ALTO Incremental Updates

draft-alto-incr-update-sse-02

W. Roome
Alcatel-Lucent/Bell Labs (NJ)
R. Yang
X. Shi
Yale

IETF 92

March 26, 2015

Motivation

- Maps can be very large
- Small subsets can change frequently
- Clients want timely & efficient updates:
  - Get updates as soon as possible (e.g., no polling delay)
  - Only get the changes

Overview

- General framework for updates to any ALTO resource
  - Including Endpoint Cost & Property Services
- Server defines one or more Update Stream resources
  - Continuous stream of update messages for a resource set
  - Messages are Server-Sent Events (SSEs)
  - Each SSE updates one resource
- Update data format:
  - Full replacement: ALTO data format
  - Incremental update: JSON Merge-Patch (RFC 7386)
  - Server decides which to use

Changes Since -v00

- Dropped GET-mode “Full” Update Stream Service; only define the POST-mode “Filtered” Update Stream Service
- Draft -v02 is a complete specification, with full examples
- Changed format of IRD entry
- Changed format of client’s POST-mode input

Server-Sent Events (SSE)

- Just like HTTP, except content is stream of events
  - W3C Standard
  - Events are separated by blank lines
  - Each event has a type and data:

```
HTTP server->client headers

event: first-type
data: content of first event

event: another-type
data: here is a second event with
data: longer content than the first
```

- Simple & robust:
  - Firewalls & NAT boxes: No problem!
  - HTTP proxy: Must pass data as server sends it, and must keep stream open for a long time

Our Update Events

- Each SSE updates one ALTO resource
- Data is JSON message with the update
- Type has ID of ALTO resource & media-type of data field
  - Complete replacements use ALTO media-types
  - Incremental updates use JSON Merge-Patch encoding (next slide)

HTTP server->client headers

```
event: my-routingcost-map,application/alto-costmap+json
data: { full cost-map message }

event: my-routingcost-map,application/merge-patch+json
data: { JSON Merge-Patch with changed costs }

event: my-routingcost-map,application/merge-patch+json
data: { JSON Merge-Patch with changed costs }
```

JSON Merge-Patch

- Standard (RFC 7386) for JSON incremental updates
  - Very efficient for data expressed as nested JSON objects
- Update algorithm:
  - Do depth-first walk of objects in merge-patch message, keeping path to current value. When encountering a value that is not a JSON object dictionary (e.g., string, number, array, etc), replace the previous value for that path with the new value. If there was no value for that path, create one. If the new value is null, delete the previous value for that path.
- Example:
  - Change (or add) the costs from PID1=>PID2 to 9, PID3=>PID3 to 1, and delete the cost from PID3=>PID1:

```
{ "cost-map": "PID1": { "PID2": 9 },
              "PID3": { "PID1": null, "PID3": 1 } }
```

  - Perfect for Cost Map changes
  - Good, but not optimal, for Network Map changes

Update Stream Service

- POST-mode
- Updates set of resources selected by server
- Media-type is text/event-stream (registered for SSE)
- Accepts application/alto−updatestreamparams+json (new)
- Server sends a stream of SSE update events
- Client's input:
  - Resource IDs for which client wants updates
  - Vtags of any resources client has already retrieved
  - Input parameters required for any POST-mode resources
  - Whether client wants incremental updates

IRD Example

```
"my-costs-update-stream": {
  "uri": "http://alto.example.com/updates/costs",
  "media-type": "text/event-stream",
  "accepts": "application/alto-updatestreamparams+json",
  "uses": ["my-network-map",
           "my-routingcost-map", "my-hopcount-map"],
  "capabilities": {
    "incremental-update-media-types": {
      "my-routingcost-map": "application/merge-patch+json",
      "my-hopcount-map": "application/merge-patch+json"
    }
  }
}
```

This stream provides updates for a Network Map and two Cost Maps (routingcost & hopcount). It offers incremental updates for the Cost Maps, but not for the Network Map.

Update Stream Semantics

- A server MAY offer multiple Update Stream resources
  - Server chooses the resources for each stream
- The updates to a dependent resource and its “parent” resource SHOULD be available via the same stream
  - Thus a client can get updates for a Cost Map and its Network Map and throught the same stream
- If a stream offers updates to a “parent” resource and a dependent resource, the server MUST send update events for the parent resource before sending updates for the dependent resource
  - E.g., the server MUST send Network Map update events before Cost Map updates

Update Stream Semantics (Part 2)

- If several streams offer updates to the same resource, the updates MUST eventually converge to the same data
  - But the server MAY group changes into different merge-patch update events in different streams
  - E.g., suppose three cost points change within a short interval. On one stream, server may send three SSEs, with one cost each. On another stream, the server may send one SSE with all three costs.
- At start-up, server MUST send a full replacement for every resource, unless client has current version
  - Exception only applies to vtag'd resources (e.g., Network Maps)
  - Server must send full Cost Maps and Post-mode resources

Update Stream Example 1: Client

```
POST /updates/costmaps HTTP/1.1
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

{ "my-network-map": {
    "tag": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
  },
  "my-routingcost-map": {}
}
```

The client just wants routingcost updates, not hopcount updates. The client already has the Network Map, and provides its tag.

Update Stream Example 1: Server

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/event-stream

event: my-routingcost-map,application/alto-costmap+json
data: { ... full routingcost Cost Map message ... }

    (pause; then a cost changes)

event: my-routingcost-map,application/merge-patch+json
data: {"cost-map": {"PID2" : {"PID3" : 31}}}

    (pause; then Network Map changes)

event: my-network-map,application/alto-networkmap+json
data: { ... full Network Map message ... }

event: my-routingcost-map,application/alto-costmap+json
data: { ... full Cost Map message ... }
```

Update Stream Example 2: Client

```
POST /updates/properties HTTP/1.1
Host: alto.example.com
Accept: text/event-stream
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

{ "my-properties": {
     "input": {
       "properties" :
           [ "priv:ietf-bandwidth" ],
       "endpoints" :
           [ "ipv4:1.0.0.1", "ipv4:1.0.0.2", "ipv4:1.0.0.3" ]
     }
  }
}
```

Update Stream Example 2: Server

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/event-stream

event: my-properties,application/alto-endpointprops+json
data: { "endpoint-properties": {
data:     "ipv4:1.0.0.1" : { "priv:ietf-bandwidth": "13" },
data:     "ipv4:1.0.0.2" : { "priv:ietf-bandwidth": "42" },
data:     "ipv4:1.0.0.3" : { "priv:ietf-bandwidth": "27" } } }

    (pause)

event: my-properties,application/merge-patch+json
data: { "endpoint-properties": {
data:     "ipv4:1.0.0.1" : { "priv:ietf-bandwidth": "3" } } }

    (pause)

event: my-properties,application/merge-patch+json
data: { "endpoint-properties": {
data:     "ipv4:1.0.0.3" : { "priv:ietf-bandwidth": "38" } } }
```

Controversial Design Decision!

- At startup, the server MUST send full maps for untagged resources
  - Cost Maps are not tagged, so server always sends full Cost Maps
- If a stream drops, when the client reconnects, the server must resend data the client already has
  - *Ugly! Inefficient!!*
  - Could add vtags to Cost Maps, but that complicates ALTO
- But only a problem for large maps **and** frequent drops
  - Clients who need large maps rarely drop connections
  - Clients who do drop connections don't need large maps
- So keep it simple, and accept the occasional inefficiency

Next Steps

Approve the key points of this approach:

1. Transport: Server-Sent Events (SSE)
2. Message format: JSON Merge-Patch & Full ALTO messages
   - Allows other incremental update formats
3. Server sends updates when they are available
4. Server may offer multiple Update Stream resources
5. Each stream updates one or more resources
6. Streams may update any resource
7. At startup, server sends full maps for untagged resources
8. Update Streams are NOT required

Assuming “yes,” adopt as working group document

- Review via mailing list
- WGLC by summer??? (Charter target was Nov. 2014)

Thank you