Proxy Operations for CoAP Group Communication

draft-ietf-core-groupcomm-proxy-01

Marco Tiloca, RISE
Esko Dijk, IoTconsultancy.nl

IETF 119 Meeting – Brisbane – March 20^{th}, 2024
Recap

› Adopted as WG document in December 2023

› **Scope:** definition of proxy operations for CoAP group communication
  – Signaling protocol between client and proxy, with two new CoAP options
  – Individual responses from the CoAP servers are relayed back to the client
  – Support for forward-proxies, reverse-proxies, chain of proxies, and HTTP-CoAP proxies
  – Updated CoAP freshness model and validation model for cached responses in groups

› **The proxy is explicitly configured to support group communication**
  – Clients are allowed-listed on the proxy, and identified by the proxy

› Address issues discussed in **Section 3.5 of draft-ietf-core-groupcomm-bis**
Gist of the protocol

› In the unicast request addressed to the proxy, the client indicates:
  – To be interested / capable of handling multiple responses
  – For how long the proxy should collect and forward responses
  – In the new CoAP option Multicast-Timeout, removed by the proxy

› In each response to the group request, the proxy includes addressing information pertaining to the server
  – In the new CoAP option Reply-To (old name: Response-Forwarding)
  – The client can distinguish responses and different servers
  – The client may later contact an individual server (directly if possible, or again via the proxy)

› Group OSCORE can be used for end-to-end security between client and servers

› Security is used between Client and Proxy, especially to identify the Client
  – (D)TLS or OSCORE (see draft-ietf-core-oscore-capable-proxies)
Updates in v -01

› Simple changes
  – Editorial fixes and readability improvements
  – IANA considerations: use the "Hypertext Transfer Protocol (HTTP) Field Name" registry

› Clarifications
  – Definition of “individual request” in the terminology:
    › A request that an origin client sends to a single origin server within a group, either
directly, or indirectly via a proxy.
  – UDP/IP multicast is the default transport
    › Alternatives are possible but out of scope here, like in draft-ietf-core-groupcomm-bis

› Considered also the CoAP options Proxy-Cri and Proxy-Scheme-Number
  – Defined in draft-ietf-core-href
Updates in v -01

› Addressed two points about reverse-proxies – Thanks, Christian!
   – Resolution based on discussions at an interim meeting [1] and on the mailing list [2]

› Point #1
   – The unicast request from the client has always to include the Multicast-Timeout Option
     › Otherwise, the proxy replies with an error. Client then includes the option
     › The client does not assume a default, pre-configured timeout at the proxy

› Point #2
   – To specify forwarding instructions, do not use a method like the one in RFC 8075
     › That is, do not use Uri-Path Options to convey host/port information
     › Use the Uri-Host and Uri-Port Options instead, as expected in CoAP
   – Revised the example in Appendix A.1 (efficient proxy with a single IP address)

[2] https://mailarchive.ietf.org/arch/msg/core/BsYKAFTozgt00ndQHTquRIMSoxk/
Updates in v -01

Reply-To Option (old name: Response-Forwarding)

› Clarified meaning of the option value
  - Addressing information pertaining to the origin server that generated the response
    › The client can use it to send an individual request intended to that server
  - Rationale: if the client sends a follow-up request using that information, then the request will eventually reach that origin server. (Different cases in a later slide)

› Name “Reply-To”: short, memorable, and aligned with the intended meaning
  - No intent to suggest/recommend/trigger a follow-up request always
  - The client can use it to distinguish responses from different origin servers
  - Possible alternative names: “Resp-From”, “Proxied-Response”, “Responder-Locator”, …
Updates in v -01

Reply-To Option (old name: Response-Forwarding)

› New encoding of the option value, using CRIs [3]
  – Binary serialization of a CBOR Sequence, of at most two elements
    1. REQUIRED: a CRI, with only the ‘scheme’ and ‘authority’
    2. OPTIONAL: a CRI reference
      - With ‘scheme’ set to null, and at least one of ‘authority’ and ‘path’ given
      - Useful only for particular setups with a reverse-proxy (see later slide)

› A proxy adds the option to the response as soon as possible
  – If the proxy caches responses, then a cached response has the option included

› Revised encoding of the corresponding HTTP header field
  – Now a base64url string without padding, encoding the value of the CoAP option

Updates in v-01

Reply-To Option used in different setups

› **Forward-proxy**
  - X1: actual address ADDR_S of the origin server; X2: not used
  - As a follow-up, the client can:
    › Send a request to ADDR_S and directly reach the server; or
    › Send a request to the proxy, specifying ADDR_S with the proxy-related options

› **Reverse-proxy, hiding the group but not the individual servers**
  - X1: actual address ADDR_S of the origin server; X2: not used
  - The client can send a follow-up request to ADDR_S and directly reach the server
Updates in v -01

Reply-To Option used in different setups

› Reverse-proxy, hiding the group and also the individual servers
  - X1: an address ADDR_P of the proxy
  - X2 (if present): components to use in the Uri-Host/Uri-Port/Uri-Path Options
  - The client can send a follow-up request to ADDR_P
    › If X2 is used, the request has that information as Uri-Host/Uri-Port/Uri-Path Options
    › X2 is good for a reverse-proxy with single IP address (see example in Appendix A.1)

› In a chain of such reverse-proxies
  - As usual, the last proxy adjacent to the origin server adds the option to the response
  - Each other proxy receiving a response with Reply-To=TARGET_OLD:
    › Replaces the option value with a new value TARGET_NEW, such that …
    › when receiving a request targeting TARGET_NEW, it is forwarded to TARGET_OLD
Next steps

› **Some points to address in the next versions**
  – Cancellation of ongoing response forwarding
  – Response revalidation between proxy and servers, when using Group OSCORE
    › Placeholder note in Sections 7.2.1 and 7.2.2: introduce an outer ETag Option
    › Perhaps it can be defined in `draft-amsuess-core-cachable-oscore`?
  – Enable response forwarding to an HTTP client via streamed delivery
    › Using the HTTP Transfer-Coding:chunked
  – Revisit and extend the RFC 8075 security considerations on HTTP-CoAP proxies
  – Add examples with an HTTP-to-CoAP proxy
  – Terminology alignment with `draft-bormann-core-responses`

› **Comments and reviews are welcome!**
Thank you!

Comments/questions?

https://github.com/core-wg/groupcomm-proxy
Updates since version -05 (1/3)

› Last presentation, of version -05, at the CoRE interim on 2021-10-27

› Version -06 submitted before IETF 113 (not presented)

› "Multicast-Timeout" Option
  – Renamed from "Multicast-Signaling", as suggested by Carsten
  – Max length reduced to 4 bytes, as suggested by Christian

› "Response-Forwarding" Option
  – Updated semantics on port number "null" or absent (swapped)
  – "null" --> same as destination port number of the group request
  – absent --> default port number
Backup

(Note: old name “Response-Forwarding” is used)
Example with forward-proxy (1/2)

```
Src: C_ADDR:C_PORT
Dst: P_ADDR:P_PORT
Proxi-URI {
    coap://G_ADDR:G_PORT/r
}

Multicast-Timeout: 60

Src: P_ADDR:P_PORT
Dst: G_ADDR:G_PORT
Uri-Path: /r

/* t = 0 : P starts accepting responses for this request */
```
Example with forward-proxy (2/2)
Example #1 with reverse-proxy (1/3)

- C→P: CoAP over TCP
- p.example.com resolves to the address of P
- group1.com resolves to the multicast address of the group
- The proxy hides the group as a whole and the individual servers

```plaintext
C

C.ADDR:C.PORT
Dst: p.example.com:P.PORT
Uri-Path:
  /cp/coap://group1.com/r
Multicast-Timeout: 60

P

/* C embeds the group URI into its request to the proxy */

Src: P_ADDR:P.PORT
Dst: G_ADDR:G.PORT
Uri-Path: /r

S1

/* t = 0: P starts accepting responses for this request */

S2
```
Example #1 with reverse-proxy (2/3)

- C→P: CoAP over TCP
- p.example.com resolves to the address of P
- group1.com resolves to the multicast address of the group
- The proxy hides the group as a whole and the individual servers
- Dx_ADDR:Dx_PORT is mapped to address and port of server Sx

```
/* At t = 60, P stops accepting responses for this request */
```
Example #1 with reverse-proxy (3/3)

- C→P: CoAP over TCP
- p.example.com resolves to the address of P
- group1.com resolves to the multicast address of the group
- The proxy hides the group as a whole and the individual servers
- Dx_ADDR:Dx_PORT is mapped to address and port of server Sx
Example #2 with reverse-proxy (1/3)

- C→P: CoAP over TCP
- group1.com resolves to the address of P
- The proxy hides the group as a whole and the individual servers

```
Sn: C_ADDR:C PORT
Dst: group1.com:P PORT
URI-Path: /r

Sn: group1.com:P PORT
Dst: C_ADDR:C PORT
4.00 Bad Request
Multicast-Timeout: (empty)
Payload: "Please use
Multicast-Timeout"

Sn: C_ADDR:C PORT
Dst: group1.com:P PORT
Multicast-timeout: 60
URI-Path: /r

Sn: P_ADDR:P PORT
Dst: G_ADDR:G PORT
URI-Path: /r

/* t = 0 : P starts accepting responses for this request */
```
Example #2 with reverse-proxy (2/3)

- C→P: CoAP over TCP
- group1.com resolves to the address of P
- The proxy hides the group as a whole and the individual servers
- Dx_ADDR:Dx_PORT is mapped to address and port of server Sx
Example #2 with reverse-proxy (3/3)

- C→P: CoAP over TCP
- group1.com resolves to the address of P
- The proxy hides the group as a whole and the individual servers
- Dx_ADDR:Dx_PORT is mapped to address and port of server Sx
Example with HTTP-CoAP proxy

POST https://proxy.url/hc/?target_uri=coap://G_ADDR:G_PORT/ HTTP/1.1
Content-Length: <REQUEST_TOTAL_CONTENT_LENGTH>
Content-Type: text/plain
Multicast-Timeout: 60

HTTP/1.1 200 OK
Content-Length: <BATCH_RESPONSE_TOTAL_CONTENT_LENGTH>
Content-Type: multipart/mixed; boundary=batch_foo_bar
--batch_foo_bar
Content-Type: application/http

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: <INDIVIDUAL_RESPONSE_1CONTENT_LENGTH>
Response-Forwarding: coap://S1_ADDR:G_PORT

Body: Done!
--batch_foo_bar
Content-Type: application/http

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: <INDIVIDUAL_RESPONSE_2CONTENT_LENGTH>
Response-Forwarding: coap://S2_ADDR:S2_PORT

Body: More than done!
--batch_foo_bar--

› C → P: HTTP unicast group request
  - P converts it to a CoAP group request
  - Forwarded to coap://G_ADDR:G_PORT

› P accepts responses for 60 s

› S1 → P: CoAP response
  - Converted to HTTP and stored

› S2 → P: CoAP response
  - Converted to HTTP and stored

... ... ... TIMEOUT!

› P prepares one HTTP “batch” response
  - Include the different individual responses, one for each replying server

› P → C: HTTP “batch” response

› C extracts the individual HTTP responses from the “batch” response