ALTO Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2022

R. Schott
Deutsche Telekom
Y. Yang
Yale University
K. Gao
Sichuan University
J. Zhang
Tongji University

March 5, 2022

ALTO New Transport using HTTP/2
draft-schott-alto-new-transport-00

Abstract

The ALTO base protocol [RFC7285] uses HTTP/1.x as the transport protocol and hence ALTO transport includes the limitations of HTTP/1.x. ALTO/SSE [RFC8895] addresses some of the limitations, but is still based on HTTP/1.x. This document introduces ALTO new transport, which provides the transport functions of ALTO/SSE on top of HTTP/2, for more efficient ALTO transport.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 6, 2022.

Table of Contents

1. Introduction

Application-Layer Traffic Optimization (ALTO) provides a means for network applications to obtain network status information. The ALTO base protocol [RFC7285] is based on the sequential request and response model of HTTP/1.1 [RFC7230]; hence, in the base protocol, an ALTO client can issue only a sequence of requests on network information resources, and the ALTO server sends the information resources one-by-one, in the order of the request sequence.

To address the use cases where an ALTO client may need to efficiently

monitor changes to a set of network information resources and the protocol is still based on the HTTP/1.1 model, the ALTO Working Group introduces ALTO/SSE (ALTO Incremental Update based on Server-Sent-Event) [RFC8895], so that an ALTO client can manage (i.e., add and remove) a set of requests maintained at an ALTO server, and the

server can continuously, concurrently, and incrementally push updates whenever a monitored network information resource changes. Figure 1 shows the architecture and message flow of ALTO/SSE, which can be considered as a more general transport protocol than the ALTO base transport protocol. Although ALTO/SSE allows the concurrent transport of multiple ALTO information resources, it has complexities and limitations. For example, it requires that the server provide a separate control URI, leading to complexity in management.

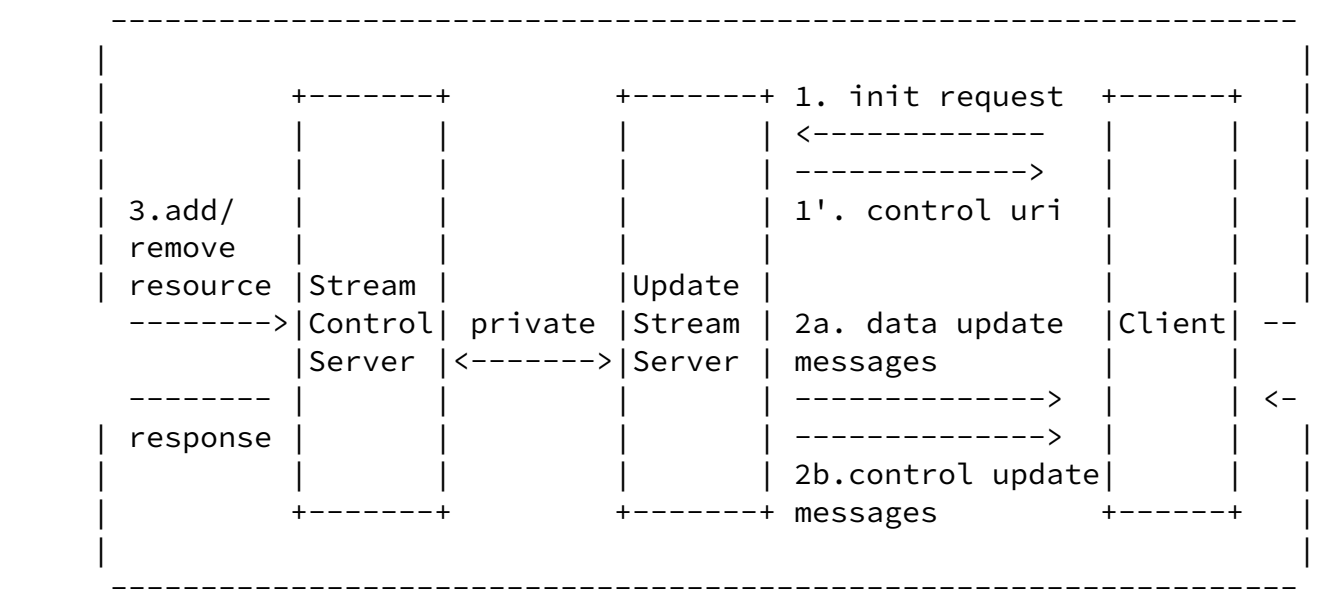

Figure 1: ALTO SSE Architecture and Message Flow.

This document specifies ALTO new transport, which realizes ALTO/SSE but takes advantage of new HTTP capabilities.

2. ALTO New Transport Design Requirements

The new transport is designed to satisfy a set of requirements.

First, it should satisfy the following requirements to realize the functions of ALTO/SSE:

- R0: Client can request any resource using the connection, just as using ALTO base protocol using HTTP/1.x.

- R1: The client can request the addition (start) of incremental updates to a resource.

- R2: The client can request the deletion (stop) of incremental updates to a resource.

- R3: The server can signal to the client the start or stop of incremental updates to a resource.

- R4: The server can choose the type of each incremental update endcoding, as long as the type is indicated to be acceptable by the client.

Following the ALTO framework, the new transport protocol should still be HTTP based:

- R5: The design follows basic principle of HTTP---Representational State Transfer and hence can use only HTTP verbs (GET, POST, PUT, DELETE, HEAD).

- R6: The design takes advantage of HTTP/2 design features such as parallel transfer and respects HTTP/2 semantics such as the semantics of PUSH_PROMISE.

To allow flexible deployment, the new transport protocol should be flexible:

- R7: The design should support capability negotiation.

3. ALTO New Transport Information Structure

A key design of ALTO new transport is to distinguish between information about ALTO resources and information about ALTO transport. It introduces the following transport information structures to distribute ALTO information resources:

- The transport state from the ALTO server to an ALTO client (or a set of clients) for an ALTO information resource is conceptually through a transport queue. A static ALTO information resource (e.g., Cost Map, Network Map) has a single transport queue, and a dynamic ALTO information resource (e.g., Filtered Cost Map) may create a queue for each unique filter request.

- Each transport queue maintains two states: (1) the incremental update message queue TQ-BASE-URI/m, and (2) the recipients set TQ-BASE-URI/r, where TQ-BASE-URI is the base URI pointing to the transport queue, TQ-BASE-URI/m/meta is the list of update messages in the update message queue, and TQ-BASE-URI/m/msg-seq-no is a specific update. A transport queue can be created by a POST URI; a client can delete a transport queue by sending DELETE the TQ-BASE-URI. A client with an interest to receive incremental updates should be in TQ-BASE-URI/r.

- The transport queue state is exposed to clients through views; that is, a client can see only a virtual view of the server state.

3.1. Transport Queue

An ALTO client creates a transport queue using ALTO SSE AddUpdateReq ([RFC 8895] Sec. 6.5). Unless the request has incremental-changes to be false, the client is added to TQ-BASE-URI/r.

```
object {
    ResourceID   resource-id;
    [JSONString  tag;]
    [Boolean     incremental-changes;]
    [Object      input;]
 } AddUpdateReq;
```

Any disconnection between the client and the server will remove the client from the receiver queue; that is, the receiver state is ephemeral. A client can also remove itself by deleting itself from the receiver set.

3.2. Incremental Update Message Queue

When a client joins a transport queue and specifies incremental push updates, the first message pushed from the server to the client is the last independent message in the incremental message queue, unless the client specifies a matching message tag.

3.3. Examples

The first example is client receiving cost map and its updates.


```
POST /tqs HTTP/2
Host: alto.example.com
Accept: application/alto-transport+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: TBD

{
    "resource-id": "my-routingcost-map"
}

HTTP/2 200 OK
Content-Type: application/alto-transport+json

{"mq": "/updates/streams/2718281828459"}
```

Note that the example above uses HTTP/1.x notation, and it is straightforward to change to use HTTP/2 notation. We use the short notation for now and will update to the HTTP/2 notation in later revisions.


```
Client -> server request

HEADERS
  - END_STREAM
  - END_HEADERS
    :method = POST
    :scheme = https
    :path = /tqs
```

```
CONTINUATION
 + END_HEADERS
    host = alto.example.com
    accept = application/alto-transport+json
    content-type = application/alto-updatestreamparams+json
    content-length = TBD

DATA
 + END_STREAM
 {
    "resource-id": "my-routingcost-map"
 }

Server -> client response:

HEADERS
  - END_STREAM
  + END_HEADERS
    :status = 200
    content-type = application/alto-transport+json
    content-length = TBD

DATA
  + END_STREAM
   {"mq": "/updates/streams/2718281828459"}
```

The client can check the status of the transport queue from the same connection:


```
GET /updates/streams/2718281828459/m/meta HTTP/2
```

```
Accept: application/alto-transport+json


HTTP/2 200 OK
Content-Type: application/alto-transport+json

{
  [
    {"seq":        101,
     "media-type": "application/alto-costmap+json",
     "tag":        "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe" },
    {"seq":        102,
     "media-type": "application/merge-patch+json",
     "tag":        "cdf0222x59740b0b2e3f8eb1d4785acd42231bfe" },
    {"seq":        103,
     "media-type": "application/merge-patch+json",
     "tag":        "8eb1d4785acd42231bfecdf0222x59740b0b2e3f",
     "equi-link":  "/updates/streams/2718281828459/m/aliase1"}

  ]
}
```

The client can directly request an element in the queue, for example,

```
GET /updates/streams/2718281828459/m/101 HTTP/2
Accept: application/application/alto-costmap+json


HTTP/2 200 OK
Content-Type: application/alto-costmap+json

{
   "meta" : {
      "dependent-vtags" : [{
         "resource-id": "my-network-map",
         "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
       }],
      "cost-type" : {
        "cost-mode"  : "numerical",
        "cost-metric": "routingcost"
      },
      "vtag": {
        "resource-id" : "my-routingcost-map",
        "tag" : "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"
      }
   },
   "cost-map" : {
     "PID1": { "PID1": 1,  "PID2": 5,  "PID3": 10 },
     "PID2": { "PID1": 5,  "PID2": 1,  "PID3": 15 },
     "PID3": { "PID1": 20, "PID2": 15  }
   }
}
```

Note from the transport queue state that the 103 message has an OPTIONAL link to a complete snapshot, which a client can request.

Instead of directly requesting, the client can wait for the server for incremental push, where the server first sends PUSH_PROMISE with the GET URI as above.

A client can leave incremental updates by sending the request:

```
DELETE /updates/streams/2718281828459/r/self HTTP/2
Accept: application/alto-transport+json


HTTP/2 200 OK
```

A second client can can request the creation for the same resource and the server can return the same transport queue.

A client can delete the transport queue from its view, and as long as there are other clients, the server will still maintain the transport queue.

```
DELETE /updates/streams/2718281828459 HTTP/2
Accept: application/alto-transport+json


HTTP/2 200 OK
```

The transport queue is not limited to only GET resources. The client can also request a filtered ALTO resource, which is shown in the example below:

```
POST /tqs HTTP/2
Host: alto.example.com
Accept: application/alto-transport+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: 382

{
   "resource-id": "my-pv",
   "input": {
      "cost-type": {
         "cost-mode": "array",
         "cost-metric": "ane-path"
      },
      "endpoints": {
        "srcs": [ "ipv4:192.0.2.2" ],
        "dsts": [ "ipv4:192.0.2.89", "ipv4:203.0.113.45" ]
      },
      "ane-properties": [ "maxresbw", "persistent-entities" ]
   }
}

HTTP/2 200 OK
Content-Type: application/alto-transport+json

{"mq": "/updates/streams/2718281828459"}
```

4. ALTO/H2 Information Resource Directory (IRD)

Extending the IRD example in Section 8.1 of [RFC8895], Figure 2 is the IRD of an ALTO server supporting ALTO base protocol, ALTO/SSE, and ALTO/H2.

In particular,


```
  "my-network-map": {
    "uri": "https://alto.example.com/networkmap",
    "media-type": "application/alto-networkmap+json",
  },
  "my-routingcost-map": {
    "uri": "https://alto.example.com/costmap/routingcost",
    "media-type": "application/alto-costmap+json",
    "uses": ["my-networkmap"],
    "capabilities": {
      "cost-type-names": ["num-routingcost"]
    }
  },
  "my-hopcount-map": {
    "uri": "https://alto.example.com/costmap/hopcount",
    "media-type": "application/alto-costmap+json",
    "uses": ["my-networkmap"],
    "capabilities": {
      "cost-type-names": ["num-hopcount"]
    }
  },
  "my-filtered-cost-map": {
    "uri": "https://alto.example.com/costmap/filtered/constraints",
    "media-type": "application/alto-costmap+json",
    "accepts": "application/alto-costmapfilter+json",
    "uses": ["my-networkmap"],
    "capabilities": {
      "cost-type-names": ["num-routingcost", "num-hopcount"],
      "cost-constraints": true
    }
  },
  "my-simple-filtered-cost-map": {
    "uri": "https://alto.example.com/costmap/filtered/simple",
```

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



```
    "my-pv": {
      "uri": "https://alto.example.com/endpointcost/pv",
      "media-type": "multipart/related;
                     type=application/alto-endpointcost+json",
      "accepts": "application/alto-endpointcostparams+json",
      "capabilities": {
        "cost-type-names": [ "path-vector" ],
        "ane-properties": [ "maxresbw", "persistent-entities" ]
      }
    },
    "update-my-costs": {
      "uri": "https://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",
         "my-simple-filtered-cost-map"
      ],
      "capabilities": {
        "incremental-change-media-types": {
          "my-network-map": "application/json-patch+json",
          "my-routingcost-map": "application/merge-patch+json",
          "my-hopcount-map": "application/merge-patch+json"
```

```
      },
      "support-stream-control": true
    }
  },
  "update-my-costs-h2": {
    "uri": "https://alto.example.com/updates-h2/costs",
    "media-type": "application/alto-h2",
    "accepts": "application/alto-updatestreamparams+json",
    "uses": [
       "my-network-map",
       "my-routingcost-map",
       "my-hopcount-map",
       "my-simple-filtered-cost-map"
    ],
    "capabilities": {
      "incremental-change-media-types": {
        "my-network-map": "application/json-patch+json",
        "my-routingcost-map": "application/merge-patch+json",
        "my-hopcount-map": "application/merge-patch+json"
      },
      "support-stream-control": true
    }
  },
```



```
  "update-my-props": {
    "uri": "https://alto.example.com/updates/properties",
    "media-type": "text/event-stream",
    "uses": [ "my-props" ],
    "accepts": "application/alto-updatestreamparams+json",
    "capabilities": {
      "incremental-change-media-types": {
        "my-props": "application/merge-patch+json"
      },
      "support-stream-control": true
    }
  },
  "update-my-pv": {
    "uri": "https://alto.example.com/updates/pv",
    "media-type": "text/event-stream",
    "uses": [ "my-pv" ],
    "accepts": "application/alto-updatestreamparams+json",
    "capabilities": {
```

```
      "incremental-change-media-types": {
        "my-pv": "application/merge-patch+json"
      },
      "support-stream-control": true
    }
  }
```

Note that it is straightforward for an ALTO sever to run HTTP/2 and support concurrent retrieval of multiple resources such as "my-network-map" and "my-routingcost-map" using multiple HTTP/2 streams with the need to introducing ALTO/H2.

The resource "update-my-costs-h2" provides an ALTO/H2 based connection, and this is indicated by the media-type "application/alto-h2". For an ALTO/H2 connection, the client can send in a sequence of control requests using media type application/alto-updatestreamparams+json. The server creates HTTP/2 streams and pushes updates to the client.

5. Security Considerations

The properties defined in this document present no security considerations beyond those in Section 15 of the base ALTO specification [RFC7285].

6. IANA Considerations

IANA will need to register the alto-h2 media type under ALTO registry as defined in [RFC7285].

7. Acknowledgments

The authors of this document would also like to thank many for the reviews and comments.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/

RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7285] Alimi, R., Ed., Penno, R., Ed., Yang, Y., Ed., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", RFC 7285, DOI 10.17487/RFC7285, September 2014, <https://www.rfc-editor.org/info/rfc7285>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8895] Roome, W. and Y. Yang, "Application-Layer Traffic Optimization (ALTO) Incremental Updates Using Server-Sent Events (SSE)", RFC 8895, DOI 10.17487/RFC8895, November 2020, <https://www.rfc-editor.org/info/rfc8895>.

8.2. Informative References

[RFC7971] Stiemerling, M., Kiesel, S., Scharf, M., Seidel, H., and S. Previdi, "Application-Layer Traffic Optimization (ALTO) Deployment Considerations", RFC 7971, DOI 10.17487/RFC7971, October 2016, <https://www.rfc-editor.org/info/rfc7971>.

Authors' Addresses

Roland Schott
Deutsche Telekom

Heinrich-Hertz-Strasse 3-7
64295 Darmstadt
Germany

Email: Roland.Schott@telekom.de

Y. Richard Yang
Yale University
51 Prospect St
New Haven, CT 06520
USA

Email: yry@cs.yale.edu

Kai Gao
Sichuan University
Chengdu 201804
China

Email: kgao@scu.edu.cn

Jingxuan Jensen Zhang
Tongji University
4800 Cao'An Hwy
Shanghai 201804
China

Email: jingxuan.n.zhang@gmail.com