the request message
    - The response to a request carried in a Non-confirmable message is always sent separately
  - The sep. res. can be Confirmable or not (server's decision)
  - Implementation Note: The protocol leaves the decision whether to piggyback a response or not (i.e., send a separate response) to the server. If possible piggyback is better. The client MUST be prepared to receive either.

Request/Response Semantics (Cont...)

- Separate response
  - Logic
    - It sends the Ack to the Confirmable req as an Empty message
    - Once it's done, server must not send back the res. in another Ack, even if the client retransmits another identical req.
    - If a retransmitted request is received (perhaps because the original Ack was delayed), another Empty Ack is sent, and any response MUST be sent as a separate response
    - If the server then sends a Confirmable response, the client's Ack to that response MUST also be an Empty message (one that carries neither a request nor a response)
    - The server MUST stop retransmitting its response on any matching Ack or Reset message
    - If sep. res. arrived before ack (it may happen in UDP), this also serves as an ack
    - It the res won't come within reasonable time, client should back off (better to to set up a timeout that is unrelated to retransmission timers)

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** (Class=0), the Code field indicates the **Request Method**
    - In case of a **response** (Class $\in \{2, 4, 5\}$), the Code field indicates 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

- Token
  - Token value may be 0 to 8 bytes as given by the Token Length field
  - Used to correlate requests and responses
  - Every message carries a token, even if it is of zero length
  - Every request carries a client-generated token that the server MUST echo (without modification) in any resulting response
  - Intended for use as a client-local identifier for differentiating between concurrent requests; could be called a "request ID"
  - SHOULD use a nontrivial, randomized token to guard against spoofing of responses (if it doesn't use Transport Layer Security)
  - An endpoint receiving a token it did not generate MUST treat the token as opaque and make no assumptions about its content or structure.

Message format - Options

- Options
  - There can be zero or more Options
  - An Option can be followed by the end of the message, by another Option, or by the Payload Marker and the payload
  - Each option instance specifies the Option Number, the length of the Option Value, and Option Value.
    - Option number is calculated using simply summing the Option Delta values of this and all previous options before it

Request/Response Matching Rules

- The source endpoint of the response MUST be the same as the destination endpoint of the original request.
- In a piggybacked response, the Message ID of the Confirmable request and the Acknowledgement MUST match, and the tokens of the response and original request MUST match.
- In a separate response, just the tokens of the response and original request MUST match.
- In case a message carrying a response is unexpected (the client is not waiting for a response from the identified endpoint, at the endpoint addressed, and/or with the given token), the response is rejected

Code registries

- Code ranges:
    - 0.00 $\rightarrow$ Indicates an Empty message
    - 0.01-0.31 $\rightarrow$ Indicates a request
        - Initial CoAP Method Codes: 0.01 $\rightarrow$ GET, 0.02 $\rightarrow$ POST, 0.03 $\rightarrow$ PUT, 0.04 $\rightarrow$ DELETE
        - All other Method Codes are Unassigned
    - 1.00-1.31 $\rightarrow$ Reserved
    - 2.00-5.31 $\rightarrow$ Indicates a response
        - The Response Codes 3.00-3.31 are Reserved for future use
    - 6.00-7.31 $\rightarrow$ Reserved

Code registries (Cont...)

- Response codes
    - 2.01 → Created, 2.02 → Deleted, 2.03 → Valid, 2.04 → Changed, 2.05 → Content
    - 4.00 → Bad Request, 4.01 → Unauthorized, 4.02 → Bad Option, 4.03 → Forbidden, 4.04 → Not Found, 4.05 → Method Not Allowed, 4.06 → Not Acceptable, 4.12 → Precondition Failed, 4.13 → Request Entity Too Large, 4.15 → Unsupported Content-Format
        - All other codes of the class should be treated as the generic response code of the class (4.00)
    - 5.00 → Internal Server Error, 5.01 → Not Implemented, 5.02 → Bad Gateway, 5.03 → Service Unavailable, 5.04 → Gateway Timeout, 5.05 → Proxying Not Supported
        - All other codes of the class should be treated as the generic response code of the class (5.00)
    - The Response Codes 3.00-3.31 are Reserved for future use
    - All other Response Codes are Unassigned

Multicast CoAP

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