ALTO
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2015

W. Roome
Alcatel-Lucent
N. Schwan
Thales Deutschland

July 2, 2014

ALTO Incremental Updates
draft-roome-alto-incr-updates-01

Abstract

The goal of Application-Layer Traffic Optimization (ALTO) is to bridge the gap between network and applications by provisioning network related information. 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 (the cost map 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

Copyright Notice

Table of Contents

1. Introduction

The goal of Application-Layer Traffic Optimization (ALTO) is to bridge the gap between network and applications by provisioning network related information. This allows applications to make informed decisions, for example when selecting a target host from a set of candidates. Typical applications are file sharing, real-time communication and live streaming peer-to-peer networks [RFC5693] as well as Content Distribution Networks [I-D.jenkins-alto-cdn-use-cases].

The ALTO protocol [RFC7285] is a client-server protocol based on the HyperText Transfer Protocol (HTTP) and encoded in JavaScript Object Notation (JSON). An ALTO server provides several services, two of which are relavent to this draft.

The ALTO Network Map Service makes the large space of endpoint addresses manageable by partitioning them into a small set of equivalence classes, called Provider-defined Identifiers, or PIDs. Each PID is defined by a set of endpoint address prefixes, or CIDRs [RFC4632]. The ALTO Server defines PIDs it sees fit. Some servers might define a fine-grained Network Map with thousands of PIDs, while others might define a course-grained Map with tens of PIDs. The only requirement is that the network costs for all endpoints in a PID are similar.

The ALTO Cost Map Service presents the unidirectional network cost between each pair of PIDs. Costs are numeric and non-negative, but an ALTO Server may omit unknown costs. Essentially a Cost Map is a (possibly) sparse NxN matrix, where N is the number of PIDs in the Network Map.

The size of these maps depends primarily on the number of PIDs the ALTO Server choses to define. Because they go with the square of the number of PIDs, Cost Maps in particular can become very large. As an example, a Network Map with 5,000 PIDs, each with 10 CIDRs, is roughly 1.25 megabytes. A fully specified Cost Map for 5,000 PIDs takes up to 417 megabytes.

These maps may change at any time. Although not a protocol requirement, we expect that for many ALTO Servers, the Cost Map will change much more frequently than the Network Map. For example, the Cost Map might change every few minutes, as opposed to hours, if not days, between changes to the Network Map. However, we expect that only a small portion of these maps will change at any given time.

Thus with the base ALTO protocol, if a client wishes to maintain an up-to-date copy of the Network and Cost Maps, it must fetch a large

amount of data very frequently, even though only a small fraction of that data will have changed. This puts additional load on the ALTO Server, the ALTO Client and the network. This draft presents an extension to the ALTO protocol to allow a client to fetch just the updated portion of those maps.

Comments and discussions about this memo should be directed to the ALTO working group: alto@ietf.org.

2. Issues With Incremental Update

There are several issues involved with incremental updates:

2.1. Communication Mechanism

How does the server send incremental updates to the client? The two basic approaches are "server-push", where the server sends updates to the client when they become available, versus "client-pull", where the client periodically asks the server to send any changes.

In general, "server-push" is more efficient than "client-pull". However, ALTO is based on HTTP ([RFC2616]), and HTTP is a "client-pull" protocol. While there are push-like extentions to HTTP, they are not as widely supported as the basic HTTP protocol. Hence we will focus on solutions in which the client periodically polls the server via simple HTTP requests.

2.2. Polling Frequency

If we use a polling method, how often should a client check the server for updates? The simplest solution is to use the HTTP Expires header ([RFC2616]). The full Network Map and Cost Map services return that header in the response, as a guideline for the client as to when to check for updates.

An alternative would be to add an "expires" field to the "meta" section of the response message, so the expiration date stays with the message body instead of being in the HTTP headers.

2.3. Version Specification

How does a client tell the server what version the client has? Rather than inventing a new mechanism for that, we propose extending the ALTO protocol’s "version tag" concept. The base protocol requires an ALTO Server to assign a unique id ("tag") to the Network Map, and update the tag every time the Network Map changes. We will extend that concept to Cost Maps as well.

2.4. Message Format

The final question is how to represent an incremental update. Fortunately the ALTO Cost Map response message works very nicely to describe incremental updates; the client can update the cost pairs in the message, and leave the other data as is.

JSON Patch ([RFC6902]) can also represent incremental changes. However, as described in Section 6.2, we believe the existing ALTO

Cost Map message is more appropriate.

However, the ALTO Network Map response message does not work as well for incremental updates, especially if PIDs have hundreds of prefixes and typical updates involve moving a few prefixes from one PID to another. Accordingly we will define a new message for Network Map Updates. This provides a compact represenation of the expected update actions: moving prefixes between PIDs, deleting unused prefixes, and adding or deleting PIDs.

3. Incremental Update Extensions

Incremental update involves two new services, plus extensions to the base protocol’s Network Map and Cost Map services.

3.1. Date and Expires HTTP Headers

If an ALTO Server supports incremental update for a Network Map or Cost Map Service, the server SHOULD return the HTTP Date and Expires headers with the responses for those services. The client SHOULD request an update no sooner than the date in the Expires header. If omitted, the client would add a reasonable guess to the date in the Date header, or if ommitted, to the current time.

3.2. Extensions to Cost Map Service

If an ALTO Server supports incremental update for a Cost Map Service, the server MUST assign a "version tag" ("vtag") to each version of the Cost Map. As with Network Map vtags, the server MUST change the tag whenever any cost in the map changes. The ALTO Server puts the tag in the "meta" section of the response message, just as it does for a Network Map response.

When the Network Map changes -- that is, when the ALTO Server assigns a new tag to the Network Map -- the ALTO Server MUST assign a new tag to the Cost Map, even if no costs change.

Here is an example Cost Map response:

```
 HTTP/1.1 200 OK
 Date: TBA
 Expires: TBA
 Content-Length: TBA
 Content-Type: application/alto-costmap+json

 {
     "meta": {
         "vtag":
             {"resource-id": "numerical-routing-cost-map",
              "tag": "3141592653"},
         "dependent-vtags" : [
             {"resource-id": "my-default-network-map",
              "tag": "1266506139"}
           ],
         "cost-type" : {"cost-mode": "numerical",
                        "cost-metric": "routingcost"}
         },
     "cost-map": { .... }
```

```
    }
```

This addition is only required for Cost Map resources for which the ALTO Server chooses to offer incremental updates.

3.3. Filtered Cost Map Service

The Filtered Cost Map Service MUST NOT return the Cost Map vtag (it does return the Network Map vtag, of course). If the client maintains a copy of the Full Cost Map, the client MUST NOT save the Filtered Cost Map costs in that table. That is, even if the ALTO Server provides an Incremental Cost Map Update Service, the Filtered Cost Map Service works exactly as described in [RFC7285].

The reason is that Full and Filtered Cost Map Services may return inconsistent costs. For example, the costs returned by the Filtered Cost Map Service may be more up-to-date than the costs returned by the Full Cost Map Service (see Section 5). This inconsistency is inherent in the base ALTO protocol, because an ALTO Server may update costs at any time. We do not believe this inconsistency will be a problem, because we do not expect clients will use both the Full and Filtered Cost Map Services. Specifically, some clients, especially high-volume clients, will fetch and save the Full Cost Map, and use that to calculate costs as needed. These clients will use the incremental update service to get changes to the full Cost Map. Other clients will use the Filtered Cost Map Service whenever they need to evaluate costs. These clients will not bother to fetch or save the Full Cost Map.

3.4. Incremental Network Map Update Service

This new service returns the changes between the current Network Map and a version previously retrieved by the client.

3.4.1. Media Type

The media type is the new type "application/alto-networkmapupdate+json".

3.4.2. HTTP Method

An Incremental Network Map Update is requested using the HTTP POST method.

3.4.3. Accept Input Parameters

An ALTO Client supplies the vtag of the previous version by specifying media type "application/alto-vtag+json" with an HTTP POST body containing a JSON object of type VersionTag, as defined in Section 10.3 of [RFC7285]:

```
    object {
        ResourceID resource-id;
        JSONString tag;
    } VersionTag;
```

3.4.4. Capabilities

None.

3.4.5. Uses

The Resource ID of the Network Map for which this resource supplies incremental updates.

3.4.6. Response

The "meta" field of an Incremental Network Map Update response MUST include the "vtag" key with the latest version of the Network Map. The "resource-id" is for the Full Network Map Service, not the Incremental Update Service. In other words, the Incremental Update Service returns the same "vtag" that the Full Network Map Service would return.

The "meta" field MUST include a "dependent-vtags" key with the "resource-id" of the Full Network Map Service and the "tag" of the client’s current version. Thus the body of the response contains the changes from the "dependent-vtags" version to the "vtag" version.

The body of the response includes three data members: "network-map-add", "network-map-delete" and "network-map-delete-pids". These members MAY be empty JSON objects. A JSON Server MAY omit any data member that would otherwise be empty.

The "network-map-add" member is a NetworkMapData object, as defined in Section 11.2.1.6 of the ALTO protocol. The syntax is identical to that of the "network-map" member in a Network Map Service response message, but the semantics are different: the client MUST add the prefixes listed in the "network-map-add" object for a PID to the prefixes previously defined for that PID. If any prefix had been in another PID, the client MUST remove that prefix from the former PID. If a PID was not defined in the previous version, the client MUST add that PID to its list of PIDs.

The "network-map-delete" member is an EndpointAddressGroup object, as defined in Section 10.4.5 of the ALTO protocol. The client MUST delete the prefixes listed in this member from whatever PID they had been in before. The client MUST ignore any prefix that was not previously in some PID.

The "network-map-delete-pids" member is an array of PID names. The client MUST delete all PIDs in that list, and remove all prefixes in those PIDs, unless "network-map-add" assigns those prefixes to another PID. The client MUST ignore any PID name that did not exist in the previous version.

An ALTO Server MUST ensure that the update actions implicit in these three members do not conflict, so an ALTO Client MAY apply those updates in any order. Specifically, the same prefix MUST NOT appear in both the "network-map-add" and "network-map-delete" lists, and the same PID MUST NOT appear in both the "network-map-add" and "network-map-delete-pids" lists.

If there have been no changes since the version specified by the client's tag, the data members MUST be empty or omitted. In this case, the "tag" in "vtag" MUST be the same as the tag supplied by the client.

If the client's tag is invalid, or if it is so old that the ALTO Server is unable to provide incremental updates relative to that version, or if there have been so many changes that the ALTO Server is unwilling to provide incremental updates relative to that version, the ALTO Server MUST return an E_INVALID_FIELD_VALUE error response. In this case, the client SHOULD use the Full Network Map Service to retrieve the latest version.

The Incremental Cost Map Update response SHOULD include the HTTP Date and Expires headers, as a hint to the client as to when to request another incremental update.

3.4.7. Information Resource Directory Example

This is an example of the Information Resource Directory (IRD) entry for an Incremental Network Map Update Service resource for a Full Network Map Service with Resource ID "my-default-network-map":

```
   {
     "meta" : { .... },
     "resources" : {
       "my-default-network-map" : {
           ...
       },
       "my-default-network-map-update" : {
           "uri" : "http://alto.example.com/networkmap-update",
           "media-type" : "application/alto-networkmapupdate+json",
           "accepts" : "application/alto-vtag+json",
           "uses" : [ "my-default-network-map" ]
       },
       ...
   }
```

3.4.8. Request And Response Example

In this example, the Incremental Network Map Update Service adds a prefix to PID1, deletes another prefix from whatever PID it had been in, and deletes PID2 altogether.

```
POST /networkmap/incremental HTTP/1.1
Host: custom.alto.example.com
Content-Length: TBA
Content-Type: application/alto-vtag+json
Accept: application/alto-networkmapupdate+json,application/alto-error+json

{"vtag": {"resource-id": "NETWORK-MAP-ID", "tag": "OLD-TAG"}}


HTTP/1.1 200 OK
Date: TBA
Expires: TBA
Content-Length: TBA
Content-Type: application/alto-networkmapupdate+json

{
    "meta": {
        "vtag":
              {"resource-id": "NETWORK-MAP-ID", "tag": "NEW-TAG"},
        "dependent-vtags":
              [{"resource-id": "NETWORK-MAP-ID", tag: "OLD-TAG"}]
         },
    "network-map-add": { "PID1": {"ipv4": ["192.0.2.0/24"]} },
    "network-map-delete": { "ipv4": [192.0.3.0/24] },
    "network-map-delete-pids": [ "PID2" ]
}
```

3.4.9. Comments

A client can discover the Incremental Update Service for a given Network Map by looking for a resource that uses the desired Network Map resource, returns the media type "application/alto-networkmapupdate+json", and accepts the media type "application/alto-vtag+json".

3.5. Incremental Cost Map Update Service

This new service returns the changes between the current Cost Map and a version previously retrieved by the client.

3.5.1. Media Type

The media type is "application/alto-costmap+json", the same as for a Full or Filtered Cost Map.

3.5.2. HTTP Method

An Incremental Cost Map Update is requested using the HTTP POST method.

3.5.3. Accept Input Parameters

An ALTO Client supplies the vtag of the previous version by specifying media type "application/alto-vtag+json" with an HTTP POST body containing a JSON object of type VersionTag, as defined in Section 10.3 of [RFC7285]:

```
    object {
        ResourceID resource-id;
        JSONString tag;
    } VersionTag;
```

3.5.4. Capabilities

There are no explicit capabilities for this service. This service uses the cost metric and cost mode of the Full Cost Map Service for which this service provides incremental updates.

3.5.5. Uses

The Resource ID of the Cost Map for which this resource supplies incremental updates. An Incremental Cost Map Update resource MUST NOT list a Network Map resource. The Network Map is implicit in the "uses" list of the Cost Map resource.

3.5.6. Response

The "meta" field of an Incremental Cost Map Update response MUST include the "vtag" key with the latest version of the Cost Map. The "resource-id" is for the Full Cost Map Service, not the Incremental Update Service; the Incremental Update Service returns the same "vtag" that the Full Cost Map Service would return.

The "meta" field MUST also include a "dependent-vtags" key with the vtag of the client's version of the Cost Map, to indicate that the body of the response contains the changes from the "dependent-vtags" version to the "vtag" version.

"dependent-vtags" must also include the vtag of the version of the Network Map resource that defines the PIDs in this Cost Map.

The body of the response has the cost points that changed between the old version and the current version. Costs not mentioned in the body keep the same values as before. If the cost for that source/destination pair is no longer known the ALTO Server MUST specify the cost as "null" (a reserved token in JSON).

An ALTO Client MUST delete all cost points with the value "null", replace (or add) the other cost points in the response, and leave unchanged any cost points defined in the previous version.

If the version supplied by the client is still current, the "network-map" body will be empty, and the "tag" in "vtag" will be the same as the tag supplied by the client.

If the client's tag is invalid, or if it is so old that the ALTO Server is unable to provide incremental updates relative to that version, or if there have been so many changes that the ALTO Server is unwilling to provide incremental updates relative to that version, the ALTO Server MUST return an E_INVALID_FIELD_VALUE error response. The client MUST use the Full Cost Map Service to retrieve the latest version.

As with the Full Cost Map service, the Incremental Cost Map Update response SHOULD include the HTTP Date and Expires headers, as a hint to the client as to when to request another incremental update.

3.5.7. Information Resource Directory Example

This is an example of the Information Resource Directory (IRD) entry for an Incremental Cost Map Update Service resource for a Full Cost Map Service with Resource ID "numerical-routing-cost-map":

```
 {
   "meta" : { .... },
   "resources" : {
     ....
     "numerical-routing-cost-map" : {
         ...
     },
     "numerical-routing-cost-map-update" : {
       "uri" : "http://alto.example.com/costmap/num/routingcost-update",
       "media-type" : "application/alto-costmap+json",
       "accepts" : "application/alto-vtag+json",
       "uses" : [ "numerical-routing-cost-map" ]
     },
     ...
 }
```

3.5.8. Request And Response Example

In this example, the Incremental Cost Map Update Service reports that the cost from PID1 to PID2 is 10, and the cost from PID1 to PID99 is no longer available. All other costs remain the same as before.

```
   POST /costmap/num/routingcost/incremental HTTP/1.1
   Host: custom.alto.example.com
   Content-Length: TBA
   Content-Type: application/alto-vtag+json
   Accept: application/alto-costmap+json,application/alto-error+json

   {"vtag": {"resource-id": "COST-MAP-ID", "tag": "OLD-CM-TAG"}}


   HTTP/1.1 200 OK
   Date: TBA
   Expires: TBA
   Content-Length: TBA
   Content-Type: application/alto-costmap+json

   {
        "meta": {
             "vtag":
                 {"resource-id": "COST-MAP-ID", "tag": "NEW-CM-TAG"},
             "dependent-vtags": [
                 {"resource-id": "COST-MAP-D", tag: "OLD-CM-TAG"},
                 {"resource-id": "NETWORK-MAP-ID", tag: "OLD-NM-TAG"}
               ]
             },
        "cost-map": {
             "PID1": {"PID2": 10, "PID99": null}
             }
   }
```

3.5.9. Comments

A client can discover the Incremental Update Service for a given Cost Map by looking for a resource that uses the desired Cost Map resource, returns the media type "application/alto-costmap+json", and accepts the media type "application/alto-vtag+json".

The ALTO protocol says that a cost must be non-negative, so it is tempting to use the value -1, instead of "null", to indicate a cost that is no longer available. However, that would preclude future ALTO extensions from allowing negative costs. It is also tempting to use "NaN", for "Not a Number". Unfortunately, the JSON specification does not allow NaN as a numerical value.

The Incremental Cost Map Update Service is independent of the Incremental Network Map Update Service. An ALTO Server can implement one without the other.

4. Impact On Existing ALTO Clients

The incremental update services do not affect clients who are not aware of this extension. According to the ALTO protocol, clients must ignore fields that are not defined in the base protocol, so existing clients should ignore the new version tag in the Cost Map response. Similarly, clients who are not aware of the new incremental update services will simply ignore those resources in the Information Resource Directory, and will never use those URIs.

5. Server Update Model

While this extension does not dictate how an ALTO Server would implement incremental updates, it is useful to outline one possible strategy.

First we will consider cost map updates. We start by assuming updates arrive individually rather than en masse. That is, if there are 1,000 PIDs, cost updates trickle in a few at a time, rather than all 1,000,000 costs arriving in one batch.

The server keeps two copies of the Cost Map: a "frozen" version and a "latest" version. The server also keeps a "change log" with the differences. The frozen version has a tag, the latest version does not. The Full Cost Map Service uses the frozen map, while the Filtered Cost Map Service uses the latest map.

As cost updates arrive, the server immediately applies them to the latest version, and saves the updated cost points in the change log. When the change log becomes large enough, the server applies all the logged updates to the frozen version, and assigns it a new tag.

Thus the frozen version of the Cost Map is updated in well defined steps. Each step has a tag as the version id, and the change logs contain the incremental changes between each version.

The server keeps the old change logs in a FIFO list indexed by the Cost Map version tags. That is, if tags are "1", "2", etc, then the change log for version "1" has the changes from "1" to "2", the change log for version "2" has the changes from "2" to "3", etc. When these logs take up too much space, the server deletes the oldest change logs. When a client requests an incremental update, the server finds the change log for the client's tag, and returns all cost updates in that log and all subsequent logs. If the server cannot find the client's tag in the change log table, the server returns an "invalid field" error code, and the client must retrieve the full Cost Map to get the updated costs. This covers the error cases of the tag being totally invalid as well as being too old.

We divide network map updates into two categories. Minor updates move some prefixes from one PID to another, perhaps to reflect temporary rerouting, but do not change the PID names. Major updates change PID names, add or delete PIDs, etc.

An ALTO Server can handle minor updates by keeping change logs with the prefixes for the updated PIDs, as described above for cost maps. When a client requests an incremental update, logically concatenate the logs from the client's tag to the current version.

For major network map changes, the server could just refuse to provide incremental updates. That is, when there is a major network map change, the server would simply discard all the old change logs.

Finally, note that the Incremental Network Map Update Service is independent of the Incremental Cost Map Update Service. An ALTO Server may choose to provide Incremental Cost Map Updates without providing Incremental Network Map Updates.

6. Alternatives

This section presents several alternative approaches, and explains why we do not think they are appropriate.

6.1. HTTP Conditional Retrieval

The HTTP Protocol ([RFC2616]) defines several conditional-retrieval mechanisms, such as the If-Modified-Since and If-None-Match headers. These allow a client to retrieve a new version of a map only if the resource has changed since the client's last access.

However, these mechanisms do not allow incremental update. If only a few costs changed, the server would still have to send the entire map. Because we expect that parts of the maps will change frequently, we do not think these approaches are satisfactory.

6.2. JSON Patch

A more promising alternative is JSON Patch ([RFC6902]). This is a standardized method of describing the changes between two versions of a JSON data structure. As such, it is ideally suited for incremental update. When a client requests an incremental update from the server, the server would return a JSON Patch description of the changes. Presumably JSON libraries will provide procedures to apply a patch to an previously retrieved JSON data structure, and to create a patch describing the differences between two versions of a JSON data structure. Clients can use the former methods to apply patches, and servers can use the latter to create them, so little additional programming is required.

Despite those advantages, we do not believe JSON Patch is a good solution for incremental update for ALTO. First, note that JSON Patch does not solve the "what version?" problem. We still need to assign version tags to cost maps, and we would still need new services similar to our Incremental Network and Cost Map Update Services. The difference would be that the body of the responses would have JSON Patch data instead of the Network and Cost Map structures.

Second, note that the Network and Cost Map response messages defined in [RFC7285] are, for all practical purposes, "patch" structures. All that is needed is the semantics that they represent changes to an existing map, rather than a completely new map. It is true that JSON Patch can represent a wider class of changes, but it is not clear that power is necessary for the incremental changes that an ALTO Server will make.

Next, JSON Patch is less efficient than our proposal. For example, suppose the cost for SRC-PID to DEST-PID changes to 123. Our proposal represents that as:

```
     {"SRC-PID": {"DEST-PID": 123}}
```

JSON Patch represents that change as:

```
     {"replace": "cost-map.SRC-PID.DEST-PID", "value": 123}
```

Finally, we have serious doubts as to whether JSON Patch can handle maps of the size we expect. To see the problem, realize that incremental updates are only important for large maps. For small maps, a client can just retrieve the full version.

For a client to take advantage of an "apply patch" method in a JSON library, the client would almost certainly have to store the Cost Map using a Document Object Model (DOM) representation provided by that library. A DOM representation of a Cost Map with (say) 1,000 PIDs requires 1,000 associative tables, each of which has 1,000 entries. That takes a considerable amount of space.

There are far more efficient ways to represent an ALTO Cost Map. For example, an implicit assumption is that costs change more frequently than network maps. So a client can sort the PID names, assign them numbers from 0 to N-1, and then store the costs in (possibly sparse) numerically-indexed NxN matrix instead of a string-based lookup table. Furthermore, a general JSON library would store numerical values as double precision. It is difficult to believe that any ALTO Server can provide costs that are accurate enough to require double precision. A single precision, numerically-indexed matrix is much smaller than a double precision string-indexed DOM representation, and can be searched much faster.

Therefore if we used JSON Patch, a client might be forced to use a very inefficient representation of a Cost Map.

JSON Patch causes similar problems for an ALTO Server. To take full advantage of JSON Patch, a server would have to present two DOM versions of the Cost Map to a "calculate patch" method. Those representations would take a lot of space. Furthermore, calculating the difference between two DOMs of that size will tax most computers. And finally, as we outlined above, we expect the ALTO Server will know the difference anyway.

To summarize, we believe that for ALTO incremental update, JSON Patch is an overly general approach that would be far too expensive to use for networks with a large number of PIDs.

6.3. Persistant HTTP Connection

Another alternative is for a client to create a persistant HTTP connection (e.g., "Keep-Alive") to the ALTO Server's Filtered Cost Map Service, and send repeated search requests on that connection. This isn't an incremental update service as such. But it avoids the overhead of setting up a TCP connection for each request, and hence allows a client to query the ALTO server more efficiently.

6.4. Web Sockets

Web Sockets [RFC6455] are an alternative to the client-pull model. Web Sockets are a standard mechanism to establish a persistent bi-directional stream of messages between a client and a server. Web Sockets are not HTTP, but the initial message looks enough like HTTP that Web Socket aware server can upgrade the connection to a Web Socket stream, while older web servers will just recognize the HTTP and will return a default page.

While there are several ways to use Web Sockets for incremental update in ALTO, the simplest would be to define a "Continous Update Service". A client would use this service instead of the Full Network Map and Full Cost Map Services. A client would establish a Web Socket connection. The server would immediately respond with a full network map, followed a full cost map. After that initial setup, the server would continue to send cost map and network map changes as they become available.

This has the advantage of providing almost immediate updates to clients, and it removes the need for version tags on cost maps. But it has the disadvantage of being a different protocol. Both the client and server must support Web Sockets. That shouldn't be a problem for most ALTO Servers. But ALTO Clients are likely to be in small, embedded systems, and might have very minimal HTTP support (Web Sockets were originally intended for browser-based applications like stock tickers and interactive games). Web Sockets also require cooperation from any proxy servers along the way. And finally, Web Sockets require maintaining a persistant connection between the client and server, as well as through any proxy server along the way, which could lead to scaling problems.

7. IANA Considerations

The Incremental Update service as proposed introduces a new MIME types "application/alto-vtag+json" and "application/alto-networkmapupdate+json", which need to be registered.

8. Security Considerations

This extension does not introduce any security issues that are not present in the base ALTO protocol.

9. Conclusion

This document describes different options that can be applied to support incremental updates of ALTO Network and Cost maps. In particular it comprises option for client and server to synchronize themselves about their current map state, and further includes options on how to encode partial updates. Finally it proposes an new incremental update service and evaluates different options numerically.

10. References

[I-D.jenkins-alto-cdn-use-cases]
Niven-Jenkins, B., Watson, G., Bitar, N., Medved, J., and S. Previdi, "Use Cases for ALTO within CDNs", draft-jenkins-alto-cdn-use-cases-01 (work in progress), June 2011.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Burners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC4632] Fuller, V. and T. Li, ""Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan", RFC 4632, BCP 122, August 2006.

[RFC5693] Seedorf, J. and E. Burger, "Application-Layer Traffic Optimization (ALTO) Problem Statement", RFC 5693, October 2009.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

[RFC6902] Bryan, P. and M. Nottingham, "JavaScript Object Notation (JSON) Patch", RFC 6902, April 2013.

[RFC7285] Alimi, R., Penno, R., and Y. Yang, "Application-Layer Traffic Optimization (ALTO) Protocol", draft-ietf-alto-protocol-27 (work in progress), June 2014.

Appendix A. Acknowledgments

The authors would like to thank Vijay Gurbani for his valuable input and excellent feedback to this document.

Nico Schwan is partially supported by the ENVISION project (http://www.envision-project.org), a research project supported by the European Commission under its 7th Framework Program (contract no. 248565). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ENVISION project or the European Commission.

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

Nico Schwan
Thales Deutschland
Lorenzstrasse 10
Stuttgart 70435
Germany

Email: nico.schwan@thalesgroup.com