DAP updates and open issues

IETF 120 – PPM
Changes since IETF 118

- draft-ietf-ppm-dap-09: *fixed_size* queries: Make max batch size optional
- draft-ietf-ppm-dap-10: Incorporate feedback from early HTTPDIR review (Mark Nottingham)
- draft-ietf-ppm-dap-11: Result of [consensus call](#) following [discussion at the interim](#): remove support for multi-collection. DAP no longer supports heavy hitters via Poplar1.

- Closed the drill down issue ([#490](#)) without action. Options for Prio3:
  - Specify variant with [plaintext labels](#)
  - Use [Mastic](#) instead (under development)
**PR #563**: Remove `max_batch_size` task parameter

- **Context**: The `fixed_size` query type is so named because it has both a minimum and maximum batch size. The maximum batch size turned out to be slightly annoying to implement, as well as not particularly useful. In fact, we made it optional in [draft-ietf-ppm-dap-09](https://datatracker.ietf.org/doc/draft-ietf-ppm-dap-09/).

- **Proposal**: Remove the maximum batch size. Related nits:
  - Rename `QueryType` to `BatchMode`
    - Aligns better with intended semantics: it refers *both* to how reports may be partitioned during aggregation *and* to how they're addressed during collection
  - Rename the query type to `leader_selected`
  - Security considerations: Describe Sybil-like attack in which the Leader attempts to partition reports by Client
PR #564: Make aggregation asynchronous

- Context: Aggregation is "synchronous": each HTTP request blocks until Helper completes VDAF preparation and commits states changes (record report IDs for anti-replay, update aggregate share).
  - Deployment experience: a small number of requests take 10+ seconds to resolve or need to be aborted and retried (resource contention).
PR #564: Make aggregation asynchronous

- **Context:** Aggregation is "synchronous": each HTTP request blocks until Helper completes VDAF preparation and commits states changes (record report IDs for anti-replay, update aggregate share).
  - Deployment experience: a small number of requests take 10+ seconds to resolve or need to be aborted and retried (resource contention).

- **Proposal:** Allow Helper to process job asynchronously
  - Helper responds to PUT with 201 Created
  - Leader sends GET to poll for result
Context: DAP implicitly overloads RFC 8446 (TLS 1.3) presentation language. The following is for fields with fixed constants (Section 3.7). We use it for specifying how a message is formatted ("the Helper replies with the following message")

```c
struct {
    PrepareStepState prepare_step_state = 2; /* reject */
    ReportId report_id;
    ReportShareError report_share_error;
} PrepareStep;
```

Proposal: Make this syntax explicit with new `struct variant` notation.

```c
struct variant {
    PrepareStepState prepare_step_state = 2; /* reject */
    ReportId report_id;
    ReportShareError report_share_error;
} PrepareStep;
```
PR #567: Drop task_id param from HPKE config endpoint

- Context: DAP Aggregators may have per-task HPKE configurations to reduce risk of key compromise. To support this, Clients indicate the task ID when requesting the configs.
  - This might complicate applications that need anonymity (e.g., DAP over OHTTP): Aggregators know which tasks a Client participates in.
  - This feature is not used in any known implementation.

- Proposal: Remove the optional task_id parameter from the /hpke_config endpoint, forcing an Aggregator endpoint to use the same set of HPKE configs for all tasks.
  - Can still do key separation across endpoints
PR #568: Content type versioning

- Context: the content type for an HTTP request (e.g., "application/dap-aggregation-job-init-req") is meant to convey how the request body is parsed. However:
  - "DAP v2" will likely be wire-incompatible with the current protocol
  - Drafts of the current protocol are often wire-incompatible

- Proposal: Clarify that major revisions to DAP will use new media types (e.g., "dap" → "dap2"). Also, note that media types have an optional parameter that can be used to convey the draft number (may be useful for debugging).
PR #569: Allow aggregation job skew recovery at any step

- **Context:**
  - Multi-round VDAFs:
    - 1-round VDAFs (Prio3) complete after initialization (the `AggregationJobInitReq` and corresponding `AggregationJobResp`)
    - 2-round VDAFs (Poplar1) require initialization and one round of continuation (`AggregationJobContinueReq`)
    - 3-round VDAFs require one continuation, 4-round VDAFs require two rounds of continuation, ...
  - If the Leader sends the request for the previous continuation step (e.g., due to state rollback), then the Helper MAY reply with the corresponding `AggregationJobResp` to help the Leader recover. We call this **step skew recovery**.

- **Proposal:** Allow the Helper to recover from any continuation step.
Life of an aggregation job

(2-round VDAF)
1. chose candidate reports
2. choose agg-job-id
3. decrypt report shares and do early report checks (Section 4.5.1.4)
4. update candidate reports
5. `vdaf.prep_init()` // round 1
6. Produce AggregationJobInitReq

At this point in time, what happens if …
1. chose candidate reports
2. choose agg-job-id
3. decrypt report shares and do early report checks (Section 4.5.1.4)
4. update candidate reports
5. `vdaf.prep_init()` // round 1
6. Produce AggregationJobInitReq

... the Leader PUTs the same agg job?

7. Consume AggregationJobInitReq

Ok: idempotency
1. chose candidate reports
2. choose agg-job-id
3. decrypt report shares and do early report checks (Section 4.5.1.4)
4. update candidate reports
5. `vdaf.prep_init()` // round 1
6. Produce AggregationJobInitReq

```
PUT /tasks/{task-id}/aggregation_jobs/{agg-job-id}
```

7. Consume AggregationJobInitReq

```
201 Created
```

T1

```
DELETE /tasks/{task-id}/aggregation_jobs/{agg-job-id}
```

```
201 Created
```

`ok: job abandoned`

... the Leader DELETEs the agg job?
1. chose candidate reports
2. choose agg-job-id
3. decrypt report shares and do early report checks (Section 4.5.1.4)
4. update candidate reports
5. `vdaf.prep_init()` // round 1
6. Produce AggregationJobInitReq

---

... the Leader PUTs the wrong data?

---

```
PUT /tasks/{task-id}/aggregation_jobs/{agg-job-id}

AggregationJobInitReq

201 Created

T1

PUT /tasks/{task-id}/aggregation_jobs/{agg-job-id}

AggregationJobInitReq2

400 Bad Request
```

7. Consume AggregationJobInitReq

```
abort: it's illegal to change the job
```
1. chose candidate reports
2. choose agg-job-id
3. decrypt report shares and do early report checks (Section 4.5.1.4)
4. update candidate reports
5. `vdaf.prep_init()` // round 1
6. Produce AggregationJobInitReq

```
PUT /tasks/{task-id}/aggregation_jobs/{agg-job-id}
```

AggregationJobInitReq

```
201 Created
```

```
T1
```

```
POST /tasks/{task-id}/aggregation_jobs/{agg-job-id}
```

```
400 Bad Request
```

7. Consume AggregationJobInitReq

... the Leader POSTs?

**abort**: the job is not POSTable yet (we're still on the first step)
At this point in time, what happens if …

7. Consume AggregationJobInitReq

8. decrypt report shares and do early report checks (Section 4.5.1.4)
9. `vdaf.prep_init()` // round 1
10. `vdaf.prep_shares_to_prep()`
11. `vdaf.prep_next()` // round 2
12. Produce AggregationJobResp
... the Leader GETs the agg job?

GET /tasks/{task-id}/aggregation_jobs/{agg-job-id}

201 Created

T1

GET /tasks/{task-id}/aggregation_jobs/{agg-job-id}

202 Accepted

ok: response not ready

7. Consume AggregationJobInitReq

8. decrypt report shares and do early report checks (Section 4.5.1.4)

9. vdaf.prep_init() // round 1

10. vdaf.prep_shares_to_prep()

11. vdaf.prep_next() // round 2

12. Produce AggregationJobResp

GET /tasks/{task-id}/aggregation_jobs/{agg-job-id}

200 OK

AggregationJobResp

ok: response ready
… the Leader GETs the agg job?

GET /tasks/{task-id}/aggregation_jobs/{agg-job-id}

201 Created

7. Consume AggregationJobInitReq

200 OK

AggregationJobResp

ok: response ready

8. decrypt report shares and do early report checks (Section 4.5.1.4)
9. vdaf.prep_init() // round 1
10. vdaf.prep_shares_to_prep()
11. vdaf.prep_next() // round 2
12. Produce AggregationJobResp
At this point, what happens if …

13. Consume AggregationJobResp
14. `vdaf.prep_next()` // round 2
15. `vdaf.prep_shares_to_prep()`
16. `vdaf.prep_next()` // finish
17. update candidate reports
18. Produce
   AggregationJobContinueReq

Leader

200 OK

Helper

AggregationJobResp
13. Consume AggregationJobResp
14. `vdaf.prep_next()` // round 2
15. `vdaf.prep_shares_to_prep()`
16. `vdaf.prep_next()` // finish
17. update candidate reports
18. Produce
   AggregationJobContinueReq

... the Leader goes back a step?

200 OK
AggregationJobResp

T2
PUT /tasks/{task-id}/aggregation_jobs/{agg-job-id}
AggregationJobInitReq

201 Created

ok: idempotency
... the Leader advances to the next step?

13. Consume AggregationJobResp
14. vdaf.prep_next() // round 2
15. vdaf.prep_shares_to_prep()
16. vdaf.prep_next() // finish
17. update candidate reports
18. Produce
    AggregationJobContinueReq
13. Consume AggregationJobResp
14. \texttt{vdaf.prep\_next()} // round 2
15. \texttt{vdaf.prep\_shares\_to\_prep()}
16. \texttt{vdaf.prep\_next()} // finish
17. update candidate reports
18. Produce
AggregationJobContinueReq

... the Leader
advances a \texttt{step}
too far?

Leader

Helper

200 OK
AggregationJobResp

POST /tasks/{task-id}/aggregation_jobs/{agg-job-id}

AggregationJobContinueReq(step=2)

T2

19. Consume
AggregationJobContinueReq

400 Bad Request

abort: the next step is 1.
… the Leader GETs the agg job?

GET /tasks/{task-id}/aggregation_jobs/{agg-job-id}
202 Accepted

T3

GET /tasks/{task-id}/aggregation_jobs/{agg-job-id}
202 Accepted

ok: response not ready

vdaf.prep_next() // finish

commit aggregates (resolve replayed reports and batch overlaps and merge agg share)

Produce AggregationJobResp

ok: response ready
... the Leader POSTs the **wrong** data?

1. Leader
   - POST `/tasks/{task-id}/aggregation_jobs/{agg-job-id}`
   - 202 Accepted

2. Helper
   - AggregationJobContinueReq
   - 202 Accepted

3. Leader
   - T3

4. Helper
   - 19. Consume `AggregationJobContinueReq2`
   - `abort: it's illegal to change the job`
... the Leader GETs the agg job?

GET /tasks/{task-id}/aggregation_jobs/{agg-job-id}

T3

200 OK
AggregationJobResp

19. Consume AggregationJobContinueReq
20. vdaf.prep_next() // finish
21. commit aggregates (resolve replayed reports and batch overlaps and merge agg share)
22. Produce AggregationJobResp

ok: response ready
23. Consume AggregationJobResp
24. update candidate reports
25. **commit aggregates** (resolve replayed reports and batch overlaps and merge agg share)
PR #564: Make aggregation asynchronous

Ambiguous GET requests - which step is the Leader requesting?

Leader sends AggregationJobContinueReq for step n+1

Leader sends AggregationJobContinueReq for step n (allowed by step skew recovery)