Aligning DAP with RFC 9205/BCP 56

Tim Geoghegan - PPM interim March 11 2025

Major DAP changes since October 2024 interim

- Catch up with most recent VDAF drafts
- Specify asynchronous aggregation job handling
- Why?
  - Experience in production shows that aggregation job handling can take minutes
    - Large measurements in Prio3SumVec or Prio3Histogram
    - Many aggregation jobs contend for a single database row
- Align collection job handling with new aggregation job semantics

RFC 9205/BCP 56: Building Protocols with HTTP

- WGLC is soon, so it's time to get picky and split hairs
- 4.1: Provide examples of requests and responses
- 3.1 and 4.6: use broadest possible status codes (200, 400, 500)
- 4.3: Refer to RFC 9110 for HTTPS
- PR(s) to draft-ietf-ppm-dap are coming!

```
GET /thing HTTP/1.1
Host: example.com
Accept:
application/things+json
User-Agent: Foo/1.0
```

```
HTTP/1.1 200 OK
Content-Type:
application/things+json
Content-Length: 500
Server: Bar/2.2

[content here]
```

Polishing DAP's semantics for long running operations

- DAP has several interactions that look like this
  - Aggregation job init
  - Aggregation job continue
  - Collection job
  - Aggregate share request?
- Goals:
  - Define a unified semantic for potentially long-running operations
  - Align it with RFC 9205/BCP 56
  - Use it consistently across DAP interactions
- Non-goals:
  - Relitigate whether async aggregation jobs should exist
  - Or whether to use HTTP
  - Change how DAP actually works

What's in draft-ietf-ppm-dap-14

Leader creates an aggregation job:

```
PUT /tasks/{task-id}/aggregation_jobs/{aggregation-job-id}
Host: example.com/helper
Content-Type: application/dap-aggregation-job-init-req

encoded(struct {...} AggregationJobInitReq)
```

What's in draft-ietf-ppm-dap-14

Helper responds synchronously:

```
HTTP/1.1 201 Created
Content-Type: application/dap-aggregation-job-resp

encoded(struct {
      status = ready,
      prepare_resps,
} AggregationJobResp)
```

OR asynchronously, and the Leader then polls:

```
HTTP/1.1 201 Created
Content-Type: application/dap-aggregation-job-resp
Location: /tasks/{task-id}/aggregation_jobs/{aggregation-job-id}?step=0
Retry-After: 300s

encoded(struct {
      status = processing,
} AggregationJobResp)
```

What's in draft-ietf-ppm-dap-14

Helper responds synchronously:

```
HTTP/1.1 201 Created
Content-Type: application/dap-aggregation-job-resp

encoded(struct {
      status = ready,
      prepare_resps,
} AggregationJobResp)
```

OR asynchronously, and the Leader then polls:

```
HTTP/1.1 201 Created
Content-Type: application/dap-aggregation-job-resp
Location: /tasks/{task-id}/aggregation_jobs/{aggregation-job-id}?step=0
Retry-After: 300s

encoded(struct {
      status = processing,
} AggregationJobResp)
```

Observation 1:
RFC 9205 section 4.6:
"[make] generous use of the general status codes (200, 400, and 500) when in doubt"

Observation 2:
Use an empty response body instead of defining struct variants

draft-ietf-ppm-dap-14 collection jobs

Similar to aggregation jobs:

```
PUT /tasks/{task-id}/collection_jobs/{collection-job-id}
Host: example.com/leader
Content-Type: application/dap-collection-job-req

encoded(struct {...} CollectionJobReq)

HTTP/1.1 201 Created
Content-Type: application/dap-collection-job-resp
Retry-After: 300s

encoded(struct {
    status = processing,
} CollectionJobResp)
```

draft-ietf-ppm-dap-14 collection jobs

Similar to aggregation jobs:

```
PUT /tasks/{task-id}/collection_jobs/{collection-job-id}
Host: example.com/leader
Content-Type: application/dap-collection-job-req

encoded(struct {...} CollectionJobReq)

HTTP/1.1 201 Created
~~Content-Type: application/dap-collection-job-resp~~
Retry-After: 300s

~~encoded(struct {~~
~~    status = processing,~~
~~} CollectionJobResp)~~
```

draft-ietf-ppm-dap-14 collection jobs

Similar to aggregation jobs:

```
PUT /tasks/{task-id}/collection_jobs/{collection-job-id}
Host: example.com/leader
Content-Type: application/dap-collection-job-req

encoded(struct {...} CollectionJobReq)

HTTP/1.1 201 Created
~~Content-Type: application/dap-collection-job-resp~~
Retry-After: 300s

~~encoded(struct {~~
~~    status = processing,~~
~~} CollectionJobResp)~~
```

Observation 3: Synchronous processing not allowed?

draft-ietf-ppm-dap-14 aggregate share requests

```
POST /tasks/{task-id}/aggregate_shares
Host: example.com/helper
Content-Type: application/dap-aggregate-share-req

encoded(struct {...} AggregateShareReq)

HTTP/1.1 200 OK
Content-Type: application/dap-aggregate-share

encoded(struct {...} AggregateShare)
```

draft-ietf-ppm-dap-14 aggregate share requests

```
POST /tasks/{task-id}/aggregate_shares/missing-identifier
Host: example.com/helper
Content-Type: application/dap-aggregate-share-req

encoded(struct {...} AggregateShareReq)

HTTP/1.1 200 OK
Content-Type: application/dap-aggregate-share

encoded(struct {...} AggregateShare)
```

Observation 3:
*A*synchronous
processing not allowed?

Observations

1. Converge on HTTP 200 everywhere
2. Use empty response bodies instead of defining bespoke body format that means empty body
3. Consistently allow async or sync handling of operations

A consistent semantic for long-running operations

- Lean into HTTP conventions for Create, Read, Update, Delete
- Resources uniquely identified by paths
- Create: PUT /tasks/{task-id}/resource_type/{resource-unique-id}
  - If synchronous: response body is the resource and Content-Type is set
  - If asynchronous: empty response body, no Content-Type, Retry-After SHOULD be set
- Read: GET /tasks/{task-id}/resource_type/{resource-unique-id}
  - If ready: response body is the resource and Content-Type is set
  - If not ready: empty response body, no Content-Type, Retry-After SHOULD be set
- Update: POST /tasks/{task-id}/resource_type/{resource-unique-id}
  - For aggregation job continuation. Other resources don't use it.
- Delete: DELETE /tasks/{task-id}/resource_type/{resource-unique-id}

What is to be done?

- Aggregation job interaction:
  - Tweak HTTP status codes
  - Change representation of processing jobs
  - No changes to how it actually works
- Aggregate share interaction:
  - Add asynchronous mode of operation
  - Requires defining a unique identifier for aggregate share that leader can poll on
  - Let the leader choose it (Collection job ID is natural choice)
- Collection job interaction:
  - Add synchronous mode of operation
  - No changes to how it actually works

Unpredictable task IDs

- draft-ietf-ppm-dap section 4.3: "The task ID value MUST be a globally unique sequence of bytes"
- taskprov derives task ID from `struct TaskConfig`
- Attacker can enumerate `TaskConfig` values to discover what tasks an aggregator currently supports
- Risk to aggregator security but NOT to client privacy or robustness
- Further discussion:
  - draft-ietf-ppm-dap/#664
  - draft-ietf-ppm-dap-taskprov/#102

Unpredictable task IDs

- Mitigation: require authentication and restrict which tasks are visible to which clients
  - No protocol action needed
- What should DAP do?
  - Current consensus: add brief security consideration
- What should taskprov do?
  - Security consideration?
  - Require mixing some entropy into `TaskConfig`?
    - Can SHA-256 work for this?