Recap

› **Observe notifications as multicast responses**
  – Many clients observe the same resource on a server (e.g., pub-sub)
  – Improved performance due to multicast delivery
  – Clients configured by the server, with a 5.03 error informative response

› **Token space managed by the server**
  – The Token space **belongs** to the group (clients)
  – The group **entrusts** the management to the server
  – All clients in a group observation use the same Token value

› **Group OSCORE to protect multicast notifications**
  – The server aligns all clients of an observation on a same `external_aad`
  – All notifications for a resource are protected with that `external_aad`
Main updates since v -02

› Expanded on possible pre- configurations
  › The client may know in advance everything needed about the phantom request
  › E.g., phantom request early published by the server and available to clients (see Appendix A)

› What does it mean on the server side?
  › Before publishing the phantom request and related information ...
  › ... the server must first have started the associated group observation

› How does the client take advantage?
  › If directly able to, it just starts listening to multicast notifications
  › It might send an observation request to the server anyway (preferably with No-Response:16)
    › The server is at least aided in counting the active clients
    › If sent back, the error informative response can omit the phantom request
Main updates since v -02

› Consistent renaming and rephrasing
  › In the error informative response, 'cli_addr' → 'cli_host'
  › When Group OSCORE is used
    › "authentication credential" instead of "public key"
    › In the error informative response, 'pub_key_enc' → 'cred_fmt'

› New Section 2 compiling all prerequisites
  › Large number of clients reachable via multicast (possibly through proxies)
  › Server provisioned with multicast addresses for which it controls Token values
    › Unmanaged multicast addresses (e.g., "All CoAP Nodes") are not good to use
  › Clients and server pre-agree out-of-band on using multicast notifications
    › This makes sense where many individual notification streams are not feasible/preferred
    › No negotiation; no influence on the server's decision to start group observations
Main updates since v-02

› **New Section 3 overviewing the available variants (requested by Göran)**
  › How clients obtain the phantom request
    › From the server, when attempting to register with an individual observation request
    › From other sources; then they just start listening or instruct a proxy to listen to
  › Setup without proxy  ➔  The clients directly listen to multicast notifications
    › Without security
    › With Group OSCORE
  › Setup with proxy  ➔  The proxy listens to multicast notifications to relay back to the clients
    › Without security
    › With Group OSCORE  ➔  The clients instruct the proxy to listen to multicast notifications
      › Easier and more efficient when using OSCORE Deterministic Requests [1]
  › Included pointers to respective content sections and examples

Open point

A phantom request may be an OSCORE Deterministic Request [1]

Proposal – For each of such group observations …
› The server assumes that all the clients support Deterministic Requests
› The server does not care of possible clients without such support
› General pre-requisites include out-of-band agreement
› The variant with Deterministic Requests already builds on a lot of agreed pre-configuration

Practically
› The server does not run the main group observation and a parallel "twin" group observation for clients that do not support OSCORE Deterministic Requests
› The server replies with a generic 5.03 response (i.e., not the error informative response) when receiving traditional, individual observation requests different from the Deterministic Request


Any objection?
To express addressing information, in the error informative response

```plaintext
informative_response_payload = {
    0 => array, ; ‘tp_info’, i.e., transport-specific information
    ? 1 => bstr, ; ‘ph_req’ (transport-independent information)
    ? 2 => bstr ; ‘last_notif’ (transport-independent information)
    ? 3 => uint ; ‘next_not_before’
}
```

**Current approach**

```plaintext
tp_info = [
    srv_addr ; Addressing information of the server
    ? req_info ; Request data extension
]

srv_addr = (
    tp_id ; Identifier of the used transport protocol
    + elements ; Number, format and encoding based on the value of ‘tp_id’
)

req_info = ( ; Number, format and encoding based on
    + elements ; the value of ‘tp_id’ in ‘srv_addr’
)```

TODO: use CRIs (draft-core-ietf-href)
To express addressing information, in the error informative response

```plaintext
informative_response_payload = {
    0 => array, ; ‘tp_info’, i.e., transport-specific information
    ? 1 => bstr, ; ‘ph_req’ (transport-independent information)
    ? 2 => bstr ; ‘last_notif’ (transport-independent information)
    ? 3 => uint ; ‘next_not_before’
}
```

Current approach

```plaintext
tp_info = [
    srv_addr ; Addressing information of the server
    ? req_info ; Request data extension
]
```

```plaintext
srv_addr = (  
    tp_id ; Identifier of the used transport protocol  
    + elements ; Number, format and encoding based on the valu of ‘tp_id’
)
```

```plaintext
req_info = (  
    + elements ; Number, format and encoding based on  
    the valu of ‘tp_id’ in ‘srv_addr’
)
```

New approach – Ongoing PR #13 [2]

```plaintext
tp_info = [
    tpi_server ; Addressing information of the server
    ? tpi_details ; Additional information about the request
]
```

```plaintext
tpi_server = CRI ; From draft-ietf-core-href, with no local part
```

```plaintext
tpi_details = (  
    + elements ; Number, format and encoding based on  
    the URI scheme of ‘tpi_server’
)
```

TODO: use CRIs (draft-core-ietf-href)

To express addressing information, in the error informative response

```plaintext
informative_response_payload = {
    0 => array, ; ‘tp_info’, i.e., transport-specific information
    ? 1 => bstr, ; ‘ph_req’ (transport-independent information)
    ? 2 => bstr ; ‘last_notif’ (transport-independent information)
    ? 3 => uint ; ‘next_not_before’
}
```

Current approach

```plaintext
tp_info = [
    tp_id : 1,                       ; UDP as transport protocol
    srv_host : #6.260(bstr),        ; Src. address of multicast notifications
    srv_port : uint,                ; Src. port of multicast notifications
    token : bstr,                   ; Token of the phantom request and
    cli_host : #6.260(bstr),        ; Dst. address of multicast notifications
    ? cli_port : uint                ; Dst. port of multicast notifications
]
```

New approach – Ongoing PR #13 [2]

```plaintext
tp_info = [
    tpi_srv ; Addressing information of the server,
    ; as a CRI with scheme “coap”
    tpi_details_udp ; Additional information about the request,
    ; when CoAP over UDP is used
]
```

```plaintext
tpi_details_udp = (;
    tpi_token : bstr, ; Token of the phantom request and
    ; associated multicast notifications
    tpi_client : CRI ; Destination of multicast notifications,
    ; as a CRI with scheme “coap”
)
```


Full example with CoAP over UDP

[13]
Next steps

- Address the open point on not having "twin" group observations

- Switch to using CRIs for encoding addressing information – see draft-ietf-core-href

- Discuss how the counting of clients is affected in the different setups
  - E.g., what makes it more or less accurate/reliable

- Suggestions from IANA
  - New registry "Informative Response Parameters" as sub-registry under the "Constrained RESTful Environments (CoRE) Parameters" registry.
  - New "CoAP Transport Information" registry would better have "Standards Action With Expert Review“, same as some of the COSE registries.

- Add discussed examples with a reverse-proxy

- Need for reviews – Previously promised: Göran, Esko, Jaime, Carsten, Thomas
Thank you!

Comments/questions?

https://github.com/core-wg/observe-multicast-notifications
Backup
Phantom request and error response

The server requests the observation on its own, e.g. when:
1. A first traditional registration request comes from a first client; or
2. Some threshold is crossed – clients can be shifted to a group observation

Consensus on Token & external_aad, by using a phantom observation request
- Generated inside the server, it does not hit the wire
- Like if sent by the group, from the multicast IP address of the group
- Multicast notifications are responses to this phantom request

The server sends to clients a 5.03 error informative response with:
- Transport-specific information, e.g., the IP multicast address where notifications are sent to
- The serialization of the phantom observation request (optional)
- The serialization of the latest multicast notification (optional)
- Minimum amount of time after which the next multicast notification will be sent (optional)
Server side

1. Build a GET phantom request; Observe option set to 0

2. Choose a value T, from the Token space for messages …
   – … coming from the multicast IP address and addressed to the target resource

3. Process the phantom request
   – As coming from the group and its IP multicast address
   – As addressed to the target resource

4. Hereafter, use T as token value for the group observation

5. Store the phantom request, store (not send) reply as ‘last_notif’
Interaction with clients

- The server sends to new/shifted clients an **error informative response** with
  - ‘tp_info’: transport-specific information
    - ‘srv_host’ and ‘srv_port’: destination address of the phantom request
    - ‘token’: the selected Token value T, used for ‘ph_req’ and ‘last_notif’
    - ‘cli_host’ and ‘cli_port’: source address of the phantom request
  - ‘ph_req’: serialization of the phantom request
  - ‘last_notif’: serialization of the latest sent notification for the target resource
  - ‘next_not_before’: minimum amount of time after which the next multicast notification will be sent

- When the value of the target resource changes:
  - The server sends an Observe notification to the multicast IP address ‘cli_host’ : ‘cli_port’
  - The notification has the Token value T of the phantom request

- When getting the error informative response, a client:
  - Configures an observation for an endpoint associated to the multicast IP address
  - Accepts observe notifications with Token value T, sent to that multicast IP address
C1 registration

C_1 --------------- [ Unicast ] --------------- \rightarrow S /r

GET
Token: 0x4a
Observe: 0 (Register)
<Other options>

(S allocates the available Token value 0x7b .)

(S sends to itself a phantom observation request PH_REQ as coming from the IP multicast address GRP_ADDR .)

/ \-----------------------------------------> /r

GET
Token: 0x7b
Observe: 0 (Register)
<Other options>

(S creates a group observation of /r .)

(S increments the observer counter for the group observation of /r .)
C1 registration

C_1 <---------------------- [ Unicast ] ---------------------- S

  5.03
  Token: 0x4a
  Content-Format: application/informative-response+cbor
  Max-Age: 0
  <Other options>
  Payload: {
    tp_info     : [1, bstr(SRV_ADDR), SRV_PORT,
                   0x7b, bstr(GRP_ADDR), GRP_PORT],
    last_notif  : bstr(0x45 | OPT | 0xff | PAYLOAD)
  }

C2 registration

GET
Token: 0x01
Observe: 0 (Register)
<Other options>

(S increments the observer counter for the group observation of /r.)

5.03
Token: 0x01
Content-Format: application/informative-response+cbor
Max-Age: 0
<Other options>
Payload:

```json
{  
  tp_info : [1, bstr(SRV_ADDR), SRV_PORT, 0x7b, bstr(GRP_ADDR), GRP_PORT],
  lastnotif : bstr(0x45 | OPT | 0xff | PAYLOAD)
}
```
Multicast notification

(The value of the resource /r changes to "5678").

C_1
+  <--------------- [ Multicast ] ---------------
C_2
   (Destination address/port: GRP_ADDR/GRP_PORT)

2.05  Token: 0x7b
     Observe: 11
     Content-Format: application/cbor
     <Other options>
     Payload: : "5678"

› Same Token value of the Phantom Request

› Enforce binding between
   - Every multicast notification for the target resource
   - The (group) observation that each client takes part in
Security with Group OSCORE

- The phantom request is protected with Group OSCORE
  - \( x \): the Sender ID (‘kid’) of the Server in the OSCORE group
  - \( y \): the current SN value (‘piv’) used by the Server in the OSCORE group
  - \( z \): the Group ID (‘kid_context’) used in the OSCORE group
  - Note: the Server consumes the value \( y \) and does not reuse it as SN in the group

- To secure/verify all multicast notifications, the OSCORE \textit{external_aad} is built with:
  - ‘request_kid’ = \( x \)
  - ‘request_piv’ = \( y \)
  - ‘request_kid_context’ = \( z \)

- The phantom request is still included in the informative response
  - Each client retrieves \( x \), \( y \) and \( z \) from the OSCORE Option value
In the error response, the server can optionally specify also:

- ‘join_uri’ : link to the Group Manager to join the OSCORE group
- ‘sec_gp’ : name of the OSCORE group
- ‘as_uri’ : link to the ACE Authorization Server associated to the Group Manager
- ‘hkdf’ : HKDF algorithm
- ‘cred_fmt’ : format used in the OSCORE group for the authentication credentials
- ‘sign_enc_alg’ : AEAD algorithm
- ‘sign_alg’ : signature algorithm
- ‘sign_params’ : parameters of the signature algorithm and signing key
- ‘sign_alg_capab’ : COSE capabilities of the ‘sign_alg’ algorithm
- ‘sign_key_type_capab’ : COSE capabilities of the keys used by ‘sign_alg’
The server protects the Phantom Request with Group OSCORE, using its Sender Context, as if it was the sender.
C1 registration w/ security

C_1 <------------ [ Unicast w/ OSCORE ] ---------- | S

2.05 (Content)
Token: 0x4a
OSCORE: { piv: 301; ...}
Max-Age: 0
<Other class U/I options>
0xff

Encrypted_payload {
  5.03 (Service Unavailable),
  Content-Format: application/informative-response+cbor,
  <Other class E options>,
  0xff,
  CBOR_payload {
    tp_info : [1, bstr(SRV_ADDR), SRV_PORT,
    0x7f, bstr(GRP_ADDR), [GRP_PORT],
    ph_req : bstr(0x05 OPT 0xff PAYLOAD SIGN),
    last_notif : bstr(0x45 OPT 0xff PAYLOAD SIGN),
    join_url : "coap://myGM/ace-group/myGroup",
    sec_gp : "myGroup"
  }
}
C2 registration w/ security

C_2: [Unicast w/ OSCORE] [Unicast w/ OSCORE] -> S/r

0.05 (FETCH)
Token: 0x01
OSCORE: {kid: 0x02; piv: 201; ...}
<Other class U/I options>
0xff
Encrypted_payload {
0x01 (GET),
    Observe: 0 (Register),
    <Other class E options>
}

(S increments the observer counter for the group observation of /r.)

C_2: [Unicast w/ OSCORE] [Unicast w/ OSCORE] -> S

2.05 (Content)
Token: 0x01
OSCORE: {piv: 401; ...}
Max-Age: 0
<Other class U/I options>
0xff,
Encrypted_payload {
5.03 (Service Unavailable),
<Other class E options>,
0xff,
CBOR_payload {
    tp_info : [1, bstr(SRV_ADDR), SRV_PORT,
    ph_req : [bstr(0x05)],
    last_notif : bstr(0x45),
    join_uri : "coap://myGM/ace-group/myGroup",
    sec_gp : "myGroup"
}}
Multicast notification w/ security

- When encrypting and signing the multicast notification:
  - The `external_aad` has `request_kid = 0x05`, `request_iv = 501` and `request_kid_context = 0x57ab2e`
  - Same for all following notifications for the same resource

- Enforce secure binding between
  - Every multicast notification for the target resource
  - The (group) observation that each client takes part in
Support for intermediary proxies

› How it works
  – The proxy (next to the server) directly listens to the IP multicast address
  – The original Token of the phantom request has to match at the proxy
  – The proxy forwards multicast notifications back to each client
    › The proxy uses the Token values offered by the clients

› Without end-to-end security (Section 11)
  – The proxy can retrieve the phantom request from the informative response
  – No need to forward the informative response back to the clients

› With end-to-end security (Section 12)
  – The informative response is also protected with OSCORE or Group OSCORE
  – The proxy **cannot** retrieve the phantom request from the informative response
  – Each client has to explicitly provide the phantom request to the proxy
  – Exception: the phantom request is a Deterministic Request (see *core-cachable-oscore*)