Workgroup: ALTO Working Group
Internet-Draft:
draft-schott-alto-new-transport-push-00
Published: 21 October 2022
Intended Status: Standards Track
Expires: 24 April 2023
Authors: R. Schott   Y. Yang
         Deutsche Telekom   Yale University
         K. Gao   L. Delwiche
         Sichuan University   Yale University

ALTO New Transport: Server Push using PUSH_PROMISE of HTTP/2

Abstract

The ALTO New Transport [draft-ietf-alto-new-transport] introduces ALTO transport information structures (TIS) at an ALTO server. The introduction of ALTO TIS allows at least two types of efficient transport using HTTP: (1) HTTP/2/3 independent client long poll allowed by non-blocking, newer HTTP, and (2) HTTP/2 specific server push. This document defines HTTP/2 specific server-push ALTO transport based on ALTO TIS.

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

Copyright Notice

Table of Contents

1. Introduction

The ALTO new transport [draft-ietf-alto-new-transport] introduces ALTO transport queues for an ALTO server to manage the transport of ALTO information to an ALTO client. The base design, however, supports only client pull. Hence, for a client to obtain the latest ALTO information, the client need to maintain a pending pull on the incremental updates queue. This document extends the base design to allow server push, potentially reducing information distribution delay.

The extension to realize server push on a transport queue is by
adding a receiver set. Figure 2 shows an example illustrating the
additional receiver set at each transport queue.

Information Resource:

a) Static resource (#1) such as NetworkMap
b) Filterable resource (#3) such as FilteredCostMap

```
                              +-------------+
                              |             |
         +--------------------| ALTO Server |-----------+
         |                  +-|             |-+         |
         |                  | +-------------+ |         |
         |                  |                 |         |
---------|------------------|-----------------|---------|------------
         |                  |                 |         | Information
         |                  |                 |         | Resource
+-------------+   +-------------+   +-------------+   +-------------+
| Information |   | Information |   | Information |   | Information |
| Resource #1 |   | Resource #2 |   | Resource #3 |   | Resource #4 |
+-------------+   +-------------+   +-------------+   +-------------+
       |                              /    \
-------|-----------------------------/------\------------------------
       |                            /        \            Transport
       |                      +----/          \------+    Queues
       |                      |                      |
  +--------+             +--------+             +--------+
  |   tq1  |-----+       |   tq2  |-----+       |   tq3  |-----+
  +----|---+     |       +----|---+     |       +----|---+     |
       |         |            |         |            |         |
  +----|---+ +---|----+  +----|---+ +---|----+  +----|---+ +---|----+
  | tq1/uq | | tq1/rs |  | tq2/uq | | tq2/rs |  | tq3/uq | | tq3/rs |
  +--------+ +--------+  +--------+ +--------+  +--------+ +--------+
       |\       /\              |         /           |          |
-------|-\-----/--\-------------|--------/------------|----------|---
       |  \   /    +-------+    |       /             |          |
       |   +-/-----------+  \   |      /              |          |
       |    /             \  \  |     /   A           +          +
       |   /            +--\--\-|----/--+ single       \        /
       |  /             +---\--\|---/---+ http2/3       \      /
   +----------+             +----------+  connection   +----------+
   | Client 1 |             | Client 2 |               | Client 3 |
   +----------+             +------- --+               +----------+

tqi    = transport queue i
tqi/uq = incremental updates queue of transport queue i
tqi/rs = receiver set of transport queue i
```

Figure 1: ALTO New Transport Information Structure.

This document specifies the operation to manage the receiver set.

2. Manage Server Push: Receiver Set

2.1. Receiver Set Operations

A client starts to receive server push when it is added to the receiver set. A client can add itself to the receiver set when creating the transport queue, or add itself explicitly to the receiver set. A client can read the status of the receiver set and delete itself from the receiver set to stop server push.

Implicit Create: As a short cut, when creating a transport queue, an ALTO client can start server push by setting the "incremental-changes" field to be true when creating a transport queue using the HTTP POST method with ALTO SSE AddUpdateReq ([RFC 8895] Sec. 6.5) as the parameter:

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

PUT Create: A client can add itself in the receiver set by using the HTTP PUT method: PUT transport-queue/rs/self

Read: A client can see only itself in the receiver set. The appearance of self in the receiver set (read does not return "not exists" error) is an indication that push starts.

Delete: A client can delete itself (stops receiving push) either explicitly or implicitly.

* Explicit delete: A client deletes itself using the HTTP DELETE method: DELETE transport-queue/rs/self.

* Implicit delete: Transport queue is connection ephemeral: the close of connection or stream for the transport queue deletes the transport queue (from the view) for the client.

2.2. Examples

The first example is a client creating a transport queue and starting server push.

```
Client -> server request

HEADERS
  - END_STREAM
  + END_HEADERS
    :method = POST
    :scheme = https
    :path = /tqs
    host = alto.example.com
    accept = application/alto-error+json,
                 application/alto-transport+json
    content-type = application/alto-transport+json
    content-length = TBD

DATA
 - END_STREAM
 {
    "resource-id": "my-routingcost-map",
    "incremental-push": true
 }



Server -> client response:

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

DATA
  - END_STREAM
   {"tq": “/tqs/2718281828459”}
```

If the client reads the status of the transport queue created above using the read operation (GET) in the same HTTP connection, the client should see itself in the receiver set:

```
Client -> server request

HEADERS
  - END_STREAM
  + END_HEADERS
    :method = GET
    :scheme = https
    :path = /tqs/2718281828459
    host = alto.example.com
    accept = application/alto-error+json,
                 application/alto-transport+json

Server -> client response:

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

DATA
  - END_STREAM
 { "uq":
    [
      {“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",
       "link":       "/tqs/2718281828459/snapshot/2e3f"}

    ],
  "rs": ["self"]
 }
```

A client can stop incremental push updates from the server to itself by sending the request:

```
DELETE /tqs/2718281828459/rs/self HTTP/2
Accept: application/alto-transport+json


HTTP/2 200 OK
```

3. Server Push of Incremental Updates

3.1. Server Push

The work flow of server push of individual updates is the following:

* Initialization: the first update pushed from the server to the client MUST be the later of the following two: (1) the last independent update in the incremental updates queue; and (2) the following entry of the entry that matches the tag when the client creates the transport queue. The client MUST set SETTINGS_ENABLE_PUSH to be consistent.

* Push state: the server MUST maintain the last entry pushed to the client (and hence per client, per connection state) and schedule next update push accordingly.

* Push management: The client MUST NOT cancel (RST_STREAM) a PUSH_PROMISE to avoid complex server state management.

3.2. Examples

A client can wait for the server for incremental push, where the server first sends PUSH_PROMISE, for the first example in Sec. 2.2:

```
Server -> client PUSH_PROMISE in current stream:

PUSH_PROMISE
  - END_STREAM
    Promised Stream 4
    HEADER BLOCK
    :method = GET
    :scheme = https
    :path = /tqs/2718281828459/uq/101
    host = alto.example.com
    accept = application/alto-error+json,
                 application/alto-costmap+json

Server -> client content Stream 4:

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

DATA
  + END_STREAM
 {
   "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  }
   }
}

Server -> client PUSH_PROMISE in current stream:

PUSH_PROMISE
```

```
  - END_STREAM
    Promised Stream 6
    HEADER BLOCK
    :method = GET
    :scheme = https
    :path = /tqs/2718281828459/uq/102
    host = alto.example.com
    accept = application/alto-error+json,
                 application/merge-patch+json

Server -> client content Stream 6

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

DATA
  + END_STREAM
 { ...}
```

4. Server Push Stream Management

4.1. Server -> Client [PUSH_PROMISE for Transport Queue on Stream SID_tq]

The server push MUST satisfy the following requirements:

- *PUSH_PROMISE MUST be sent in stream SID_tq to serialize to allow the client to know the push order;

- *Each PUSH_PROMISE chooses a new server-selected stream ID, and the stream is closed after push.

5. Server Push Information Resource Directory (IRD)

Extending the IRD example in Section 8.1 of [RFC8895], below is the IRD of an ALTO server supporting ALTO base protocol, ALTO/SSE, and Server Push.

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-transport+json",
    "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
    }
  }
```

6. Security Considerations

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

7. IANA Considerations

IANA will need to register server push.

8. Acknowledgments

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

9. References

9.1. Normative References

[draft-ietf-alto-new-transport] Schott, R. and Y. Yang, "ALTO New Transport: ALTO Transport Information Structures", Internet Draft ID, October 2022, <https://datatracker.ietf.org/doc/draft-ietf-alto-new-transport/02/>.

[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, "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>.

9.2. 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
United States of America

Email: yry@cs.yale.edu

Kai Gao
Sichuan University
Chengdu
201804
China

Email: kgao@scu.edu.cn

Lauren Delwiche
Yale University
51 Prospect St
New Haven, CT 06520
United States of America

Email: lauren.delwiche@yale.edu