Proxy Operations for CoAP Group Communication

Work in progress towards
draft-tiloca-core-groupcomm-proxy-04

Marco Tiloca, RISE
Esko Dijk, IoTconsultancy.nl

CoRE WG Interim Meeting, June 9th, 2021

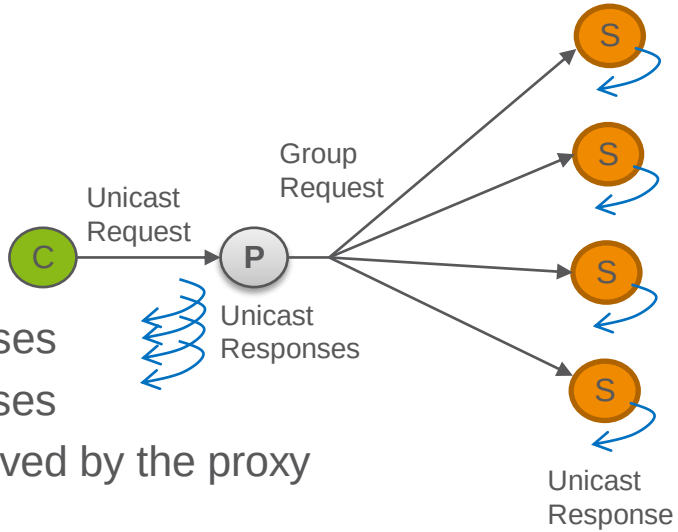
Recap

- CoAP supports group communication over IP multicast
  - Section 3.4 of *draft-ietf-core-groupcomm-bis* discusses issues when using a proxy
  - The proxy forwards a request to the group of servers, over IP multicast
  - Handling responses and forwarding them back to the client is not trivial

- Contribution – Description of proxy operations for CoAP group communication
  - Addresses all issues mentioned in *draft-ietf-core-groupcomm-bis*
  - Signaling protocol between client and proxy, with two new CoAP options
  - Forwarding back of individual responses to the client
  - Support for both forward-proxies (main case) and reverse-proxies
  - Response caching at the proxy (**added to the Editor's copy**)

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

Recap (how it works)



- In the unicast request addressed to the proxy, the client indicates:
  - To be interested / capable of handling multiple responses
  - How long the proxy should collect and forward responses
  - With the new CoAP option **Multicast-Signaling**, removed by the proxy
- In each response to a group request, the proxy includes the server address
  - In the new CoAP option **Response-Forwarding**
  - Now, the client can distinguish responses and different servers
  - The client can contact an individual server (directly, or again via the proxy)
- Group OSCORE can be used for e2e security between client and servers
- Required security association between Client and Proxy (e.g., OSCORE or DTLS)

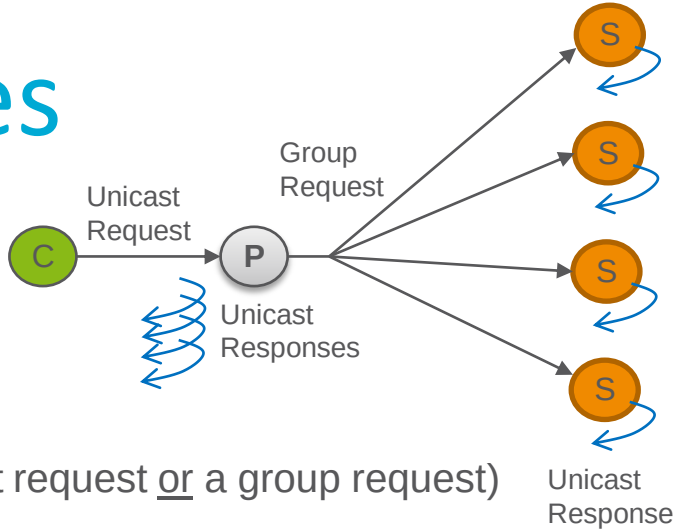
Content moved in

- Taken out from *draft-ietf-core-groupcomm-bis*
  - General caching model at the proxy
  - Response re-validation between Client and Proxy
    - Based on the new Group-ETag option
  - Above 2 points are now updating RFC 7252
  - Covering also the case with end-to-end security based on Cacheable OSCORE
    - Limited to (REST) safe requests with no side-effects on resource at the servers
    - https://datatracker.ietf.org/doc/draft-amsuess-core-cachable-oscore/

- Included in the Editor's copy, with some clarifications
  - https://gitlab.com/crimson84/draft-tiloca-core-groupcomm-proxy/-/tree/v-04

Caching model at proxies

- Two types of cache entries



- "**Individual**" cache entry
  - Populated with the response from one server (to a unicast request or a group request)
  - Hit by a matching unicast request intended to that server
  - Response added to the cache entry right away

- "**Aggregated**" cache entry
  - Populated with all the responses to a group request, from any server in the group
  - Can be updated with responses to unicast request too
  - Hit by a matching group request intended to all servers

Caching model clarifications

- As requested by Christian (**added to the Editor's copy**)

- "Send immediately – Update later" policy
  - Multiple responses to the same Group Request can come from the **same server**
  - Already the case for the origin client; the Proxy has to think the same way too
  - Responses are simply forwarded back as they come
  - Only when it stops collecting responses, Proxy actually updates the "Aggregated" cache entry

- Rules for retention and management of Aggregated cache entries
  - Servers can join at any time → The cache entry might not fully reflect the group membership
  - Policies to invalidate/refresh an Aggregated cache entry
    - If the Proxy is sitting on a (multicast) router, it can see servers as they join the group
    - A Proxy with application/network context is aware of a maximum "uncertainty" time interval
    - If Proxy has no information and context, better not to keep Aggregated cache entries

Github issues

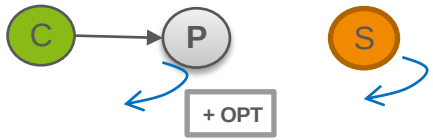
- #19 and #20 – Reverse-proxies
  - More details on the use of Multicast-Signaling option / Response-Forwarding Option
  - More examples using URI templates taken from RFC 8075

- Example of RFC 8075 URI embedding
  - Reverse proxy located at: coaps://myproxy.example.com/
  - Proxy resource located at: /p
  - CoAP group to access: coap://group3.example.net
  - Group resource to access: /light
  - Client's request to Proxy: coaps://myproxy.example.com/p/coap://group3.example.net/light
    or alternative access by: coap://myproxy.example.com/p/coap://group3.example.net/light
    or related to issue #14: https://myproxy.example.com/**hc**/coap://group3.example.net/light

- Questions:
  - Can the Multicast-Signaling Option be received & used by a reverse proxy? [assume: YES]
  - Can the Response-Forwarding Option be used by a reverse proxy? [assume: YES]
  - Can a reverse proxy send back Response-Forwarding Option in responses, even if the client did not use Multicast-Signaling Option in its request? [assume: YES – seems useful ; option is elective anyhow]

Github issues

- #29 – Use CRI [1] in the Response-Forwarding Option
  - Same goal: indicate the server address (and port)
  - Just in a more compact way
- Still using a subset of the ‘tp_info’ array
  - Defined in *draft-ietf-core-observe-multicast-notifications* [2]
  - Change the definition there, then adapt its use here
  - Related to that document’s issue #5, see [3]
- Adapt the “CoAP Transport Information” registry
  - Defined in [2] but used and populated here too
  - Here relevant for ‘tp_id’ (coap over UDP → -1)
  - Will need cross-references to the “Schemes” registry



No.	C	U	N	R	Name	Format	Length	Default
TBD2			-		Response-Forwarding	(*)	9-24	(none)

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Response-Forwarding Option

Current format

```
tp_info = [
      tp_id : 1,              ; UDP as transport protocol
   srv_host : #6.260(bstr),   ; IP address where to reach the server
 ? srv_port : uint / null     ; Port number where to reach the server
]
```

Possible new format

```
tp_info = [
  // in absolute form, with no path or query
  CORI (tp_id, srv_host, ?srv_port)
]
```

[1] https://datatracker.ietf.org/doc/draft-ietf-core-href/
[2] https://datatracker.ietf.org/doc/draft-ietf-core-observe-multicast-notifications/
[3] https://github.com/core-wg/observe-multicast-notifications/issues/5

Github issues

- #23 – “OSCORE between client and proxy” to be moved out

- Convenient for the security association between Client and Proxy, which is required
  - Especially if Group OSCORE is also used between Client and Servers → “OSCORE in OSCORE”
  - Currently defined in Appendix A of this document, at a functional high-level

- Agreed at IETF 110 to take out this part and move it to a separate document
  - It can be specified and analyzed properly
    - E.g., “OSCORE-in-OSCORE” is currently not allowed by RFC 8613
  - It can serve different use cases, other than this particular one

- Ongoing writing of new dedicated draft
  - Appendix A of *groupcomm-proxy* can be removed soon

Github issues (for later on)

- #28 – “Client-Server and Proxy-Server response revalidation using TBD”
  - To be based on what will be defined in *draft-ietf-core-groupcomm-bis*
  - The Client-Server leg can entirely be defined in *groupcomm-bis*
  - The Proxy-Server leg can be in this document, based on the same approach
    - With additional considerations in case e2e security is also used

- #14 “Enable also HTTP-to-CoAP forward proxies”
  - Define HTTP headers for Cross-Proxies
  - Define forwarding of responses back to the client (to be aggregated into a single one)
  - Enable a HTTP client to talk to a CoAP group

Next steps

- Address most of the above points and open issues
- Submit v-04 before the IETF 111 cut-off
- More feedback is welcome

Thank you!

Comments/questions?

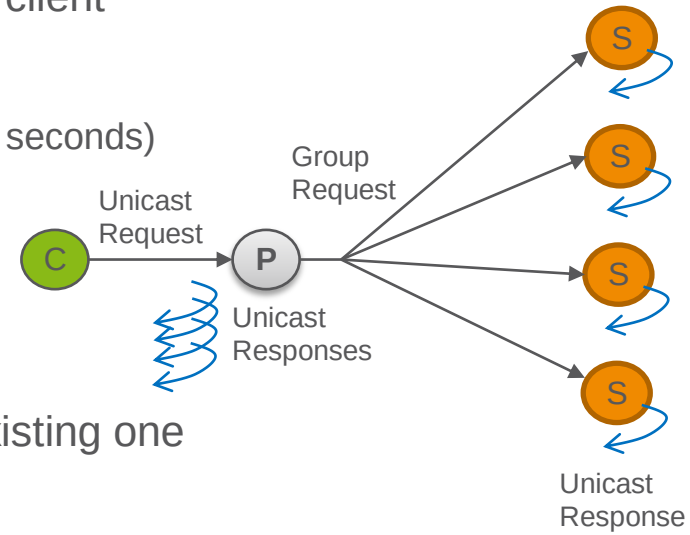
https://gitlab.com/crimson84/draft-tiloca-core-groupcomm-proxy/-/tree/v-04

Backup

Caching model at proxies

› As it receives responses to a group request, the proxy:

1. Forwards each response from the origin server S to the client
2. Adds each response to the individual cache entry for S
   – Same lifetime as Max-Age of the response (or default to 60 seconds)
3. Adds the response to a list L



› After forwarding back all the responses, the proxy:

1. Creates an aggregated cache entry, or cleans up the existing one
2. Copies the responses from the list L to the cache entry
3. Set the cache entry lifetime to the smallest Max-Age of the added responses
4. Set the cache entry as active

Caching model at proxies

› When it receives a response to a unicast request, the proxy:

1. Forwards back the response from the origin server S to the client
2. Creates an Individual cache entry for S, or updates the existing one
   – Same lifetime as Max-Age of the response (or default to 60 seconds)
3. Looks for existing Aggregated cache entries, such that:
   – They would produce a hit, if receiving a group request matching the forwarded unicast request
4. In each found Aggregated cache entry:
   – Store the response, possibly overwriting a currently stored one
   – Set the lifetime of the cache entry to *min*(current entry lifetime, Max-Age of the response)

› Same when the proxy sends requests to the servers, to refresh its cache

C<->P response validation model

Between Client and Proxy

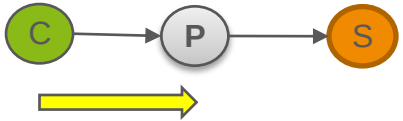
- New Group-Etag Option
  - Only for Aggregated cache entries
  - For group requests and related responses

No.	C	U	N	R	Name	Format	Length	Default
TBD2				x	Group-ETag	opaque	1-8	(none)

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

The Group-ETag Option

- Option value: an entity-tag value, as CBOR byte string
  - Basically, a version number of the Aggregated cache entry (maintained by the proxy)



- A 2.05 (Content) response may include one Group-ETag Option
- In a GET/FETCH group request
  - One option instance per e-tag value to revalidate against the proxy's Aggregated cache entries
- A 2.03 (Valid) response revalidates all responses in the Aggregated cache entry
  - MUST include one Group-Etag Option indicating the revalidated responses set