ALTO WG
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2015

W. Roome
Alcatel-Lucent
X. Shi
Y. Yang
Yale University

October 27, 2014

ALTO Incremental Updates Using Server-Sent Events (SSE)
draft-roome-alto-incr-update-sse-00

Abstract

The goal of Application-Layer Traffic Optimization (ALTO) [RFC7285] is to bridge the gap between network and applications by providing network related information to non-priviledged, application-level clients. This allows applications to make informed decisions, for example when selecting a target host from a set of candidates.

Therefore an ALTO Server provides network and cost maps to its clients. However, those maps can be very large, and portions of those maps may change frequently (cost maps in particular).

This draft presents a method to provide incremental updates for these maps. The goal is to reduce the load on the ALTO Client and Server by transmitting just the updated portions of those maps.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of this Memo

Table of Contents

1. Introduction

The Application-Layer Traffic Optimization (ALTO) [RFC7285] protocol provides network related information to client applications so that clients may make informed decisions. An ALTO Server provides network and cost maps, which may be very large and change very frequently.

Instead of having the clients request for a new complete network map or cost map every time, an incremental update from the server is much more efficient. The goals are to reduce the load on the ALTO Client and Server by efficiently transmitting only the updated portions of those maps, and to provide timely updates to clients.

This draft uses the JSON Merge Patch message format [RFC7386] to encode the incremental update messages for network maps and cost maps, and uses Server-Sent Events (SSE) as the transport mechanism to deliver those updates to clients.

2. Incremental Update Message Format

2.1. JSON Merge Patch

[RFC7386] defines JSON Merge Patch format and transport, which enables applications to update the server resources via the PATCH method [RFC5789] of HTTP. This draft adopts the format of the Merge Patch messages to encode our incremental updates objects, but uses a different transport mechanism.

The process of applying a Merge Patch is defined by the following algorithm, as specified in [RFC7386]:

```
define MergePatch(Target, Patch) {
  if Patch is an Object {
    if Target is not an Object {
      Target = {} # Ignore the contents and
                  # set it to an empty Object
    }
    for each Name/Value pair in Patch {
      if Value is null {
        if Name exists in Target {
          remove the Name/Value pair from Target
        }
      } else {
        Target[Name] = MergePatch(Target[Name], Value)
      }
    }
    return Target
  } else {
    return Patch
  }
}
```

Note that null as the value of a name/value pair will remove the pair with "name" in the original JSON document.

2.2. JSON Merge Patch Applied to Network Map Messages

Section 11.2.1.6 of [RFC7285] defines the format of a Network Map message. Here is a simple example:

```
     {
       "meta" : {
         "vtag": {
           "resource-id": "my-default-network-map",
           "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
         }
       },
       "network-map" : {
         "PID1" : {
           "ipv4" : [
             "192.0.2.0/24",
             "198.51.100.0/25"
           ]
         },
         "PID2" : {
           "ipv4" : [ "198.51.100.128/25" ]
         },
         "PID3" : {
           "ipv4" : [ "0.0.0.0/0" ],
           "ipv6" : [ "::/0" ]
         }
       }
     }
```

When applied to that message, the following Merge Patch update message adds the ipv6 prefix "2000::/3" to "PID1", deletes "PID2", and assigns a new "tag" to the Network Map:

```
     {
       "meta" : {
         "vtag" : {
           "tag" : "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
         }
       },
       "network-map": {
         "PID1" : {
           "ipv6" : [ "2000::/3" ]
         },
         "PID2" : null
       }
     }
```

Here is the updated Network Map:

```
     {
       "meta" : {
         "vtag": {
           "resource-id": "my-default-network-map",
           "tag": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
         }
       },
       "network-map" : {
         "PID1" : {
           "ipv4" : [
             "192.0.2.0/24",
             "198.51.100.0/25"
           ],
           "ipv6" : [ "2000::/3" ]
         },
         "PID3" : {
           "ipv4" : [ "0.0.0.0/0" ],
           "ipv6" : [  "::/0" ]
         }
       }
     }
```

2.3. JSON Merge Patch Applied to Cost Map Messages

Section 11.2.3.6 of [RFC7285] defines the format of a Cost Map message. Here is a simple example:

```
     {
       "meta" : {
         "dependent-vtags" : [
           {"resource-id": "my-default-network-map",
            "tag": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
           }
         ],
         "cost-type" : {"cost-mode"  : "numerical",
                        "cost-metric": "routingcost"
         }
       },
       "cost-map" : {
         "PID1": { "PID1": 1,  "PID2": 5,  "PID3": 10 },
         "PID2": { "PID1": 5,  "PID2": 1,  "PID3": 15 },
         "PID3": { "PID1": 20, "PID2": 15  }
       }
     }
```

The following Merge Patch message updates that cost map so that (1) PID1->PID2 is 9 instead of 5; (2) PID3->PID1 is no longer available; and (3) PID3->PID3 is now 1:

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

Here is the updated Cost Map:

```
     {
       "meta" : {
         "dependent-vtags" : [
           {"resource-id": "my-default-network-map",
            "tag": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
           }
         ],
         "cost-type" : {"cost-mode"  : "numerical",
                        "cost-metric": "routingcost"
         }
       },
       "cost-map" : {
         "PID1": { "PID1": 1,  "PID2": 9,  "PID3": 10 },
         "PID2": { "PID1": 5,  "PID2": 1,  "PID3": 15 },
         "PID3": { "PID1": 20,             "PID3": 1  }
       }
     }
```

3. Server-Sent Events

3.1. Overview of SSEs

Server-Sent Events [SSE] enable a server to send new data to a client by pushing messages to the client. To summarize the protocol, the client establishes an HTTP connection to the server, and keeps the connection open. The server continually sends messages. Messages are delimited by two new-lines (this is a slight simplification of the full specification), and contain three fields: an event type, an id, and data. All fields are strings. The data field may contain new-lines; the other fields cannot. The event type and id fields are optional.

Here is a sample SSE stream, starting with the client request. The server sends three events and then closes the stream.

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

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

event: start
data: hello there

event: middle
data: let’s chat some more ... and more ...

event: end
data: good bye
```

3.2. ALTO SSE Update Messages

In our events, the data field is a JSON object. There two types of data objects. One is a message describing an ALTO resource, such as a Network Maps or Cost Map, as defined in [RFC7285]. We will refer to these as full-map messages. The other type is a Merge Patch message to apply to an ALTO resource.

Our event types have two sub-fields: the media-type of the JSON message in the data field, and the resource-id of the ALTO resource. The media-types for ALTO resource messages are defined by [RFC7285], and include "application/alto-networkmap+json" for Network Map messages and "application/alto-costmap+json" for Cost Map messages. The media-type for a Merge Patch message is "application/merge-patch+json", and is defined by [RFC7285].

We do not use the SSE id field.

Because commas (character code 0x2c) are not allowed in media-type names, we encode the event type sub-fields as

```
      media-type , resource-id
```

Here examples of ALTO update events:

```
     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: { ... Merge Patch update for previous Cost Map ... }
```

3.3. Keep-Alive Messages

An SSE event with an empty event type is a keep-alive message. An ALTO Server MAY send keep-alive messages as needed. An ALTO Client MUST ignore any keep-alive messages.

4. Update Stream Service

An Update Stream Service returns a stream of SSE messages, as defined in Section 3.2.

4.1. Media Type

The media type of an ALTO Update Stream resource is "text/event-stream".

4.2. HTTP Method

An ALTO Update Stream resource is requested using the HTTP GET method.

4.3. Accept Input Parameters

None.

4.4. Capabilities

The capabilities are defined by an object of type UpdateStreamCapabilities:

```
     object {
       JSONString events<1..*>;
     } UpdateEventStreamCapabilities;
```

The strings in the array are the event types (see Section 3.2) sent by this Update Stream.

If an Update Event Service’s event capability list has an event with a media-type of "text/merge-patch+json" for a resource-id, then the event capability list MUST also have an full-map event for that resource-id. For example, suppose "my-costmap" is the resource-id of a Cost Map. Then if the event list has "text/merge-patch+json,my-costmap", it MUST also have the event "application/alto-costmap+json,my-costmap".

4.5. Uses

An array with the resource-ids of the resources for which this stream sends updates. This array MUST contain the resource-ids of every event type in the "events" capability.

4.6. Event Order Requirements

There are several requirements on the order in which an ALTO Server sends SSE Update messages on the event stream:

o For any given resource-id, the ALTO Server MUST send a full-map update event (media-type "application/alto-networkmap+json" or "application/alto-costmap+json") before the first Merge Patch event (media-type "application/merge-patch+json") for that resource-id.

o The ALTO Server SHOULD send full-map update events for all resource-ids covered by this Update Stream resource as soon as possible after the client initiates the connection.

o If the event list contains a resource-id R0 on which resource-id R1 depends, when R0 changes, the ALTO Server MUST send the update for R0 before sending the update for R1. For example, suppose the event list includes a Network Map resource and its dependent Cost Map resources. When the Network Map changes, the ALTO Server MUST send an update event for that Network Map before sending the update events for the dependent Cost Maps.

o If the event list contains a resource-id R0 on which resource-id R1 depends, the ALTO Server SHOULD send an update for R1 as soon as possible after sending the update for R0. For example, when a Network Map changes, the ALTO Server SHOULD send update events for all dependent Cost Maps as soon as possible after the update event for the Network Map.

4.7. Response

Here is an example of a client's request and the server's immediate response, using the Update Stream resource "my-routingcost-update-stream" defined in the IRD in Section 6. This assumes the Update

Stream service sends updates for a Network Map with resource-id "my-network-map" and an associated Cost Map with resource-id "my-routingcost-map":

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

  HTTP/1.1 200 OK
  Connection: keep-alive
  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 ... }
```

After sending those two events immediately, the ALTO Server will send additional events as the maps change. For example, the following represents a small change to the Cost Map:

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

If a major change to the Network Map occurs, the ALTO Server MAY choose to send full Network and Cost Map messages rather than Merge Patch messages:

```
  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 ... }
```

4.8. Client Actions When Receiving Update Messages

In general, when a client receives a full-map update message for a resource, the client should replace the current version with the new version. When a client receives a Merge Patch update message for a resource, the client should apply those patches to the current version of the resource.

However, because resources can depend on other resources (e.g., Cost Maps depend on Network Maps), an ALTO Client MUST NOT use a dependent resource when the resource on which it depends changes. There are at least two ways a client may do that. We will illustrate these techniques by referring to Network and Cost Map messages, although these techniques apply to any dependent resources.

One approach is for the ALTO Client to save the Network Map update message in a buffer, and continue to use the previous Network Map, and the associated Cost Maps, until the client receives the update messages for all dependent Cost Maps. The client then applies all Network and Cost Map updates atomically.

Alternatively, the client MAY update the Network Map immediately. In this case, the client MUST mark each dependent Cost Map as temporarily invalid, and MUST NOT use that map until the client receives a Cost Map update message with the new Network Map version tag. Note that the client MUST NOT delete the Cost Maps, because the server may send Merge Patch update messages.

The ALTO Server SHOULD send updates to dependent resources in a timely fashion. However, if the client does not receive the expected updates, the client MUST close the Update Stream connection, discard the dependent resources, and reestablish the Update Stream. If the client uses the Filtered Update Stream service, the client MAY retain the version tag of the last version of any tagged resources, and give those version tags when requesting the new Update Stream. In this case, if a version is still current, the ALTO Server will not re-send that resource.

Although not as efficient as possible, this recovery method is simple and reliable.

5. Filtered Update Stream Service

The Filtered Update Stream service is similar to the Update Stream service, except that the client can select the types of update events. Specifically, except as noted below, the Filtered Update Stream service is identical to the Update Stream service (Section 4).

5.1. HTTP Method

A Filtered ALTO Update Stream resource is requested using the HTTP POST method.

5.2. Accept Input Parameters

An ALTO Client supplies filtering parameters by specifying media type "application/alto-updatestreamfilter+json" with HTTP POST body containing a JSON object of type ReqFilteredUpdateStream, where:

```
     object {
       [UpdateEventType  events<1..*>;]
       [VersionTag       vtags<1..*>;]
       [ResourceInputs   inputs<1..*>;]
     } ReqFilteredUpdateStream;

     object-map {
       ResourceID -> JSONObject;
     } ResourceInputs;
```

The "events" field gives the types of the events the ALTO Client wishes to receive. These events MUST be a subset of the "events" capability of this resource. If the "events" list is omitted, the ALTO Server MUST send all event types in the "events" capability of this resource.

The "vtags" field gives the version tags, as defined in Section 10.3 of [RFC7285], for any resources which the client already has. If those versions are still current, the server SHOULD NOT send the full version of that resource at startup.

The "inputs" field gives the client input needed for any POST-mode resources requested by the client. The value is a JSON object; the key is the resource-id of the POST-mode resource, and the value is the JSON object that it requires as "accepts" input.

If a client requests Merge Patch update events for a given resource-id, the client MUST also request the corresponding full map update events for that resource-id.

If a client requests the full-map update event for given resource-id, but does not request the Merge Patch update event for that resource-id, then the ALTO Server MUST send full-map update events whenever the map changes. For Network Map resources, the ALTO Server SHOULD send the full map as soon as it would have sent the Merge Patch event. For Cost Map and other resources, the ALTO Server MAY delay sending the full-map until more changes are available.

5.3. Response

Here is an example of a client's request and the server's immediate response, using the Filtered Update Stream resource "my-allresources-update-stream" defined in the IRD in Section 6. The client requests updates for the Network Map and the "routingcost" Cost Map, but does not want updates for the "hopcount" Cost Map. The "vtags" field gives the client's version of the Network Map. Because that version is still current, the server does not send the full Network Map update event at the beginning of the stream:

```
     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 ... }
```

After that, the ALTO Server sends updates for the Network Map and "routingcost" Cost Map as they become available.

As another example, here is how a client can request updates for the property "priv:ietf-bandwidth" for a set of endpoints. The ALTO Server immediately sends a full-map message with the property values for all endpoints. After that, the server sends update events for the individual endpoints as their property values change.

```
     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-endpointprop+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"
           ]
         }
       }
     }


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

     event: application/alto-endpointprop+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: }
```

6. IRD Example

Here is an example of an IRD that offers both regular and Filtered Update Stream services. The unfiltered Update Stream provides updates for the Network Map and "routingcost" Cost Map. The Filtered Update Stream provides update to both those maps, plus the "hopcount" Cost Map and the Endpoint Properties service.

```
     "my-network-map": {
       "uri": "http://alto.example.com/networkmap",
       "media-type": "application/alto-networkmap+json",
     },
     "my-routingcost-map": {
       "uri": "http://alto.example.com/costmap",
       "media-type": "application/alto-costmap+json",
       "uses": ["my-networkmap+json"],
       "capabilities": {
         "cost-type-names": ["num-routingcost"]
       }
     },
     "my-hopcount-map": {
       "uri": "http://alto.example.com/costmap",
       "media-type": "application/alto-costmap+json",
       "uses": ["my-networkmap+json"],
       "capabilities": {
         "cost-type-names": ["num-hopcount"]
       }
     },
     "my-properties": {
       "uri": "http://alto.example.com/properties",
       "media-type": "application/alto-endpointprops+json",
       "accepts": "application/alto-endpointpropparams+json",
       "capabilities": {
         "prop-types": ["priv:ietf-bandwidth"]
       }
     },
     "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"
         ]
       }
     },
     "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"
         ]
       }
     }
```

7. Design Decisions and Discussion

7.1. Not Allowing Stream Restart

If an update stream is closed accidentally, when the client reconnects, the server must resend the full maps. This is clearly inefficient. To avoid that inefficiency, the SSE specification allows a server to assign an id to each event. When a client reconnects, the client can present the id of the last successfully received event, and the server restarts with the next event.

However, that mechanism adds a lot of complication. The server would have to save SSE messages in a buffer, in case clients reconnect. But that mechanism will never be perfect: if the client waits too long to reconnect, or if the client's last id is bogus, then the server will have to resend the complete maps anyway.

In short, using event ids to avoid resending the full map adds a lot of complication to avoid a situation which is hopefully very rare. Hence we decided to keep it simple.

The Filtered Update Stream service does allow the client to specify the vtag of the last received Network Map, and if that is still current, the server can avoid retransmitting the Network Map.

7.2. Is Incremental Update Useful for Network Maps?

It is not clear whether incremental update (that is, Merge Patch update) is useful for Network Maps. For minor changes, such as moving a prefix from one PID to another, it might be useful. But more involved changes to the Network Map are likely to be "flag days": they represent a completely new Network Map, rather than a simple, well-defined change.

This is not to say that Network Map updates are not useful. Clearly Network Maps will change, and update events are necessary to inform clients of the new map. But we expect most Network Map updates will be full updates with full Network Map message, rather than incremental Merge Patch updates.

Note that while we allow a server to use Merge Patch on Network Maps, we do not require the server to do so.

8. Security Considerations

Allowing persistent update stream connections does enable a new class of Denial-of-Service attacks. An ALTO Server MAY choose to limit the number of active streams, and reject new requests when that threshold is reached. In this case the server should return the HTTP status "503 Service Unavailable".

Alternatively an ALTO Server MAY return the HTTP status "307 Temporary Redirect" to redirect the client to another ALTO Server which can better handle a large number of update streams.

This extension does not introduce any privacy issues not already present in the ALTO protocol.

9. IANA Considerations

This draft defines a new media-type, "application/ alto-updatestreamfilter+json", as described in Section 5.2. That type must be registered with IANA.

All other media-types used in this document have already been registered, either for ALTO or JSON Merge Patch.

10. References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

Requirement Levels", RFC 2119, BCP 14, March 1997.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[RFC7285] Almi, R., Penno, R., Yang, Y., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", RFC 7285, September 2014.

[RFC7386] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7386, October 2014.

[SSE] Hickson, I., "Server-Sent Events (W3C)", December 2012.

Authors' Addresses

Wendy Roome
Alcatel-Lucent/Bell Labs
600 Mountain Ave, Rm 3B-324
Murray Hill, NJ 07974
USA

Phone: +1-908-582-7974
Email: w.roome@alcatel-lucent.com

Xiao Shi
Yale University
51 Prospect Street
New Haven, CT 06511
USA

Email: xiao.shi@yale.edu

Y. Richard Yang
Yale University
51 Prospect St
New Haven CT
USA

Email: yang.r.yang@gmail.com