ALTO Incremental Updates

draft-alto-incr-update-sse-00

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

IETF 91

November 13, 2014

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 of Approach

- Design a general framework for continuous & incremental updates to any ALTO resource
  - ALTO Server defines an Update Service for one or more resources
  - Client establishes an HTTP stream to an Update Service
  - Server sends updates as they become available
  - Updates are Server-Sent Events (SSEs)
  - Each SSE updates one resource
  - Updates can be complete replacement, or just changes
- This replaces previous “client-pull/polling” proposal, draft-roome-alto-incr-update-00

Server-Sent Events (SSE)

- Just like HTTP, except content is stream of events
  - W3C Standard
  - New content type, not a new protocol (IMHO)
  - 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 the media-type of the JSON message, and the ID of an ALTO resource
  - Complete replacements use ALTO media-types
  - Incremental updates use JSON Merge-Patch encoding (next slide)

```
HTTP server->client headers

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

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

event: application/merge-patch+json,my-routingcost-map
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
  - Okay, but not optimal, for Network Map changes

ALTO Update Stream Service

- GET-mode
- Media-type is text/event-stream (registered for SSE)
- URI gives a stream of SSE update events for a set of resources
- “event” capability gives the IDs of the resources for which this stream provides updates, and the types of update messages for each resource
  - Server selects the resources and update types
  - Server MUST offer complete replacement updates
  - Server MAY offer merge-patch updates

IRD Example

- This resource provides both complete-replacement and merge-patch updates for a cost map, but only complete-replacement updates for the network map:

```
"my-routingcost-update-stream": {
  "uri": "http://alto.example.com/updates/routingcost",
  "media-type": "text/event-stream",
  "uses": ["my-network-map", "my-routingcost-map"],
  "capabilities": {
    "events": [
      "application/alto-networkmap+json,my-network-map",
      "application/alto-costmap+json,my-routingcost-map",
      "application/merge-patch+json,my-routingcost-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 give 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.
- When a client establishes a stream, the server MUST immediately send complete replacement update events for every resource
  - But there are exceptions, as described later

Update Stream Example

```
GET /updates/routingcost HTTP/1.1
Host: alto.example.com
Accept: text/event-stream

HTTP/1.1 200 OK
Content-Type: text/event-stream

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

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

event: application/merge-patch+json,my-routingcost-map
data: {"cost-map": {"PID1" : {"PID2" : 9}}}

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

event: application/merge-patch+json,my-routingcost-map
data: {"meta": {"dependent-vtags": {"tag": "new-network-map-tag"}}}
```

Filtered Update Stream Service

- Like full Update Stream Service, except POST-mode
  - Client selects subset of available events
  - Client provides input for any POST-mode resources
  - Client may provide vtags of versioned resources, such as Network Maps, which it has previously retrieved; the server should send merge/patch updates relative to that version, and omit the complete map
- Examples:
  - Server offers a Filtered Update Stream for a Network Map and routingcost & hopcount Cost Maps. Then client may skip hopcount cost updates and initial full Network Map.
  - Server offers a Filtered Update Stream for an Endpoint Property Service for frequently changing properties (server load, bandwidth, etc.). Client requests updates for those properties for a set of endpoints. Server sends new values when the properties change.

Controversial Design Decision!

- At startup, the server MUST send full maps for all 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 this is only a problem for large maps and frequent disconnects
  - Clients who need large maps rarely drop connections
  - Clients who do drop connections don't need large maps
- So we propose to keep it simple, and accept the occasional inefficiency

Next Steps

- Tentatively approve the key points of this approach:
  - Transport: Server-Sent Events (SSE)
  - Message format: JSON Merge-Patch & Full ALTO messages
  - Server sends updates when they are available
  - One stream may provide updates for many resources
  - Server may offer multiple Update Stream resources
  - Server may provide an update stream for any resource
  - Server is not required to provide an update stream for every resource
  - At startup, server sends full replacement events for all untagged resources
- Between now & next meeting:
  - Review proposal via mailing list

Thank you

(Backup Slides)

IRD For Filtered Update Stream

```
"my-allresources-update-stream": {
   "uri": "http://alto.example.com/updates/allresources",
   "media-type": "text/event-stream",
   "uses": [
      "my-network-map",
      "my-routingcost-map","my-hopcount-map",
      "my-properties"
   ],
   "accepts": "application/alto-updatestreamfilter+json",
   "capabilities": {
     "events": [
       "application/alto-networkmap+json,my-network-map",
       "application/alto-costmap+json,my-routingcost-map",
       "application/merge-patch+json,my-routingcost-map"
       "application/alto-costmap+json,my-hopcount-map",
       "application/merge-patch+json,my-hopcount-map"
       "application/alto-endpointprops+json,my-properties",
       "application/merge-patch+json,my-properties"
     ]
   }
}
```

Filtered Update Stream Example 1

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

{ "events": [
      "application/alto-networkmap+json,my-network-map",
      "application/alto-costmap+json,my-routingcost-map",
      "application/merge-patch+json,my-routingcost-map"
      ],
  "vtags": [
      "resource-id": "my-network-map", "tag": "314159265359”}
] }

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

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

Filtered Update Ex 2: Request

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

{ "events": [
    "application/alto-endpointprops+json,my-properties",
    "application/merge-patch+json,my-properties"
  ],
  "inputs": {
    "my-properties": {
      "properties" : [ "priv:ietf-bandwidth" ],
      "endpoints" : [
         "ipv4:1.0.0.1",
         "ipv4:1.0.0.2",
         "ipv4:1.0.0.3"
      ]
    }
  }
}
```

Filtered Update Ex 2: Response

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

event: application/alto-endpointprops+json,my-properties
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" }
data:  } }

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

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

Update Stream and Resource Dependency

- Stream composition requirement
  - If a resource y depends on another resource x, then an update stream that includes y should also include x
- Event ordering requirement
  - If an update event x should appear before dependent events (e.g., ev_n10 that updates network map from v0 to v1 should appear before ev_c21)