Abstract

The Content Delivery Networks Interconnection (CDNI) framework [RFC6707] defines a set of protocols to interconnect CDNs, to achieve multiple goals such as extending the reach of a given CDN to areas that are not covered by that particular CDN. One component that is needed to achieve the goal of CDNI described in [RFC7336] is the CDNI Request Routing Footprint & Capabilities Advertisement interface (FCI). [RFC8008] defines precisely the semantics of FCI and provides guidelines on the FCI protocol, but the exact protocol is explicitly outside the scope of that document. In this document, we follow the guidelines to define an FCI protocol using the Application-Layer Traffic Optimization (ALTO) protocol.
1. Introduction

The ability to interconnect multiple content delivery networks (CDNs) has many benefits, including increased coverage, capability, and reliability. The Content Delivery Networks Interconnection (CDNI) framework [RFC6707] defines four interfaces to achieve interconnection of CDNs: (1) the CDNI Request Routing Interface; (2) the CDNI Metadata Interface; (3) the CDNI Logging Interface; and (4) the CDNI Control Interface.

Among the four interfaces, the CDNI Request Routing Interface provides key functions, as specified in [RFC6707]: "The CDNI Request Routing interface enables a Request Routing function in an Upstream CDN to query a Request Routing function in a Downstream CDN to determine if the Downstream CDN is able (and willing) to accept the delegated Content Request. It also allows the Downstream CDN to control what should be returned to the User Agent in the redirection message by the upstream Request Routing function." On a high level, the scope of the CDNI Request Routing Interface, therefore, contains two main tasks: (1) determining if the downstream CDN (dCDN) is willing to accept a delegated content request; (2) redirecting the
content request coming from an upstream CDN (uCDN) to the proper entry point or entity in the downstream CDN.

Correspondingly, the request routing interface is broadly divided into two functionalities: (1) CDNI Footprint & Capabilities Advertisement interface (FCI); (2) CDNI Request Routing Redirection interface (RI). Since this document focuses on the first functionality, CDNI FCI, we will describe it in a more detailed way. CDNI FCI is an advertisement from a dCDN to a uCDN (push) or a query from a uCDN to a dCDN (pull) so that the uCDN knows whether it can redirect a particular user request to that dCDN.

A key component in defining CDNI FCI is defining objects describing the footprints and capabilities of a dCDN. Such objects are already in [RFC8008]. A protocol to transport and update such objects between a uCDN and a dCDN, however, is not defined. Hence, the scope of this document is to define such a protocol by introducing a new Application-Layer Traffic Optimization (ALTO) [RFC7285] service called "CDNI FCI Map Service".

There are multiple benefits in using ALTO as a transport protocol, as we discuss in Section 2.2.

The rest of this document is organized as follows. Section 2 provides non-normative background on both CDNI FCI and ALTO. Section 3 introduces the most basic service, called CDNI FCI Map, to realize CDNI FCI using ALTO. Section 4 demonstrates a key benefit of using ALTO: the ability to integrate CDNI FCI with ALTO network maps. Such integration provides a new granularity to describe footprints. Section 5 builds on filtered ALTO maps to introduce filtered CDNI FCI maps using capabilities so that a uCDN can get footprints with given capabilities instead of getting the full map which can be huge. Section 6 further shows a benefit of using ALTO: the ability to query footprint properties using ALTO unified properties. In this way, a uCDN can effectively fetch capabilities of some footprints in which it is interested. IANA and security considerations are discussed in Section 8 and Section 9 respectively.

Throughout this document, we use the terminology for CDNI defined in [RFC6707], [RFC8006], [RFC8008] and we use the terminology for ALTO defined in [RFC7285], [I-D.ietf-alto-unified-props-new].

2. Background

The design of CDNI FCI transport using ALTO depends on the understanding of both FCI semantics and ALTO. Hence, we start with a review of both.
2.1. Semantics of FCI Advertisement

The CDNI document on "Footprint and Capabilities Semantics" [RFC8008] defines the semantics of CDNI FCI, and provides guidance on what Footprint and Capabilities mean in a CDNI context and how a protocol solution should in principle look like. The definitions in [RFC8008] depend on [RFC8006]. Here we briefly summarize key related points of [RFC8008] and [RFC8006]. For a detailed discussion, the reader is referred to the RFCs.

- Footprint and capabilities are tied together and cannot be interpreted independently from each other. Hence, capabilities must be expressed on a per footprint basis. [RFC8008] integrates footprint and capabilities with an approach of "capabilities with footprint restrictions".

- Given that a large part of Footprint and Capabilities Advertisement will actually happen in contractual agreements, the semantics of CDNI Footprint and Capabilities advertisement refers to answering the following question: what exactly still needs to be advertised by the CDNI FCI? For instance, updates about temporal failures of part of a footprint can be useful information to convey via the CDNI request routing interface. Such information would provide updates on information previously agreed in contracts between the participating CDNs. In other words, the CDNI FCI is a means for a dCDN to provide changes/updates regarding a footprint and/or capabilities that it has prior agreed to serve in a contract with a uCDN. Hence, server push and incremental encoding will be necessary techniques.

- Multiple types of footprints (ipv4cidr, ipv6cidr, asn and countrycode) are defined in [RFC8006].

- A "Set of IP-prefixes" can contain both full IP addresses (i.e., a /32 for IPv4 and a /128 for IPv6) and IP prefixes with an arbitrary prefix length. There must also be support for multiple IP address versions, i.e., IPv4 and IPv6, in such a footprint.

- For all of these mandatory-to-implement footprint types, footprints can be viewed as constraints for delegating requests to a dCDN: A dCDN footprint advertisement tells the uCDN the limitations for delegating a request to the dCDN. For IP prefixes or ASN(s), the footprint signals to the uCDN that it should consider the dCDN a candidate only if the IP address of the request routing source falls within the prefix set (or ASN, respectively). The CDNI specifications do not define how a given uCDN determines what address ranges are in a particular ASN. Similarly, for country codes, a uCDN should only consider the dCDN
a candidate if it covers the country of the request routing source. The CDNI specifications do not define how a given uCDN determines the country of the request routing source. Multiple footprint constraints are additive, i.e., the advertisement of different types of footprint narrows the dCDN candidacy cumulatively.

- The following capabilities are defined as "base" capabilities; that is, they are required in all cases and therefore constitute mandatory capabilities to be supported by the CDNI FCI: (1) Delivery Protocol; (2) Acquisition Protocol; (3) Redirection Mode; (4) Capabilities related to CDNI Logging; (5) Capabilities related to CDNI Metadata.

2.2. ALTO Background and Benefits

Application-Layer Traffic Optimization (ALTO) [RFC7285] is an approach for guiding the resource provider selection process in distributed applications that can choose among several candidate resources providers to retrieve a given resource. By conveying network layer (topology) information, an ALTO server can provide important information to "guide" the resource provider selection process in distributed applications. Usually, it is assumed that an ALTO server conveys information that these applications cannot or have difficulty to measure themselves [RFC5693].

Originally, ALTO was motivated by optimizing cross-ISP traffic generated by P2P applications [RFC5693]. Recently, however, ALTO is also being considered for improving the request routing in CDNs [I-D.jenkins-alto-cdn-use-cases]. The CDNI problem statement explicitly mentions ALTO as a candidate protocol for "actual algorithms for selection of CDN or Surrogate by Request-Routing systems" [RFC6707].

The following reasons make ALTO a suitable candidate protocol for downstream CDN selection as part of CDNI request routing and in particular for an FCI protocol:

- ALTO is a protocol specifically designed to improve application layer traffic (and application layer connections among hosts on the Internet) by providing additional information to applications that these applications could not easily retrieve themselves. For CDNI, this is exactly the case: a uCDN wants to improve application layer CDN request routing by using dedicated information (provided by a dCDN) that the uCDN could not easily obtain otherwise. ALTO can help a uCDN to select a proper dCDN by first providing dCDNs’ capabilities as well as footprints (see
Section 3) and then providing costs of surrogates in a dCDN by ALTO cost maps.

- The semantics of an ALTO network map is an exact match for the needed information to convey a footprint by a downstream CDN, in particular if such a footprint is being expressed by IP-prefix ranges. Please see Section 4.

- Security: Identifications between uCDNs and dCDNs are extremely important. ALTO maps can be signed and hence provide inherent integrity protection. Please see Section 9.

- RESTful-Design: The ALTO protocol has undergone extensive revisions in order to provide a RESTful design regarding the client-server interaction specified by the protocol. A CDNI FCI interface based on ALTO would inherit this RESTful design. Please see Section 3.

- Error-handling: The ALTO protocol has undergone extensive revisions in order to provide sophisticated error-handling, in particular regarding unexpected cases. A CDNI FCI interface based on ALTO would inherit this thought-through and mature error-handling. Please see Section 5.

- Filtered map service: The ALTO map filtering service would allow a uCDN to query only for parts of an ALTO map. For example, filtered unified property map service can enable a uCDN to query properties of a part of footprints in an effective way (see Section 6).

- Server-initiated Notifications and Incremental Updates: When the footprint or the capabilities of a downstream CDN change (i.e., unexpectedly from the perspective of an upstream CDN), server-initiated notifications would enable a dCDN to directly inform an upstream CDN about such changes. Consider the case where - due to failure - part of the footprint of the dCDN is not functioning, i.e., the CDN cannot serve content to such clients with reasonable QoS. Without server-initiated notifications, the uCDN might still use a very recent network and cost map from dCDN, and therefore redirect requests to dCDN which it cannot serve. Similarly, the possibility for incremental updates would enable efficient conveyance of the aforementioned (or similar) status changes by the dCDN to the uCDN. The newest design of ALTO supports server pushed incremental updates [I-D.ietf-alto-incr-update-sse].

- Content Availability on Hosts: A dCDN might want to express CDN capabilities in terms of certain content types (e.g., codecs/formats, or content from certain content providers). The new
endpoint property for ALTO would enable a dCDN to make such information available to an upstream CDN. This would enable a uCDN to determine if a given dCDN actually has the capabilities for a given request with respect to the type of content requested.

- Resource Availability on Hosts or Links: The capabilities on links (e.g. maximum bandwidth) or caches (e.g. average load) might be useful information for an upstream CDN for optimized downstream CDN selection. For instance, if a uCDN receives a streaming request for content with a certain bitrate, it needs to know if it is likely that a dCDN can fulfill such stringent application-level requirements (i.e., can be expected to have enough consistent bandwidth) before it redirects the request. In general, if ALTO could convey such information via new endpoint properties, it would enable more sophisticated means for downstream CDN selection with ALTO. ALTO Path Vector Extension [I-D.ietf-alto-path-vector] is designed to allow ALTO clients to query information such as capacity regions for a given set of flows.

3. CDNI FCI Map

The ALTO protocol is based on an ALTO Information Service Framework which consists of several services, where all ALTO services are "provided through a common transport protocol, messaging structure and encoding, and transaction model" [RFC7285]. The ALTO protocol specification [RFC7285] defines several such services, e.g. the ALTO map service.

This document defines a new ALTO Map Service called "CDNI FCI Map Service" which conveys JSON objects of media type "application/alto-cdni-fcimap+json". These JSON objects are used to transport BaseAdvertisementObject objects defined in [RFC8008]; this document specifies how to transport such BaseAdvertisementObject objects via the ALTO protocol with the ALTO "CDNI FCI Map Service". Given that the "CDNI FCI Map Service" is very similar in structure to the two already defined map services (network maps and cost maps), the specification of CDNI FCI Map below uses the same specification structure for Cost Map specification in Section 11.2.3 of [RFC7285] when specifying cost maps.

3.1. Media Type

The media type of the CDNI FCI Map is "application/alto-cdni-fcimap+json".
3.2. HTTP Method

A CDNI FCI map resource is requested using the HTTP GET method.

3.3. Accept Input Parameters

None.

3.4. Capabilities

None.

3.5. Uses

The resource ID of the resource based on which the CDNI FCI map will be defined. For example, if a CDNI FCI map depends on a network map, the resource ID of the network map MUST be included in "uses" field.

Please see Section 4 for details. If the CDNI FCI map does not depend on any other resources, "uses" field MUST NOT appear.

3.6. Response

The "meta" field of a CDNI FCI map response MUST include the "vtag" field defined in Section 10.3 of [RFC7285]. This field provides the version of the retrieved CDNI FCI map.

If a CDNI FCI map response depends on a resource such as a network map, it MUST include the "dependent-vtags" field, whose value is an array to indicate the version tags of the resources used, where each resource is specified in "uses" of the IRD. The current defined dependent resource is only network map, and the usage of it is described in Section 4.

The data component of an ALTO CDNI FCI map response is named "cdni-fci-map", which is a JSON object of type CDNIFCIMapData:

```json
object {
  CDNIFCIMapData cdni-fci-map;
} InfoResourceCDNIFCIMap : ResponseEntityBase;

object {
  BaseAdvertisementObject capabilities<1..*>;
} CDNIFCIMapData;
```

Specifically, a CDNIFCIMapData object is a JSON object that includes only one property named "capabilities", whose value is an array of BaseAdvertisementObject objects.
The syntax and semantics of BaseAdvertisementObject are well defined in Section 5.1 of [RFC8008]. A BaseAdvertisementObject object includes multiple properties, including capability-type, capability-value and footprints, where footprints are defined in Section 4.2.2.2 of [RFC8006].

To be self-contained, we give a non-normative specification of BaseAdvertisementObject below. As mentioned above, the normative specification of BaseAdvertisementObject is in [RFC8008].

```json
object {
    JSONString capability-type;
    JSONValue capability-value;
    Footprint footprints<0..*>;
} BaseAdvertisementObject;

object {
    JSONString footprint-type;
    JSONString footprint-value<1..*>;
} Footprint;
```

For each BaseAdvertisementObject, the ALTO client MUST interpret footprints appearing multiple times as if they appeared only once. If footprints in a BaseAdvertisementObject is null or empty or not appearing, the ALTO client MUST understand that the capabilities in this BaseAdvertisementObject have the "global" coverage.

Note: Further optimization of BaseAdvertisement objects to effectively provide the advertisement of capabilities with footprint restrictions is certainly possible. For example, these two examples below both describe that the dCDN can provide capabilities ["http/1.1", "https/1.1"] for the same footprints. However, the latter one is smaller in its size.

**EXAMPLE 1**
```json
{  
    "meta": {...},  
    "cdni-fci-map": {  
        "capabilities": [  
            {  
                "capability-type": "FCI.DeliveryProtocol",  
                "capability-value": {  
                    "delivery-protocols": [  
                        "http/1.1"  
                    ]  
                }  
            },  
            "footprints": [  
                <Footprint objects>  
            ]  
        ]  
    }  
}  
```
EXAMPLE 2

Since such optimizations are not necessary for the basic interconnection of CDNs, the specifics of such mechanisms are outside the scope of this document.

3.7. Examples
3.7.1. IRD Example

Below is the information resource directory (IRD) of a simple, example ALTO server. The server provides both base ALTO information resources (e.g., network maps) and CDNI FCI information resources (e.g., CDNI FCI map), demonstrating a single, integrated environment.

Specifically, the IRD announces two network maps, one CDNI FCI map without dependency, one CDNI FCI map depending on a network map, one filtered CDNI FCI map to be defined in Section 5, one unified property map including "cdni-fci-capabilities" as its entities' property, one filtered unified property map including "cdni-fci-capabilities" and "pid" as its entities' properties, and two update stream services (one for updating CDNI FCI maps, and the other for updating property maps).

```
GET /directory HTTP/1.1
Host: alto.example.com
Accept: application/alto-directory+json,application/alto-error+json

{
  "meta": { ... },
  "resources": {
    "my-default-network-map": {
      "uri": "http://alto.example.com/networkmap",
      "media-type": "application/alto-networkmap+json"
    },
    "my-eu-netmap": {
      "uri": "http://alto.example.com/myeunetmap",
      "media-type": "application/alto-networkmap+json"
    },
    "my-default-cdnifci-map": {
      "uri": "http://alto.example.com/cdnifcimap",
      "media-type": "application/alto-cdnifcimap+json"
    },
    "my-filtered-cdnifci-map": {
      "uri": "http://alto.example.com/cdnifcimap/filtered",
      "media-type": "application/alto-cdnifcimap+json",
      "accepts": "application/alto-cdnifcimapfilter+json",
      "uses": [ "my-default-cdnifci-map" ]
    },
    "my-cdnifci-map-with-network-map-footprints": {
      "uri": "http://alto.example.com/networkcdnifcimap",
      "media-type": "application/alto-cdnifcimap+json",
      "uses": [ "my-eu-netmap" ]
    },
    "cdnifci-property-map": {
      "uri": "http://alto.example.com/propmap/full/cdnifci",
```
"media-type": "application/alto-propmap+json",
"capabilities": {
  "domain-types": ["ipv4", "ipv6", "countrycode", "asn"],
  "prop-types": ["cdni-fci-capabilities"]
},
"filtered-cdnifci-property-map": {
  "uri": "http://alto.example.com/propmap/lookup/cdnifci-pid",
  "media-type": "application/alto-propmap+json",
  "accepts": "application/alto-propmapparams+json",
  "capabilities": {
    "domain-types": ["ipv4", "ipv6", "countrycode", "asn"],
    "prop-types": ["cdni-fci-capabilities", "pid"]
  }
},
"update-my-cdnifci-maps": {
  "uri": "http://alto.example.com/updates/cdnifcimaps",
  "media-type": "text/event-stream",
  "accepts": "application/alto-updatestreamparams+json",
  "uses": [
    "my-default-network-map",
    "my-eu-netmap",
    "my-default-cdnifci-map",
    "my-filtered-cdnifci-map",
    "my-cdnifci-map-with-network-map-footprints"
  ],
  "capabilities": {
    "incremental-change-media-types": {
      "my-default-network-map": "application/json-patch+json",
      "my-eu-netmap": "application/json-patch+json",
      "my-default-cdnifci-map": "application/merge-patch+json,application/json-patch+json",
      "my-filtered-cdnifci-map": "application/merge-patch+json,application/json-patch+json",
      "my-cdnifci-map-with-network-map-footprints": "application/merge-patch+json,application/json-patch+json"
    }
  }
},
"update-my-props": {
  "uri": "http://alto.example.com/updates/properties",
  "media-type": "text/event-stream",
  "uses": [
    "cdnifci-property-map",
    "filtered-cdnifci-property-map"
  ],
  "capabilities": {
    "incremental-change-media-types": {
      "my-default-network-map": "application/json-patch+json",
      "my-eu-netmap": "application/json-patch+json",
      "my-default-cdnifci-map": "application/merge-patch+json,application/json-patch+json",
      "my-filtered-cdnifci-map": "application/merge-patch+json,application/json-patch+json",
      "my-cdnifci-map-with-network-map-footprints": "application/merge-patch+json,application/json-patch+json"
    }
  }
}
3.7.2. Basic Example

In this example, we demonstrate a simple CDNI FCI map; this map does not depend on other resources. There are three BaseAdvertisementObjects in this map and these objects’ capabilities are http/1.1 delivery protocol, [http/1.1, https/1.1] delivery protocol and https/1.1 acquisition protocol respectively.

GET /cdnifcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-cdnifcimap+json
Accept: application/alto-error+json

HTTP/1.1 200 OK
Content-Length: XXX
Content-Type: application/alto-cdnifcimap+json

{ "meta": { "vtag": { "resource-id": "my-default-cdnifci-map", "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785" } }, "cdni-fci-map": { "capabilities": [ { "capability-type": "FCI.DeliveryProtocol", "capability-value": { "delivery-protocols": [ "http/1.1" ] } }, { "footprints": [ <Footprint objects> ] }, { "capability-type": "FCI.DeliveryProtocol", } ] } }
3.7.3. Incremental Updates Example

A benefit of using ALTO to provide CDNI FCI maps is that such maps can be updated using ALTO incremental updates. Below is an example that also shows the benefit of having both JSON merge patch and JSON patch to encode updates.

At first, an ALTO client requests the ALTO server updates for "my-default-cdnifcimap", and the ALTO server returns the "control-uri" followed by the full CDNI FCI map. Then when there is a huge change in footprint objects in delivery-protocol http/1.1, the ALTO server uses JSON merge patch to encode the change and sends it to the ALTO client. Later on, the ALTO server notifies the ALTO client that "ipv4:192.0.2.0/24" is added into the footprints in delivery-protocol http/1.1 by sending the change encoded by JSON patch to the ALTO client.

```
POST /updates/cdnifcimaps HTTP/1.1
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###
```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/event-stream

event: application/alto-updatestreamcontrol+json
data: {"control-uri": "http://alto.example.com/updates/streams/3141592653589"}

event: application/alto-cdnifcimap+json,my-default-cdnifci-map
data: { ... full CDN FCI map ... }

event: application/merge-patch+json,my-default-cdnifci-map
data: {
data:  "meta": {
data:   "vtag": {data:     "tag": "dasdfal0ce8b059740bdddadasd8eb1d47853716"
data:   }
data:  },
data:  "capability-type": "FCI.DeliveryProtocol",
data:  "capability-value": {
data:   "delivery-protocols": [
data:     "http/1.1"
data:   ],
data:   "footprints": [
data:     <Footprint objects that are different from footprint objects in delivery-protocols http/1.1>
data:   ]
data:  }
data: }

event: application/json-patch+json,my-default-cdnifci-map
data: {
data:  {"op": "replace",
data:   "path": "/meta/vtag/tag",
data:   "value": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
data:  },
data:  {"op": "add",
data:   "path": "/cdnif-cmi-capabilities/0/footprints/-",
data:   "value": "ipv4:192.0.2.0/24"}
4. CDNI FCI Map using ALTO Network Map

4.1. Network Map Footprint Type: altonetworkmap

In addition to the already defined CDNI footprint types (e.g., ipv4cidr, ipv6cidr, asn, countrycode), ALTO network maps can be a type of FCI footprint.

Specifically, CDNI footprints using ALTO network maps should use a new CDNI Footprint Type called "altonetworkmap".

"altonetworkmap" footprint type indicates that the corresponding footprint value is a list of PIDNames as defined in [RFC7285]. These PIDNames are references of PIDs in a network map resource. Hence a CDNI FCI map with "altonetworkmap" footprints depends on a network map. For such a CDNI FCI map, the "dependent-vtag" field with a reference to a network map it depends on MUST be included in it (see the example in Section 4.2.3).

4.2. Examples

4.2.1. IRD Example

We use the same IRD example given in Section 3.7.1.

4.2.2. ALTO Network Map for CDNI FCI Footprints Example

Below is an example network map whose resource id is "my-eu-netmap", and this map is referenced by the CDNI FCI map example in Section 4.2.3.

GET /networkmap HTTP/1.1
Host: http://alto.example.com/myeunetmap
Accept: application/alto-networkmap+json, application/alto-error+json
HTTP/1.1 200 OK
Content-Length: XXX
Content-Type: application/alto-networkmap+json

{
  "meta": {
    "vtag": [
      {
        "resource-id": "my-eu-netmap",
        "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"
      }
    ]
  },
  "network-map": {
    "south-france": {
      "ipv4": ["192.0.2.0/24", "198.51.100.0/25"]
    },
    "germany": {
      "ipv4": ["192.0.3.0/24"]
    }
  }
}

4.2.3. ALTO Network Map Footprints in CDNI FCI Map

In this example, we show a CDNI FCI map that depends on a network map described in Section 4.2.2.

GET /networkcdnifcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-cdnifcimap+json, application/alto-error+json
HTTP/1.1 200 OK
Content-Length: 618
Content-Type: application/alto-cdnifcimap+json

{
    "meta": {
        "dependent-vtags": [
            {
                "resource-id": "my-eu-netmap",
                "tag": "3ee2cbe8d63d9fab71b9b34cbbf764436315542e"
            }
        ]
    },
    "cdni-fci-map": {
        "capabilities": [
            {
                "capability-type": "FCI.DeliveryProtocol",
                "capability-value": [
                    "http/1.1"
                ]
            },
            {
                "capability-type": "FCI.DeliveryProtocol",
                "capability-value": [
                    "https/1.1"
                ],
                "footprints": [
                    {
                        "footprint-type": "altonetworkmap",
                        "footprint-value": [
                            "germany",
                            "south-france"
                        ]
                    }
                ]
            }
        ]
    }
}

4.2.4. Incremental Updates Example

In this example, the ALTO client is interested in changes of "my-cdnifci-map-with-network-map-footprints". Considering two changes, the first one is to change footprints of http/1.1 Delivery Protocol capability, and the second one is to remove "south-france" from the footprints of https/1.1 delivery protocol capability.

POST /updates/cdnifcimaps HTTP/1.1
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

```json
{ "add": {
   "my-network-map-cdnifci-stream": {
      "resource-id": "my-cdnifci-map-with-network-map-footprints"
   }
}
```

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

```json
event: application/alto-updatestreamcontrol+json
data: {"control-uri":
data: "http://alto.example.com/updates/streams/3141592653590"
}
event: application/alto-cdnifcimap+json,my-fci-stream
data: { ... full CDNI FCI map ... }
event: application/merge-patch+json,my-fci-stream
data: {
data: "meta": {
data: "dependent-vtags": [
data:  
data:  "resource-id": "my-eu-netmap",
data:  "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"
data:  },
data: },
data: "vtag": {
data:  "tag": "dasdfa10ce8b059740bd1b3d4d81e3716"
data:  },
data: 

data: "capability-type": "FCI.DeliveryProtocol",
data: "capability-value": {
data:  "delivery-protocols": [
data:  "http/1.1"
data:  ],
data:  "footprints": [
data:   <Footprint objects that are different from

data:    footprint objects in delivery-protocols http/1.1>
data:  ]
data: }
data: }
```
event: application/json-patch+json,my-fci-stream

5. Filtered CDNI FCI Map using Capabilities

Section 3 and Section 4 describe CDNI FCI Map Service which can be used to enable a uCDN to get capabilities with footprints constrains from dCDNs. However, always getting full CDNI FCI maps from dCDNs is very inefficient, hence we introduce a new service named "Filtered CDNI FCI Map Service" to allow a client to filter a CDNI FCI map using a client-given set of capabilities. For each entry of the CDNI FCI map, only if the entry contains at least one of the client-given capabilities will it be returned to the client. The relationship between a filtered CDNI FCI map and a CDNI FCI map is similar to the relationship between a filtered network/cost map and a network/cost map.

5.1. Media Type

Since a filtered CDNI FCI map is still a CDNI FCI map, it uses the media type defined for CDNI FCI maps in Section 3.1.

5.2. HTTP Method

A filtered CDNI FCI map is requested using the HTTP POST method.

5.3. Accept Input Parameters

The input parameters for a filtered CDNI FCI map are supplied in the entity body of the POST request. This document specifies the input parameters with a data format indicated by the media type "application/alto-cdnifcifilter+json" which is a JSON object of type ReqFilteredCDNIFCIMap, where:
object {
    JSONString capability-type;
    JSONValue capability-value;
} CDNIFCICapability;

object {
    [CDNIFCICapability cdni-fci-capabilities<0..*>;]
} ReqFilteredCDNIFCIMap;

with fields:

capability-type: The same as Base Advertisement Object’s capability-type defined in Section 5.1 of [RFC8008].

capability-value: The same as Base Advertisement Object’s capability-value defined in Section 5.1 of [RFC8008].

cdni-fci-capabilities: A list of CDNI FCI capabilities defined in Section 5.1 of [RFC8008] for which footprints are to be returned. If a list is empty or not appearing, the ALTO server MUST interpret it as a request for the full CDNI FCI Map. The ALTO server MUST interpret entries appearing in a list multiple times as if they appeared only once. If a "capability-type" or a "capability-value" is not defined, the ALTO server MUST ignore this capability. If there is only one capability in the list and its "capability-type" or "capability-value" is not defined, the ALTO server MUST return nothing.

5.4. Capabilities

None.

5.5. Uses

The resource ID of the CDNI FCI map based on which the filtering is performed.

5.6. Response

The response MUST indicate an error, using ALTO protocol error handling specified in Section 8.5 of the ALTO protocol [RFC7285], if the request is invalid.

Specifically, a filtered CDNI FCI map request can be invalid as follows:

- The value of "capability-type" is null;
The value of "capability-value" is null;

- The value of "capability-value" is inconsistent with "capability-type".

When the request is invalid, the ALTO server MUST return an "E_INVALID_FIELD_VALUE" error defined in Section 8.5.2 of [RFC7285], and the "value" field of the error message SHOULD indicate this CDNI FCI capability.

The ALTO server return a filtered CDNI FCI map for a valid request. The format of a filtered CDNI FCI map is the same as an unfiltered CDNI FCI map. See Section 3.6 for the format.

The returned CDNI FCI map MUST contain only BaseAdvertisementObject objects whose CDNI capability object is the superset of one of CDNI capability object in "cdni-fci-capabilities". Specifically, that a CDNI capability object A is the superset of another CDNI capability object B means that these two CDNI capability objects have the same capability type and mandatory properties in capability value of A MUST include mandatory properties in capability value of B semantically. See Section 5.7.2 for a concrete example.

The version tag included in the "vtag" field of the response MUST correspond to the full CDNI FCI map resource from which the filtered CDNI FCI map is provided. This ensures that a single, canonical version tag is used independently of any filtering that is requested by an ALTO client.

5.7. Examples

5.7.1. IRD Example

We use the same IRD example by Section 3.7.1.

5.7.2. Basic Example

This example filters the full CDNI FCI map in Section 3.7.2 by selecting only http/1.1 delivery protocol capability. Only the first two BaseAdvertisementObjects in the full map will be returned because the first object’s capability is http/1.1 delivery protocol and the second object’s capability is http/1.1 and https/1.1 delivery protocols which is the superset of http/1.1 delivery protocol.

```
POST /cdnifcimap/filtered HTTP/1.1
HOST: alto.example.com
Content-Type: application/cdnifilter+json
Accept: application/alto-cdnifcimap+json
```
HTTP/1.1 200 OK
Content-Length: XXX
Content-Type: application/alto-cdnifcimap+json

{
   "meta": {
      "vtag": {
         "resource-id": "my-default-cdnifci-map",
         "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
      }
   },
   "cdni-fci-map": {
      "capabilities": [
      {
         "capability-type": "FCI.DeliveryProtocol",
         "capability-value": {
            "delivery-protocols": [
               "http/1.1"
            ]
         },
         "footprints": [
            <Footprint objects>
         ]
      },
      {
         "capability-type": "FCI.DeliveryProtocol",
         "capability-value": {
            "delivery-protocols": [
               "https/1.1",
               "http/1.1"
            ]
         },
         "footprints": [
            <Footprint objects>
         ]
      }
   }
}

5.7.3. Incremental Updates Example

In this example, the ALTO client only cares about the updates of one Delivery Protocol object whose value is "http/1.1". So it adds its limitation of capabilities in "input" field of the POST request.

```
POST /updates/cdnifcimaps HTTP/1.1
Host: fcialtoupdate.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

{ "add": {
    "my-fci-stream": {
      "resource-id": "my-filtered-cdnifci-map",
      "input": {
        "cdnifci-capabilities": [
          { "capability-type": "FCI.DeliveryProtocol",
            "capability-value": {
              "delivery-protocols": [ "http/1.1"
            ]
          }
        ]
      }
    }
  }
}
```

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

```
event: application/alto-updatestreamcontrol+json
data: {"control-uri":
data: "http://alto.example.com/updates/streams/3141592653590"}
event: application/alto-cdnifcimap+json,my-fci-stream
data: { ... full filtered CDNI FCI map ... }
event: application/merge-patch+json,my-fci-stream
data: {
```
6. Query Footprint Properties using ALTO Unified Property Service

Above sections describe how a uCDN can get the whole capabilities and footprints from dCDNs and how a uCDN can get the footprints of given capabilities. But there is another important case which is how a uCDN can get properties (i.e., capabilities) of given footprints.

The most natural way to solve this problem is to use ALTO unified property map defined in [I-D.ietf-alto-unified-props-new] since footprints can be easily presented as groups of entities and Filtered Property Maps are already well-defined. In this section, we describe how ALTO clients look up properties for individual footprints. We firstly describe how to represent footprint objects as unified property map entities, and then we provide examples of the full property map, the filtered property map and the incremental updates.
6.1. Representing Footprint Objects as Unified Property Map Entities

A footprint object has two properties: footprint-type and footprint-value. A footprint-value is an array of footprint values conforming to the specification associated with the registered footprint type ("ipv4cidr", "ipv6cidr", "asn", and "countrycode"). Since each unified property map entity has a unique address and each pair of footprint-type and a footprint value determines a group of unique addresses, a footprint object can be represented as a set of entities according to their different footprint-type and footprint values. However, [I-D.ietf-alto-unified-props-new] only defines IPv4 Domain and IPv6 Domain which represent footprint-type "ipv4cidr" and "ipv6cidr" respectively. To represent footprint-type "asn" and "countrycode", this document registers two new domains in Section 8.

Here gives an example of representing a footprint object as a set of unified property map entities.

{"footprint-type": "ipv4cidr", "footprint-value": ["192.0.2.0/24", "198.51.100.0/24"]} --> "ipv4:192.168.2.0/24", "ipv4:198.51.100.0/24"

6.1.1. ASN Domain

This document specifies a new domain in addition to the ones in [I-D.ietf-alto-unified-props-new]. ASN is the abbreviation of Autonomous System Number.

6.1.1.1. Domain Name

asn

6.1.1.2. Domain-Specific Entity Addresses

The entity address of asn domain is encoded as a string consisting of the characters "as" (in lowercase) followed by the ASN [RFC6793].

6.1.1.3. Hierarchy and Inheritance

There is no hierarchy or inheritance for properties associated with ASN.

6.1.2. COUNTRYCODE Domain

This document specifies a new domain in addition to the ones in [I-D.ietf-alto-unified-props-new].
6.1.2.1. Domain Name
countrycode

6.1.2.2. Domain-Specific Entity Addresses

The entity address of countrycode domain is encoded as an ISO 3166-1 alpha-2 code [ISO3166-1] in lowercase.

6.1.2.3. Hierarchy and Inheritance

There is no hierarchy or inheritance for properties associated with country codes.

6.2. Examples

6.2.1. IRD Example

We use the same IRD example given by Section 3.7.1.

6.2.2. Property Map Example

This example shows a full unified property map in which entities are footprints and entities’ property is "cdni-fci-capabilities".
GET /propmap/full/cdnifci HTTP/1.1
HOST: alto.example.com
Accept: application/alto-propmap+json, application/alto-error+json

HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json

{
    "property-map": {
        "meta": {
            "dependent-vtags": [
                {
                    "resource-id": "my-default-cdnifci-map",
                    "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf62"
                }
            ],
            "countrycode:us": {
                "cdni-fci-capabilities": [
                    {
                        "capability-type": "FCI.DeliveryProtocol",
                        "capability-value": {
                            "delivery-protocols": ["http/1.1"]
                        }
                    }
                ]
            },
            "ipv4:192.0.2.0/24": {
                "cdni-fci-capabilities": [
                    {
                        "capability-type": "FCI.DeliveryProtocol",
                        "capability-value": {
                            "delivery-protocols": ["http/1.1"]
                        }
                    }
                ]
            },
            "ipv4:198.51.100.0/24": {
                "cdni-fci-capabilities": [
                    {
                        "capability-type": "FCI.DeliveryProtocol",
                        "capability-value": {
                            "delivery-protocols": ["http/1.1"]
                        }
                    }
                ]
            },
            "ipv6:2001:db8::/32": {
                "cdni-fci-capabilities": [
                    {
                        "capability-type": "FCI.DeliveryProtocol",
                        "capability-value": {
                            "delivery-protocols": ["http/1.1"]
                        }
                    }
                ]
            },
            "asn:as64496": {
                "cdni-fci-capabilities": [
                    {
                        "capability-type": "FCI.DeliveryProtocol",
                        "capability-value": {
                            "delivery-protocols": ["http/1.1", "https/1.1"]
                        }
                    }
                ]
            }
        }
    }
}
6.2.3. Filtered Property Map Example

In this example, we use filtered property map service to get "pid" and "cdni-fci-capabilities" properties for two footprints "ipv4:192.0.2.0/24" and "ipv6:2001:db8::/32".

```plaintext
POST /propmap/lookup/cdnifci-pid HTTP/1.1
HOST: alto.example.com
Content-Type: application/alto-propmapparams+json
Accept: application/alto-propmap+json,application/alto-error+json
Content-Length:

{
  "entities": [
    "ipv4:192.0.2.0/24",
    "ipv6:2001:db8::/32"
  ],
  "properties": [ "cdni-fci-capabilities", "pid" ]
}

HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json

{
  "property-map": {
    "meta": {
      "dependent-vtags": [
        {"resource-id": "my-default-cdnifci-map",
         "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf62"},
        {"resource-id": "my-default-networkmap",
         "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf63"}
      ],
      "ipv4:192.0.2.0/24": {
        "cdni-fci-capabilities": [
          {"capability-type": "FCI.DeliveryProtocol",
           "capability-value": {"delivery-protocols": ["http/1.1"]}},
          "pid": "pid1"
        },
        "ipv6:2001:db8::/32": {
          "cdni-fci-capabilities": [
            {"capability-type": "FCI.DeliveryProtocol",
             "capability-value": {"delivery-protocols": ["http/1.1"]}},
            "pid": "pid3"
          ]
        }
      }
    }
  }
}
```
6.2.4.  Incremental Updates Example

In this example, here is a client want to request updates for the properties "cdni-fci-capabilities" and "pid" for two footprints "ipv4:192.0.2.0/24" and "ipv6:2001:db8::/32".

POST /updates/properties HTTP/1.1
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

{
  "add": {
    "property-map-including-capability-property": {
      "resource-id": "filtered-cdnifci-property-map",
      "input": {
        "properties": ["cdni-fci-capabilities", "pid"],
        "entities": {
          "ipv4:192.0.2.0/24",
          "ipv6:2001:db8::/32"
        }
      }
    }
  }
}

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

event: application/alto-updatestreamcontrol+json
data: ("control-uri":
data: "http://alto.example.com/updates/streams/1414213562373")

event: application/alto-cdnifcimap+json,my-fci-stream
data: { ... full filtered unified property map ... }

event: application/merge-patch+json,my-fci-stream
data: {
data:    "property-map":
data:      {
data:        "meta": {
data:          "dependent-vtags": {
data:            {
            "resource-id": "my-default-cdnifci-map",
data:              "tag": "2beac8ee23c3dd1e9d9a73fd30df0e6e8a5627"},
data:            {
            "resource-id": "my-default-networkmap",
data:              "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf63"}
data:          }
        }
      }
}

7. Design Decisions and Discussions

7.1. Table versus Map

A major design decision is if the Map service is suitable to provide the CDNI FCI. Current ALTO protocol uses Map service to provide network information, such as Network Maps, Cost Maps and Property Maps. Their common idea is to use Map-like data structure to represent information. It is different from the data structure of the CDNI FCI designed in [RFC8008], which suggests to use a set of BaseAdvertisementObjects to represent the CDNI FCI information, which actually is Table-like data structure. Both Table and Map can be represented as a set of data entries. But the difference of them is whether there is a primary key to index each data entry.

The main advantage of Map-like data design is to simplify the filter-based query. According to the discussion in [RFC8008] about benefits and concerns of advertisement-based design and query-based design, filter-based query can make the CDNI FCI scalable when the dCDN has thousands or tens of thousands of FCI objects. To transfer Table-like data to Map-like data, introducing the primary key is necessary. This document already defines two different solution to introduce the primary key: (1) set unique identifiers for CDNI capability objects; (2) set unique identifiers for CDNI footprint objects.
But the major concern of the Map-like data design is the redundancy. In Map-like data design, whatever we choose CDNI capability objects or footprint objects as the key, each data entry can only represent the 1-N relation. But there are lots of CDNI FCI objects have the N-N relation.

7.2. Filter-based Query versus Test-based Query

Another design decision is the query approach. ALTO is a query-based protocol. So using ALTO, uCDN should send a query request to the dCDN to pull the CDNI FCI proactively. To make the query efficiently instead of pulling the whole FCI data base every time, query approach design is very important.

This document only defines the filter-based query. A uCDN can specify a set of FCI capability objects or footprint objects to only query the information including them. But there are two limitations: (1) uCDN cannot filter both of them simultaneously; (2) cannot specify complex filters.

One example is that uCDN wants to filter all CDNI FCI objects whose capabilities are in range C1 and footprints are in range F1, or capabilities are in range C2 and footprints are in range F2.

8. IANA Considerations

8.1. CDNI Metadata Footprint Type Registry

<table>
<thead>
<tr>
<th>Footprint Type</th>
<th>Description</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>altonetworkmap</td>
<td>A list of PID-names</td>
<td>RFCthis</td>
</tr>
</tbody>
</table>

Table 1: CDNI Metadata Footprint Type

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

8.2. ALTO Entity Domain Registry

As proposed in Section 9.2 of [I-D.ietf-alto-unified-props-new], "ALTO Entity Domain Registry" is requested. Besides, two new domains are to be registered, listed in Table 2.
### 8.3. ALTO CDNI FCI Property Type Registry

The "ALTO CDNI FCI Property Type Registry" is required by the ALTO Entity Domain "asn", "countrycode", "pid", "ipv4" and "ipv6", listed in Table 3.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Intended Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdni-fci-capabilities</td>
<td>An array of CDNI FCI capability objects</td>
</tr>
</tbody>
</table>

Table 3: ALTO CDNI FCI Property Type

### 9. Security Considerations

Although CDNI FCI Map resource defined in this document is relatively different from other existed resources defined in the base protocol, the Security Considerations of the base protocol (Section 15 of RFC7285) still apply.

For authenticity and Integrity of ALTO information, an attacker may disguise itself as an ALTO server in a dCDN, and it may provide false capabilities and footprints to an ALTO client in a uCDN by the CDNI FCI map. Such false information may lead a uCDN to select a wrong dCDN to serve user requests or even block uCDNs utilizing some dCDNs in good condition.

For potential undesirable guidance from authenticated ALTO information, dCDNs can provide a uCDN with limited capabilities and smaller footprint coverage so that dCDNs can avoid transferring traffic for a uCDN which they should have to transfer.

For confidentiality of ALTO information, an attacker may infer the whole and exact capabilities and footprints of a dCDN by means of pretending it is one of different uCDNs of a dCDN respectively, getting different CDNI FCI maps from a dCDN and combining these maps together.
For privacy for ALTO users, querying footprint properties using ALTO unified property may expose network location identifiers (IP addresses or fine-grained PIDs) to the ALTO server in a dCDN. In such case, a dCDN may potentially monitor and analyze user behaviors and communication patterns of uCDNs’ customers.

For availability of ALTO services, an attacker may get the potential huge full CDNI FCI maps from an ALTO server in a dCDN continuously to run out of bandwidth resources of that ALTO server or may query filtered CDNI FCI services with complex capabilities to run out of computation resources of an ALTO server.

Protection Strategies described in RFC 7285 can solve problems mentioned above well. However, the isolation of full/filtered CDNI FCI maps should also be considered.

If a dCDN signs agreements with multiple uCDNs, it must isolate full/filtered CDNI FCI maps for different uCDNs in that uCDNs will not redirect requests which should not have to served by this dCDN to this dCDN and it may not disclose extra information to uCDNs.

To avoid this risk, a dCDN may consider generating URIs of different full/filtered CDNI FCI maps by hashing its company ID, a uCDN’s company ID as well as their agreements. And it needs to avoid exposing all full/filtered CDNI FCI maps resources in one of its IRDs.

10. Acknowledgments

The authors would like to thank Daryl Malas, Matt Caulfield for their timely reviews and invaluable comments.

Jan Seedorf is partially supported by the GreenICN project (GreenICN: Architecture and Applications of Green Information Centric Networking), a research project supported jointly by the European Commission under its 7th Framework Program (contract no. 608518) and the National Institute of Information and Communications Technology (NICT) in Japan (contract no. 167). 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 GreenICN project, the European Commission, or NICT.

11. References
11.1. Normative References


11.2. Informative References

[I-D.ietf-alto-incr-update-sse]

[I-D.ietf-alto-path-vector]

[I-D.ietf-alto-unified-props-new]

[I-D.jenkins-alto-cdn-use-cases]

Authors’ Addresses

Jan Seedorf
HFT Stuttgart - Univ. of Applied Sciences
Schellingstrasse 24
Stuttgart 70174
Germany
Phone: +49-0711-8926-2801
Email: jan.seedorf@hft-stuttgart.de

Y.R. Yang
Tongji/Yale University
51 Prospect Street
New Haven, CT 06511
United States of America
Email: yry@cs.yale.edu
URI: http://www.cs.yale.edu/~yry/
Kevin J. Ma  
Ericsson  
43 Nagog Park  
Acton, MA  01720  
United States of America  

Phone: +1-978-844-5100  
Email: kevin.j.ma@ericsson.com  

Jon Peterson  
NeuStar  
1800 Sutter St Suite 570  
Concord, CA  94520  
United States of America  

Email: jon.peterson@neustar.biz  

Xiao Shawn Lin  
Tongji University  
4800 Cao‘an Hwy  
Shanghai  201804  
China  

Email: x.shawn.lin@gmail.com
Abstract

This document is an extension to the base Application-Layer Traffic Optimization (ALTO) protocol. It extends the ALTO cost information service so that applications decide not only ‘where’ to connect, but also ‘when’. This is useful for applications that need to perform bulk data transfer and would like to schedule these transfers during an off-peak hour, for example. This extension introduces ALTO Cost Calendar, with which an ALTO Server exposes ALTO cost values in JSON arrays where each value corresponds to a given time interval. The time intervals as well as other Calendar attributes, are specified in the Information Resources Directory and ALTO Server responses.

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.

When the words appear in lower case, they are to be interpreted with their natural language meanings.

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 https://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 November 15, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.  Introduction ................................................. 3
2.  Overview of ALTO Cost Calendars and terminology ............ 4
   2.1.  Terminology ........................................... 4
   2.2.  ALTO Cost Calendar overview ............................ 5
   2.3.  ALTO Cost Calendar information features .............. 6
   2.4.  ALTO Calendar design characteristics ................ 6
      2.4.1.  ALTO Cost Calendar for all cost modes .......... 7
      2.4.2.  Compatibility with legacy ALTO Clients .......... 8
3.  ALTO Calendar specification: IRD extensions ............... 8
   3.1.  Calendar attributes in the IRD resources capabilities .. 9
   3.2.  Calendars in a delegate IRD ........................... 10
   3.3.  Example IRD with ALTO Cost Calendars ................. 10
4.  ALTO Calendar specification: Service Information Resources . 14
   4.1.  Calendar extensions for Filtered Cost Maps (FCM) .... 14
      4.1.1.  Calendar extensions in Filtered Cost Map requests .. 14
      4.1.2.  Calendar extensions in Filtered Cost Map responses .. 15
      4.1.3.  Use case and example: FCM with a bandwidth Calendar .. 18
   4.2.  Calendar extensions in the Endpoint Cost Service ...... 20
      4.2.1.  Calendar specific input in Endpoint Cost requests .. 20
      4.2.2.  Calendar attributes in the Endpoint Cost response .. 20
      4.2.3.  Use case and example: ECS with a routingcost Calendar 21
4.2.4. Use case and example: ECS with a multi-cost Calendar
for routingcost and owdelay .......................... 24
5. IANA Considerations ................................. 26
6. Security Considerations .............................. 26
7. Operational Considerations .......................... 27
8. Acknowledgements .................................... 28
9. References ........................................... 28
  9.1. Normative References .............................. 28
  9.2. Informative References ............................ 29
Authors’ Addresses ...................................... 30

1. Introduction

The base Application-Layer Traffic Optimization (ALTO) protocol specified in [RFC7285] provides guidance to overlay applications that need to select one or several hosts from a set of candidates able to provide a desired resource. This guidance is based on parameters that affect performance and efficiency of the data transmission between the hosts such as the topological distance. The goal of ALTO is to improve the Quality of Experience (QoE) in the application while optimizing resource usage in the underlying network infrastructure.

The ALTO protocol in [RFC7285] specifies a network map which defines groupings of endpoints in provider-defined network regions identified by Provider-defined Identifiers (PIDs). The Cost Map Service, Endpoint Cost Service (ECS) and Endpoint Ranking Service then provide ISP-defined costs and rankings for connections among the specified endpoints and PIDs and thus incentives for application clients to connect to ISP preferred locations, for instance, to reduce their costs. For the reasons outlined in the ALTO problem statement [RFC5693] and requirement AR-14 of [RFC6708], ALTO does not disseminate network metrics that change frequently. In a network, the costs can fluctuate for many reasons having to do with instantaneous traffic load or due to diurnal patterns of traffic demand or planned events such as network maintenance, holidays or highly publicized events. Thus, an ALTO application wishing to use the Cost Map and Endpoint Cost Service at some future time will have to estimate the state of the network at that time, a process that is, at best, fragile and brittle since the application does not have any visibility into the state of the network. The need of such future scheduling of large scale traffic that can be addressed by the ALTO protocol is motivated by Unicorn, a unified resource orchestration framework for multi-domain, geo-distributed data analytics, see [draft-xiang-alto-multidomain-analytics].

In case the ALTO Cost value changes are predictable over a certain period of time and the application does not require immediate data...
transfer, it can save time to get the whole set of cost values over this period in one single ALTO response. Using this set to schedule data transfers allows optimizing the network resources usage and QoE. ALTO Clients and Servers can also minimize their workload by reducing and accordingly scheduling their data exchanges.

This document extends [RFC7285] to allow an ALTO server to provide network costs for a given duration of time. A sequence of network costs across a time span for a given pair of network locations is named an "ALTO Cost Calendar". The Filtered Cost Map Service and Endpoint Cost Service are extended to provide Cost Calendars. In addition to this functional ALTO enhancement, we expect to further save network and storage resources by gathering multiple Cost Values for one Cost Type into one single ALTO Server response.

In this draft, an "ALTO Cost Calendar" is specified in terms of information resources capabilities that are applicable to time-sensitive ALTO metrics. An ALTO Cost Calendar exposes ALTO Cost Values in JSON arrays, see [RFC8259], where each value corresponds to a given time interval. The time intervals as well as other Calendar attributes are specified in the Information Resources Directory (IRD) and in the Server response to allow the ALTO Client to interpret the received ALTO values. Last, the extensions for ALTO Calendars are applicable to any Cost Mode and they ensure backwards compatibility with legacy ALTO clients.

In the rest of this document, Section 2 provides the design characteristics. Sections 3 and 4 define the formal specifications for the IRD and the information resources. IANA, security and operational considerations are addressed respectively in sections Section 5, Section 6 and Section 7.

2. Overview of ALTO Cost Calendars and terminology

2.1. Terminology

- ALTO transaction: A request/response exchange between an ALTO Client and an ALTO Server.
- Client: When used with a capital "C", this term refers to an ALTO Client.
- Calendar, Cost Calendar: When used with a capital "C", these terms refer to an ALTO Cost Calendar.
- Endpoint (EP): An endpoint is defined as in Section 2.1 of [RFC7285]. It can be, for example, a peer, a CDN storage location, a physical server involved in a virtual server-supported
application, a party in a resource-sharing swarm such as a
computation grid, or an online multi-party game.

- Server: When used with a capital "S", this term refers to an ALTO
  Server.

2.2. ALTO Cost Calendar overview

An ALTO Cost Calendar provided by the ALTO Server provides 2
information items:

- an array of values for a given metric, where each value
  corresponds to a time interval, where the value array can
  sometimes be a cyclic pattern that repeats a certain number of
times.

- attributes describing the time scope of the Calendar such as the
  size and number of the intervals and the date of the starting
  point of the Calendar, allowing an ALTO Client to interpret the
  values properly.

An ALTO Cost Calendar can be used like a "time table" to figure out
the best time to schedule data transfers and also to proactively
manage application traffic given predictable events such as expected
spike in traffic due to crowd gathering (concerts, sports, etc.),
traffic-intensive holidays and network maintenance. It may be viewed
as a synthetic abstraction of, for example, real measurements
gathered over previous periods on which statistics have been
computed. However, like for any schedule, unexpected network
incidents may require the current ALTO Calendar to be updated and re-
sent to the ALTO Clients needing it. To this end, it is RECOMMENDED
that ALTO Servers providing ALTO Calendars also provide the "ALTO
Incremental Updates Using Server-Sent Events (SSE)" Service that is
specified in [draft-ietf-alto-incr-update-sse]. Likewise, ALTO
Clients capable of using ALTO Calendars SHOULD also use the SSE
Service.

Most likely, the ALTO Cost Calendar would be used for the Endpoint
Cost Service, assuming that a limited set of feasible Endpoints for a
non-real time application is already identified, that they do not
need to be accessed immediately and that their access can be
scheduled within a given time period. The Filtered Cost Map Service
is also applicable as long as the size of the Map allows it.
2.3. ALTO Cost Calendar information features

The Calendar attributes are provided in the Information Resources Directory (IRD) and in ALTO Server responses. The IRD announces attributes without date values in its information resources capabilities, whereas attributes with time dependent values are provided in the "meta" section of Server responses. The ALTO Cost Calendar attributes provide the following information:

- attributes to describe the time scope of the Calendar value array:
  - generic time zone,
  - applicable time interval size for each Calendar value, defined in seconds, that can cover a wide range of values.
  - duration of the Calendar: e.g., the number of intervals provided in the Calendar.
- "calendar-start-date": specifying when the Calendar starts, that is to which date the first value of the Cost Calendar is applicable.
- "repeated": an optional attribute indicating how many iterations of the provided Calendar will have the same values. The server may use it to allow the client to schedule its next request and thus save its own workload by reducing processing of similar requests.

Attribute "repeated" may take a very high value if a Calendar represents a cyclic value pattern that the Server considers valid for a long period. In this case, the Server will only update the Calendar values once this period has elapsed or if an unexpected event occurs on the network.

2.4. ALTO Calendar design characteristics

The extensions in this document and encode requests and responses using JSON [RFC8259].

Formally, the cost entries in an ALTO cost map can be any type of JSON value [RFC8259], (see the DstCosts object in Section 11.2.3.6 of [RFC7285]). However, that section states that an implementation of [RFC7285] SHOULD assume that the cost is a JSON number and fail to parse if it is not, unless the implementation is using an extension that signals a different data type. This document extends the definition of a legacy cost map given in [RFC7285] to allow a cost...
entry to be an array of values, one per time interval, instead of just one number.

To realize an ALTO Calendar, this document extends: the IRD, the ALTO requests and responses for Cost Calendars.

This extension is designed to be light and to ensure backwards compatibility with base protocol ALTO Clients and with other extensions. It relies on section 8.3.7 "Parsing of Unknown Fields" of [RFC7285] that writes: "Extensions may include additional fields within JSON objects defined in this document. ALTO implementations MUST ignore unknown fields when processing ALTO messages."

The Calendar-specific capabilities are integrated in the information resources of the IRD and in the "meta" member of ALTO responses to Cost Calendars requests. A Calendar and its capabilities are associated with a given information resource and within this information resource with a given cost type. This design has several advantages:

- it does not introduce a new mode,
- it does not introduce new media types,
- it allows an ALTO Server to offer Calendar capabilities on a cost type, with attributes values adapted to each information resource.

The applicable Calendared information resources are:

- the Filtered Cost Map,
- the Endpoint Cost Map.

The ALTO Server can choose in which frequency it provides cost Calendars to ALTO Clients. It may either provide Calendar updates starting at the request date, or carefully schedule its updates so as to take profit from a potential repetition/periodicity of Calendar values.

2.4.1. ALTO Cost Calendar for all cost modes

An ALTO Cost Calendar is well-suited for values encoded in the "numerical" mode. Actually, a Calendar can also represent metrics in other modes considered as compatible with time-varying values. For example, types of Cost values such as JSONBool can also be calendared, as their value may be 'true' or 'false' depending on given time periods or likewise, values represented by strings, such as "medium", "high", "low", "blue", "open".
Note also that a Calendar is suitable as well for time-varying metrics provided in the "ordinal" mode, if these values are time-varying and the ALTO Server provides updates of cost value based preferences.

2.4.2. Compatibility with legacy ALTO Clients

The ALTO protocol extensions for Cost Calendars have been defined so as to ensure that Calendar capable ALTO Servers can provide legacy ALTO Clients with legacy information resources as well. That is a legacy ALTO Client can request resources and receive responses as specified in [RFC7285].

A Calendar-aware ALTO Server MUST implement the base protocol specified in [RFC7285].

As a consequence, when a metric is available as a Calendar array, it also MUST be available as a single value as required by [RFC7285]. The Server, in this case, provides the current value of the metric to either Calendar-aware Clients not interested in future or time-based values, or Clients implementing [RFC7285] only.

For compatibility with legacy ALTO Clients specified in [RFC7285], calendared information resources are not applicable for full cost maps for the following reason: a legacy ALTO client would receive a calendared cost map via an HTTP 'GET' command. As specified in section 8.3.7 of [RFC7285], it will ignore the Calendar Attributes indicated in the "meta" of the responses. Therefore, lacking information on Calendar attributes, it will not be able to correctly interpret and process the values of the received array of Calendar cost values.

Therefore, calendared information resources MUST be requested via the Filtered Cost Map Service or the Endpoint Cost Service, using a POST method.

3. ALTO Calendar specification: IRD extensions

The Calendar attributes in the IRD information resources capabilities carry constant dateless values. A Calendar is associated with an information resource rather than a cost type. For example, a Server can provide a "routingcost" Calendar for the Filtered Cost Map Service at a granularity of one day and a "routingcost" Calendar for the Endpoint Cost Service at a finer granularity but for a limited number of endpoints. An example IRD with Calendar specific features is provided in Section 3.3.
3.1. Calendar attributes in the IRD resources capabilities

A Cost Calendar for a given Cost Type MUST be indicated in the IRD by an object of type CalendarAttributes. A CalendarAttribute object is represented by the "calendar-attributes" member of a resource entry. Each CalendarAttributes object applies to a set of one or more cost types. A Cost Type name MUST appear no more than once in the "calendar-attributes" member of a resource entry; multiple appearances of a Cost Type name in CalendarAttributes object of the "calendar-attributes" member MUST cause the ALTO client to ignore any occurrences of this name beyond the first encountered occurrence.

It is RECOMMENDED for an ALTO Server that the time interval size specified in the IRD is the smallest possible one that it can provide. The Client can aggregate cost values on its own if it needs a larger granularity.

The encoding format for object CalendarAttributes, using JSON [RFC8259], is as follows:

```
CalendarAttributes calendar-attributes <1..*>
{
  JSONString cost-type-names <1..*>
  JSONNumber time-interval-size
  JSONNumber number-of-intervals
} CalendarAttributes
```

- "cost-type-names":
  * An array of one or more elements indicating the cost-type-names in the IRD entry to which the capabilities apply.

- "time-interval-size":
  * is the duration of an ALTO Calendar time interval in seconds. A "time-interval-size" value contains a JSONNumber. ALTO servers SHOULD use at least IEEE 754 double-precision floating point [IEEE.754.2008] to store this value. Example values are: 300, 7200, meaning that each Calendar value applies on a time interval that lasts respectively 5 minutes and 2 hours.

- "number-of-intervals":
  * the integer number of values of the Cost Calendar array, at least equal to 1.
- Attribute "cost-type-names" provides a better readability to the Calendar attributes specified in the IRD and avoids confusion with Calendar attributes of other cost-types.

- Multiplying 'time-interval-size' by 'number-of-intervals' provides the duration of the provided Calendar. For example, an ALTO Server may provide a Calendar for ALTO values changing every 'time-interval-size' equal to 5 minutes. If 'number-of-intervals' has the value 12, then the duration of the provided Calendar is "1 hour".

3.2. Calendars in a delegate IRD

It may be useful to distinguish IRD resources supported by the base ALTO protocol from resources supported by its extensions. To achieve this, one option, is that a "root" ALTO Server implementing base protocol resources delegates "specialized" information resources such as the ones providing Cost Calendars, to another ALTO Server running in a subdomain that is specified with its URI in the "root" ALTO Server. This option is described in Section 9.2.4 "Delegation using IRDs" of [RFC7285].

This document provides an example, where a "root" ALTO Server runs in a domain called "alto.example.com". It delegates the announcement of Calendars capabilities to an ALTO Server running in a subdomain called "custom.alto.example.com". The location of the "delegate Calendar IRD" is assumed to be indicated in the "root" IRD by the resource entry: "custom-calendared-resources".

Another advantage is that some Cost Types for some resources may be more advantageous as Cost Calendars and it makes few sense to get them as a single value. For example, Cost Types with predictable and frequently changing values, calendared in short time intervals such as a minute.

3.3. Example IRD with ALTO Cost Calendars

This section provides an example ALTO Server IRD that supports various cost metrics and cost modes. In particular, since [RFC7285] makes it mandatory, the Server uses metric "routingcost" in the "numerical" mode.

For illustrative purposes, this section introduces 3 other fictitious example metrics and modes that should be understood as examples and should not be used or considered as normative.

The cost type names used in the example IRD as thus as follows:
o "num-routingcost": refers to metric "routingcost" in the numerical mode as defined in [RFC7285] and registered with IANA.

o "num-owdelay": refers to fictitious performance metric "owdelay" in the "numerical" mode, to reflect the one-way packet transmission delay on a path. A related performance metric is currently under definition in [draft-ietf-alto-performance-metrics].

o "num-throughputrating": refers to fictitious metric "throughputrating" in the "numerical" mode, to reflect the provider preference in terms of end to end throughput.

o "string-servicestatus": refers to fictitious metric "servicestatus" in some example mode "string", to reflect the availability, defined by the provider, of for instance path connectivity.

The example IRD includes 2 particular URIs providing Calendars:

o "https://custom.alto.example.com/calendar/costmap/filtered": a filtered cost map in which Calendar capabilities are indicated for cost type names: "num-routingcost", "num-throughputrating" and "string-servicestatus",

o "https://custom.alto.example.com/calendar/endpointcost/lookup": an endpoint cost map in which Calendar capabilities are indicated for cost type names: "num-routingcost", "num-owdelay", "num-throughputrating", "string-servicestatus".

The design of the Calendar capabilities allows that some Calendars on a cost type name are available in several information resources with different Calendar Attributes. This is the case for Calendars on "num-routingcost", "num-throughputrating" and "string-servicestatus", available in both the Filtered Cost map and Endpoint Cost Service, but with different time interval sizes for "num-throughputrating" and "string-servicestatus".

--- Client to Server request for IRD ----------

GET /calendars-directory HTTP/1.1
Host: custom.alto.example.com
Accept: application/alto-directory+json, application/alto-error+json

--- Server response to Client -------------

HTTP/1.1 200 OK
Content-Length: 2542
Content-Type: application/alto-directory+json

{
    "meta": {
        "default-alto-network-map": "my-default-network-map",
        "cost-types": {
            "num-routingcost": {
                "cost-mode": "numerical",
                "cost-metric": "routingcost"
            },
            "num-owdelay": {
                "cost-mode": "numerical",
                "cost-metric": "owdelay"
            },
            "num-throughputrating": {
                "cost-mode": "numerical",
                "cost-metric": "throughputrating"
            },
            "string-servicestatus": {
                "cost-mode": "string",
                "cost-metric": "servicestatus"
            }
        }
    },
    "resources": {
        "filtered-cost-map-calendar": {
            "uri": "https://custom.alto.example.com/calendar/costmap/filtered",
            "media-type": "application/alto-costmap+json",
            "accepts": "application/alto-costmapfilter+json",
            "capabilities": {
                "cost-constraints": true,
                "cost-type-names": ["num-routingcost",
                                    "num-throughputrating",
                                    "string-servicestatus"],
                "calendar-attributes": [
                    {"cost-type-names": ["num-routingcost",
                                             "num-throughputrating"],
                     "time-interval-size": 7200,
                     "number-of-intervals": 12}
                ,
                {"cost-type-names": ["string-servicestatus"],
                 "time-interval-size": 1800,
                 "number-of-intervals": 48}
                ]
            },
            "uses": ["my-default-network-map"]
        }
    }
}
In this example IRD, for the Filtered Cost Map Service:

- the Calendar for "num-routingcost" and "num-throughputrating" is an array of 12 values each provided on a time interval lasting 7200 seconds (2 hours).

- the Calendar for "string-servicestatus": is an array of 48 values each provided on a time interval lasting 1800 seconds (30 minutes).

For the Endpoint Cost Service:
the Calendar for "num-routingcost": is an array of 24 values each provided on a time interval lasting 3600 seconds (1 hour).

o the Calendar for "owdelay": is an array of 12 values each provided on a time interval lasting 300 seconds (5 minutes).

o the Calendar for "num-throughputrating": is an array of 60 values each provided on a time interval lasting 60 seconds (1 minute).

o the Calendar for "string-servicestatus": is an array of 30 values each provided on a time interval lasting 120 seconds (2 minutes).

4. ALTO Calendar specification: Service Information Resources

This section documents the individual information resources defined to provide the calendared information services defined in this document.

The reference time zone for the provided time values is UTC. The option chosen to express the time format is the HTTP header fields format specified in [RFC7231] where, however, timestamps are still displayed with the acronym "GMT" rather than "UTC":

```
Date: Tue, 15 Nov 2014 08:12:31 GMT
```

The value of a Calendar time interval size is expressed in seconds.

4.1. Calendar extensions for Filtered Cost Maps (FCM)

A legacy ALTO client requests and gets Filtered Cost Map responses as specified in [RFC7285].

4.1.1. Calendar extensions in Filtered Cost Map requests

The input parameters of a "legacy" request for a filtered cost map, defined by object ReqFilteredCostMap in section 11.3.2 of [RFC7285], are augmented with one additional member.

A Calendar-aware ALTO client requesting a Calendar on a given Cost Type for a filtered cost map resource having Calendar capabilities MUST add the following field to its input parameters:

```
JSONBoolean calendared<1..*>;
```

This field is an array of 1 to N boolean values, where N is the number of requested metrics. Each entry corresponds to the requested metric at the same array position. Each boolean value indicates
whether or not the ALTO Server should provide the values for this Cost Type as a Calendar. The array MUST contain exactly N boolean values, otherwise, the Server returns an error.

This field MUST NOT be included if no member "calendar-attributes" is specified in this information resource.

If a value of field ‘calendared’ is ‘true’ for a cost type name for which no Calendar attributes have been specified: an ALTO Server, whether it implements the extensions of this document or only implements [RFC7285], MUST ignore it and return a response with a single cost value as specified in [RFC7285].

If this field is not present, it MUST be assumed to have only values equal to ‘false’.

A Calendar-aware ALTO client that supports requests for only one cost type at a time and wants to request a Calendar MUST provide an array of 1 element:

   "calendared" : [true];

A Calendar-aware ALTO client that supports requests for more than one Cost Types at a time, as specified in [RFC8189] MUST provide an array of N values set to ‘true’ or ‘false’, depending whether it wants the applicable Cost Type values as a single or calendared value.

4.1.2. Calendar extensions in Filtered Cost Map responses

In a calendared ALTO Filtered Cost Map, a cost value between a source and a destination is a JSON array of JSON values. An ALTO Calendar values array has a number of values equal to the value of member "number-of-intervals" of the Calendar attributes that are indicated in the IRD. These attributes will be conveyed as metadata in the Filtered Cost Map response. Each element of the array is valid for the time-interval that matches its array position.

The FCM response conveys metadata among which:

  o some are not specific to Calendars and ensure compatibility with [RFC7285] and [RFC8189]

  o some are specific to Calendars.

The non Calendar specific "meta" fields of a calendared Filtered Cost Map response MUST include at least:
if the ALTO Client requests cost values for one Cost Type at a time only: the "meta" fields specified in [RFC7285] for these information service responses:

* "dependent-vtags",
* "cost-type" field.

if the ALTO Client implements the Multi-Cost ALTO extension specified in [RFC8189] and requests cost values for several Cost Types at a time: the "meta" fields specified in [RFC8189] for these information service responses:

* "dependent-vtags",
* "cost-type" field with value set to ‘{}’, for backwards compatibility with [RFC7285].
* "multi-cost-types" field.

If the client request does not provide member "calendared" or if it provides it with a value equal to 'false', for all the requested Cost Types, then the ALTO Server response is exactly as specified in [RFC7285] and [RFC8189].

If the value of member "calendared" is equal to ‘false’ for a given requested Cost Type, the ALTO Server MUST return, for this Cost Type, a single cost value as specified in [RFC7285].

If the value of member "calendared" is equal to ‘true’ for a given requested Cost Type, the ALTO Server returns, for this Cost Type, a cost value Calendar as specified above in this section. In addition to the above cited non Calendar specific "meta" members, the Server MUST provide a Calendar specific metadata field.

The Calendar specific "meta" field that a calendared Filtered Cost Map response MUST include is a member called "calendar-response-attributes", that describes properties of the Calendar and where:

o member "calendar-response-attributes" is an array of one or more objects of type "CalendarResponseAttributes".

o each "CalendarResponseAttributes" object in the array is specified for one or more Cost Types for which the value of member "calendared" is equal to 'true' and for which a Calendar is provided for the requested information resource.
the "CalendarResponseAttributes" object that applies to a cost
type name has a corresponding "CalendarAttributes" object defined
for this cost type name in the IRD capabilities of the requested
information resource. The members of a
"CalendarResponseAttributes" object include all the members of the
corresponding "CalendarAttributes" object.

The format of member "CalendarResponseAttributes is defined as
follows:

CalendarResponseAttributes calendar-response-attributes <1..*>;

object{
  [JSONString cost-type-names <1..*>];
  JSONString calendar-start-time;
  JSONNumber time-interval-size;
  JSONNumber number-of-intervals;
  [JSONNumber repeated;]
} CalendarResponseAttributes;

Object CalendarResponseAttributes has the following attributes:

- "cost-type-names": is an array of one or more cost-type-names to
  which the capabilities apply and for which a Calendar has been
  requested. The value of this member is a subset of the "cost-
  type-names" array specified in the corresponding IRD Calendar
  attributes.

- "calendar-start-time": indicates the date at which the first value
  of the Calendar applies. The value provided for the "calendar-
  start-time" attribute SHOULD NOT be later than the request date.

- "time-interval-size": as specified in Section 3.1 and with the
  same value.

- "number-of-intervals": as specified in Section 3.1 and with the
  same value.

- "repeated": is an optional field provided for Calendars. It is an
  integer N greater or equal to '1' that indicates how many
  iterations of the Calendar value array starting at the date
  indicated by "calendar-start-time" have the same values. The
  number N includes the provided iteration.

For example: suppose the "calendar-start-time" member has value "Mon,
30 Jun 2014 at 00:00:00 GMT", the "time-interval-size" member has
value '3600', the "number-of-intervals" member has value '24' and the
value of member "repeated" is equal to '4'. This means that the
Calendar values are the same on Monday, Tuesday, Wednesday and Thursday on a period of 24 hours starting at 00:00:00 GMT. The ALTO Client thus may use the same Calendar for the next 4 days starting at "calendar-start-time" and will only need to request a new one for Friday July 4th at 00:00:00 GMT.

Attribute "repeated" may take a very high value if a Calendar represents a cyclic value pattern that the Server considers valid for a long period and hence will only update once this period has elapsed or if an unexpected event occurs on the network. In the latter case, the client will be notified if it uses the "ALTO Incremental Updates Using Server-Sent Events (SSE)" Service, specified in [draft-ietf-alto-incr-update-sse]. See also discussion in Section 7 "Operational Considerations".

4.1.3. Use case and example: FCM with a bandwidth Calendar

An example of non-real time information that can be provisioned in a Calendar is the expected path throughput. While the transmission rate can be measured in real time by end systems, the operator of a data center is in the position of formulating preferences for given paths, at given time periods to avoid traffic peaks due to diurnal usage patterns. In this example, we assume that an ALTO Client requests a Calendar of network provider defined throughput ratings, as specified in the IRD, to schedule its bulk data transfers as described in the use cases.

In the example IRD, Calendars for cost type name "num-throughputrating" are available for the information resources: "filtered-cost-calendar-map" and "endpoint-cost-calendar-map". The ALTO Client requests a Calendar for "num-throughputrating" via a POST request for a filtered cost map.

We suppose in the present example that the ALTO Client sends its request on Tuesday July 1st 2014 at 13:15. The Server returns Calendars with arrays of 12 numbers for each source and destination pair. The values for metric "throughputrating", in this example, are assumed to be encoded in 2 digits. For representation brevity, the arrays in the provided example are symbolized by expression "[v1,v2, ... v12]", that is otherwise not valid in JSON. The value of field "Content-Length" in the responses is computed as if "throughputrating" values were encoded in 2 digits. The same type of symbolization is used in the other example Server responses in Section 4.2.3 and Section 4.2.4 of this document.
POST /calendar/costmap/filtered HTTP/1.1
Host: alto.example.com
Content-Length: 208
Content-Type: application/alto-costmapfilter+json
Accept: application/alto-costmap+json,application/alto-error+json

{
    "cost-type" : {
        "cost-mode" : "numerical",
        "cost-metric" : "throughputrating"},
    "calendared" : [true],
    "pids" : {
        "srcs" : [ "PID1", "PID2" ],
        "dsts" : [ "PID1", "PID2", "PID3" ]
    }
}

HTTP/1.1 200 OK
Content-Length: 815
Content-Type: application/alto-costmap+json

{
    "meta" : {
        "dependent-vtags" : [  
            {"resource-id": "my-default-network-map",
             "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"  
            },
        ],
        "cost-type" : {
            "cost-mode" : "numerical",
            "cost-metric" : "throughputrating"},
        "calendar-response-attributes" : [  
            {"calendar-start-time" : "Tue, 1 Jul 2014 13:00:00 GMT",
             "time-interval-size" : 7200,
             "number-of-intervals" : 12}
        ],
    },
    "cost-map" : {
        "PID1": { "PID1": [v1,v2, ... v12],
            "PID2": [v1,v2, ... v12],
            "PID3": [v1,v2, ... v12] },
        "PID2": { "PID1": [v1,v2, ... v12],
            "PID2": [v1,v2, ... v12],
            "PID3": [v1,v2, ... v12] }  
    }
}
4.2. Calendar extensions in the Endpoint Cost Service

This document extends the Endpoint Cost Service, as defined in
(11.5.1) of [RFC7285], by adding new input parameters and
capabilities, and by returning JSONArrays instead of JSONNumbers as
the cost values. The media type (11.5.1.1) and HTTP method
(11.5.1.2) are unchanged.

4.2.1. Calendar specific input in Endpoint Cost requests

The extensions to the requests for calendared Endpoint Cost Maps are
the same as for the Filtered Cost Map Service, specified in section
Section 4.1.1 of this draft.

The ReqEndpointCostMap object for a calendared ECM request will have
the following format:

object {
  [CostType cost-type;]
  [CostType multi-cost-types<1..*>;]
  [JSONBoolean calendared<1..*>;]
  EndpointFilter endpoints;
} ReqEndpointCostMap;

object {
  [TypedEndpointAddr srcs<0..*>;]
  [TypedEndpointAddr dsts<0..*>;]
} EndpointFilter;

4.2.2. Calendar attributes in the Endpoint Cost response

The "meta" field of a calendared Endpoint Cost response MUST include
at least:

- if the ALTO Client supports cost values for one Cost Type at a
time only: the "meta" fields specified in (11.5.1.6) of [RFC7285]
for the Endpoint Cost response:
  * "cost-type" field.

- if the ALTO Client supports cost values for several Cost Types at
  a time, as specified in [RFC8189] : the "meta" fields specified in
  [RFC8189] for the the Endpoint Cost response:
  * "cost-type" field with value set to '{}', for backwards
    compatibility with [RFC7285].
If the client request does not provide member "calendared" or if it provides it with a value equal to 'false', for all the requested Cost Types, then the ALTO Server response is exactly as specified in [RFC7285] and [RFC8189].

If the ALTO client provides member "calendared" in the input parameters with a value equal to 'true' for given requested Cost Types, the "meta" member of a calendared Endpoint Cost response MUST include, for these Cost Types, an additional member "calendar-response-attributes", the contents of which obey the same rules as for the Filtered Cost Map Service, specified in Section 4.1.2. The Server response is thus changed as follows, w.r.t [RFC7285] and [RFC8189]:

1. the "meta" member has one additional field "CalendarResponseAttributes", as specified for the Filtered Cost Map Service,
2. the calendared costs are JSONArrays instead of the JSONNumbers format used by legacy ALTO implementations. All arrays have a number of values equal to 'number-of-intervals'.

If the value of member "calendared" is equal to 'false' for a given requested Cost Type, the ALTO Server MUST return, for this Cost Type, a single cost value as specified in [RFC7285].

4.2.3. Use case and example: ECS with a routingcost Calendar

Let us assume an Application Client is located in an end system with limited resources and having access to the network that is either intermittent or provides an acceptable quality in limited but predictable time periods. Therefore, it needs to both schedule its resource-greedy networking activities and its ALTO transactions.

The Application Client has the choice to trade content or resources with a set ofEndpoints and needs to decide with which one it will connect and at what time. For instance, the Endpoints are spread in different time-zones, or have intermittent access. In this example, the 'routingcost' is assumed to be time-varying, with values provided as ALTO Calendars.

The ALTO Client associated with the Application Client queries an ALTO Calendar on 'routingcost' and will get the Calendar covering the 24 hours time period "containing" the date and time of the ALTO client request.
For Cost Type "num-routingcost", the solicited ALTO Server has defined 3 different daily patterns each represented by a Calendar, to cover the week of Monday June 30th at 00:00 to Sunday July 6th 23:59:

- C1 for Monday, Tuesday, Wednesday, Thursday, (weekdays)
- C2 for Saturday, Sunday, (weekend)
- C3 for Friday (maintenance outage on July 4, 2014 from 02:00:00 GMT to 04:00:00 GMT, or big holiday such as New Year evening).

In the following example, the ALTO Client sends its request on Tuesday July 1st 2014 at 13:15.

The "routingcost" values are assumed to be encoded in 3 digits.

For representation brevity, the arrays in the provided example are symbolized by expression "[v1,v2, ... v24]", that is otherwise not valid in JSON. The value of field "Content-Length" in the responses is computed as if "routingcost" values were encoded in 3 digits.
POST /calendar/endpointcost/lookup HTTP/1.1
Host: alto.example.com
Content-Length: 290
Content-Type: application/alto-endpointcostparams+json
Accept: application/alto-endpointcost+json,application/alto-error+json

{
   "cost-type" : {"cost-mode" : "numerical",
                  "cost-metric" : "routingcost"},
   "calendared" : [true],
   "endpoints" : {
      "srcs": [ "ipv4:192.0.2.2" ],
      "dsts": [ "ipv4:192.0.2.89",
                "ipv4:198.51.100.34",
                "ipv4:203.0.113.45",
                "ipv6:2001:db8::10" ]
   }
}

HTTP/1.1 200 OK
Content-Length: 877
Content-Type: application/alto-endpointcost+json

{
   "meta" : {
      "cost-type" : {"cost-mode" : "numerical",
                     "cost-metric" : "routingcost"},
      "calendar-response-attributes" : [
         {"calendar-start-time" : "Mon, 30 Jun 2014 00:00:00 GMT",
          "time-interval-size" : 3600,
          "number-of-intervals" : 24,
          "repeated": 4 }
      ],
   },
   "endpoint-cost-map" : {
      "ipv4:192.0.2.2": {
         "ipv4:192.0.2.89" : [v1, v2, ... v24],
         "ipv4:198.51.100.34" : [v1, v2, ... v24],
         "ipv4:203.0.113.45" : [v1, v2, ... v24],
         "ipv6:2001:db8::10" : [v1, v2, ... v24]
      }
   }
}
When the Client gets the Calendar for "routingcost", it sees that the "calendar-start-time" is Monday at 00h00 GMT and member "repeated" is equal to ‘4’. It understands that the provided values are valid until Thursday included and will only need to get a Calendar update on Friday.

4.2.4. Use case and example: ECS with a multi-cost Calendar for routingcost and owdelay

In this example, it is assumed that the ALTO Server implements multi-cost capabilities, as specified in [RFC8189]. That is, an ALTO client can request and receive values for several cost types in one single transaction. An illustrating use case is a path selection done on the basis of 2 metrics: routing cost and owdelay.

As in the previous example, the IRD indicates that the ALTO Server provides "routingcost" Calendars in terms of 24 time intervals of 1 hour (3600 seconds) each.

For metric "owdelay", the IRD indicates that the ALTO Server provides Calendars in terms of 12 time intervals values lasting each 5 minutes (300 seconds).

In the following example transaction, the ALTO Client sends its request on Tuesday July 1st 2014 at 13:15.

This example assumes that the values of metric "owdelay" and "routingcost" are encoded in 3 digits.

For representation brevity, the arrays in the provided example are symbolized by expression "[r1, r2, ... r24], [o1, o2, ... o12]" , that is otherwise not valid in JSON. The value of field "Content-Length" in the responses is computed as if "routingcost" and "owdelay" values were encoded in 3 digits.

POST calendar/endpointcost/lookup HTTP/1.1
Host: alto.example.com
Content-Length: 373
Content-Type: application/alto-endpointcostparams+json
Accept: application/alto-endpointcost+json,application/alto-error+json

```
{  
  "cost-type" : {},  
  "multi-cost-types" : [  
    {"cost-mode" : "numerical", "cost-metric" : "routingcost"},  
    {"cost-mode" : "numerical", "cost-metric" : "owdelay"}  
  ],
```

Randriamasy, et al. Expires November 15, 2019
"calendared" : [true, true],
"endpoints" : {
  "srcs" : [ "ipv4:192.0.2.2" ],
  "dsts" : [ "ipv4:192.0.2.89",
             "ipv4:198.51.100.34",
             "ipv4:203.0.113.45",
             "ipv6:2001:db8::10"
  ]
}

HTTP/1.1 200 OK
Content-Length: 1377
Content-Type: application/alto-endpointcost+json

{
  "meta" : {
    "multi-cost-types" : [{
      "cost-mode" : "numerical", "cost-metric" : "routingcost"},
      "cost-mode" : "numerical", "cost-metric" : "owdelay"
    }],
    "calendar-response-attributes" : [{
      "cost-type-names" : "num-routingcost",
      "calendar-start-time" : "Mon, 30 Jun 2014 00:00:00 GMT",
      "time-interval-size" : 3600,
      "number-of-intervals" : 24,
      "repeated" : 4
    },
    {
      "cost-type-names" : "num-owdelay",
      "calendar-start-time" : "Tue, 1 Jul 2014 13:00:00 GMT",
      "time-interval-size" : 300,
      "number-of-intervals" : 12
    }],
  }
}

"endpoint-cost-map" : {
  "ipv4:192.0.2.2" : {
    "ipv4:192.0.2.89" : [[r1, r2, ... r24], [o1, o2, ... o12]],
    "ipv4:198.51.100.34" : [[r1, r2, ... r24], [o1, o2, ... o12]],
    "ipv4:203.0.113.45" : [[r1, r2, ... r24], [o1, o2, ... o12]],
    "ipv6:2001:db8::10" : [[r1, r2, ... r24], [o1, o2, ... o12]]
  }
}

When receiving the response, the client sees that the Calendar values for 'routing cost' are repeated for 4 iterations. Therefore, in its
next requests until the routing cost Calendar is expected to change, the client will only need to request a Calendar for "owdelay".

Without the ALTO Calendar extensions, the ALTO client would have no clue on the dynamicity of the metric value change and would spend needless time requesting values at an inappropriate pace. In addition, without the Multi-Cost ALTO capabilities, the ALTO client would duplicate this waste of time as it would need to send one request per cost metric.

5. IANA Considerations

This document does not define any new media types or introduce any new IANA considerations.

6. Security Considerations

As an extension of the base ALTO protocol [RFC7285], this document fits into the architecture of the base protocol, and hence the Security Considerations (Section 15) of the base protocol fully apply when this extension is provided by an ALTO server. For example, the same authenticity and integrity considerations (Section 15.1 of [RFC7285]) still fully apply; the same considerations for the privacy of ALTO users (Section 15.4 of [RFC7285]) also still fully apply.

The calendaring information provided by this extension requires additional considerations on three security considerations discussed in the base protocol: potential undesirable guidance to clients (Section 15.2 of [RFC7285]), confidentiality of ALTO information (Section 15.2 of [RFC7285]), and availability of ALTO (Section 15.5 of [RFC7285]). For example, by providing network information in the future in a Calendar, this extension may improve availability of ALTO, when the ALTO server is unavailable but related information is already provided in the Calendar.

For confidentiality of ALTO information, an operator should be cognizant that this extension may introduce a new risk: an ALTO client may get information for future events that are scheduled through Calendaring. Possessing such information, the client may use it to achieve its goal: (1) initiating connections only at advantageous network costs, leading to unexpected network load; (2) generating massive connections to the network at times where its load is expected to be high.

To mitigate this risk, the operator should address the risk of ALTO information being leaked to malicious clients or third parties. As specified in Section 15.3.2 ("Protection Strategies") of [RFC7285], the ALTO server should authenticate ALTO clients and use the
Transport Layer Security (TLS) protocol so that Man In The Middle (MITM) attacks to intercept an ALTO Calendar are not possible. [RFC7285] ensures the availability of such a solution in its Section 8.3.5. "Authentication and Encryption", which specifies that: "ALTO server implementations as well as ALTO client implementations MUST support the "https" URI scheme of [RFC2818] and Transport Layer Security (TLS) of [RFC5246]."

[RFC8446] specifies TLS 1.3 and writes in its section 1: "While TLS 1.3 is not directly compatible with previous versions, all versions of TLS incorporate a versioning mechanism which allows clients and servers to interoperably negotiate a common version if one is supported by both peers". So ALTO clients and servers MAY use newer versions (e.g., 1.3) of TLS as long as the negotiation process succeeds. To ensure backward compatibility with [RFC7285], it is RECOMMENDED for both Calendar-aware Clients and Servers to both support at least TLS 1.2, until it gets deprecated.

To avoid malicious or erroneous guidance from ALTO information, an ALTO client should be cognizant that using calendaring information can have risks: (1) Calendar values, especially in "repeated" Calendars may be only statistical, and (2) future events may change. Hence, a more robust ALTO client should adapt and extend protection strategies specified in Section 15.2 of the base protocol: it should develop self-check and also ensure information update, to reduce the impact of this risk. To address the risk of unexpected ALTO Values changes that the ALTO Client would be unaware of, it is RECOMMENDED that Servers supporting Calendars also support the "ALTO Incremental Updates Using Server-Sent Events (SSE)" Service, specified in [draft-ietf-alto-incr-update-sse]. Likewise, it is RECOMMENDED that Clients using Calendars also support the SSE Service.

7. Operational Considerations

Conveying ALTO Cost Calendars tends to reduce the on-the-wire data exchange volume compared to multiple single cost ALTO transactions. An application using Calendars has a set of time-dependent values upon which it can plan its connections in advance with no need for the ALTO Client to query information at each time. Additionally, the Calendar response attribute "repeated", when provided, saves additional data exchanges in that it indicates that the ALTO Client does not need to query Calendars during a period indicated by this attribute. Unexpected changes during this period can be handled by using the SSE Service as discussed in Section 6, if the Server and the Client support it.

High-resolution intervals may be needed when values change, sometimes during very small time intervals but in a significant manner. A way
to avoid conveying too many entries is to leverage on the "repeated" feature. A server can smartly set the Calendar start time and number of intervals so as to declare them "repeated" for a large number of periods, until the Calendar values change and are conveyed to requesting Clients.

Clients and Servers supporting ALTO Calendars use [RFC8259]. [RFC7285] encodes its requests and responses using the JSON Data Interchange Format specified in [RFC7159]. In the meantime, [RFC7159] has been obsoleted by [RFC8259], that among others makes UTF-8 mandatory for text encoding to improve interoperability. Therefore, ALTO Clients and Servers implementations using UTF-{16,32} need to be cognizant of the subsequent interoperability risks and it is RECOMMENDED for them to switch to UTF-8 encoding, if they want to interoperate with Calendar-aware Servers and Clients.

8. Acknowledgements

The authors would like to thank Fred Baker, Li Geng, Diego Lopez, He Peng and Haibin Song for fruitful discussions and feedback on earlier draft versions. Dawn Chan, Kai Gao, Vijay Gurbani, Yichen Qian and Jensen Zhang provided substantial review feedback and suggestions to the protocol design.

9. References

9.1. Normative References


9.2. Informative References

[draft-ietf-alto-incr-update-sse]
"ALTO Incremental Updates Using Server-Sent Events (SSE) (work in progress)", December 2018.

[draft-ietf-alto-performance-metrics]
"ALTO Performance Cost Metrics (work in progress)", June 2018.

[draft-xiang-alto-multidomain-analytics]

[IEEE.754.2008]


Authors’ Addresses

Sabine Randriamasy
Nokia Bell Labs
Route de Villejust
NOZAY  91460
FRANCE

Email: Sabine.Randriamasy@nokia-bell-labs.com

Richard Yang
Yale University
51 Prospect st
New Haven, CT  06520
USA

Email: yry@cs.yale.edu

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu  210012
China

Email: sunseawq@huawei.com

Lingli Deng
China Mobile
China

Email: denglingli@chinamobile.com
Abstract

The Application-Layer Traffic Optimization (ALTO) [RFC7285] protocol provides network related information, called network information resources, to client applications so that clients can make informed decisions in utilizing network resources. For example, an ALTO server can provide network and cost maps so that an ALTO client can use the maps to determine the costs between network endpoints when choosing communicating endpoints.

However, the ALTO protocol does not define a mechanism to allow an ALTO client to obtain updates to the information resources, other than by periodically re-fetching them. Because some information resources (e.g., the aforementioned maps) may be large (potentially tens of megabytes), and because only parts of the information resources may change frequently (e.g., only some entries in a cost map), complete re-fetching can be extremely inefficient.

This document presents a mechanism to allow an ALTO server to push updates to ALTO clients, to achieve two benefits: (1) updates can be immediate, in that the ALTO server can send updates as soon as they are available; and (2) updates can be incremental, in that if only a small section of an information resource changes, the ALTO server can send just the changes.

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

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 11, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ................................................. 3
2. Major Changes Since Version -01 .......................... 4
3. Terms ...................................................... 5
4. Background ................................................ 6
   4.1. Server-Sent Events (SSEs) ............................ 6
   4.2. JSON Merge Patch ................................... 7
     4.2.1. JSON Merge Patch Encoding ...................... 7
     4.2.2. JSON Merge Patch ALTO Messages ............... 9
   4.3. JSON Patch ........................................... 12
     4.3.1. JSON Patch Encoding ............................ 12
     4.3.2. JSON Patch ALTO Messages ...................... 13
5. Overview of Approach ...................................... 14
6. Update Messages: Data Update and Control Update Messages ... 16
   6.1. ALTO Update Message Format .......................... 16
   6.2. ALTO Data Update Message ............................ 17
   6.3. ALTO Control Update Message ......................... 18
7. Update Stream Service ...................................... 19
   7.1. Media Type ........................................... 19
   7.2. HTTP Method .......................................... 19
   7.3. Accept Input Parameters ............................. 19
   7.4. Capabilities ......................................... 20
The Application-Layer Traffic Optimization (ALTO) [RFC7285] protocol provides network related information called network information resources to client applications so that clients may make informed decisions in utilizing network resources. For example, an ALTO server provides network and cost maps, where a network map partitions the set of endpoints into a manageable number of sets each defined by a Provider-Defined Identifier (PID), and a cost map provides directed costs between PIDs. Given network and cost maps, an ALTO client can
obtain costs between endpoints by first using the network map to get
the PID for each endpoint, and then using the cost map to get the
costs between those PIDs. Such costs can be used by the client to
choose communicating endpoints with low network costs.

The ALTO protocol defines only an ALTO client pull model, without
defining a mechanism to allow an ALTO client to obtain updates to
network information resources, other than by periodically re-fetching
them. In settings where an information resource may be large but
only parts of it may change frequently (e.g., some entries of a cost
map), complete re-fetching can be inefficient.

This document presents a mechanism to allow an ALTO server to push
incremental updates to ALTO clients. Integrating server-push and
incremental updates provides two benefits: (1) updates can be
immediate, in that the ALTO server can send updates as soon as they
are available; and (2) updates can be small, in that if only a small
section of an information resource changes, the ALTO server can send
just the changes.

While primarily intended to provide updates to GET-mode network and
cost maps, the mechanism defined in this document can also provide
updates to POST-mode ALTO services, such as the endpoint property and
endpoint cost services. We intend that the mechanism can also
support new ALTO services to be defined by future extensions, but a
future service needs to satisfy requirements specified in Section 12.

The rest of this document is organized as follows. Section 4 gives
background on the basic techniques used in this design: (1) Server-
Sent Events to allow server push; (2) JSON merge patch and JSON patch
to allow incremental update. With the background, Section 5 gives a
non-normative overview of the design. Section 6 defines individual
messages in an update stream, and Section 7 defines the overall
update stream service; Section 8 defines the stream control service;
Section 9 gives several examples; Section 10 describes operation and
processing considerations by both ALTO servers and clients;
Section 14 discusses two design features that are not supported;
Section 11 discusses security issues; The last two sections review
the requirements for future ALTO services to use SSE and IANA
considerations, respectively.

2. Major Changes Since Version -01

To RFC editor: This will be removed in the final version. We keep
this section to make clear major changes in the technical content.

- Incremental encoding: Added JSON patch as an alternative
  incremental delta encoding.
Client-assigned client-id to allow concurrent updates of the same server resource: The client now assigns a unique client-id to each resource in an update stream. The server puts the client-id in each update event for that resource (before, the server used the server’s resource-id). This allows a client to use one update stream to receive updates to multiple requests for the same server resource, for example, for a POST-mode resource; before, that required separate update streams.

Flexible control: Defined a new "stream control" resource (Section 8) to allow a client to add or remove resources from a previously created update stream. The ALTO server creates a new stream control resource for each update stream instance, assigns a unique URI to it, and sends the URI to the client as the first event in the stream.

3. Terms

This document uses the following terms: Update Stream, Update Stream Server, Update Message, Data Update Message, Full Replacement, Incremental Change, Control Update Message, Stream Control Service, Stream Control.

Update Stream: An update stream is an HTTP connection between an ALTO client and an ALTO server so that the server can push a sequence of update messages using SSE to the client.

Update Stream Server: We refer to an ALTO server providing an update stream as an ALTO update stream server, or update stream server for short. Note that the ALTO server mentioned in this document refers to a general server that provides various kinds of services; it can be an update stream server or stream control server (see below); it can also be a server providing ALTO IRD information.

Update Message: An update message is either a data update message or a control update message.

Data Update Message: A data update message is for a single ALTO information resource and sent from the update stream server to the ALTO client when the resource changes. A data update message can be either a full-replacement message or an incremental-change message. Full replacement is a shorthand for a full-replacement message, and incremental change is a shorthand for an incremental-change message.

Full Replacement: A full replacement for a resource encodes the content of the resource in its original ALTO encoding.
Incremental Change: An incremental change specifies only the difference between the new content and the previous version. An incremental change can be encoded using either JSON merge patch or JSON patch in this document.

Stream Control Service: An stream control service provides an HTTP URI so that the ALTO client of an update stream can use it to send stream control requests on the addition or removal of resources receiving update messages from the update stream.

Stream Control: A shorthand for stream control service.

Stream Control Server: An stream control server providing the stream control service.

Control Update Message: A control update message is a message in an update stream for the update stream server to notify the ALTO client of related control information of the update stream. The first message of an update stream is a control update message and provides the URI using which the ALTO client can send stream control requests to the stream control server. Additional control update messages in an update stream allow the update stream server to notify the ALTO client of status changes (e.g., the server will no longer send updates for an information resource).

4. Background

The design requires two basic techniques: server push and encoding of incremental changes. Using existing techniques whenever possible, this design uses Server-Sent Events (SSEs) for server push; JSON merge patch and JSON patch to encode incremental changes. Below we give a non-normative summary of these two techniques.

4.1. Server-Sent Events (SSEs)

The following is a non-normative summary of SSE; see [SSE] for its normative definition.

Server-Sent Events enable a server to send new data to a client by "server-push". The client establishes an HTTP ([RFC7230], [RFC7231]) connection to the server and keeps the connection open. The server continually sends messages. Each message has one or more lines, where a line is terminated by a carriage-return immediately followed by a new-line, a carriage-return not immediately followed by a new-line, or a new-line not immediately preceded by a carriage-return. A message is terminated by a blank line (two line terminators in a row).
Each line in a message is of the form "field-name: string value". Lines with a blank field-name (that is, lines which start with a colon) are ignored, as are lines which do not have a colon. The protocol defines three field names: event, id, and data. If a message has more than one "data" line, the value of the data field is the concatenation of the values on those lines. There can be only one "event" and "id" line per message. The "data" field is required; the others are optional.

Figure 1 is a sample SSE stream, starting with the client request. The server sends three events and then closes the stream.

(Client request)
GET /stream HTTP/1.1
Host: example.com
Accept: text/event-stream

(Server response)
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/event-stream

event: start
id: 1
data: hello there

event: middle
id: 2
data: let’s chat some more ...
data: and more and more and ...

event: end
id: 3
data: goodbye

Figure 1: A Sample SSE stream.

4.2. JSON Merge Patch

4.2.1. JSON Merge Patch Encoding

To avoid always sending complete data, a server needs mechanisms to encode incremental changes. This design uses JSON merge patch as one mechanism. Below is a non-normative summary of JSON merge patch; see [RFC7396] for the normative definition.

JSON merge patch is intended to allow applications to update server resources via the HTTP patch method [RFC5789]. This document adopts
the JSON merge patch message format to encode incremental changes, but uses a different transport mechanism.

Informally, a JSON merge patch object is a JSON data structure that defines how to transform one JSON value into another. Specifically, JSON merge patch treats the two JSON values as trees of nested JSON objects (dictionaries of name-value pairs), where the leaves are values (e.g., JSON arrays, strings, numbers) other than JSON objects and the path for each leaf is the sequence of keys leading to that leaf. When the second tree has a different value for a leaf at a path, or adds a new leaf, the JSON merge patch tree has a leaf, at that path, with the new value. When a leaf in the first tree does not exist in the second tree, the JSON merge patch tree has a leaf with a JSON "null" value. The JSON merge patch tree does not have an entry for any leaf that has the same value in both versions.

As a result, if all leaf values are simple scalars, JSON merge patch is a quite efficient representation of incremental changes. It is less efficient when leaf values are arrays, because JSON merge patch replaces arrays in their entirety, even if only one entry changes.

Formally, the process of applying a JSON merge patch is defined by the following recursive algorithm, as specified in [RFC7396]:

```
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 delete the element with "name" in the original JSON value.
4.2.2. JSON Merge Patch ALTO Messages

To provide both examples of JSON merge patch and a demonstration of the feasibility of applying JSON merge patch to ALTO, we look at the application of JSON merge patch to two key ALTO messages.

4.2.2.1. JSON Merge Patch Network Map Messages

Section 11.2.1.6 of [RFC7285] defines the format of an ALTO network map message. Assume a simple example ALTO message sending an initial network map:

```json
{
    "meta": {
        "vtag": {
            "resource-id": "my-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" ]
        }
    }
}
```

Consider the following JSON merge patch update message, which (1) adds an ipv4 prefix "193.51.100.0/25" and an ipv6 prefix "2001:db8:8000::/33" to "PID1", (2) deletes "PID2", and (3) assigns a new "tag" to the network map:
Applying the JSON merge patch update to the initial network map is equivalent to the following ALTO network map:

```
{
  "meta": {
    "vtag": {
      "tag": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
    }
  },
  "network-map": {
    "PID1": {
      "ipv4": ["192.0.2.0/24", "198.51.100.0/25",
                "193.51.100.0/25" ],
      "ipv6": ["2001:db8:8000::/33" ]
    },
    "PID2": null
  }
}
```

4.2.2.2. JSON Merge Patch Cost Map Messages

Section 11.2.3.6 of [RFC7285] defines the format of an ALTO cost map message. Assume a simple example ALTO message for an initial cost map:
The following JSON merge patch message updates the example cost map so that (1) the "tag" field of the cost map is updated, (2) the cost of PID1->PID2 is 9 instead of 5, (3) the cost of PID3->PID1 is no longer available, and (4) the cost of PID3->PID3 is defined as 1.

```
{
  "meta": {
    "vtag": {
      "tag": "c0ce023b8678a7b9ec00324673b98e54656d1f6d"
    }
  }
  "cost-map": {
    "PID1": { "PID2": 9 },
    "PID3": { "PID1": null, "PID3": 1 }
  }
}
```

Hence applying the JSON merge patch to the initial cost map is equivalent to the following ALTO cost map:
4.3. JSON Patch

4.3.1. JSON Patch Encoding

One issue of JSON merge patch is that it does not handle array changes well. In particular, JSON merge patch considers an array as a single object and hence can only replace an array in its entirety. When the change is to make a small change to an array such as the deletion of an element from a large array, whole-array replacement is inefficient. Consider the example in Section 4.2.2.1. To add a new entry to the ipv4 array for PID1, the server needs to send a whole new array. Another issue is that JSON merge patch cannot change a value to be null, as the JSON merge patch processing algorithm (MergePatch in Section 4.2.1) interprets a null as a removal instruction. On the other hand, some ALTO resources can have null values, and it is possible that the update will want to change the new value to be null.

JSON patch [RFC6902] can address the preceding issues. It defines a set of operators to modify a JSON object. Below is a non-normative description of JSON patch; see [RFC6902] for the normative definition.
4.3.2. JSON Patch ALTO Messages

To provide both examples of JSON patch and a demonstration of the difference between JSON patch and JSON merge patch, we take a look at the application of JSON patch to the same updates shown in Section 4.2.2.

4.3.2.1. JSON Patch Network Map Messages

First consider the same update as in Section 4.2.2.1 for the network map. Below is the encoding using JSON patch:

```
[
  {
    "op": "replace",
    "path": "/meta/vtag/tag",
    "value": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
  },
  {
    "op": "add",
    "path": "/network-map/PID1/ipv4/2",
    "value": "193.51.100.0/25"
  },
  {
    "op": "add",
    "path": "/network-map/PID1/ipv6",
    "value": ["2001:db8:8000::/33"
  },
  {
    "op": "remove",
    "path": "/network-map/PID2"
  }
]
```

4.3.2.2. JSON Patch Cost Map Messages

Compared with JSON merge patch, JSON patch does not encode cost map updates efficiently. Consider the cost map update shown in Section 4.2.2.2, the encoding using JSON patch is:

```
5. Overview of Approach

With the preceding background, we now give a non-normative overview of the update mechanism to be defined in later sections of this document.

The building block of the update mechanism defined in this document is the update stream service (defined in Section 7), where each update stream service is a POST-mode service that provides update streams. When an ALTO client requests an update stream service, the ALTO client establishes a persistent connection to the update stream server, creating an update stream. The update stream server uses the update stream to continuously send a sequence of update messages (defined in Section 6) to the ALTO client. An update stream can provide updates to both GET-mode resources, such as ALTO network and cost maps, and POST-mode resources, such as ALTO endpoint property service.

An ALTO server may provide any number of update stream services, where each update stream may provide updates for a given subset of the ALTO server’s resources. An ALTO server’s Information Resource Directory (IRD) defines the update stream services and declares the set of resources for which each update stream service provides updates. The ALTO server selects the resource set for each update stream service. It is recommended that if a resource depends on one or more other resource(s) (indicated with the "uses" attribute
defined in [RFC7285]), these other resource(s) should also be part of that update stream. Thus the update stream for a cost map should also provide updates for the network map on which that cost map depends.

An ALTO client may request any number of update streams simultaneously. Because each update stream consumes resources on the update stream server, an update stream server may require client authorization and/or authentication, limit the number of open update streams, close inactive streams, or redirect an ALTO client to another update stream server.

The key objective of an update stream is to update the ALTO client on data value changes to ALTO resources. We refer to messages sending such updates as data update messages. Although an update stream may update one or more ALTO resources, each data update message updates only one resource and is sent as a Server-Sent Event (SSE), as defined by [SSE]. A data update message is encoded either as a full replacement or as an incremental change. A full replacement uses the JSON message format defined by the ALTO protocol. There can be multiple encodings for incremental changes. The current design supports incremental changes using JSON merge patch ([RFC7396]) or JSON patch ([RFC6902]) to describe the changes of the resource. Future documents may define additional mechanisms for incremental changes. The update stream server decides when to send data update messages, and whether to send full replacements or incremental changes. These decisions can vary from resource to resource and from update to update.

An update stream can run for a long time, and hence there can be status changes at the update stream server side during the lifetime of an update stream; for example, the update stream server may encounter an error or need to shut down for maintenance. To support robust, flexible protocol design, this design allows the update stream server to send server control updates (vs. data updates) to the ALTO client as well, showing as control update messages from the update stream server to the ALTO client.
Figure 2: ALTO SSE Architecture.

In addition to control changes triggered from the update stream server side, in a flexible design, an ALTO client may initiate control changes as well, in particular, by adding or removing ALTO resources receiving updates. An ALTO client initiates such changes using the stream control service. Although one may use a design that the client uses the same HTTP connection to send the control requests, it requires stronger server support such as HTTP pipeline. For more flexibility, this document introduces stream control service. In particular, the update stream server of an update stream uses the first message to provide the URI of the stream control service. The ALTO client can then use the URI to ask the update stream server to (1) send data update messages for additional resources, (2) stop sending data update messages for previously requested resources, or (3) gracefully stop and close the update stream altogether. Figure 2 shows the complete ALTO SSE architecture.

6. Update Messages: Data Update and Control Update Messages

We now define the details of ALTO SSE. Specifically, an update stream consists of a stream of data update messages (Section 6.2) and control update messages (Section 6.3).

6.1. ALTO Update Message Format

Data update and control update messages have the same basic structure. The data field is a JSON object, and the event field contains the media type of the data field and an optional client-id. Data update messages use the client-id to identify the ALTO resource to which the update message applies. Client-ids MUST follow the
rules for ALTO ResourceIds (Section 10.2 of [RFC7285]). Client-ids
MUST be unique within an update stream, but need not be globally
unique. For example, if an ALTO client requests updates for both a
cost map and its dependent network map, the ALTO client might assign
client-id "1" to the network map and client-id "2" to the cost map.
Alternatively, the ALTO client could use the client-ids for those two
maps.

JSON specifications use the type ClientId for a client-id, and the
type ClientId conforms to the specification of ResourceId as defined
in Section 10.2 of [RFC7285].

The two sub-fields (media-type and client-id) of the event field are
encoded as comma-separated strings:

    media-type [ ',', client-id ]

The media-type name used by ALTO SSE MUST NOT contain a comma
(character code 0x2c). [TODO: conform to Section 4.2 of [RFC6838];
confirm with IANA.]

Note that an update message does not use the SSE "id" field. [TODO:
Discuss to move client-id to SSE id?]

6.2. ALTO Data Update Message

A data update message is sent when a monitored resource changes. The
data is either a complete specification of the resource, or else a
patch (either JSON merge patch or JSON patch) describing the changes
from the last version. We will refer to these as full replacement
and incremental change, respectively. The encoding of full
replacement is defined by [RFC7285]; examples are network and cost
map messages. They have the media types defined in that document.
The encoding of JSON merge patch is defined by [RFC7396], with media
type "application/merge-patch+json"; the encoding of JSON patch is
defined by [RFC6902], with media type "application/json-patch+json".

Figure 3 shows some examples of ALTO data update messages:
event: application/alto-networkmap+json,1
data: { ... full network map message ... }

event: application/alto-costmap+json,2
data: { ... full cost map message ... }

event: application/merge-patch+json,2
data: { ... JSON merge patch update for the cost map ... }

Figure 3: Examples of ALTO data update messages.

6.3. ALTO Control Update Message

Control update messages have the media type "application/alto-updatestreamcontrol+json", and the data is of type UpdateStreamControlEvent:

object {
  [String control-uri;]
  [ClientId started<1..*>;]
  [ClientId stopped<1..*>;]
  [String description;]
} UpdateStreamControlEvent;

control-uri: the URI providing stream control for this update stream (see Section 8). The server MUST send a control update message with an URI as the first event in an update stream. If the URI is NULL, the update stream server does not support stream control for this update stream; otherwise, the update stream server provides stream control through the given URI.

started: a list of client-ids of resources. It notifies the ALTO client that the update stream server will start sending data update messages for each resource listed.

stopped: a list of client-ids of resources. It notifies the ALTO client that the update stream server will no longer send data update messages for the listed resources. There can be multiple reasons for an update stream server to stop sending data update messages for a resource, including a request from the ALTO client using stream control (Section 7.7.1) or an internal server event.

description: a non-normative text providing an explanation for the control event. When an update stream server stops sending data update messages for a resource, it is RECOMMENDED that the update stream server use the description field to provide details.
7. Update Stream Service

An update stream service returns a stream of update messages, as defined in Section 6. An ALTO server’s IRD (Information Resource Directory) MAY define one or more update stream services, which ALTO clients use to request new update stream instances.

7.1. Media Type

The media type of an ALTO update stream service is "text/event-stream", as defined by [SSE].

7.2. HTTP Method

An ALTO update stream service is requested using the HTTP POST method.

7.3. Accept Input Parameters

An ALTO client specifies the parameters for the new update stream by sending an HTTP POST body with the media type "application/alto-updatestreamparams+json". That body contains a JSON Object of type UpdateStreamReq, where:

```json
object {
    [AddUpdatesReq   add;]
    [ClientId        remove<0..*>;]
} UpdateStreamReq;

object-map {
    ClientId -> AddUpdateReq;
} AddUpdatesReq;

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

add: specifies the resources for which the ALTO client wants updates, and has one entry for each resource. An ALTO client creates a unique client-id (Section 6.1) for each such resource, and uses those client-ids as the keys in the "add" field.

resource-id: the resource-id of an ALTO resource, and MUST be in the update stream’s "uses" list (Section 8.5.2 of Section 7.5). If the resource-id is a GET-mode resource with a version tag (or
"vtag"), as defined in Section 6.3 and Section 10.3 of [RFC7285], and the ALTO client has previously retrieved a version of that resource from the update stream server, the ALTO client MAY set the "tag" field to the tag part of the client’s version of that resource. If that version is not current, the update stream server MUST send a full replacement before sending any incremental changes, as described in Section 7.7.1. If that version is still current, the update stream server MAY omit the initial full replacement.

incremental-changes: the ALTO client specifies whether it is willing to receive incremental changes from the update stream server for a specific resource-id. If the "incremental-changes" field for a resource-id is "true", the update stream server MAY send incremental changes for that resource (assuming the update stream server supports incremental changes for) that resource; see Section 7.4). If the "incremental-changes" field is "false", the update stream server MUST NOT send incremental changes for that resource. The default value for "incremental-changes" is "true", so to suppress incremental changes, the ALTO client MUST explicitly set "incremental-changes" to "false". Note that the ALTO client cannot suppress full replacement. When the ALTO client sets "incremental-changes" to "false", the update stream server MUST send a full replacement instead of an incremental change to the ALTO client. The update stream server MAY wait until more changes are available, and send a single full replacement with those changes. Thus an ALTO client which declines to accept incremental changes may not get updates as quickly as an ALTO client which does.

input: If the resource is a POST-mode service which requires input, the ALTO client MUST set the "input" field to a JSON Object with the parameters that resource expects.

remove: it is used in update stream control requests (Section 8), and is not allowed in the update stream request. The update stream server SHOULD ignore this field if it is included in the request.

If a request has any errors, the update stream server MUST NOT create an update stream. Also, the update stream server will send an error response to the ALTO client as specified in Section 7.6.

7.4. Capabilities

The capabilities are defined as an object of type UpdateStreamCapabilities:
If this update stream can provide data update messages with incremental changes for a resource, the "incremental-change-media-types" field has an entry for that resource-id, and the value is the media-type of the incremental change. Normally this will be "application/merge-patch+json", "application/json-patch+json", or "application/merge-patch+json,application/json-patch+json", because, as described in Section 6, they are the only incremental change types defined by this document. However future extensions may define other types of incremental changes.

When choosing the media-type to encode incremental changes for a resource, the update stream server SHOULD consider the limitations of the encoding. For example, when a JSON merge patch specifies that the value of a field is null, its semantics is that the field is removed from the target, and hence the field is no longer defined (i.e., undefined); see the MergePatch algorithm in Section 4.2.1 on how null value is processed. This, however, may not be the intended result for the resource, when null and undefined have different semantics for the resource. In such a case, the update stream server SHOULD choose JSON patch over JSON merge patch.

The "support-stream-control" field specifies whether the given update stream supports stream control. If "support-stream-control" field is "true", the update stream server will uses the stream control specified in this document; else, the update stream server may use other mechanisms to provide the same functionality as stream control.

7.5. Uses

The "uses" attribute MUST be an array with the resource-ids of every resource for which this update stream can provide updates. Each resource specified in the "uses" MUST support full replacement: the update stream server can always send full replacement, and the ALTO client MUST accept full replacement.

This set may be any subset of the ALTO server’s resources, and may include resources defined in linked IRDs. However, it is RECOMMENDED that the ALTO server selects a set that is closed under the resource dependency relationship. That is, if an update stream’s "uses" set
includes resource R1, and resource R1 depends on ("uses") resource R0, then the update stream’s "uses" set SHOULD include R0 as well as R1. For example, an update stream for a cost map SHOULD also provide updates for the network map upon which that cost map depends.

7.6. Response

If the update stream request has any errors, the update stream server MUST return an HTTP "400 Bad Request" to the ALTO client. The body part of the HTTP response is the JSON object defined in Section 8.5.2 in [RFC7285]. Hence, an ALTO error response has the format:

HTTP/1.1 400 Bad Request
Content-Length: [TBD]
Content-Type: application/alto-error+json
Connection: Closed

{
    "meta":{
        "code": "***",
        "field": "***",
        "value": "***"
    }
}

Note that "field" and "value" are optional fields. If the "value" field exists, the "field" field MUST exist.

- If an update stream request does not have an "add" field specifying one or more resources, the error code of the error message MUST be E_MISSING_FIELD and the "field" field SHOULD be "add". The update stream server MUST close the stream without sending any events.

- If the "resource-id" field is invalid, or is not associated with the update stream, the error code of the error message MUST be E_INVALID_FIELD_VALUE; the "field" field SHOULD be "resource-id" and the "value" field SHOULD be the invalid resource-id. If there are more than one invalid resource-ids, the update stream server SHOULD pick one and return it. The update stream server MUST close the stream without sending any events.

- If the resource is a POST-mode service which requires input, the client MUST set the "input" field to a JSON Object with the parameters that that resource expects. If the "input" field is missing or invalid, the update stream server MUST return the same error response that that resource would return for missing or
invalid input (see [RFC7285]). In this case, the update stream server MUST close the update stream without sending any events. If the input for several POST-mode resources are missing or invalid, the update stream server MUST pick one and return it.

The response to a valid request is a stream of update messages. Section 6 defines the update messages, and [SSE] defines how they are encoded into a stream.

An update stream server SHOULD send updates only when the underlying values change. However, it may be difficult for an update stream server to guarantee that in all circumstances. Therefore a client MUST NOT assume that an update message represents an actual change.

7.7. Additional Requirements on Update Messages

7.7.1. Event Sequence Requirements

- The first event MUST be a control update message with the URI of the update stream control service Section 8 for this update stream.

- As soon as possible after the ALTO client initiates the connection, the update stream server MUST send a full replacement for each resource-id requested with a version tag. In this case the update stream server MAY omit the initial full replacement for that resource, if the "tag" field the ALTO client provided for that resource-id matches the tag of the update stream's current version.

- If this update stream provides update for resource-ids and R0 and R1, and if R1 depends on R0, then the update stream server MUST send the update for R0 before sending the related updates for R1. For example, suppose an update stream provides updates to a network map and its dependent cost maps. When the network map changes, the update stream server MUST send the network map update before sending the cost map updates.

- When the ALTO client uses the stream control service to stop updates for one or more resources Section 8, the ALTO client MUST send a stream control request. The update stream server MUST send a control update message whose "stopped" field has the client-ids of all active resources.
7.7.2. Cross-Stream Consistency Requirements

If several ALTO clients create multiple update streams for updates to the same resource, the update stream server MUST send the same updates to all of them. However, the update stream server MAY pack data items into different patch events, as long as the net result of applying those updates is the same.

For example, suppose two different ALTO clients create update streams for the same cost map, and suppose the update stream server processes three separate cost point updates with a brief pause between each update. The server MUST send all three new cost points to both clients. But the update stream server MAY send a single patch event (with all three cost points) to one ALTO client, while sending three separate patch events (with one cost point per event) to the other ALTO client.

A update stream server MAY offer several different update stream resources that provide updates to the same underlying resource (that is, a resource-id may appear in the "uses" field of more than one update stream resource). In this case, those update stream resources MUST return the same update data.

7.8. Keep-Alive Messages

In an SSE stream, any line which starts with a colon (U+003A) character is a comment, and an ALTO client MUST ignore that line ([SSE]). As recommended in [SSE], an update stream server SHOULD send a comment line (or an event) every 15 seconds to prevent ALTO clients and proxy servers from dropping the HTTP connection.

8. Stream Control Service

An stream control service allows an ALTO client to remove resources from the set of resources that are monitored by an update stream, or add additional resources to that set. The service also allows an ALTO client to gracefully shut down an update stream.

When an update stream server creates a new update stream, and if the update stream server supports stream control for the update stream, the update stream server creates a stream control service for that update stream. An ALTO client uses the stream control service to remove resources from the update stream instance, or to request updates for additional resources. An ALTO client cannot obtain the stream control service through the IRD. Instead, the first event that the update stream server sends to the ALTO client has the URI for the associated stream control service (see Section 6.3).
Each stream control request is an individual HTTP request. If the ALTO client and the stream control server the ALTO client MAY send multiple stream control requests to the stream control server using the same HTTP connection.

8.1. URI

The URI for an stream control service, by itself, MUST uniquely specify the update stream instance which it controls. The stream control server MUST NOT use other properties of an HTTP request, such as cookies or the client’s IP address, to determine the update stream. Furthermore, an update stream server MUST NOT reuse a control service URI once the associated update stream has been closed.

The ALTO client MUST evaluate a non-absolute control URI (for example, a URI without a host, or with a relative path) in the context of the URI used to create the update stream. The stream control service’s host MAY be different from the update stream’s host.

It is expected that the update stream server will assign a unique stream id to each update stream instance and will embed that id in the associated stream control URI. However, the exact mechanism is left to the update stream server. ALTO clients MUST NOT attempt to deduce a stream id from the control URI.

To prevent an attacker from forging a stream control URI and sending bogus requests to disrupt other update streams, stream control URIs SHOULD contain sufficient random redundancy to make it difficult to guess valid URIs.

8.2. Media Type

An ALTO stream control response does not have a specific media type.

8.3. HTTP Method

An ALTO update stream control resource is requested using the HTTP POST method.

8.4. Accept Input Parameters

An stream control service accepts the same input media type and input parameters as the update stream service (Section 7.3). The only difference is that a stream control service also accepts the "remove" field.
If specified, the "remove" field is an array of client-ids the ALTO client previously added to this update stream. An empty "remove" array is equivalent to a list of all currently active resources; the update stream server responds by removing all resources and closing the stream.

An ALTO client MAY use the "add" field to add additional resources. However, the ALTO client MUST assign a unique client-id to each resource. Client-ids MUST be unique over the lifetime of this update stream: an ALTO client MUST NOT reuse a previously removed client-id.

If a request has any errors, the update stream server MUST NOT add or remove any resources from the associated update stream. Also, the stream control server will return an error response to the client as specified in Section 8.6.

8.5. Capabilities & Uses

None (Stream control services do not appear in the IRD).

8.6. Response

The stream control server MUST process the "add" field before the "remove" field. If the request removes all active resources without adding any additional resources, the update stream server MUST close the update stream. Thus an update stream cannot have zero resources.

If the request has any errors, the stream control server MUST return an HTTP "400 Bad Request" to the ALTO client. The body part of the HTTP response is the JSON object defined in Section 8.5.2 in [RFC7285]. An error response has the same format as specified in Section 7.6. Detailed error code and error information are specified as below.

- If the "add" request does not satisfy the requirements in Section 7.3, the stream control server MUST return the ALTO error message defined in Section 7.6.
- If any client-id in the "remove" field was not added in a prior request, the error code of the error message MUST be E_INVALID_FIELD_VALUE; the "field" field SHOULD be "remove" and the "value" field SHOULD be the array of the invalid client-ids. Thus it is illegal to "add" and "remove" the same client-id in the same request. However, it is legal to remove a client-id twice.
- If any client-id in the "add" field has been used before in this stream, the error code of the error message MUST be
E_INVALID_FIELD_VALUE, the "field" field SHOULD be "add" and the "value" field SHOULD be the array of invalid client-ids.

- If the request has a non-empty "add" field and a "remove" field with an empty list of client-ids (to replace all active resources with a new set, the client MUST explicitly enumerate the client-ids to be removed), the error code of the error message MUST be E_INVALID_FIELD_VALUE; the "field" field SHOULD be "remove" and the "value" field SHOULD be an empty array.

If the request is valid but the associated update stream has been closed. The stream control server MUST return an HTTP "404 Not Found".

If the request is valid and the stream control server successfully processes the request without error, the stream control server should return either an HTTP "202 Accepted" response or an HTTP "204 No Content" response. The difference is that for the latter case, the stream control server is sure that the update stream server has also processed the request. Regardless of 202 or 204 HTTP response, the final updates of related resources will be notified by the update stream server using its control update message(s), due to our modular design.

9. Examples

9.1. Example: IRD Announcing Update Stream Services

Below is an example IRD announcing two update stream services. The first, which is named "update-my-costs", provides updates for the network map, the "routingcost" and "hopcount" cost maps, and a filtered cost map resource. The second, which is named "update-my-prop", provides updates to the endpoint properties service.

Note that in the "update-my-costs" update stream shown in the example IRD, the update stream server uses JSON patch for network map, and it uses JSON merge patch to update the other resources. Also, the update stream will only provide full replacements for "my-simple-filtered-cost-map".

Also, note that this IRD defines two filtered cost map resources. They use the same cost types, but "my-filtered-cost-map" accepts cost constraint tests, while "my-simple-filtered-cost-map" does not. To avoid the issues discussed in Section 10.4, the update stream provides updates for the second, but not the first.

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

9.2. Example: Simple Network and Cost Map Updates

Given the update streams announced in the preceding example IRD, below we show an example of an ALTO client’s request and the update stream server’s immediate response, using the update stream resource "update-my-costs". In the example, the ALTO client requests updates for the network map and "routingcost" cost map, but not for the "hopcount" cost map. The ALTO client uses the ALTO server’s resource-ids as the client-ids. Because the client does not provide a "tag" for the network map, the update stream server must send a full replacement for the network map as well as for the cost map. The ALTO client does not set "incremental-changes" to "false", so it defaults to "true". Thus, the update stream server will send patch updates for the cost map and the network map.
POST /updates/costs HTTP/1.1
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

{  
  "add": {
    "my-network-map": {
      "resource-id": "my-network-map"
    },
    "my-routingcost-map": {
      "resource-id": "my-routingcost-map"
    }
  }
}

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

event: application/alto-updatestreamcontrol+json
data: {"control-uri":
data: "http://alto.example.com/updates/streams/3141592653589"}

event: application/alto-networkmap+json,my-network-map
data: {
data:  
  "meta": {
  "vtag": {
  "resource-id": "my-network-map",
data:     "tag": "da65eca2eb7a10ce8b059740b0b2eb3f8eb1d4785"
data:   },
data:       "network-map": {
  "PID1": {
  "ipv4": ["192.0.2.0/24", "198.51.100.0/25" ]
data:   },
data:   "PID2": {
  "ipv4": ["198.51.100.128/25" ]
data:   },
data:   "PID3": {
  "ipv4": ["0.0.0.0/0" ],
data:  "ipv6": [ "::/0" ]
data: 
  }
data: }
data: }

event: application/alto-costmap+json,my-routingcost-map
data: {
  data: "meta": {
    data: "dependent-vtags": [{
      data: "resource-id": "my-network-map",
      data: "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
    }],
    data: "cost-type": {
      data: "cost-mode": "numerical",
      data: "cost-metric": "routingcost"
    },
    data: "vtag": {
      data: "resource-id": "my-routingcost-map",
      data: "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"
    }
  },
  data: "cost-map": {
    data: "PID1": { "PID1": 1, "PID2": 5, "PID3": 10 },
    data: "PID2": { "PID1": 5, "PID2": 1, "PID3": 15 },
    data: "PID3": { "PID1": 20, "PID2": 15 }
  }
}

After sending those events immediately, the update stream server will send additional events as the maps change. For example, the following represents a small change to the cost map. PID1->PID2 is changed to 9 from 5, PID3->PID1 is no longer available and PID3->PID3 is now defined as 1:

```
event: application/merge-patch+json,my-routingcost-map
data: {
  data: "meta": {
    data: "vtag": {
      data: "tag": "c0ce023b8678a7b9ec00324673b98e54656d1f6d"
    }
  },
  data: "cost-map": {
    data: "PID1": { "PID2": 9 },
    data: "PID3": { "PID1": null, "PID3": 1 }
  }
}
```

As another example, the following represents a change to the network map: an ipv4 prefix "193.51.100.0/25" is added to PID1. It triggers changes to the cost map. The update stream server chooses to send an incremental change for the network map and send a full replacement instead of an incremental change for the cost map:
event: application/json-patch+json,my-network-map
data: {
data:   {
data:     "op": "replace",
data:     "path": "/meta/vtag/tag",
data:     "value": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
data:   },
data:   {
data:     "op": "add",
data:     "path": "/network-map/PID1/ipv4/2",
data:     "value": "193.51.100.0/25"
data:   }
data: }

event: application/alto-costmap+json,my-routingcost-map
data: {
data:   "meta": {
data:     "vtag": {
data:       "tag": "c0ce023b8678a7b9ec00324673b98e54656d1f6d"
data:     }
data:   },
data:   "cost-map": {
data:     "PID1": { "PID1": 1, "PID2": 3, "PID3": 7 },
data:     "PID2": { "PID1": 12, "PID2": 1, "PID3": 9 },
data:     "PID3": { "PID1": 14, "PID2": 8 }ndata:   }
data: }

9.3. Example: Advanced Network and Cost Map Updates

This example is similar to the previous one, except that the ALTO client requests updates for the "hopcount" cost map as well as the "routingcost" cost map and provides the current version tag of the network map, so the update stream server is not required to send the full network map data update message at the beginning of the stream. In this example, the client uses the client-ids "net", "routing" and "hops" for those resources. The update stream server sends the stream control URI and the full cost maps, followed by updates for the network map and cost maps as they become available:
POST /updates/costs HTTP/1.1
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

{
    "add": {
        "net": {
            "resource-id": "my-network-map",
            "tag": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
        },
        "routing": {
            "resource-id": "my-routingcost-map"
        },
        "hops": {
            "resource-id": "my-hopcount-map"
        }
    }
}

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

event: application/alto-updatestreamcontrol+json
data: {"control-uri":
data: "http://alto.example.com/updates/streams/2718281828459"}

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

event: application/alto-costmap+json,hops
data: { ... full hopcount cost map message ... }

(event pause)

event: application/merge-patch+json,routing
data: {"cost-map": {"PID2" : {"PID3" : 31}}}

event: application/merge-patch+json,hops
data: {"cost-map": {"PID2" : {"PID3" : 4}}}

If the ALTO client wishes to stop receiving updates for the
"hopcount" cost map, the ALTO client can send a "remove" request on
the stream control URI:

Roome & Yang           Expires September 11, 2019              [Page 33]
POST /updates/streams/2718281828459" HTTP/1.1
Host: alto.example.com
Accept: text/plain,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

{
    "remove": [ "hops" ]
}

HTTP/1.1 204 No Content
Content-Length: 0

(stream closed without sending data content)

The update stream server sends a "stopped" control update message on
the original request stream to inform the ALTO client that updates
are stopped for that resource:

event: application/alto-updatestreamcontrol+json
data: {
    data: "stopped": ["hops"]
data: }

Below is an example of an invalid stream control request. The
"remove" field of the request includes an undefined client-id and the
stream control server will return an error response to the ALTO
client.
If the ALTO client no longer needs any updates, and wishes to shut the update stream down gracefully, the client can send a "remove" request with an empty array:

```
POST /updates/streams/2718281828459 HTTP/1.1
Host: alto.example.com
Accept: text/plain,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###
{
   "remove": [ ]
}
```

HTTP/1.1 204 No Content
Content-Length: 0

(stream closed without sending data content)

The update stream server sends a final control update message on the original request stream to inform the ALTO client that all updates are stopped and then closes the stream:
9.4. Example: Endpoint Property Updates

As another example, here is how an ALTO client can request updates for the property "priv:ietf-bandwidth" for one set of endpoints and "priv:ietf-load" for another. The update stream server immediately sends full replacements with the property values for all endpoints. After that, the update stream server sends data update messages for the individual endpoints as their property values change.

POST /updates/properties HTTP/1.1
Host: alto.example.com
Accept: text/event-stream
Content-Type: application/alto-updatestreamparams+json

{ "add": {
  "props-1": {
    "resource-id": "my-props",
    "input": {
      "properties": [ "priv:ietf-bandwidth" ],
      "endpoints": [
        "ipv4:198.51.100.1",
        "ipv4:198.51.100.2",
        "ipv4:198.51.100.3"
      ]
    }
  },
  "props-2": {
    "resource-id": "my-props",
    "input": {
      "properties": [ "priv:ietf-load" ],
      "endpoints": [
        "ipv6:2001:db8:100::1",
        "ipv6:2001:db8:100::2",
        "ipv6:2001:db8:100::3"
      ]
    }
  }
}
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/event-stream

event: application/alto-updatestreamcontrol+json
data: {"control-uri":
data: "http://alto.example.com/updates/streams/1414213562373"}

event: application/alto-endpointprops+json,props-1
data: { "endpoint-properties": {
data: "ipv4:198.51.100.1" : { "priv:ietf-bandwidth": "13" },
data: "ipv4:198.51.100.2" : { "priv:ietf-bandwidth": "42" },
data: "ipv4:198.51.100.3" : { "priv:ietf-bandwidth": "27" }
data: } }

event: application/alto-endpointprops+json,props-2
data: { "endpoint-properties": {
data: "ipv6:2001:db8:100::1" : { "priv:ietf-load": "8" },
data: "ipv6:2001:db8:100::2" : { "priv:ietf-load": "2" },
data: "ipv6:2001:db8:100::3" : { "priv:ietf-load": "9" }
data: } }

(pause)

event: application/merge-patch+json,props-1
data: { "endpoint-properties":
data: {"ipv4:198.51.100.1" : {"priv:ietf-bandwidth": "3"}}
data: }

(pause)

event: application/merge-patch+json,props-2
data: { "endpoint-properties":
data: {"ipv6:2001:db8:100::3" : {"priv:ietf-load": "7"}}
data: }

If the ALTO client needs the "bandwidth" property for additional endpoints, the ALTO client can send an "add" request on the stream control URI:
POST /updates/streams/1414213562373" HTTP/1.1
Host: alto.example.com
Accept: text/plain,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

{
    "add": {
        "props-3": {
            "resource-id": "my-props",
            "input": {
                "properties": [ "priv:ietf-bandwidth" ],
                "endpoints": [
                    "ipv4:198.51.100.4",
                    "ipv4:198.51.100.5",
                ]
            }
        },
        "props-4": {
            "resource-id": "my-props",
            "input": {
                "properties": [ "priv:ietf-load" ],
                "endpoints": [
                    "ipv6:2001:db8:100::4",
                    "ipv6:2001:db8:100::5",
                ]
            }
        }
    }
}

HTTP/1.1 204 No Content
Content-Length: 0

(stream closed without sending data content)

The update stream server sends full replacements for the two new resources, followed by incremental changes for all four requests as they arrive:
event: application/alto-endpointprops+json,props-3
data: { "endpoint-properties": {
data:    "ipv4:198.51.100.4" : { "priv:ietf-bandwidth": "25" },
data:    "ipv4:198.51.100.5" : { "priv:ietf-bandwidth": "31" },
data: } }

(event)

event: application/alto-endpointprops+json,props-4
data: { "endpoint-properties": {
data:    "ipv6:2001:db8:100::4" : { "priv:ietf-load": "6" },
data:    "ipv6:2001:db8:100::5" : { "priv:ietf-load": "4" },
data: } }

(event)

(event)

(event)

(event)

10. Operation and Processing Considerations

10.1. Considerations for Choosing SSE Line Lengths

SSE was designed for events that consist of relatively small amounts of line-oriented text data, and SSE clients frequently read input one line-at-a-time. However, an update stream sends full cost maps as single events, and a cost map may involve megabytes, if not tens of megabytes, of text. This has implications for both the update stream server and the ALTO Client.

First, SSE clients might not be able to handle a multi-megabyte data "line". Hence it is RECOMMENDED that an update stream server limit data lines to at most 2,000 characters.
Second, some SSE client packages read all the data for an event into memory, and then present it to the client as a single character array. However, a client computer may not have enough memory to hold the entire JSON text for a large cost map. Hence an ALTO client SHOULD consider using an SSE library which presents the event data in manageable chunks, so the ALTO client can parse the cost map incrementally and store the underlying data in a more compact format.

10.2. Considerations for Choosing Data Update Messages

The choice on data update messages depends on both how frequently the resources will change, and how extensive those changes will be. For stable resources with minor changes, the update stream server may choose to send incremental changes; for resources that frequently change, the update stream server may choose to send a full replacement after a while. Whether to send full replacement or incremental change depends on the update stream server.

For incremental updates, this design allows both JSON patch and JSON merge patch for incremental changes. JSON merge patch is clearly superior to JSON patch for describing incremental changes to Cost Maps, Endpoint Costs, and Endpoint Properties. For these data structures, JSON merge patch is more space-efficient, as well as simpler to apply; we see no advantage to allowing a server to use JSON patch for those resources.

The case is not as clear for incremental changes to network maps.

First, consider small changes such as moving a prefix from one PID to another. JSON patch could encode that as a simple insertion and deletion, while JSON merge patch would have to replace the entire array of prefixes for both PIDs. On the other hand, to process a JSON patch update, the ALTO client would have to retain the indexes of the prefixes for each PID. Logically, the prefixes in a PID are an unordered set, not an array; aside from handling updates, a client has no need to retain the array indexes of the prefixes. Hence to take advantage of JSON patch for network maps, ALTO clients would have to retain additional, otherwise unnecessary, data.

Second, consider more involved changes such as removing half of the prefixes from a PID. JSON merge patch would send a new array for that PID, while JSON patch would have to send a list of remove operations and delete the prefix one by one.

Therefore, each update stream server may decide on its own whether to use JSON merge patch or JSON patch according to the changes in network maps.
Other JSON-based incremental change formats may be introduced in the future.

10.3. Considerations for Client Processing Data Update Messages

In general, when an ALTO client receives a full replacement for a resource, the ALTO client should replace the current version with the new version. When an ALTO client receives an incremental change for a resource, the ALTO 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 if the resource on which it depends has changed. There are at least two ways an ALTO client can do that. We will illustrate these techniques by referring to network and cost map messages, although these techniques apply to any dependent resources.

Note that when a network map changes, the update stream server MUST send the network map update message before sending the updates for the dependent cost maps (see Section 7.7.1).

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 ALTO client receives the update messages for all dependent cost maps. The ALTO client then applies all network and cost map updates atomically.

Alternatively, the ALTO client MAY update the network map immediately. In this case, the ALTO client MUST mark each dependent cost map as temporarily invalid and MUST NOT use that map until the ALTO client receives a cost map update message with the new network map version tag. Note that the ALTO client MUST NOT delete the cost maps, because the update stream server may send incremental changes.

The update stream server SHOULD send updates for dependent resources in a timely fashion. However, if the ALTO client does not receive the expected updates, the ALTO client MUST close the update stream connection, discard the dependent resources, and reestablish the update stream. The ALTO 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 update stream server will not re-send that resource.

Although not as efficient as possible, this recovery method is simple and reliable.
10.4. Considerations for Updates to Filtered Cost Maps

If an update stream provides updates to a Filtered cost map which allows constraint tests, then an ALTO client MAY request updates to a Filtered cost map request with a constraint test. In this case, when a cost changes, the update stream server MUST send an update if the new value satisfies the test. If the new value does not, whether the update stream server sends an update depends on whether the previous value satisfied the test. If it did not, the update stream server SHOULDN'T send an update to the ALTO client. But if the previous value did, then the update stream server MUST send an update with a "null" value, to inform the ALTO client that this cost no longer satisfies the criteria.

An update stream server can avoid such issues by offering update streams only for filtered cost maps which do not allow constraint tests.

10.5. Considerations for Updates to Ordinal Mode Costs

For an ordinal mode cost map, a change to a single cost point may require updating many other costs. As an extreme example, suppose the lowest cost changes to the highest cost. For a numerical mode cost map, only that one cost changes. But for an ordinal mode cost map, every cost might change. While this document allows an update stream server to offer incremental updates for ordinal mode cost maps, update stream server implementors should be aware that incremental updates for ordinal costs are more complicated than for numerical costs, and ALTO clients should be aware that small changes may result in large updates.

An update stream server can avoid this complication by only offering full replacements for ordinal cost maps.

11. Security Considerations

As an extension of the base ALTO protocol [RFC7285], this document fits into the architecture of the base protocol, and hence the Security Considerations (Section 15) of the base protocol fully apply when this extension is provided by an ALTO server. For example, the same authenticity and integrity considerations (Section 15.1 of [RFC7285]) still fully apply; the same considerations for the privacy of ALTO users (Section 15.4 of [RFC7285]) also still fully apply.

The addition of update streams and stream control can introduce additional risks which we discuss below.
11.1. Update Stream Server: Denial-of-Service Attacks

Allowing persistent update stream connections enables a new class of Denial-of-Service attacks.

For the update stream server, an ALTO client might create an unreasonable number of update stream connections, or add an unreasonable number of client-ids to one update stream.

To avoid these attacks on the update stream server, the server MAY choose to limit the number of active streams and reject new requests when that threshold is reached. An update stream server MAY also choose to limit the number of active client-ids on any given stream, or limit the total number of client-ids used over the lifetime of a stream, and reject any stream control request which would exceed those limits. In these cases, the update stream server SHOULD return the HTTP status "503 Service Unavailable".

While the preceding techniques prevent update stream DoS attacks from disrupting an update stream server’s other services, it does make it easier for a DoS attack to disrupt the update stream service. Therefore an update stream server may prefer to restrict update stream services to authorized clients, as discussed in Section 15 of [RFC7285].

Alternatively, an update stream 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.

11.2. ALTO Client: Update Overloading or Instability

The availability of continuous updates can also cause overload for an ALTO client, in particular an ALTO client with limited processing capabilities. The current design does not include any flow control mechanisms for the client to reduce the update rates from the server. Under overloading, the client may choose to remove the information resources with high update rates.

Also, under overloading, the client may no longer be able to detect whether an information is still fresh or has become stale. In such a case, the client should be careful in how it uses the information to avoid stability or efficiency issues.

11.3. Stream Control: Spoofed Control Requests

An outside party which can read the update stream response, or which can observe stream control requests, can obtain the control URI and use that to send a fraudulent "remove" requests, thus disabling
updates for the valid ALTO client. This can be avoided by encrypting
the update stream and stream control requests (see Section 15 of
[RFC7285]). Also, the update stream server echoes the "remove"
requests on the update stream, so the valid ALTO client can detect
unauthorized requests.

12. Requirements on Future ALTO Services to Use this Design

Although this design is quite flexible, it has underlying
requirements.

The key requirements are that (1) each data update message is for a
single resource; (2) an incremental change can be applied only to a
resource that is a single JSON object, as both JSON merge patch and
JSON patch can apply only to a single JSON object. Hence, if a
future ALTO resource can contain multiple objects, then either each
individual object also has a resource-id or an extension to this
design is made.

At the low level encoding level, new line in SSE has its own
semantics. Hence, this design requires that resource encoding does
not include new lines that can confuse with SSE encoding. In
particular, the data update message MUST NOT include "event: " or
"data: " at a new line as part of data message.

If an update stream provides updates to a filtered cost map that
allows constraint tests, the requirements for such services are
stated in Section 10.4.

13. IANA Considerations

This document defines two new media-types, "application/alto-
updatestreamparams+json", as described in Section 7.3, and
"application/alto-updatestreamcontrol+json", as described in
Section 6.3. All other media-types used in this document have
already been registered, either for ALTO, JSON merge patch, or JSON
patch.

Type name: application

Subtype name: alto-updatestreamparams+json

Required parameters: n/a

Optional parameters: n/a
Encoding considerations: Encoding considerations are identical to those specified for the "application/json" media type. See [RFC7159].

Security considerations: Security considerations relating to the generation and consumption of ALTO Protocol messages are discussed in Section 11 of this document and Section 15 of [RFC7285].

Interoperability considerations: This document specifies format of conforming messages and the interpretation thereof.

Published specification: Section 7.3 of this document.

Applications that use this media type: ALTO servers and ALTO clients either stand alone or are embedded within other applications.

Additional information:

Magic number(s): n/a

File extension(s): This document uses the mime type to refer to protocol messages and thus does not require a file extension.

Macintosh file type code(s): n/a

Person & email address to contact for further information: See Authors' Addresses section.

Intended usage: COMMON

Restrictions on usage: n/a

Author: See Authors' Addresses section.

Change controller: Internet Engineering Task Force (mailto:iesg@ietf.org).

Type name: application

Subtype name: alto-updatestreamcontrol+json

Required parameters: n/a

Optional parameters: n/a

Encoding considerations: Encoding considerations are identical to those specified for the "application/json" media type. See [RFC7159].
14. Alternative Designs Not Supported

14.1. Why Not HTTP/2 Server-Push

HTTP/2 ([RFC7540]) provides a Server Push facility. Although the name implies that it might be useful for sending asynchronous updates from the update stream server to the client, in reality Server Push is not well suited for that task. To see why it is not, here is a quick summary of HTTP/2.

HTTP/2 allows an client and a server to multiplex many HTTP requests and responses over a single TCP connection. The requests and responses can be interleaved on a block by block basis, avoiding the head-of-line blocking problem encountered with the Keep-Alive mechanism in HTTP/1.1. Server Push allows a server to send a
resource (an image, a CSS file, a javascript file, etc.) to the client before the client explicitly requests it. A server can only push cacheable GET-mode resources. By pushing a resource, the server implicitly tells the client, "Add this resource to your cache, because a resource you have requested needs it."

One approach for using Server Push for updates is for the update stream server to send each data update message as a separate Server Push item and let the client apply those updates as they arrive. Unfortunately, there are several problems with that approach.

First, HTTP/2 does not guarantee that pushed resources are delivered to the client in the order they were sent by the client, so each data update message would need a sequence number, and the client would have to re-sequence them.

Second, an HTTP/2-aware client library will not necessarily inform a client application when the server pushes a resource. Instead, the library might cache the pushed resource, and only deliver it to the client when the client explicitly requests that URI.

But the third problem is the most significant: Server Push is optional and can be disabled by any proxy between the client and the server. This is not a problem for the intended use of Server Push: eventually the client will request those resources, so disabling Server Push just adds a delay. But this means that Server Push is not suitable for resources which the client does not know to request.

Thus we do not believe HTTP/2 Server Push is suitable for delivering asynchronous updates. Hence we have chosen to base ALTO updates on HTTP/1.1 and SSE.

14.2. Why Not Allowing Stream Restart

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

However, that mechanism adds additional complexity. The update stream server must save SSE messages in a buffer, in case ALTO clients reconnect. But that mechanism will never be perfect: if the ALTO client waits too long to reconnect, or if the ALTO client sends an invalid id, then the update stream server will have to resend the complete maps anyway.
Furthermore, this is unlikely to be a problem in practice. ALTO
clients who want continuous updates for large resources, such as full
Network and cost maps, are likely to be things like P2P trackers.
These ALTO clients will be well connected to the network; they will
rarely drop connections.

Mobile devices certainly can and do drop connections and will have to
reconnect. But mobile devices will not need continuous updates for
multi-megabyte cost maps. If mobile devices need continuous updates
at all, they will need them for small queries, such as the costs from
a small set of media servers from which the device can stream the
currently playing movie. If the mobile device drops the connection
and reestablishes the update stream, the update stream server will
have to retransmit only a small amount of redundant data.

In short, using event ids to avoid resending the full map adds a
considerable amount of complexity to avoid a situation which we
expect is very rare. We believe that complexity is not worth the
benefit.

The Update Stream service does allow the ALTO client to specify the
tag of the last received version of any tagged resource, and if that
is still current, the update stream server need not retransmit the
full resource. Hence ALTO clients can use this to avoid
retransmitting full network maps. cost maps are not tagged, so this
will not work for them. Of course, the ALTO protocol could be
extended by adding version tags to cost maps, which would solve the
retransmission-on-reconnect problem. However, adding tags to cost
maps might add a new set of complications.

15. Acknowledgments

Thank you to Dawn Chen (Tongji University), Shawn Lin (Tongji
University) and Xiao Shi (Yale University) for their contributions to
an earlier version of this document.

16. References

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

5789, March 2010.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type
Specifications and Registration Procedures", RFC 6838,
January 2013.


Authors' Addresses

Wendy Roome
Nokia Bell Labs (Retired)
124 Burlington Rd
Murray Hill, NJ 07974
USA
Phone: +1-908-464-6975
Email: wendy@wdroome.com

Y. Richard Yang
Yale University
51 Prospect St
New Haven CT
USA
Email: yry@cs.yale.edu
Abstract

The Application-Layer Traffic Optimization (ALTO) protocol [RFC7285] has defined cost maps and endpoint cost maps to provide basic network information. However, they provide only scalar (numerical or ordinal) cost mode values, which are insufficient to satisfy the demands of solving more complex network optimization problems. This document introduces an extension to the base ALTO protocol, namely the path-vector extension, which allows ALTO clients to query information such as the capacity region for a given set of flows (called co-flows). A non-normative example called co-flow scheduling is presented to illustrate the limitations of existing ALTO endpoint cost maps. After that, details of the extension are defined.

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

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 https://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 December 20, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ................................................. 3
2. Terminology ............................................... 5
3. Use Case: Capacity Region for Co-Flow Scheduling ............ 5
4. Overview of Path Vector Extensions .......................... 7
   4.1. New Cost Mode to Encode Path Vectors .................... 7
   4.2. New ALTO Entity Domain for ANE Properties ............. 8
   4.3. Multipart/Related Resource for Consistency ............ 8
5. Path-Vector Cost Type ....................................... 9
   5.1. Cost Mode: path-vector ................................. 10
   5.2. Cost Metric: Link Maximum Reservable Bandwidth ........ 10
6. ANE Domain .................................................. 10
   6.1. Domain Name ........................................... 11
   6.2. Domain-Specific Entity Identifier ...................... 11
   6.3. Hierarchy and Inheritance .............................. 11
7. Multipart Filtered Cost Map for Path Vector .................. 11
   7.1. Media Type ............................................ 11
   7.2. HTTP Method ........................................... 11
   7.3. Accept Input Parameters ................................. 12
   7.4. Capabilities .......................................... 12
   7.5. Uses .................................................. 12
   7.6. Response .............................................. 12
8. Multipart Endpoint Cost Service for Path Vector .............. 13
   8.1. Media Type ............................................ 13
   8.2. HTTP Method ........................................... 13
   8.3. Accept Input Parameters ................................. 13
   8.4. Capabilities .......................................... 13
1. Introduction

The base ALTO protocol [RFC7285] is designed to expose network information through services such as cost maps and endpoint cost service. These services use an extreme "single-node" network abstraction, which represents a whole network as a single node, and hosts as "endpoint groups" directly connected to the node.

Although the "single-node" abstraction works well in many settings, it lacks the ability to support emerging use cases, such as co-flow scheduling for large-scale data analytics. For such a use case, applications require a more powerful network view abstraction beyond the "single-node" abstraction.

To support capabilities like co-flow scheduling, this document uses a "path vector" abstraction to represent more detailed network graph information like capacity regions. A path vector is a sequence of abstract network elements (ANEs), and each ANE represents a network device that end-to-end traffic goes through, such as links, switches, middleboxes, and their aggregations. An ANE can have properties such as "bandwidth", and "delay". Providing such information can help
Providing path vector abstraction using ALTO introduces the following additional requirements (ARs):

AR-1: The path vector abstraction requires the encoding of array-like cost values rather than scalar cost values in cost maps or endpoint cost maps.

Specifically, the path vector abstraction requires the specification of the sequence of ANEs between sources and destinations. Such a sequence, however, cannot be encoded by the scalar types (numerical or ordinal) which the base ALTO protocol supports.

AR-2: The path vector abstraction requires the encoding of the properties of aforementioned ANEs.

Specifically, only the sequences of ANEs are not enough for existing use cases. Properties of ANEs such as "bandwidth" and "delay" are needed by applications to properly construct network constraints or states.

AR-3: The path vector abstraction requires consistent encoding of path vectors (AR-1) and the properties of the ANEs in a path vector (AR-2).

Specifically, path vectors and the properties of ANEs in the vectors are dependent. A mechanism to query both of them consistently is necessary.

This document proposes the path vector extension to the ALTO protocol to satisfy these additional requirements.

Specifically, the extension encodes the array (AR-1) of ANEs over an end-to-end path using a new cost type, and conveys the properties of ANEs (AR-2) using unified property map [I-D.ietf-alto-unified-props-new]. The path vector and ANE properties are conveyed in a single message encoded as a multipart/related message to satisfy AR-3.

The rest of this document is organized as follows. Section 3 gives an example of co-flow scheduling and illustrates the limitations of the base ALTO protocol in such a use case. Section 4 gives an overview of the path vector extension. Section 5 introduces a new cost type. Section 6 registers a new domain in Domain Registry. Section 7 and Section 8 define new ALTO resources to support Path
Vector query by using the request format of Filtered Cost Map and Endpoint Cost Service. Section 9 presents several examples. Section 10 and Section 11 discuss compatibility issues with other existing ALTO extensions and design decisions. Section 12 and Section 13 review the security and IANA considerations.

2. Terminology

Besides the terms defined in [RFC7285] and [I-D.ietf-alto-unified-props-new], this document also uses the following additional terms: Abstract Network Element and Path Vector.

- Abstract Network Element (ANE): An abstract network element is an abstraction of network components. It can be an aggregation of links, middleboxes, virtualized network function (VNF), etc. An abstract network element has two types of attributes: a name and a set of properties.

- Path Vector: A path vector is an array of ANEs. It presents an abstract network path between source/destination points such as PIDs or endpoints.

3. Use Case: Capacity Region for Co-Flow Scheduling

Assume that an application has control over a set of flows, which may go through shared links or switches and share a bottleneck. The application hopes to schedule the traffic among multiple flows to get better performance. The capacity region information for those flows will benefit the scheduling. However, existing cost maps cannot reveal such information.

Specifically, consider a network as shown in Figure 1. The network has 7 switches (sw1 to sw7) forming a dumb-bell topology. Switches sw1/sw3 provide access on one side, sw2/sw4 provide access on the other side, and sw5-sw7 form the backbone. Endhosts eh1 to eh4 are connected to access switches sw1 to sw4 respectively. Assume that the bandwidth all links are 100 Mbps.
Consider an application overlay (e.g., a large data analysis system) which wants to schedule the traffic among a set of end host source-destination pairs, say eh1 -> eh2 and eh3 -> eh4. The application can request a cost map providing end-to-end available bandwidth, using "availbw" as cost-metric and "numerical" as cost-mode.

The application will receive from ALTO server that the bandwidth of eh1 -> eh2 and eh3 -> eh4 are both 100 Mbps. But this information is not enough. Consider the following two cases:

- **Case 1:** If eh1 -> eh2 uses the path eh1 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh2 and eh3 -> eh4 uses path eh3 -> sw3 -> sw5 -> sw7 -> sw4 -> eh4, then the application will obtain 200 Mbps.
Case 2: If eh1 -> eh2 uses the path eh1 -> sw1 -> sw5 -> sw7 -> sw2 -> eh2 and eh3 -> eh4 uses the path eh3 -> sw3 -> sw5 -> sw7 -> sw4 -> eh4, then the application will obtain only 100 Mbps due to the shared link from sw5 to sw7.

To allow applications to distinguish the two aforementioned cases, the network needs to provide more details. In particular:

- The network needs to expose more detailed routing information to show the shared bottlenecks;
- The network needs to provide the necessary abstraction to hide the real topology information while providing enough information to applications.

The path vector extension defined in this document provides a solution to address the preceding issue.

See [I-D.bernstein-alto-topo] for a more comprehensive survey of use cases where extended network topology information is needed.

4. Overview of Path Vector Extensions

This section presents an overview of approaches adopted by the path vector extension. It assumes that the readers are familiar with cost map and endpoint cost service defined in [RFC7285]. The path vector extension also requires the support of Filtered Property Map defined in [I-D.ietf-alto-unified-props-new].

The path vector extension is composed of three building blocks: (1) a new cost mode to encode path vectors in a cost map or an endpoint cost map; (2) a new ALTO entity domain to enable ANE property encoding using the unified property extension [I-D.ietf-alto-unified-props-new]; and (3) a generic mechanism to put multiple ALTO information objects in a single response to enforce consistency, to preserve modularity and to avoid complex linking of multiple responses.

4.1. New Cost Mode to Encode Path Vectors

Existing cost modes defined in [RFC7285] allow only scalar cost values. However, the "path vector" abstraction requires to convey vector format information (AR-1). To fulfill this requirement, this document defines a new "cost-mode" named path vector to indicate that the cost value is an array of ANEs. A path vector abstraction should be computed for a specific performance metric, and this is achieved using the existing "cost-metric" component of cost type. The details of the new "cost-mode" is given in Section 5.
4.2. New ALTO Entity Domain for ANE Properties

A path vector of ANEs contains only the abstracted routing elements between a source and a destination. Hence, an application can find shared ANEs of different source-destination pairs but cannot know the shared ANEs’ properties. For the capacity region use case in Section 3, knowing that eh1→eh2 and eh3→eh4 share ANEs but not the available bandwidth of the shared ANEs, is not enough.

To encode ANE properties like the available bandwidth in a path vector query response, this document uses the unified property extension defined in [I-D.ietf-alto-unified-props-new]. Specifically, for each path vector query, the ALTO server generates a property map associated to the (endpoint) cost map as follows:

- A dynamic entity domain of an entity domain type "ane" is generated to contain the generated ANEs. Each ANE has the same unique identifier in the path vectors and in the dynamic entity domain;

- Each entity in this dynamic entity domain has the property defined by the "cost-metric" that generated the ANEs in the query.

Detailed information and specifications are given in Section 6.

4.3. Multipart/Related Resource for Consistency

Path vectors and the property map containing the ANEs are two different types of objects, but they require strong consistency. One approach to achieving strong consistency is to define a new media type to contain both objects, but this violates modular design.

Another approach is to provide the objects in two different information resources. Thus, an ALTO client needs to make separate queries to get the information of related services. This may cause a data synchronization problem between two queries. Also, as the generation of ANE is dynamic, an ALTO server must cache the results of a query before a client fully retrieves all related resources, which hurts the scalability and security of an ALTO server.

This document uses standard-conforming usage of "multipart/related" media type defined in [RFC2387] to elegantly solve the problem.

Specifically, using "multipart/related" needs to address two issues:

- ALTO uses media type to indicate the type of an entry in the information resource directory (IRD) (e.g., "application/alto-costmap+json" for cost map and "application/alto-
endpointcostmap+json" for endpoint cost map). Simply putting "multipart/related" as the media type, however, makes it impossible for an ALTO client to identify the type of service provided by related entries.

- The ALTO SSE extension (see [I-D.ietf-alto-incr-update-sse]) depends on resource-id to identify push updates, but resource-id is provided only in IRD and hence each entry in the IRD has only one resource-id.

This design addresses the two issues as follows:

- To address the first issue, the multipart/related media type includes the type parameter to allow type indication of the root object. For a cost map service, the "media-type" will be "multipart/related" with the parameter "type=application/alto-costmap+json"; for an endpoint cost map service, the parameter will be "type=application/alto-endpointcostmap+json". This design is highly extensible. The entries can still use "application/alto-costmapfilter+json" or "application/alto-endpointcostparams+json" as the accept input parameters, and hence an ALTO client still sends the filtered cost map request or endpoint cost service request. The ALTO server sends the response as a "multipart/related" message. The body of the response includes two parts: the first one is of the media type specified by the "type" parameter; the second one is a property map associated to the first map.

- To address the second issue, each part of the "multipart/related" response message has the MIME part header information including "Content-Type" and "Resource-Id". An ALTO server MAY generate incremental updates (see [I-D.ietf-alto-incr-update-sse]) for each part separately using the "Resource-Id" header.

By applying the design above, for each path vector query, an ALTO server returns the path vectors and the associated property map modularly and consistently. An ALTO server can reuse the data models of the existing information resources. And an ALTO client can subscribe to the incremental updates for the dynamic generated information resources without any changes, if the ALTO server provides incremental updates for them.

5. Path-Vector Cost Type

This document extends the cost types defined in Section 6.1 of [RFC7285] by introducing a new cost mode "path-vector". In the rest of the document, we use "path-vector" to indicate the cost type with the cost-mode "path-vector" for short.
5.1. Cost Mode: path-vector

This document extends the CostMode defined in Section 10.5 of [RFC7285] with a new cost mode: "path-vector". This cost mode indicates that every cost value in a cost map represents an array of ANEs which are defined in Section 6.2, rather than a JSON number or a ranking order.

The ANEs computed by the ALTO server associate to the cost metric for the "path-vector" cost mode. This document re-defines some cost metrics for "path-vector", which are motivated by the co-flow scheduling use case. The ALTO client SHOULD ignore the "path-vector" cost mode with any other cost metrics, unless the future documents define other cost metrics or specify the semantics of existing cost metrics for "path-vector" cost mode for some additional requirements.

5.2. Cost Metric: Link Maximum Reservable Bandwidth

This document uses the same metric name, units of measurement and measurement point(s) with potential measurement domain defined by section 4.1 of [I-D.ietf-alto-performance-metrics], but specifies different metric description and method of measurement or calculation for "path-vector" cost mode only.

Metric Description: When used with "path-vector" cost mode, it is to specify the path vector computed by using the spatial and temporal maximum reservable bandwidth over each network link. The value of the maximum reservable bandwidth of each ANE in the path vector is specified in the associated property map.

Method of Measurement or Calculation: The value of Maximum Reservable Bandwidth is the bandwidth measured between two directly connected IS-IS neighbors, OSPF neighbors or BGP neighbors. The associated ANEs are computed by some algorithm which can guarantee the equivalent Maximum Reservable Bandwidth constraints.

6. ANE Domain

This document specifies a new ALTO entity domain called "ane" in addition to the ones in [I-D.ietf-alto-unified-props-new]. The ANE domain associates property values with the ANEs in a network. The entity in ANE domain is often used in the path vector by cost maps or endpoint cost resources. Accordingly, the ANE domain always depends on a cost map or an endpoint cost map.
6.1. Domain Name

ane

6.2. Domain-Specific Entity Identifier

The entity identifier of ane domain is encoded as a JSON string. The string MUST be no more than 64 characters, and it MUST NOT contain characters other than US-ASCII alphanumeric characters (U+0030-U+0039, U+0041-U+005A, and U+0061-U+007A), the hyphen ("-", U+002D), the colon (":", U+003A), the at sign ("@", code point U+0040), the low line ("_", U+005F), or the "." separator (U+002E). The "." separator is reserved for future use and MUST NOT be used unless specifically indicated in this document, or an extension document.

To simplify the description, we use "ANE name" to indicate the identifier of an entity in ANE domain in this document.

The ANE name is usually unrelated to the physical device information. It is usually generated by the ALTO server on demand and used to distinguish from other ANEs in its dependent cost map or endpoint cost map.

6.3. Hierarchy and Inheritance

There is no hierarchy or inheritance for properties associated with ANEs.

7. Multipart Filtered Cost Map for Path Vector

This document introduces a new ALTO resource called Multipart Filtered Cost Map resource, which allows an ALTO server to provide other ALTO resources associated to the Cost Map resource in the same response.

7.1. Media Type

The media type of the Multipart Filtered Cost Map Resource is "multipart/related;type=application/alto-costmap+json".

7.2. HTTP Method

The Multipart Filtered Cost Map is requested using the HTTP POST method.
7.3. Accept Input Parameters

The input parameters of the Multipart Filtered Cost Map MUST be encoded as a JSON object in the body of an HTTP POST request. The media type of the request MUST be one of "application/alto-costmapfilter+json". The format of the request body MUST be the same type as defined by section 11.3.2.3 of [RFC7285].

7.4. Capabilities

The Multipart Filtered Cost Map resource uses the same capabilities as defined by section 11.3.2.4 of [RFC7285]. But the "cost-type-names" field SHOULD only include cost types in "path-vector" cost mode. Otherwise, the ALTO client SHOULD ignore a cost type in other cost mode, unless additional documents define the specification of it for the Multipart Filtered Cost Map resource.

7.5. Uses

The resource ID of the network map based on which the PIDs in the returned cost map will be defined.

7.6. Response

The response MUST indicate an error, using ALTO protocol error handling, as defined in Section 8.5 of [RFC7285], if the request is invalid.

The response to a valid request MUST be a "multipart/related" message as defined by [RFC2387]. The body consists of two parts:

- the first part MUST include "Resource-Id" and "Content-Type" in its header. The value of "Resource-Id" MUST be prefixed by the resource id of the Multipart Filtered Cost Map appended by a "." character. The body of this part MUST be a JSON object with the same format as defined in Section 11.2.3.6 of [RFC7285]; The JSON object MUST include the "vtag" field in the "meta" field, which provides the version tag of the returned cost map. The resource id of the version tag MUST be as same as the value of the "Resource-Id" header. The "meta" field MUST also include the "dependent-vtags" field, whose value is a single-element array to indicate the version tag of the network map used, where the network map is specified in the "uses" attribute of the Multipart Cost Map resource in IRD.

- the second part MUST also include "Resource-Id" and "Content-Type" in its header. The value of "Resource-Id" MUST be prefixed by the resource id of the Multipart Filtered Cost Map appended by a "."
character. The body of this part MUST be a JSON object with the same format as defined in Section 4.6 of [I-D.ietf-alto-unified-props-new]. The JSON object MUST include the "dependent-vtags" field in the "meta" field. The value of the "dependent-vtags" field MUST be an array with a single VersionTag object as defined by section 10.3 of [RFC7285]. The "resource-id" of this VersionTag MUST be the value of "Resource-Id" header of the first part. The "tag" of this VersionTag MUST be the "tag" of "vtag" of the first part body.

8. Multipart Endpoint Cost Service for Path Vector

This document introduces a new ALTO resource called Multipart Endpoint Cost resource, which allows an ALTO server to provide other ALTO resources associated to the Endpoint Cost resource in the same response.

8.1. Media Type

The media type of the Multipart Endpoint Cost Resource is "multipart/related;type=application/alto-endpointcostmap+json".

8.2. HTTP Method

The Multipart Endpoint Cost resource is requested using the HTTP POST method.

8.3. Accept Input Parameters

The input parameters of the Multipart Endpoint Cost resource MUST be encoded as a JSON object in the body of an HTTP POST request. The media type of the request MUST be one of "application/alto-endpointcostparams+json". The format of the request body MUST be the same type as defined by section 11.5.1.3 of [RFC7285].

8.4. Capabilities

The Multipart Endpoint Cost resource uses the same capabilities as defined by section 11.3.2.4 of [RFC7285]. But the "cost-type-names" field SHOULD only includes cost types in "path-vector" cost mode. Otherwise, the ALTO client SHOULD ignore a cost type in other cost mode, unless additional documents define the specification of it for the Multipart Endpoint Cost resource.
8.5. Uses

The Multipart Endpoint Cost resource MUST NOT specify the "uses" attribute.

8.6. Response

The response MUST indicate an error, using ALTO protocol error handling, as defined in Section 8.5 of [RFC7285], if the request is invalid.

The response to a valid request MUST be a "multipart/related" message as defined by [RFC2387]. The body consists of two parts:

- the first part MUST include "Resource-Id" and "Content-Type" in its header. The value of "Resource-Id" MUST be prefixed by the resource id of the Multipart Filtered Cost Map appended by a "." character (U+002E). The body of this part MUST be a JSON object with the same format as defined in Section 11.5.1.6 of [RFC7285]; The JSON object MUST include the "vtag" field in the "meta" field, which provides the version tag of the returned endpoint cost map. The resource id of the version tag MUST be as same as the value of the "Resource-Id" header.

- the second part MUST also include "Resource-Id" and "Content-Type" in its header. The value of "Resource-Id" MUST be prefixed by the resource id of the Multipart Filtered Cost Map appended by a "." character (U+002E). The body of this part MUST be a JSON object with the same format as defined in Section 4.6 of [I-D.ietf-alto-unified-props-new]. The JSON object MUST include the "dependent-vtags" field in the "meta" field. The value of the "dependent-vtags" field MUST be an array with a single VersionTag object as defined by section 10.3 of [RFC7285]. The "resource-id" of this VersionTag MUST be the value of "Resource-Id" header of the first part. The "tag" of this VersionTag MUST be the "tag" of "vtag" of the first part body.

9. Examples

This section lists some examples of path vector queries and the corresponding responses.

9.1. Information Resource Directory Example

Here is an example of an Information Resource Directory. In this example, the "cost-map-pv" information resource provides a Multipart Cost Map resource for path-vector; the "endpoint-cost-pv" information resource provides a MultipartEndpoint Cost resource for path-vector.
Both of them support the Maximum Reservable Bandwidth ("maxresbw") cost metric in "path-vector" cost mode.

```json
{
    "meta": {
        "cost-types": {
            "pv-maxresbw": {
                "cost-mode": "path-vector",
                "cost-metric": "maxresbw"
            }
        }
    },
    "resources": {
        "my-default-networkmap": {
            "uri": "http://alto.example.com/networkmap",
            "media-type": "application/alto-networkmap+json"
        },
        "cost-map-pv": {
            "uri": "http://alto.example.com/costmap/pv",
            "media-type": 'multipart/related;
            type=application/alto-costmap+json',
            "accepts": "application/alto-costmapfilter+json",
            "capabilities": {
                "cost-type-names": [ "pv-maxresbw" ]
            },
            "uses": [ "my-default-networkmap" ]
        },
        "endpoint-cost-pv": {
            "uri": "http://alto.exmaple.com/endpointcost/pv",
            "media-type": 'multipart/related;
            type=application/alto-endpointcost+json',
            "accepts": "application/alto-endpointcostparams+json",
            "capabilities": {
                "cost-type-names": [ "pv-maxresbw" ]
            }
        },
        "update-pv": {
            "uri": "http://alto.example.com/updates/pv",
            "media-type": "text/event-stream",
            "uses": [ "endpoint-cost-pv" ],
            "accepts": "application/alto-updatestreamparams+json",
            "capabilities": {
                "support-stream-control": true
            }
        }
    }
}
```

9.2. Example #1

Query filtered cost map to get the path vectors.

POST /costmap/pv HTTP/1.1
Host: alto.example.com
Accept: multipart/related;
    type=application/alto-costmap+json,
    application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-costmapfilter+json

{
    "cost-type": {
        "cost-mode": "path-vector",
        "cost-metric": "maxresbw"
    },
    "pids": {
        "srcs": [ "PID1" ],
        "dsts": [ "PID2", "PID3" ]
    }
}

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=example-1;
    start=cost-map-pv.costmap
type=application/alto-costmap+json

--example-1
Resource-Id: cost-map-pv.costmap
Content-Type: application/alto-costmap+json

{
    "meta": {
        "vtag": {
            "resource-id": "cost-map-pv.costmap",
            "tag": "d827f484cb66ce6df6b5077cb8562b0a"
        },
        "dependent-vtags": [ {
            "resource-id": "my-default-networkmap",
            "tag": "75ed013b3cb58f896e839582504f6228"
        } ],
        "cost-type": {
            "cost-mode": "path-vector",
            "cost-metric": "maxresbw"
        }
    }
}
"cost-map": {
    "PID1": {
        "PID2": [ "ane:L001", "ane:L003" ],
        "PID3": [ "ane:L001", "ane:L004" ]
    }
}

--example-1
Resource-Id: cost-map-pv.propmap
Content-Type: application/alto-propmap+json

{
    "meta": {
        "dependent-vtags": [
            {
                "resource-id": "cost-map-pv.costmap",
                "tag": "d827f484cb66ce6df6b5077c8562b0a"
            }
        ]
    },
    "property-map": {
        "ane:L001": { "maxresbw": 100000000},
        "ane:L003": { "maxresbw": 150000000},
        "ane:L004": { "maxresbw": 50000000}
    }
}

9.3. Example #2
POST /endpointcost/pv HTTP/1.1
Host: alto.example.com
Accept: multipart/related;

type=application/alto-endpointcost+json,
application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-endpointcostparams+json

{
    "cost-type": {
        "cost-mode": "path-vector",
        "cost-metric": "maxresbw"
    },
    "endpoints": {
        "srcs": ["ipv4:192.0.2.2"],
        "dsts": ["ipv4:192.0.2.89",
                   "ipv4:203.0.113.45",
                   "ipv6:2001:db8::10"
                ]
    }
}

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=example-2;
start=endpoint-cost-pv.ecs
type=application/alto-endpointcost+json

--example-2
Resource-Id: endpoint-cost-pv.ecs
Content-Type: application/alto-endpointcost+json

{
    "meta": {
        "vtags": {
            "resource-id": "endpoint-cost-pv.ecs",
            "tag": "bb6b72eafe8f9bd9f335c7ed3b10822a391cef"
        },
        "cost-type": {
            "cost-mode": "path-vector",
            "cost-metric": "maxresbw"
        }
    },
    "endpoint-cost-map": {
        "ipv4:192.0.2.2": {
            "ipv4:192.0.2.89": ["ane:L001", "ane:L003",
                                 "ane:L004"],
            "ipv4:203.0.113.45": ["ane:L001", "ane:L004",
                                  "ane:L005"],
        }
    }
}
"ipv6:2001:db8::10": [ "ane:L001", "ane:L005", "ane:L007" ]
}
}
--example-2
Resource-Id: endpoint-cost-pv.propmap
Content-Type: application/alto-propmap+json
{
  "meta": {
    "dependent-vtags": [
      { "resource-id": "endpoint-cost-pv.ecs", "tag": "bb6bb72eafe8f9bdc4f335c7ed3b10822a391cef" }
    ]
  },
  "property-map": {
    "ane:L001": { "maxresbw": 50000000 },
    "ane:L003": { "maxresbw": 48000000 },
    "ane:L004": { "maxresbw": 55000000 },
    "ane:L005": { "maxresbw": 60000000 },
    "ane:L007": { "maxresbw": 35000000 }
  }
}

9.4. Example for Incremental Update

In this example, an ALTO client subscribe the incremental update for the Multipart Endpoint Cost resource "endpoint-cost-pv".

POST /updates/pv HTTP/1.1
Host: alto.example.com
Accept: text/event-stream
Content-Type: application/alto-updatestreamparams+json
Content-Length: [TBD]
{
  "add": {
    "ecspvsub1": {
      "resource-id": "endpoint-cost-pv",
      "input": <ecs-input>
    }
  }
}
Based on the server process defined in [I-D.ietf-alto-incr-update-sse], the ALTO server will send the control-uri first using Server-Sent Event (SSE), and follow the full response of the multipart message.

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

event: application/alto-updatestreamcontrol+json
data: {"control-uri": "http://alto.example.com/updates/streams/1414"}

event: multipart/related;boundary=example-3;start=pvmap;
type=application/alto-endpointcost+json,ecspvsub1

data: --example-3

data: Content-ID: pvmap

data: Content-Type: application/alto-endpointcost+json

data: <endpoint-cost-map-entry>
data: --example-3

data: Content-ID: nepmap

data: Content-Type: application/alto-propmap+json

data: <property-map-entry>
data: --example-3--

Then, the ALTO server will subscribe the whole tree of the multipart message automatically.

When the data updated, the ALTO server will publish the data updates for each node in this tree separately.

event: application/merge-patch+json,ecspvsub1.pvmap
data: <Merge patch for endpoint-cost-map-update>

event: application/merge-patch+json,ecspvsub2.nepmap
data: <Merge patch for property-map-update>

10. Compatibility

10.1. Compatibility with Base ALTO Clients/Servers

The Multipart Filtered Cost Map resource and the Multipart Endpoint Cost resource has no backward compatibility issue with the base ALTO clients and servers. Although these two types of resources reuse the media types defined in the base ALTO protocol for the accept input parameters, they have different media types for responses. If the ALTO server provides these two types of resources, but the ALTO
client does not support them, the ALTO client will ignore the resources without conducting any incompatibility.

10.2. Compatibility with Multi-Cost Extension

This document does not specify how to integrate the "path-vector" cost mode with the multi-cost extension [RFC8189]. Although there is no reason why somebody has to compound the path vectors with other cost types in a single query, there is no compatible issue doing it without constraint tests.

10.3. Compatibility with Incremental Update

As this document still follows the basic request/response protocol with JSON encoding, it is surely compatible with the incremental update service as defined by [I-D.ietf-alto-incr-update-sse]. But the following details are to be noticed:

- When using the compound response, updates on both cost map and property map SHOULD be notified.
- When not using the compound response, because the cost map is in the "uses" attribute of the property map, once the path vectors in the cost map change, the ALTO server MUST send the updates of the cost map before the updates of the property map.

11. General Discussions

11.1. Provide Calendar for Property Map

Fetching the historical network information is useful for many traffic optimization problem. [I-D.ietf-alto-cost-calendar] already proposes an ALTO extension called Cost Calendar which provides the historical cost values using Filtered Cost Map and Endpoint Cost Service. However, the calendar for only path costs is not enough.

For example, as the properties of ANEs (e.g., available bandwidth and link delay) are usually the real-time network states, they change frequently in the real network. It is very helpful to get the historical value of these properties. Applications may predicate the network status using these information to better optimize their performance.

So the coming requirement may be a general calendar service for the ALTO information resources.
11.2. Constraint Tests for General Cost Types

The constraint test is a simple approach to query the data. It allows users to filter the query result by specifying some boolean tests. This approach is already used in the ALTO protocol. [RFC7285] and [RFC8189] allow ALTO clients to specify the "constraints" and "or-constraints" tests to better filter the result.

However, the current defined syntax is too simple and can only be used to test the scalar cost value. For more complex cost types, like the "array" mode defined in this document, it does not work well. It will be helpful to propose more general constraint tests to better perform the query.

In practice, it is too complex to customize a language for the general-purpose boolean tests, and can be a duplicated work. So it may be a good idea to integrate some already defined and widely used query languages (or their subset) to solve this problem. The candidates can be XQuery and JSONiq.

11.3. General Multipart Resources Query

Querying multiple ALTO information resources continuously MAY be a general requirement. And the coming issues like inefficiency and inconsistency are also general. There is no standard solving these issues yet. So we need some approach to make the ALTO client request the compound ALTO information resources in a single query.

12. Security Considerations

This document is an extension of the base ALTO protocol, so the Security Considerations [RFC7285] of the base ALTO protocol fully apply when this extension is provided by an ALTO server.

The path vector extension requires additional considerations on two security considerations discussed in the base protocol: confidentiality of ALTO information (Section 15.3 of [RFC7285]) and availability of ALTO service (Section 15.5 of [RFC7285]).

For confidentiality of ALTO information, a network operator should be aware of that this extension may introduce a new risk: the path vector information may make network attacks easier. For example, as the path vector information may reveal more network internal structures than the more abstract single-node abstraction, an ALTO client may detect the bottleneck link and start a distributed denial-of-service (DDoS) attack involving minimal flows to conduct the in-network congestion.
To mitigate this risk, the ALTO server should consider protection mechanisms to reduce information exposure or obfuscate the real information, in particular, in settings where the network and the application do not belong to the same trust domain. But the implementation of path vector extension involving reduction or obfuscation should guarantee that the constraints on the requested properties are still accurate.

For availability of ALTO service, an ALTO server should be cognizant that using path vector extension might have a new risk: frequent requesting for path vectors might conduct intolerable increment of the server-side storage and break the ALTO server. It is known that the computation of path vectors is unlikely to be cacheable, in that the results will depend on the particular requests (e.g., where the flows are distributed). Hence, the service providing path vectors may become an entry point for denial-of-service attacks on the availability of an ALTO server. To avoid this risk, authenticity and authorization of this ALTO service may need to be better protected.

Even if there is no intentional attack, the dependent property map of path vector might be still dynamically enriched, in that every new request for path vectors will make the ALTO server generate a new property map. So the properties of the abstract network elements can consume a large amount of resources when cached. To avoid this, the ALTO server providing the path vector extension should support a time-to-live configuration for the property map, so that the outdated entries can be removed from the property map resource.

13. IANA Considerations

13.1. ALTO Cost Mode Registry

This document specifies a new cost mode "path-vector". However, the base ALTO protocol does not have a Cost Mode Registry where new cost mode can be registered. This new cost mode will be registered once the registry is defined either in a revised version of [RFC7285] or in another future extension.

13.2. ALTO Entity Domain Registry

As proposed in Section 9.2 of [I-D.ietsf-alto-unified-props-new], "ALTO Domain Entity Registry" is requested. Besides, a new domain is to be registered, listed in Table 1.
### Table 1: ALTO Entity Domain

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Entity Address Encoding</th>
<th>Hierarchy &amp; Inheritance</th>
</tr>
</thead>
<tbody>
<tr>
<td>ane</td>
<td>See Section 6.2</td>
<td>None</td>
</tr>
</tbody>
</table>

### 13.3. ALTO Property Type Registry

The "ALTO Property Type Registry" is required by the ALTO Domain "ane", listed in Table 2.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Intended Semantics</th>
<th>Dependencies and Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ane:maxresbw</td>
<td>The maximum reservable bandwidth for the ANE</td>
<td>application/alto-costmap+json, or application/alto-endpointcostmap+json, where the ANE names are used.</td>
</tr>
</tbody>
</table>

### Table 2: ALTO Abstract Network Element Property Types

### 14. Acknowledgments

The authors would like to thank discussions with Andreas Voellmy, Erran Li, Haibin Son, Haizhou Du, Jiayuan Hu, Qiao Xiang, Tianyuan Liu, Xiao Shi, Xin Wang, and Yan Luo. The authors thank Greg Bernstein (Grotto Networks), Dawn Chen (Tongji University), Wendy Roome, and Michael Scharf for their contributions to earlier drafts.

### 15. References

15.1. Normative References

15.2.  Informative References

[I-D.bernstein-alto-topo]

[I-D.ietf-alto-cost-calendar]

[I-D.ietf-alto-incr-update-sse]

[I-D.ietf-alto-performance-metrics]

[I-D.ietf-alto-unified-props-new]


Authors’ Addresses
Kai Gao  
Tsinghua University  
Beijing Beijing  
China  

Email: gaok12@mails.tsinghua.edu.cn

Young Lee  
Huawei  
TX  
USA  

Email: leeyoung@huawei.com

Sabine Randriamasy  
Nokia Bell Labs  
Route de Villejust  
NOZAY  91460  
FRANCE  

Email: Sabine.Randriamasy@nokia-bell-labs.com

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

Email: yry@cs.yale.edu

Jingxuan Jensen Zhang  
Tongji University  
4800 Caoan Road  
Shanghai  201804  
China  

Email: jingxuan.n.zhang@gmail.com
Unified Properties for the ALTO Protocol
draft-ietf-alto-unified-props-new-07

Abstract

This document extends the Application-Layer Traffic Optimization (ALTO) Protocol [RFC7285] by generalizing the concept of "endpoint properties" to domains of other entities, and by presenting those properties as maps, similar to the network and cost maps in ALTO.

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

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 https://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 12, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction .................................................. 3
2. Definitions and Concepts ...................................... 4
   2.1. Entity .................................................... 4
   2.2. Entity Domain ............................................ 5
   2.3. Domain Name .............................................. 5
   2.4. Entity Identifier ........................................ 6
   2.5. Property Type and Property Name .......................... 6
   2.6. Hierarchy and Inheritance ................................ 7
   2.7. Relationship with Other ALTO Resources .................. 7
3. Entity Domains .................................................. 8
   3.1. Internet Address Domains .................................. 9
      3.1.1. IPv4 Domain ......................................... 9
      3.1.2. IPv6 Domain ......................................... 9
      3.1.3. Hierarchy and Inheritance of ipv4/ipv6 Domains ....... 9
   3.2. PID Domain ............................................... 11
      3.2.1. Domain Name .......................................... 11
      3.2.2. Domain-Specific Entity Identifiers .................. 11
      3.2.3. Hierarchy and Inheritance ............................ 11
      3.2.4. Relationship To Internet Addresses Domains .......... 11
   3.3. Internet Address Properties vs. PID Properties ........... 11
4. Property Map ................................................... 12
   4.1. Media Type ............................................... 12
   4.2. HTTP Method ............................................. 12
   4.3. Accept Input Parameters .................................. 12
   4.4. Capabilities ............................................. 12
   4.5. Uses .................................................... 13
   4.6. Response ................................................ 13
5. Filtered Property Map .......................................... 14
   5.1. Media Type ............................................... 14
   5.2. HTTP Method ............................................. 14
   5.3. Accept Input Parameters .................................. 14
   5.4. Capabilities ............................................. 15
   5.5. Uses .................................................... 15
   5.6. Response ................................................ 15
6. Impact on Legacy ALTO Servers and ALTO Clients ............... 17
   6.1. Impact on Endpoint Property Service ..................... 17
1. Introduction

The ALTO protocol [RFC7285] introduces the concept of "properties" attached to "endpoint addresses", and defines the Endpoint Property Service (EPS) to allow ALTO clients to retrieve those properties. While useful, the EPS, as defined in [RFC7285], has at least two limitations.

First, it only allows properties to be associated with a particular domain of entities, namely individual IP addresses. It is reasonable to think that collections of endpoints, as defined by CIDRs [RFC4632] or PIDs, may also have properties. Since the EPS cannot be extended to new entity domains, new services, with new request and response messages, would have to be defined for new entity domains.

Second, the EPS is only defined as a POST-mode service. Clients must request the properties for an explicit set of endpoint addresses. By contrast, [RFC7285] defines a GET-mode cost map resource which returns all available costs, so a client can get a full set of costs once, and then processes costs lookups without querying the ALTO server. [RFC7285] does not define an equivalent service for endpoint properties. At first a map of endpoint properties might seem impractical, because it could require enumerating the property value for every possible endpoint. But in practice, it is highly unlikely
that properties will be defined for every endpoint address. It is much more likely that properties will only be defined for a subset of endpoint addresses, and that subset would be small enough to be enumerated. This is particularly true if blocks of endpoint addresses with a common prefix (e.g., a CIDR) have the same value for a property. Furthermore, entities in other domains may very well be enumerable.

This document proposes a new approach to retrieve ALTO properties. Specifically, it defines two new types of resources, namely Property Map (see Section 4) and Filtered Property Map (see Section 5). The former is a GET-mode resource which returns the property values for all entities in a domain, and is analogous to a network map or a cost map in [RFC7285]. The latter is a POST-mode resource which returns the values for a set of properties and entities requested by the client, and is analogous to a filtered network map or a filtered cost map.

Additionally, this document introduces ALTO Entity Domains, where an entity is a generalization of an endpoint to also represent a PID, a network element, or a cell in a cellular network, etc. As a consequence, ALTO Entity Domains defined in this document are a super-set of ALTO Address Types defined in [RFC7285]. Their exact relationship is specified in Section 9.2.1.

Entity domains and property names are extensible. New entity domains can be defined without revising the messages defined in this document, in the same way that new cost metrics and new endpoint properties can be defined without revising the messages defined in [RFC7285].

This proposal subsumes the Endpoint Property Service defined in [RFC7285], although that service may be retained for legacy clients (see Section 6).

2. Definitions and Concepts

2.1. Entity

The entity generalizes the concept of the endpoint defined in Section 2.1 of [RFC7285]. An entity is an object that can be an endpoint and is identified by its network address, but can also be an object that has a defined mapping to a set of one or more network addresses or is even not related to any network address.

Examples of eligible entities are:
o a PID, defined in [RFC7285], that has a provider defined human
readable abstract identifier and maps to a set of ipv4 and ipv6
addresses,

o an ASN number, that has a specified identifier and direct mapping
to network addresses,

o a country code, that specified in ISO 3166 format and that can be
retrieved from an IP of cellular address. As a consequence, all
endpoints are entities while not all entities are endpoints,

o a TCP/IP network flow, that has a server defined identifier and
represents in a TCP/IP 5-Tuple,

o a routing element, that specified in [RFC7921] and includes
routing capability information,

o an abstract network element, that has a server defined identifier
and represents a network node, link or their aggregation.

2.2. Entity Domain

Each entity MUST be in one and only one entity domain. An entity
domain is a class of entities. Examples of entity domains are the
Internet address domains (see Section 3.1 and the PID domain (see
Section 3.2). The future documents can define new entity domains to
satisfy the additional requirements such as cellular network
information and routing capability exposure. But they are not in the
scope of this document.

2.3. Domain Name

Each entity domain has a unique name. A domain name MUST be no more
than 32 characters, and MUST NOT contain characters other than US-
ASCII alphanumeric characters (U+0030-U+0039, U+0041-U+005A, and
U+0061-U+007A), hyphen ("-", U+002D), and low line ("_", U+005F).
For example, the names "ipv4" and "ipv6" identify entities in the
Internet address domains (see Section 3.1).

The type DomainName is used in this document to denote a JSON string
with a domain name in this format.

Domain names MUST be registered with the IANA, and the format of the
entity addresses (see Section 2.4) in that entity domain, as well as
any hierarchical or inheritance rules (see Section 2.6) for those
entities, MUST be specified at the same time.
2.4. Entity Identifier

Each entity has an identifier of the format:

    EntityId ::= DomainName : DomainSpecificEntityId

An entity identifier uniquely identifies a particular entity within an ALTO property map resource (see Section 4).

Examples from the IP domains include individual IP addresses such as "ipv4:192.0.2.14" and "ipv6:2001:db8::12", as well as address blocks such as "ipv4:192.0.2.0/26" and "ipv6:2001:db8::1/48".

The type EntityId is used in this document to denote a JSON string with an entity identifier in this format.

The format of the second part of an entity identifier depends on the entity domain, and MUST be specified when registering a new entity domain. Identifiers MAY be hierarchical, and properties MAY be inherited based on that hierarchy. Again, the rules defining any hierarchy or inheritance MUST be defined when the entity domain is registered.

Note that an entity address MAY have different textual representations, for a given entity domain. For example, the strings "ipv6:2001:db8::1" and "ipv6:2001:db8::1/48" refer to the same entity.

2.5. Property Type and Property Name

Every entity in some domain MAY have one or more properties. Every property is identified by a Property Type and is specific to a domain. Every property MUST have a unique Property Type.

This document defines property types in the domain-specific semantics. Multiple property types with similar semantics MAY share the same Property Name in different entity domains. But each property type MUST be registered for a single specific entity domain for the following reasons:

- Some properties may only be applicable for particular entity domains, not all. For example, the "pid" property is not applicable for entities in the "pid" domain.

- The interpretation of the value of a property may depend on the entity domain. For different entity domains, not only the intended semantics but also the dependent resource types may be totally different. For example, suppose that the "geo-location"
property is defined as the coordinates of a point, encoded as (say) "latitude longitude [altitude]." When applied to an entity that represents a specific host computer, identified by an address in the ipv4 or ipv6 domain, the property defines the host’s location and has no required dependency. However, when applied to an entity in the "pid" domain, the property would indicate the location of the center of all hosts in this "pid" entity and depend on the Network Map defining this "pid" entity.

Therefore, each property type has a unique identifier encoded with the following format:

PropertyType ::= DomainName : PropertyName

- The "DomainName" indicates which entity domain the property type applies to.
- The "PropertyName" SHOULD relate to the semantics of this property type. It does not have to be globally unique. In other words, different property types could have the same property name applied to different entity domains, if they have the similar semantics. For example, the property types "ipv4:pid" and "ipv6:pid" have the same property name "pid" applied to both "ipv4" and "ipv6" domains.

Property types MUST be registered with the IANA, and the intended semantics, as well as the media types of dependent resources and the interpretation, MUST be specified at the same time.

2.6. Hierarchy and Inheritance

Entities in a given domain MAY form a hierarchy based on entity identifiers, and introducing hierarchy allows the introduction of inheritance. Each entity domain MUST define its own hierarchy and inheritance rules when registered. The hierarchy and inheritance rule makes it possible for an entity to inherit a property value from another entity in the same domain.

2.7. Relationship with Other ALTO Resources

[RFC7285] recognizes that some properties for some entity domains MAY be specific to an ALTO resource, such as a network map. Accordingly Section 10.8.1 of [RFC7285] defines the concept of "resource-specific endpoint properties", and indicates that dependency by prefixing the property name with the ID of the resource on which it depends. That document defines one resource-specific property, namely the "pid" property, whose value is the name of the PID containing that endpoint in the associated network map.
Because a property may be associated to more than one information resources within an entity domain, this document takes a different approach as follows:

- Firstly, instead of defining the dependency by prefixing the property name with a specific dependent resource identifier, this document introduces a Property Type that appends a property name to an entity domain name, and registers the dependency types for this Property Type. This gives a hint on the types of dependent resources. For example, the fictitious property "pid:region" applying to entities in the PID domain depends on the network map in which the input PID entities have been defined; but the fictitious property "ipv4:region" does not depend on any information resource.

- Secondly, it sets a rule saying that in a property map, all provided property types MUST have the same dependency types. For example, "pid:region" and "ipv4:region" cannot be provided by an individual property map.

- Finally, it identifies, in the IRD and Server responses, the sequence of information resources associated to all provided properties in a particular property map. If a property depends on some different information resources from other properties, the ALTO server should define a different property map to provide it. For example, the property "ipv4:pid" provided by a particular property map MUST depend on one and only one network map. If an ALTO server wants to provide "ipv4:pid" for PIDs defined in both network maps "net1" and "net2", it MUST define two individual property maps.

To specify the aforementioned dependencies, this document uses the "uses" and "dependent-vtags" fields defined respectively in Sections 9.1.5 and 11.1 of [RFC7285].

- the "uses" field is included in the IRD entry of a resources-dependent information resource and specifies the dependent IRD resource.

- the "dependent-vtags" member is used in a Server response message to specify the dependent resource.

3. Entity Domains

This document defines three entity domains. The definition of each entity domain below includes the following: (1) domain name, (2) domain-specific entity identifiers, and (3) hierarchy and inheritance semantics.
3.1. Internet Address Domains

The document defines two entity domains (IPv4 and IPv6) for Internet addresses. Both entity domains include individual addresses and blocks of addresses. Since the two domains use the same hierarchy and inheritance semantics, we define the semantics together, instead of repeating for each.

3.1.1. IPv4 Domain

3.1.1.1. Domain Name
ipv4

3.1.1.2. Domain-Specific Entity Identifiers

Individual addresses are strings as specified by the IPv4Addresses rule of Section 3.2.2 of [RFC3986]; blocks of addresses are prefix-match strings as specified in Section 3.1 of [RFC4632]. For the purpose of defining properties, an individual Internet address and the corresponding full-length prefix are considered aliases for the same entity. Thus "ipv4:192.0.2.0" and "ipv4:192.0.2.0/32" are equivalent.

3.1.2. IPv6 Domain

3.1.2.1. Domain Name
ipv6

3.1.2.2. Domain-Specific Entity Identifiers

Individual addresses are strings as specified by Section 4 of [RFC5952]; blocks of addresses are prefix-match strings as specified in Section 7 of [RFC5952]. For the purpose of defining properties, an individual Internet address and the corresponding 128-bit prefix are considered aliases for the same entity. That is, "ipv6:2001:db8::1" and "ipv6:2001:db8::1/128" are equivalent, and have the same set of properties.

3.1.3. Hierarchy and Inheritance of ipv4/ipv6 Domains

Both Internet address domains allow property values to be inherited. Specifically, if a property P is not defined for a specific Internet address I, but P is defined for some block C which prefix-matches I, then the address I inherits the value of P defined for block C. If more than one such block defines a value for P, I inherits the value of P in the block with the longest prefix. It is important to notice
that this longest prefix rule will ensure no multiple inheritance, and hence no ambiguity.

Address blocks can also inherit properties: if a property P is not defined for a block C, but is defined for some block C' which covers all IP addresses in C, and C' has a shorter mask than C, then block C inherits the property from C'. If there are several such blocks C’, C inherits from the block with the longest prefix.

As an example, suppose that a server defines a property P for the following entities:

```
ipv4:192.0.2.0/26: P=v1
ipv4:192.0.2.0/28: P=v2
ipv4:192.0.2.0/30: P=v3
ipv4:192.0.2.0:    P=v4
```

Figure 1: Defined Property Values.

Then the following entities have the indicated values:

```
ipv4:192.0.2.0:    P=v4
ipv4:192.0.2.1:    P=v3
ipv4:192.0.2.16:   P=v1
ipv4:192.0.2.32:   P=v1
ipv4:192.0.2.64:   (not defined)
ipv4:192.0.2.0/32: P=v4
ipv4:192.0.2.0/31: P=v3
ipv4:192.0.2.0/29: P=v2
ipv4:192.0.2.0/27: P=v1
ipv4:192.0.2.0/25: (not defined)
```

Figure 2: Inherited Property Values.

An ALTO server MAY explicitly indicate a property as not having a value for a particular entity. That is, a server MAY say that property P of entity X is "defined to have no value", instead of "undefined". To indicate "no value", a server MAY perform different behaviours:

- If that entity would inherit a value for that property, then the ALTO server MUST return a "null" value for that property. In this case, the ALTO client MUST recognize a "null" value as "no value" and "do not apply the inheritance rules for this property."

- If the entity would not inherit a value, then the ALTO server MAY return "null" or just omit the property. In this case, the ALTO client cannot infer the value for this property of this entity.
from the Inheritance rules. So the client MUST interpret that this property has no value.

If the ALTO server does not define any properties for an entity, then the server MAY omit that entity from the response.

3.2. PID Domain

The PID domain associates property values with the PIDs in a network map. Accordingly, this entity domain always depends on a network map.

3.2.1. Domain Name

pid

3.2.2. Domain-Specific Entity Identifiers

The entity identifiers are the PID names of the associated network map.

3.2.3. Hierarchy and Inheritance

There is no hierarchy or inheritance for properties associated with PIDs.

3.2.4. Relationship To Internet Addresses Domains

The PID domain and the Internet address domains are completely independent; the properties associated with a PID have no relation to the properties associated with the prefixes or endpoint addresses in that PID. An ALTO server MAY choose to assign some or all properties of a PID to the prefixes in that PID.

For example, suppose "PID1" consists of the prefix "ipv4:192.0.2.0/24", and has the property "P" with value "v1". The Internet address entities "ipv4:192.0.2.0" and "ipv4:192.0.2.0/24", in the IPv4 domain MAY have a value for the property "P", and if they do, it is not necessarily "v1".

3.3. Internet Address Properties vs. PID Properties

Because the Internet address and PID domains are completely separate, the question may arise as to which entity domain is the best for a property. In general, the Internet address domains are RECOMMENDED for properties that are closely related to the Internet address, or are associated with, and inherited through, blocks of addresses.
The PID domain is RECOMMENDED for properties that arise from the
definition of the PID, rather than from the Internet address prefixes
in that PID.

For example, because Internet addresses are allocated to service
providers by blocks of prefixes, an "ISP" property would be best
associated with the Internet address domain. On the other hand, a
property that explains why a PID was formed, or how it relates a
provider’s network, would best be associated with the PID domain.

4. Property Map

A property map returns the properties defined for all entities in one
or more domains, e.g., the "location" property of entities in "pid"
domain, and the "ASN" property of entities in "ipv4" and "ipv6"
domains.

Section 7.4 gives an example of a property map request and its
response.

4.1. Media Type

The media type of a property map is "application/alto-propmap+json".

4.2. HTTP Method

The property map 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
PropertyMapCapabilities:

    object {
        DomainName entity-domains<1..*>;
        PropertyName properties<1..*>;
    } PropertyMapCapabilities;

where "entity-domains" is an array specifying the entity domains, and
"properties" is an array specifying the property names returned for
entities in those domains. The semantics is that this property map
provides all property types generated by the cross product of
"entity-domains" and "properties". If a property in "properties" is
NOT supported by a domain in "entity-domains", the server can declare different property maps to conform to the semantics.

For example, the capability{"entity-domains": ["ipv4", "ipv6"],
"properties": ["pid"]} means the property map provides both property types "ipv4:pid" and "ipv6:pid".

4.5. Uses

The "uses" field of a property map resource in an IRD entry specifies dependencies as discussed in Section 2.7. It is an array of the resource ID(s) of the resource(s) that properties of entities in domains specified in "entity-domains" depend on.

In a single property map, every property value of every entity depends on the same array of resources. Thus, if properties depending on different resources arrays would be provided, they MUST be split into different property maps.

Note that according to [RFC7285], a legacy ALTO server with two network maps, with resource IDs "net1" and "net2", could offer a single Endpoint Property Service for the two properties "net1.pid" and "net2.pid". An ALTO server which supports the property map resource defined in this document, would, instead, offer two different property maps for the "pid" property, one depending on "net1", and the other on "net2".

4.6. Response

If the entity domains in this property map depend on other resources, the "dependent-vtags" field in the "meta" field of the response MUST be an array that includes the version tags of those resources, and the order MUST be consistent with the "uses" field of this property map resource. The data component of a property map response is named "property-map", which is a JSON object of type PropertyMapData, where:

object {
  PropertyMapData property-map;
} InfoResourceProperties : ResponseEntityBase;

object-map {
  EntityId -> EntityProps;
} PropertyMapData;

object {
  PropertyName -> JSONValue;
} EntityProps;
The ResponseEntityBase type is defined in Section 8.4 of [RFC7285].

Specifically, a PropertyMapData object has one member for each entity in the property map. The entity’s properties are encoded in the corresponding EntityProps object. EntityProps encodes one name/value pair for each property, where the property names are encoded as strings of type PropertyName. A protocol implementation SHOULD assume that the property value is either a JSONString or a JSON "null" value, and fail to parse if it is not, unless the implementation is using an extension to this document that indicates when and how property values of other data types are signaled.

For each entity in the Property Map, the ALTO server returns the value defined for each of the properties specified in this resource’s "capabilities" list. For efficiency, the ALTO server SHOULD omit property values that are inherited rather than explicitly defined; if a client needs inherited values, the client SHOULD use the entity domain’s inheritance rules to deduce those values.

5. Filtered Property Map

A filtered property map returns the values of a set of properties for a set of entities selected by the client.

Section 7.5, Section 7.6, Section 7.7 and Section 7.8 give examples of filtered property map requests and responses.

5.1. Media Type

The media type of a property map resource is "application/alto-propmap+json".

5.2. HTTP Method

The filtered property map is requested using the HTTP POST method.

5.3. Accept Input Parameters

The input parameters for a filtered property map request are supplied in the entity body of the POST request. This document specifies the input parameters with a data format indicated by the media type "application/alto-propmapparams+json", which is a JSON object of type ReqFilteredPropertyMap:

```json
object {
    EntityId     entities<1..*>;
    PropertyName   properties<1..*>;
} ReqFilteredPropertyMap;
```

Roome, et al. Expires September 12, 2019
with fields:

entities: List of entity identifiers for which the specified properties are to be returned. The ALTO server MUST interpret entries appearing multiple times as if they appeared only once. The domain of each entity MUST be included in the list of entity domains in this resource’s "capabilities" field (see Section 5.4).

properties: List of properties to be returned for each entity. Each specified property MUST be included in the list of properties in this resource’s "capabilities" field (see Section 5.4). The ALTO server MUST interpret entries appearing multiple times as if they appeared only once.

Note that the "entities" and "properties" fields MUST have at least one entry each.

5.4. Capabilities

The capabilities are defined by an object of type PropertyMapCapabilities, as defined in Section 4.4.

5.5. Uses

The "uses" field of a filtered property map is an array with the resource ID(s) of resource(s) that each domain in "entity-domains" depends on, in order to provide the properties specified in the "properties" capability. The same "uses" rule as defined by the property map resource applies (see Section 4.5).

5.6. Response

The response MUST indicate an error, using ALTO protocol error handling, as defined in Section 8.5 of [RFC7285], if the request is invalid.

Specifically, a filtered property map request can be invalid as follows:

- An entity identifiers in "entities" in the request is invalid if:
  - The domain of this entity is not defined in the "entity-domain-types" capability of this resource in the IRD;
  - The entity identifier is an invalid identifier in the entity domain.
A valid entity identifier is never an error, even if this filtered property map resource does not define any properties for it.

If an entity identifier in "entities" in the request is invalid, the ALTO server MUST return an "E_INVALID_FIELD_VALUE" error defined in Section 8.5.2 of [RFC7285], and the "value" field of the error message SHOULD indicate this entity identifier.

- A property name in "properties" in the request is invalid if this property name is not defined in the "property-types" capability of this resource in the IRD.

It is not an error that a filtered property map resource does not define a requested property’s value for a particular entity. In this case, the ALTO server MUST omit that property from the response for that endpoint.

If a property name in "properties" in the request is invalid, the ALTO server MUST return an "E_INVALID_FIELD_VALUE" error defined in Section 8.5.2 of [RFC7285]. The "value" field of the error message SHOULD indicate the property name.

The response to a valid request is the same as for the Property Map (see Section 4.6), except that:

- The "dependent-vtags" field in its "meta" field only includes the version tags of resources on which the requested properties of the entity domains depend, and the order MUST be consistent with the "uses" field of this filtered property map resource.

- It only includes the entities and properties requested by the client. If an entity in the request is an identifier block (e.g., an "ipv4" or "ipv6" entity), the response MUST cover properties for all identifiers in this block.

It is important that the filtered property map response MUST include all inherited property values for the requested entities and all the entities which are able to inherit property values from them. To achieve this goal, the ALTO server MAY follow three rules:

- If a property for a requested entity is inherited from another entity not included in the request, the response SHOULD include this property for the requested entity. For example, A full property map may skip a property P for an entity A (e.g., ipv4:192.0.2.0/31) if P can be derived using inheritance from another entity B (e.g., ipv4:192.0.2.0/30). A filtered property map request may include only A but not B. In such a case, the property P SHOULD be included in the response for A.
o If there are entities covered by a requested entity but having different values for the requested properties, the response SHOULD include all those entities and the different property values for them. For example, considering a request for property P of entity A (e.g., ipv4:192.0.2.0/31), if P has value v1 for A1=ipv4:192.0.2.0/32 and v2 for A2=ipv4:192.0.2.1/32, then, the response SHOULD include A1 and A2.

o If an entity in the response is already covered by some other entities in the same response, it SHOULD be removed from the response for compactness. For example, in the previous example, the entity A=ipv4:192.0.2.0/31 SHOULD be removed because A1 and A2 cover all the addresses in A.

An ALTO client should be aware that the entities in the response MAY be different from the entities in its request.

6. Impact on Legacy ALTO Servers and ALTO Clients

6.1. Impact on Endpoint Property Service

Since the property map and the filtered property map defined in this document provide the functionality of the Endpoint Property Service (EPS) defined in Section 11.4 of [RFC7285], it is RECOMMENDED that the EPS be deprecated in favor of Property Map and Filtered Property Map. However, ALTO servers MAY provide an EPS for the benefit of legacy clients.

6.2. Impact on Resource-Specific Properties

Section 10.8 of [RFC7285] defines two categories of endpoint properties: "resource-specific" and "global". Resource-specific property names are prefixed with the ID of the resource they depend upon, while global property names have no such prefix. The property map and the filtered property map defined in this document do not distinguish between those two types of properties. Instead, if there is a dependency, it is indicated by the "uses" capability of a property map, and is shared by all properties and entity domains in that map. Accordingly, it is RECOMMENDED that resource-specific endpoint properties be deprecated, and no new resource-specific endpoint properties be defined.

6.3. Impact on the pid Property

Section 7.1.1 of [RFC7285] defines the resource-specific endpoint property name "pid", whose value is the name of the PID containing that endpoint. For compatibility with legacy clients, an ALTO server...
which provides the "pid" property via the EPS MUST use that
definition, and that syntax.

However, when used with property maps, this document amends the
definition of the "pid" property as follows.

First, the name of the property is simply "pid"; the name is not
prefixed with the resource ID of a network map. The "uses"
capability of the property map indicates the associated network map.
This implies that a property map can only return the "pid" property
for one network map; if an ALTO server provides several network maps,
it MUST provide a Property Map for each of the network maps.

Second, a client MAY request the "pid" property for a block of
Internet addresses. An ALTO server determines the value of "pid" for
an address block C as the rules defined in Section 5.6.

Note that although an ALTO server MAY provide a GET-mode property map
which returns the entire map for the "pid" property, there is no need
to do so, because that map is simply the inverse of the network map.

6.4. Impact on Other Properties

In general, there should be little or no impact on other previously
defined properties. The only consideration is that properties can
now be defined on blocks of identifiers, rather than just individual
identifiers, which might change the semantics of a property.

7. Examples

7.1. Network Map

The examples in this section use a very simple default network map:

defaultpid: ipv4:0.0.0.0/0  ipv6::/0
pid1: ipv4:192.0.2.0/25
pid2: ipv4:192.0.2.0/28  ipv4:192.0.2.16/28
pid3: ipv4:192.0.3.0/28
pid4: ipv4:192.0.3.16/28

Figure 3: Example Network Map

7.2. Property Definitions

Beyond "pid", the examples in this section use four additional
properties for Internet address domains, "ISP", "ASN", "country" and
"state", with the following values:
Figure 4: Example Property Values for Internet Address Domains

And the examples in this section use the property "region" for PID domain with the following values:

region  
  pid:defaultpid: -  
  pid:pid1:  west  
  pid:pid2:  east  
  pid:pid3:  south  
  pid:pid4:  north

Figure 5: Example Property Values for PID Domain

Note that "-" means the value of the property for the entity is "undefined". So the entity would inherit a value for this property by the inheritance rule if possible. For example, the value of the "ISP" property for "ipv4:192.0.2.0" is "BitsRus" because of "ipv4:192.0.2.0/24". But the "region" property for "pid:defaultpid" has no value because no entity from which it can inherit.

7.3. Information Resource Directory (IRD)

The following IRD defines the relevant resources of the ALTO server. It provides two property maps, one for the "ISP" and "ASN" properties, and another for the "country" and "state" properties. The server could have provided a single property map for all four properties, but did not, presumably because the organization that runs the ALTO server believes any given client is not interested in all four properties.

The server provides two filtered property maps. The first returns all four properties, and the second just returns the "pid" property for the default network map.

The filtered property maps for the "ISP", "ASN", "country" and "state" properties do not depend on the default network map (it does not have a "uses" capability), because the definitions of those properties do not depend on the default network map. The Filtered Property Map for the "pid" property does have a "uses" capability for
the default network map, because that defines the values of the "pid" property.

Note that for legacy clients, the ALTO server provides an Endpoint Property Service for the "pid" property for the default network map.

"meta" : {
  ...
  "default-alto-network-map" : "default-network-map"
}
"resources" : {
  "default-network-map" : {
    "uri" : "http://alto.example.com/networkmap",
    "media-type" : "application/alto-networkmap+json"
  },
  ... property map resources ....
  "country-state-property-map" : {
    "uri" : "http://alto.example.com/propmap/full/inet-cs",
    "media-type" : "application/alto-propmap+json",
    "capabilities" : {
      "entity-domains" : [ "ipv4", "ipv6" ],
      "properties" : [ "country", "state" ]
    }
  },
  "isp-asn-property-map" : {
    "uri" : "http://alto.example.com/propmap/full/inet-ia",
    "media-type" : "application/alto-propmap+json",
    "capabilities" : {
      "entity-domains" : [ "ipv4", "ipv6" ],
      "properties" : [ "ISP", "ASN" ]
    }
  },
  "iacs-property-map" : {
    "uri" : "http://alto.example.com/propmap/lookup/inet-iacs",
    "media-type" : "application/alto-propmap+json",
    "accepts" : "application/alto-propmapparams+json",
    "capabilities" : {
      "entity-domains" : [ "ipv4", "ipv6" ],
      "properties" : [ "ISP", "ASN", "country", "state" ]
    }
  },
  "pid-property-map" : {
    "uri" : "http://alto.example.com/propmap/lookup/pid",
    "media-type" : "application/alto-propmap+json",
    "accepts" : "application/alto-propmapparams+json",
    "uses" : [ "default-network-map" ]
    "capabilities" : {
      "entity-domains" : [ "ipv4", "ipv6" ],
"properties" : [ "pid" ]
}

"region-property-map": {
  "uri": "http://alto.example.com/propmap/region",
  "media-type": "application/alto-propmap+json",
  "accepts": "application/alto-propmapparams+json",
  "uses" : [ "default-network-map" ],
  "capabilities": {
    "domain-types": [ "pid" ],
    "properties": [ "region" ]
  }
}

"legacy-pid-property" : {
  "uri" : "http://alto.example.com/legacy/eps-pid",
  "media-type" : "application/alto-endpointprop+json",
  "accepts" : "application/alto-endpointpropparams+json",
  "capabilities" : {
    "properties" : [ "default-network-map.pid" ]
  }
}

Figure 6: Example IRD

7.4. Property Map Example

The following example uses the properties and IRD defined above to retrieve a Property Map for entities with the "ISP" and "ASN" properties.

Note that, to be compact, the response does not include the entity "ipv4:192.0.2.0", because values of all those properties for this entity are inherited from other entities.

Also note that the entities "ipv4:192.0.2.0/28" and "ipv4:192.0.2.16/28" are merged into "ipv4:192.0.2.0/27", because they have the same value of the "ASN" property. The same rule applies to the entities "ipv4:192.0.3.0/28" and "ipv4:192.0.3.0/28". Both of "ipv4:192.0.2.0/27" and "ipv4:192.0.3.0/27" omit the value for the "ISP" property, because it is inherited from "ipv4:192.0.2.0/23".

GET /propmap/full/inet-ia HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json
HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json

{
    "property-map": {
        "ipv4:192.0.2.0/23":   {"ISP": "BitsRus"},
        "ipv4:192.0.2.0/27":   {"ASN": "12345"},
        "ipv4:192.0.3.0/27":   {"ASN": "12346"}
    }
}

7.5. Filtered Property Map Example #1

The following example uses the filtered property map resource to request the "ISP", "ASN" and "state" properties for several IPv4 addresses.

Note that the value of "state" for "ipv4:192.0.2.0" is the only explicitly defined property; the other values are all derived by the inheritance rules for Internet address entities.

POST /propmap/lookup/inet-iacs HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json
Content-Length: ###
Content-Type: application/alto-propmapparams+json

{
    "entities" : [ "ipv4:192.0.2.0",
                   "ipv4:192.0.2.1",
                   "ipv4:192.0.2.17" ],
    "properties" : [ "ISP", "ASN", "state" ]
}
HTTP/1.1 200 OK
Content-Length: 
Content-Type: application/alto-propmap+json

{
   "property-map": {
      "ipv4:192.0.2.0":
      {"ISP": "BitsRus", "ASN": "12345", "state": "PA"},
      "ipv4:192.0.2.1":
      {"ISP": "BitsRus", "ASN": "12345", "state": "NJ"},
      "ipv4:192.0.2.17":
      {"ISP": "BitsRus", "ASN": "12345", "state": "CT"}
   }
}

7.6.  Filtered Property Map Example #2

The following example uses the filtered property map resource to request the "ASN", "country" and "state" properties for several IPv4 prefixes.

Note that the property values for both entities "ipv4:192.0.2.0/26" and "ipv4:192.0.3.0/26" are not explicitly defined. They are inherited from the entity "ipv4:192.0.2.0/23".

Also note that some entities like "ipv4:192.0.2.0/28" and "ipv4:192.0.2.16/28" in the response are not listed in the request explicitly. The response includes them because they are refinements of the requested entities and have different values for the requested properties.

The entity "ipv4:192.0.4.0/26" is not included in the response, because there are neither entities which it is inherited from, nor entities inherited from it.

POST /propmap/lookup/inet-iacs HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json
Content-Length: 
Content-Type: application/alto-propmapparams+json

{
   "entities" : [ "ipv4:192.0.2.0/26",
                   "ipv4:192.0.3.0/26",
                   "ipv4:192.0.4.0/26" ],
   "properties" : [ "ASN", "country", "state" ]
}
HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json

{
    "property-map": {
        "ipv4:192.0.2.0/26": {
            "country": "us"},
        "ipv4:192.0.2.0/28": {
            "ASN": "12345",
            "state": "NJ"},
        "ipv4:192.0.2.16/28": {
            "ASN": "12345",
            "state": "CT"},
        "ipv4:192.0.2.0": {
            "state": "PA"},
        "ipv4:192.0.3.0/26": {
            "country": "us"},
        "ipv4:192.0.3.0/28": {
            "ASN": "12345",
            "state": "TX"},
        "ipv4:192.0.3.16/28": {
            "ASN": "12345",
            "state": "MN"}
    }
}

7.7. Filtered Property Map Example #3

The following example uses the filtered property map resource to request the "pid" property for several IPv4 addresses and prefixes.

Note that the entity "ipv4:192.0.3.0/27" is redundant in the response. Although it can inherit a value of "defaultpid" for the "pid" property from the entity "ipv4:0.0.0.0/0", none of addresses in it is in "defaultpid". Because blocks "ipv4:192.0.3.0/28" and "ipv4:192.0.3.16/28" have already cover all addresses in that block. So an ALTO server who wants a compact response can omit this entity.

POST /propmap/lookup/pid HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json, application/alto-error+json
Content-Length: ###
Content-Type: application/alto-propmapparams+json

{
    "entities": [
        "ipv4:192.0.2.128",
        "ipv4:192.0.3.0/27"
    ],
    "properties": [ "pid" ]
}
HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json

```json
{
  "meta": {
    "dependent-vtags": [
      {"resource-id": "default-network-map",
       "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf62"}
    ],
  "property-map": {
    "ipv4:192.0.2.128": {"pid": "defaultpid"},
    "ipv4:192.0.2.0/27": {"pid": "defaultpid"},
    "ipv4:192.0.3.0/28": {"pid": "pid3"},
    "ipv4:192.0.3.16/28": {"pid": "pid4"}
  }
}
```

7.8. Filtered Property Map Example #4

The following example uses the filtered property map resource to request the "region" property for several PIDs defined in "default-network-map". The value of the "region" property for each PID is not defined by "default-network-map", but the reason why the PID is defined by the network operator.

POST /propmap/lookup/region HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json
Content-Length: ###
Content-Type: application/alto-propmapparams+json

```json
{
  "entities": ["pid:pid1",
               "pid:pid2"],
  "properties": [ "region" ]
}
```
HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json

{
    "meta": {
        "dependent-vtags": [
            {
                "resource-id": "default-network-map",
                "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf62"
            }
        ],
        "property-map": {
            "pid:pid1": {
                "region": "west"
            },
            "pid:pid2": {
                "region": "east"
            }
        }
    }
}

8. Security Considerations

Both Property Map and Filtered Property Map defined in this document fit into the architecture of the ALTO base protocol, and hence the Security Considerations (Section 15 of [RFC7285]) of the base protocol fully apply: authenticity and integrity of ALTO information (i.e., authenticity and integrity of Property Maps), potential undesirable guidance from authenticated ALTO information (e.g., potentially imprecise or even wrong value of a property such as geo-location), confidentiality of ALTO information (e.g., exposure of a potentially sensitive entity property such as geo-location), privacy for ALTO users, and availability of ALTO services should all be considered.

A particular fundamental security consideration when an ALTO server provides a Property Map is to define precisely the policies on who can access what properties for which entities. Security mechanisms such as authentication and confidentiality mechanisms then should be applied to enforce the policy. For example, a policy can be that a property P can be accessed only by its owner (e.g., the customer who is allocated a given IP address). Then, the ALTO server will need to deploy corresponding mechanisms to realize the policy. The policy may allow non-owners to access a coarse-grained value of the property P. In such a case, the ALTO server may provide a different URI to provide the information.
9. IANA Considerations

This document defines additional application/alto-* media types, and extends the ALTO endpoint property registry.

9.1. application/alto-* Media Types

This document registers two additional ALTO media types, listed in Table 1.

<table>
<thead>
<tr>
<th>Type</th>
<th>Subtype</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>alto-propmap+json</td>
<td>Section 4.1</td>
</tr>
<tr>
<td>application</td>
<td>alto-propmapparams+json</td>
<td>Section 5.3</td>
</tr>
</tbody>
</table>

Table 1: Additional ALTO Media Types.

Type name: application

Subtype name: This document registers multiple subtypes, as listed in Table 1.

Required parameters: n/a

Optional parameters: n/a

Encoding considerations: Encoding considerations are identical to those specified for the "application/json" media type. See [RFC7159].

Security considerations: Security considerations related to the generation and consumption of ALTO Protocol messages are discussed in Section 15 of [RFC7285].

Interoperability considerations: This document specifies formats of conforming messages and the interpretation thereof.

Published specification: This document is the specification for these media types; see Table 1 for the section documenting each media type.

Applications that use this media type: ALTO servers and ALTO clients either stand alone or are embedded within other applications.

Additional information:
9.2. ALTO Entity Domain Registry

This document requests IANA to create and maintain the "ALTO Entity Domain Registry", listed in Table 2.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Entity Identifier Encoding</th>
<th>Hierarchy &amp; Inheritance</th>
<th>Mapping to ALTO Address Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipv4</td>
<td>See Section 3.1.1</td>
<td>See Section 3.1.3</td>
<td>Yes</td>
</tr>
<tr>
<td>ipv6</td>
<td>See Section 3.1.2</td>
<td>See Section 3.1.3</td>
<td>Yes</td>
</tr>
<tr>
<td>pid</td>
<td>See Section 3.2</td>
<td>None</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 2: ALTO Entity Domains.

This registry serves two purposes. First, it ensures uniqueness of identifiers referring to ALTO entity domains. Second, it states the requirements for allocated entity domains.
9.2.1. Consistency Procedure between ALTO Address Type Registry and ALTO Entity Domain Registry

One potential issue of introducing the "ALTO Entity Domain Registry" is its relationship with the "ALTO Address Types Registry" already defined in Section 14.4 of [RFC7285]. In particular, the entity identifier of an entity domain registered in the "ALTO Entity Domain Registry" MAY match an address type defined in "ALTO Address Type Registry". It is necessary to precisely define and guarantee the consistency between "ALTO Address Type Registry" and "ALTO Entity Domain Registry".

We define that the ALTO Entity Domain Registry is consistent with ALTO Address Type Registry if two conditions are satisfied:

- When an address type is already or able to be registered in the ALTO Address Type Registry [RFC7285], the same identifier MUST be used when a corresponding entity domain is registered in the ALTO Entity Domain Registry.

- If an ALTO entity domain has the same identifier as an ALTO address type, their addresses encoding MUST be compatible.

To achieve this consistency, the following items MUST be checked before registering a new ALTO entity domain in a future document:

- Whether the ALTO Address Type Registry contains an address type that can be used as an entity identifier for the candidate domain identifier. This has been done for the identifiers "ipv4" and "ipv6" in Table 2.

- Whether the candidate entity identifier of the entity domain is able to be an endpoint address, as defined in Sections 2.1 and 2.2 of [RFC7285].

When a new ALTO entity domain is registered, the consistency with the ALTO Address Type Registry MUST be ensured by the following procedure:

- Test: Do corresponding entity identifiers match a known "network" address type?
  * If yes (e.g., cell, MAC or socket addresses):
    + Test: Is such an address type present in the ALTO Address Type Registry?
- If yes: Set the new ALTO entity domain identifier to be the found ALTO address type identifier.

- If no: Define a new ALTO entity domain identifier and use it to register a new address type in the ALTO Address Type Registry following Section 14.4 of [RFC7285].

+ Use the new ALTO entity domain identifier to register a new ALTO entity domain in the ALTO Entity Domain Registry following Section 9.2.2 of this document.

* If no (e.g., pid name, ane name or country code): Proceed with the ALTO Entity Domain registration as described in Section 9.2.2.

9.2.2. ALTO Entity Domain Registration Process

New ALTO entity domains are assigned after IETF Review [RFC5226] to ensure that proper documentation regarding the new ALTO entity domains and their security considerations has been provided. RFCs defining new entity domains SHOULD indicate how an entity in a registered domain is encoded as an EntityId, and, if applicable, the rules defining the entity hierarchy and property inheritance. Updates and deletions of ALTO entity domains follow the same procedure.

Registered ALTO entity domain identifiers MUST conform to the syntactical requirements specified in Section 2.3. Identifiers are to be recorded and displayed as strings.

Requests to the IANA to add a new value to the registry MUST include the following information:

- Identifier: The name of the desired ALTO entity domain.

- Entity Identifier Encoding: The procedure for encoding the identifier of an entity of the registered type as an EntityId (see Section 2.4). If corresponding entity identifiers of an entity domain match a known "network" address type, the Entity Identifier Encoding of this domain identifier MUST include both Address Encoding and Prefix Encoding of the same identifier registered in the ALTO Address Type Registry [RFC7285]. For the purpose of defining properties, an individual entity identifier and the corresponding full-length prefix MUST be considered aliases for the same entity.

- Hierarchy: If the entities form a hierarchy, the procedure for determining that hierarchy.
o Inheritance: If entities can inherit property values from other entities, the procedure for determining that inheritance.

o Mapping to ALTO Address Type: A boolean value to indicate if the entity domain can be mapped to the ALTO address type with the same identifier.

o Security Considerations: In some usage scenarios, entity identifiers carried in ALTO Protocol messages may reveal information about an ALTO client or an ALTO service provider. Applications and ALTO service providers using addresses of the registered type should be made aware of how (or if) the addressing scheme relates to private information and network proximity.

This specification requests registration of the identifiers "ipv4", "ipv6" and "pid", as shown in Table 2.

9.3. ALTO Entity Property Type Registry

This document requests IANA to create and maintain the "ALTO Entity Property Type Registry", listed in Table 3.

To distinguish with the "ALTO Endpoint Property Type Registry", each entry in this registry is an ALTO entity property type defined in Section 2.5. Thus, registered ALTO entity property type identifier MUST conform to the syntactical requirements specified in that section.

The initial registered ALTO entity property types are listed in Table 3.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Intended Semantics</th>
<th>Dependencies and Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipv4:pid</td>
<td>PID for the IPv4 entity</td>
<td>application/alto-networkmap+json, where the PID names are defined</td>
</tr>
<tr>
<td>ipv6:pid</td>
<td>PID for the IPv6 entity</td>
<td>application/alto-networkmap+json, where the PID names are defined</td>
</tr>
</tbody>
</table>

Table 3: ALTO Entity Property Types.

Requests to the IANA to add a new value to the registry MUST include the following information:

o Identifier: The unique id for the desired ALTO entity property type. The format MUST be as defined in Section 2.5 of this
document. It includes the information of the applied ALTO entity domain and the property name.

- Intended Semantics: ALTO entity properties carry with them semantics to guide their usage by ALTO clients. Hence, a document defining a new type SHOULD provide guidance to both ALTO service providers and applications utilizing ALTO clients as to how values of the registered ALTO entity property should be interpreted.

- Dependencies and Interpretation: Dependent ALTO resources MAY be required by ALTO clients to interpret ALTO entity properties. Hence, a document defining a new type SHOULD provide a sequence of media types in which the dependent ALTO resources are and the guidance how ALTO clients use them to interpret the property.

This specification requests registration of the identifiers "ipv4:pid" and "ipv6:pid", as shown in Table 3.

9.4. Acknowledgments

The authors would like to thank discussions with Kai Gao, Qiao Xiang, Shawn Lin, Xin Wang, Danny Perez, and Vijay Gurbani. The authors thank Dawn Chen (Tongji University), and Shenshen Chen (Tongji/Yale University) for their contributions to earlier drafts.

10. Normative References


Internet-Draft             Unified Properties                 March 2019

               Address Text Representation", RFC 5952,
               DOI 10.17487/RFC5952, August 2010,

               Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March

               Previdi, S., Roome, W., Shalunov, S., and R. Woundy,
               "Application-Layer Traffic Optimization (ALTO) Protocol",
               RFC 7285, DOI 10.17487/RFC7285, September 2014,

               Nadeau, "An Architecture for the Interface to the Routing
               System", RFC 7921, DOI 10.17487/RFC7921, June 2016,

Authors’ Addresses

Wendy Roome
Nokia Bell Labs (Retired)
124 Burlington Rd
Murray Hill, NJ 07974
USA

Phone: +1-908-464-6975
Email: wendy@wdroome.com

Sabine Randriamasy
Nokia Bell Labs
Route de Villejust
NOZAY 91460
FRANCE

Email: Sabine.Randriamasy@nokia-bell-labs.com

Y. Richard Yang
Yale University
51 Prospect Street
New Haven, CT 06511
USA

Phone: +1-203-432-6400
Email: yry@cs.yale.edu

Jingxuan Jensen Zhang
Tongji University
4800 Caoan Road
Shanghai 201804
China

Email: jingxuan.n.zhang@gmail.com
Abstract

Evolving networking scenarios (e.g., 5G) are considering the provision of value-added and on-demand end-to-end (E2E) network services in multi-domain (multi-operator/multi-technology) environments. This document presents different initiatives, mainly within standardization efforts and research projects, working on E2E network services across multiple domains. Problem statement and a layered network model are also described. In addition, this document raises an initial proposal towards a new ALTO service in support of E2E network service requirements. Finally, another important objective of this document is to begin a discussion about motivating use cases in scope of the ALTO WG after the re-chartering process.

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 https://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 12, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document.
The fifth generation (5G) of cellular networks is not only considered an evolution but a revolution in the field of information and communication technologies [WHITE-PAPER-5G]. 5G will support the creation of new and novel End-to-End (E2E) services, applications and complex use case scenarios, such as massive Internet of Things, extreme real-time communications, broadband access everywhere, higher user mobility. All these scenarios and services are triggering a modification in the way telecommunications sector deploy new network services, shifting from a commonly manual and long process to a flexible and programmable process.

In this context, cloud computing, Software Defined Networking (SDN), and Network Function Virtualization (NFV) arise as technological
pillars to achieve the necessary function programmability, network programmability, and resource virtualization during the provision of E2E network services.

The delivery of an E2E network service, or simply E2E service, often requires VNFs and their specific order [RFC7665]. Network operators start offering to their customers the possibility of configuring network services with specific requirements in terms of resources (e.g., cpu, memory, hard-disk) and performance objectives (e.g., bandwidth, latency) [VNF-PLAC]. Such demands are usually composed by distributed resources which are expected to available across multiple domains with different technology and/or administration.

This document offers an overview of standardization activities and research projects, including problem statement, behind building E2E services traversing different domains (technological and/or administrative). Moreover, from a layered network model, it is proposed a potential ALTO extension related to E2E Network Service requirements representation based on the ETSI NFV MANO data model.

The overall rationale of this document is to arouse discussions into the ALTO WG concerning potential new items to be considered for the re-charter.

2. Context and Motivation

Different standardization efforts (e.g., IETF, MEF, ETSI) and research projects activities (e.g., 5GEx [H2020.5GEX], 5G-Transformer [H2020-5G-TRANSFORMER], T-NOVA [T-NOVA]) have been focused on multi-domain network service chaining. Standardization is essential to provide recommendations to create interoperable architectures with standardized protocols, and solutions (being developed by different projects) are addressing a diverse range of requirements to provide network services provided using multiple domains.

This section briefly describes, on the one hand, main standardization efforts delivering collections of norms and recommendations, while on the other hand it also provides an overview of several projects formed to develop network services across multiple domains.

2.1. Standardization Activities

2.1.1. IETF

SFC that span domains owned by single or multiple administrative entities are being proposed. The Hierarchical Service Function Chaining (hSFC) [RFC8459], for example, defines an architecture to deploy SFC in large networks. This RFC proposes to decompose the
network into smaller domains (domains under the control of a single organization). Another proposed initiative is [DRAFT-HH-MDSFC] that describes SFC crossing different domains owned by various organizations (e.g., ISPs) or by a single organization with administration partitions. The proposed architecture uses a SFC eXchange Platform (SXP) to collect and exchange information (topology, service states, policies, etc.) between different organizations and it works both in centralized (Multiple SFC domains connected by a logical SXP) and distributed (SXP server as a broker) environments.

More recently, the IETF ALTO WG started to discuss the uses of ALTO as an information model for representing network resource and services in multi-domain scenarios:

- [DRAFT-ALTO-BROKER-MDO] proposes an ALTO-based Broker-assisted architecture where a broker plane works as a coordinator between a set of top-level control planes, i.e., Domain Orchestrators (DOs) and Multi-Domain Orchestrators (MdOs). The ALTO services (with the proposed extensions) provides abstract maps with a simplified, yet enough information view about MdOs involved in the federation. This information includes the abstract network topology, resource availability (e.g., CPUs, Memory, and Storage) and capabilities (e.g., supported network functions).

- [DRAFT-ALTO-UNICORN] presents Unicorn, a resource orchestration framework for multi-domain, geo-distributed data analytics. This work resorts in ALTO as the information model to support the accurate, yet privacy-preserving resource discovery across different domains. The key information to be provided by the use of ALTO including different types of resources, e.g., the computing, storage, and networking resources.

- [DRAFT-MD-SFC-ALTO] describes different standardization activities and research projects addressing the challenges posed by Service Function Chaining (SFC) across multiple domains (specifically, multiple administrative domains). In addition, this document presents an initial approach to realize inter-domain service chaining leveraging the ALTO protocol. Finally, another important concern of this document is to initiate a discussion (ALTO, SFC as well as 9other WGs) regarding if, how, and under what conditions ALTO can be useful to improve the multi-domain SFC process.

2.1.2. ETSI

The ETSI NFV ISG is paving the way toward viable architectural options supporting the efficient placement of functions in different administrative domains. More specifically, the document
[ETSI-NFV-IFA028] reports different NFV MANO architectural approaches with use cases related to network services provided using multiple administrative domains. Besides, it gives a non-exhaustive list of key information to be exchanged between administrative domains (monitoring parameters, topology view, resource capabilities, etc.) and recommendations related to security to permit the correct and proper operation of the final service.

2.1.3. MEF

With its work on the Service Operations Specification MEF 55 [MEF-SOE-MEF55], MEF has defined a reference architecture and framework for describing functional management entities (and interfaces between them) needed to support Lifecycle Service Orchestration (LSO). This LSO architecture enables automated management and control of E2E connectivity services across multiple operator networks. The automated service management includes fulfillment, control, performance, assurance, usage, security, analytics, and policy capabilities that make it possible, for example, expanding the footprint of service providers to interact with potentially several operators to manage and control the access portions of E2E services.

2.2. Research projects

Several projects include an architectural model integrating NFV management with SDN control capabilities to address the challenges towards flexible, dynamic, cost-effective, and on-demand service chaining.

[H2020.5GEX] aims to integrate multiple administrations and technologies through the collaboration between operators in the context of emerging 5G networking. [VITAL][T-NOVA] follow a centralized approach where each domain advertises its capabilities to a federation layer which will act as a broker. In order to avoid one network operator per country or regions, [H2020-5G-NORMA] proposes the use of management and control into a single virtual domain. Also, the 5G-Transformer project [H2020-5G-TRANSFORMER] is defining flexible slicing and federation of transport networking and computing resources across multiple domains. The NECOS project [H2020-NECOS], focused on the realization of E2E multi-domain cloud network slicing, proposes an architectural approach with slice information interfaces for resource exposure and resource discovery during slice provisioning. In addition, the architecture includes a slice marketplace interface between domain orchestrators and a marketplace broker.
3. Problem Statement

3.1. Network Function Placement Decisions

An E2E service request specify virtual nodes (a set of required VNFs) as well as virtual links (the order in which they must be executed). Virtual nodes are deployed in virtual machines hosted by different physical servers, and virtual links correspond to physical paths that connect those servers hosting VNFs. Both virtual nodes and virtual links are limited resources and both may also be located on different technological domains in a single administration and even crossing multiple administrations [VNF-MOB][SFC-ORC]. So that the placement decision problem involves to discover "best" candidate resources and "best" feasible paths between such resources.

3.2. Network Inventory

Placement decisions are a fundamental step for the management and orchestration of network services. Management systems (e.g., DOs, MdOs) need to maintain an inventory of the network providing a real-time representation or view of available infrastructure resources, software resources, and their relationships. However, the size of a network inventory can be very large in scenarios, such as distributed cloud and edge computing. As a result, management systems experiment scalability problems processing large amounts of data to decide where to instantiate a service or part of the service. Therefore, building a network inventory, under these circumstances, needs aggregation mechanisms to reduce time for discovery of resources and to simplify and optimize management of them.

3.3. Publishing Information

Once a network inventory is built, a mechanism for publishing information is also necessary so that the network inventory can provide a simplified, yet enough network information view to management systems. In order to retrieve such information to perform placement decisions, a communication protocol between management systems and network inventory is also necessary.

Therefore, on the one hand, network information (e.g., network locations, costs between them, endhost properties) needs to be advertised to the network applications and, on the other hand, network applications (e.g., DOs, MdOs) needs to describe their requirements and obtain information about resources that suit such requirements.
4. Network Function Virtualization Architectures and Infrastructures

With the introduction of NFV, network functions (e.g., switches, routers, firewalls), and also complex network functions (e.g., EPC) are able to be virtualized and implemented as a collection of virtual machines (VMs) deployed over the virtualized infrastructure. In turn, the virtualized infrastructure is instantiated on a substrate network.

In this context, one of the most accepted NFV architectural frameworks is the proposed by the ETSI ISG NFV working group [ETSI-NFV-WHITEPAPER]. Figure 1 [DRAFT-MD-VIRT] shows this NFV reference architecture. On the left, we can see the data plane: NFVIs hardware/software, VNFs, and optional element management systems. On the right, we see the control plane: VIM which is something like Openstack or Kubernetes, virtual network function managers, and the NFV Orchestrator on the top.
4.1. Layered Network Model

Based on the ETSI NFV reference architecture, a layered network model is identified: Network Service layer, VNF layer, and Resource Layer. This model allows a separation of network relationships in different levels of abstraction. For example, network services can be queried at different levels of abstraction or we can map service paths in different layers (from an abstract to a more concrete layer).

In Figure 2, we have a network service with a set of interconnected VNFs. This network service topology is represented by the Network Service layer.
A VNF is typically divided into a set of virtual function components (VFCs) which comprise the VNF Layer. Each VFC is an application running within a single VM or container.

In case of the resource layer, we have virtual layer and physical layer. The virtual layer represents the virtual overlay network and the physical layer represents the substrate network. Virtualized infrastructures (e.g., VMs, virtual routers) are instantiated on a physical infrastructure.
5. ALTO Extension: E2E Network Service Requirements Representation

From the layered network model described in the previous section, we are considering an ALTO extension related to E2E Network Service requirements representation. An initial proposal has been presented in the ALTO-based Broker-assisted MdO draft [DRAFT-ALTO-BROKER-MDO] where network applications (as ALTO clients) can specify a set of
basic E2E service requirements to an ALTO server in order to obtain 
candidate resources (domains) and candidate paths.

This initial E2E service requirement representation is inspired on 
the ETSI NFV MANO data model [ETSI-NFV-MAN001]. This model defines 
network services as a composition of network functions including the 
specification of deployment and operational requirements. Such 
specifications are captured in templates called Network Service 
Descriptor (NSD) and Virtual Network Function Descriptor (VNFD) that 
contain (relatively) static information used in the process of on-
boarding network services and VNFs, respectively.

- High level objects in a NSD include (among others) 
  [ETSI-NFV-MAN001][OSM-DM]:
  - Constituent VNFs: List of VNFDs that are part of the network 
    service.
  - VNF Dependencies: This describes dependencies between VNFs. 
    For example, the order in which the VNFs inside a network 
    service should be started.
  - Network service Connection Points: Each network service has one 
    or more external connection points (which act as endpoints) 
    used to link two network services or to link external networks.
  - Virtual Links: List of Virtual Link Descriptors (VLDs) that 
    describe how VNFs (in the NSD) are connected.

- High level objects in a VNFD include (among others) 
  [ETSI-NFV-MAN001][OSM-DM]:
  - Constituent VDUs: List of virtual deployment units (VDUs) in a 
    specific VNF. Each VDU (also referred to VFC) describes the 
    VM/Container capabilities (e.g., CPU, RAM, disks).
  - VDU Dependencies: List of VDU dependencies used for determining 
    the order of startup for VDUs.
  - VNF Connection Points: List of external connection points used 
    for connecting a VNF to other VNFs or to external networks.
  - Internal VLDs: List of internal virtual links to connect 
    various VDUs/VFCs.
6. IANA Considerations

This document includes no request to IANA.

7. Security Considerations

TBD.

8. Acknowledgments

This work is supported by the Innovation Center of Ericsson S.A., Brazil (grant agreement UNI.64). This document however does not necessarily represent Ericsson’s official viewpoint.

9. References

9.1. Normative References


9.2. Informative References


Authors’ Addresses
ALTO for Multi-Domain Applications: A Review of Use Cases and Design Requirements
draft-xiang-alto-multidomain-usecases-00.txt

Abstract

With the development of novel network technology, such as software defined networking and network function virtualization, many novel multi-domain applications, such as flexible interdomain routing, distributed, federated machine learning and multi-domain collaborative dataset transfer, have been deployed. These applications can benefit substantially from the ALTO protocol [RFC7285], through which the information of multiple networks can be provided to applications. This document first introduces several multi-domain applications and how they can benefit from ALTO. It then describes a generic framework for multi-domain applications to use ALTO to improve the performance, followed by a discussion on new requirements and challenges for ALTO to better support these applications.

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 12, 2019.
1. Introduction

The ALTO protocol [RFC7285] provides network information to applications so that applications can make network informed decisions to improve the performance. Not only traditional applications such as peer-to-peer systems, many recent, novel multi-domain applications,
which orchestrate resources across multiple networks, can also benefit substantially from ALTO.

The goal of this document is to explore how ALTO can help improve the performance of novel multi-domain applications, what ALTO extension services are needed, and what are the corresponding requirements and challenges for designing such extensions. To this end, this document first give a case-by-case review of emerging multi-domain applications and how they can benefit from ALTO. It then describes a generic framework for multi-domain applications to use ALTO to improve the performance, followed by a discussion on the need of new ALTO services and the corresponding requirements and challenges for these extensions to better support these applications.

2. 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 [RFC2119].

3. Review of Multi-Domain Applications

3.1. Flexible Interdomain Routing

Flexible interdomain routing can be a highly valuable service for network providers. Specifically, an autonomous system (AS) providing such a service (the provider) allows other ASes (clients) to specify routing actions at the provider based on flexible matching conditions (e.g., match on TCP/IP 5-tuple). In this way, a client AS using the flexible interdomain routing service can offload access and traffic control to provider ASes, leading to a simpler client network configuration while giving the provider ASes additional business opportunities.

3.1.1. How flexible interdomain routing can benefit from ALTO?

ALTO provides provider ASes a standardized approach to expose its routing capability to client ASes. Traditional interdomain routing protocols such as BGP are not good options because they only expose the currently used routes, limiting client ASes’ choices to specify flexible routes. In contrast, ALTO and its extensions provide interfaces for provider ASes to expose not only currently used routes, but also available yet unused routes, to client ASes so that they can have the flexibility to specify different routes for different data traffic.
3.1.2. Example

Consider the example in Figure 1. AS A is compromised and being used to send DDoS traffic to AS E. Without flexible interdomain routing, AS E can set up a firewall locally, but normal traffic from B to E will still be congested at C-D-E due to the existence of malicious traffic from A to E. If AS C provides flexible interdomain routing service, AS E can specify such a firewall at AS C to block DDoS traffic from A, and at the same time avoid the congestion of normal traffic from B to E.

```
+-----+ DDoS traffic
| AS A |
|      \\
| AS C ---------- AS D -- AS E |
|      /|
| AS B --|
| +-----+ Normal traffic

Figure 1: Flexible interdomain routing for DDoS mitigation.
```

3.2. Resource Orchestration for Collaborative Data Sciences

As the data volume increases exponentially over time, data analytics is transitioning from a single-domain network to a multi-domain, geographically distributed network, where different member networks contribute various resources, e.g., computation, storage and networking resources, to collaboratively collect, share and analyze extremely large amounts of data. Such a paradigm calls for a unified resource orchestration framework to manage a large set of distributively-owned, heterogeneous resources, with the objective of efficient resource utilization, following the autonomy and privacy of different domains.

3.2.1. How multi-domain resource orchestration can benefit from ALTO

One key design challenge for multi-domain resource orchestration is its resource information model. Existing design options such as resource graph and ClassAds are inadequate because they cannot simultaneously (1) allow member networks to provide accurate information on different types of resource, (2) avoid the exposure of private information of member networks such as topology, and (3) allow data analytics jobs to accurately describe their requirements of different types of resources. In contrast, the section 7.1 of
Figure 2 discusses the advantages of choosing ALTO as the resource information model for multi-domain resource orchestration, and how ALTO can simultaneously satisfy the aforementioned design requirements.

3.2.2. Example

Consider an example of three member networks in Figure 2, where s1 and s2 are storage endpoints and d1 and d2 are computation endpoints. Assume a data analytics job is composed of two parallel tasks T1 and T2. T1 needs dataset X as input and T2 needs dataset Y as input.

![Diagram showing multi-domain resource orchestration](image.png)

Using the ALTO endpoint property service, an ALTO client in the resource orchestrator can discover that d1 satisfies the computing requirements of T1 and d2 satisfies the computing requirements of T2. Hence there are only two candidate endpoint pairs: (s1, d1) and (s2, d2).

Afterwards, using the ALTO path vector extension, the ALTO client can retrieve the bandwidth sharing information of task T1 and T2, denoted as x1 and x2, respectively, as follows.

\[
\begin{align*}
A: & \quad x_1 + x_2 \leq 10\text{Mbps} \\
B: & \quad x_1 \leq 3\text{Mbps} \\
C: & \quad x_2 \leq 3\text{Mbps}
\end{align*}
\]

With such information, the resource orchestrator can make the optimal resource orchestration decision to reserve 3 Mbps bandwidth for task T1, and 3 Mbps bandwidth for task T2.
3.3. Federated Machine Learning

Instead of moving large-scale datasets from multiple devices / networks to a centralized location for training, federated learning, is a distributed machine learning approach which enables training on distributed datasets residing on different autonomous systems (devices or networks). In this way, only updates on the training model need to be communicated between networks, leading to substantial reduction of networking resource consumption (e.g., saving bandwidth).

3.3.1. How federated machine learning can benefit from ALTO

Federated machine learning requires efficient scheduling algorithms to decide how networking resources should be allocated to transmit training model updates between different ASes. Similar as moving large-scale datasets between multiple ASes, moving updates of training model between ASes can also benefit from the availability of networking information, such as the AS-path and bandwidth sharing. ALTO provides a standardized approach for federated machine learning schedulers to retrieve such information from networks so that adaptive scheduling decisions can be made.

3.3.2. Example

Consider the example in Figure 3, where machine learning workers are located in AS A and D, while AS B and C are transit networks for data traffic transmitted between A and D. When AS A has a large, critical training model update to send to D. It first queries the ALTO servers and B and C for the endpoint cost (e.g., bandwidth) to transmit data from A to D. Suppose the ALTO server at AS B returns an endpoint cost of 10Mbps, while the ALTO server at AS C returns an endpoint cost of 100 Mbps. AS A can then use such information to make the optimal model update scheduling algorithm to send the training model update to AS D via AS C, instead of AS B.
4. A Generic Framework

After reviewing several important, novel multi-domain applications that can benefit substantially from ALTO, this document describes a generic framework for such applications to use ALTO to retrieve information from networks to improve their performance. The high-level architecture of this framework is given in Figure 4.

Figure 3: Federated machine learning.
The top layer of this framework is the application layer, in which each application deploy one or more ALTO clients to query for information provided by networks. The middle layer is the service layer. In this layer, each network deploys one more more ALTO servers to respond the queries sent by the ALTO clients from applications, and deploys one or more execution agents to respond to the applications’ resource consumption actions. The bottom layer is the signaling layer, in which each network deploys interdomain protocols / systems, such as routing protocol BGP and resource reservation system OSCARS.

### 4.1. Workflow

The basic workflow of this framework is as follows.

- An application identifies the networks whose resources (e.g., networking, computation and storage) it may want to consume, and invokes its ALTO clients to query the ALTO servers deployed in those networks for detailed resource information using base ALTO

---

**Figure 4: Generic framework of using ALTO in multi-domain applications.**

---

```
<table>
<thead>
<tr>
<th>Application Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Application N</td>
</tr>
<tr>
<td>ALTO Client 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>ALTO Client N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network 1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>ALTO Server 1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Execution Agent 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>ALTO Server N</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Execution Agent N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signaling Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Network N</td>
</tr>
</tbody>
</table>
```

---

protocol and its extension services (e.g., path vector, cost calendar and so on);

- Upon receiving a query from an ALTO client, an ALTO server checks its local information, contacts the underlying signaling layer protocol / system of its residing network if local information is outdated, and returns the latest resource information to the querying ALTO client;

- The applications uses the resource information collected from ALTO servers to make resource allocation decisions (e.g., route selection, resource reservation, etc.), and send such decisions to corresponding execute agents in the corresponding networks (e.g., the simple-reservation-interface of OSCARS).

5. Requirements of ALTO in Multi-Domain Applications

Using ALTO to improve the performance of recent novel multi-domain applications poses several new design requirements. This section discusses these requirements and briefly review existing efforts in the ALTO working group aiming to satisfy them.

5.1. Design Requirements

- Exposing information of alternative resources. Current ALTO protocols and its extensions only provide information of currently used resources (e.g., currently used interdomain route). However, exposing information of alternative resources (e.g., available but not used interdomain routes) may provide the users of new multi-domain applications (e.g., flexible interdomain routing) more flexibility on choosing different resources, giving networks that provide such applications additional business opportunities.

- Providing a unified, accurate representation of multiple types of resources. Current ALTO protocols and its extensions mainly focus on providing network information to applications, with the exception of endpoint property service. However, as new multi-domain applications often consume multiple types of resources across multiple networks, encoding such information accurately in a unified approach is crucial for deploying ALTO to improve such applications’ performance.

- Providing interfaces for more flexible query. Current ALTO protocol and its extensions allows applications to query resource information by specifying IP addresses of endpoints and simple filters. However, with the emerging of new networking architecture (e.g., software defined networking and network function virtualization) and the fine-grained resource requirement...
of applications (e.g., link-disjoint paths and endpoint precedence), applications need a more flexible interface to specify queries of resource information.

5.2. Existing Efforts in the ALTO Working Group

Several documents have been submitted to the ALTO working group, with the aim to satisfy one or more of the design requirements discussed above. For example, [DRAFT-PV], [DRAFT-RSA], [DRAFT-UNICORN-INFO] and several other documents propose and apply the ALTO path vector extension to provide accurate networking resource information to support multi-domain resource orchestration. [DRAFT-NFCHAIN] proposes to use ALTO to support resource orchestration for multi-domain service function chaining, and proposes a new ALTO extension to retrieve AS path of network functions across different networks. [DRAFT-CONTEXT] proposes to extend cost information specified in RFC7285 by providing several possible cost values for the same cost metric where each value depends on qualitative criteria as opposed to quantitative criteria such as time. [DRAFT-UR] makes a proposal to use mathematical programming constraint as a generic representation of multiple resources. [DRAFT-FCS] proposes a flexible flow query extension service to allow applications to specify query entities based on flexible matching conditions (e.g., TCP/IP 5-tuple) instead of IP addresses only.

6. Summary

This document reviews several emerging multi-domain applications and how they can benefit from ALTO. It then describes a generic framework for multi-domain applications to use ALTO to improve the performance. In addition, several design requirements are discussed. Though different drafts in the working group have been trying to address one or more these design requirements, a systematic investigation of these issues is still missing. The authors of this document plan to perform such an investigation and make a unified design proposal in the next version of this document.

7. References

7.1. Normative References


7.2. Informative References

[DRAFT-CONTEXT]

[DRAFT-FCS]

[DRAFT-NFCHAIN]

[DRAFT-PV]

[DRAFT-RSA]

[DRAFT-UNICORN-INFO]

[DRAFT-UP]


Authors’ Addresses

Qiao Xiang
Yale University
51 Prospect Street
New Haven, CT
USA

Email: qiao.xiang@cs.yale.edu

Franck Le
IBM
Thomas J. Watson Research Center
Yorktown Heights, NY
USA

Email: fle@us.ibm.com

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

Email: yry@cs.yale.edu
Abstract

The ALTO protocol [RFC7285] provides network information to applications so that applications can make network informed decisions to improve the performance. However, the base ALTO protocol only provides coarse-grained end-to-end metrics, which are insufficient to satisfy the demands of applications to solve more complex network optimization problems. The ALTO Path Vector extension [DRAFT-PV] has been introduced to allow ALTO clients to query information such as capacity regions for a given set of flows. However, the current design of this extension has a limited expressiveness. The goal of this document is to introduce a unified resource representation service as an extension of ALTO (ALTO-UR), which allows the ALTO clients to query and get the capacity regions of more complex resource information, such as Shared-Risk-Link-Group (SRLG), multi-path routing, multicast and on-demand routing, for a given set of flows.

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 12, 2019.
1. Introduction

As discussed in [DRAFT-PV], the "one-big-switch" abstraction used in the ALTO base protocol lacks the ability to support emerging use cases, such as inter-datacenter data transfers, because this abstraction cannot reveal the resource sharing, i.e., the capacity
The ALTO Path Vector extension addresses this insufficiency by using the path vector abstraction to express the capacity region in a set of linear inequalities. However, in an internal discussion with the leading persons of several important ALTO use cases, it is revealed that the expressiveness of the ALTO Path Vector extension is limited in three aspects:

- It cannot provide compact encoding of the SRLG for a set of flows;
- It assumes that each flow in the client’s query will use a single-path route, and hence cannot encode the resource sharing for flows that are forwarded along multi-path routes or multicast flows;
- It assumes that the route of each flow in the client’s query is pre-computed, and hence cannot encode the resource sharing for flows that use on-demand routing, e.g., the path computation element (PCE) protocol.

To cope with these issues, this document introduces a new ALTO extension, the unified representation (ALTO-UR). This extension expands the linear inequality encoding of capacity regions used in ALTO-PV, to a generic, complete encoding, which uses mathematical programming constraints to represent the capacity regions for a set of flows.

2. 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 [RFC2119].

3. Changes Since Version -00

- Add an initial proposal to design a more flexible resource information query interface in Section 5.5.

4. Limitations of the ALTO Path Vector Extension

The limitations of the ALTO-PV extension are illustrated with the same dumbbell topology used in [DRAFT-PV]. Assume that the bandwidth of every link is 100 Mbps, and that the SRLG of each link is shown in Figure 1. Consider an application overlay (e.g., a large data analytics system) which wants to schedule the traffic among a set of end host source-destination pairs, say eh1 -> eh2 and eh1 -> eh4.
o Assume the application is only interested in the SRLG of both flows, not the bandwidth. The route of eh1 -> eh2 is eh1 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh2, and the route of eh3 -> eh4 is eh3 -> sw1 -> sw5 -> sw7 -> sw2 -> eh4. The minimal yet accurate information returned to the application should be \( (2, 3, 4) \), the SRLG of both flows, since flow 1 has a set of SRLG \( (1, 2, 3, 4) \) and flow 2 has a set of SRLG \( (2, 3, 4) \). In contrast, in the current ALTO-PV service, the ALTO server needs to return the ane-path of each flow and the SRLG of every ane, where ane and ane-path are defined in [DRAFT-PV]. This response is redundant and causes unnecessary information exposure to the application, e.g., the information of flow has an SRLG 1 should not be returned to the application.

o Assume the application is only interested in the bandwidth of both flows. The route of eh1 -> eh2 is eh1 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh2, and the route of eh3 -> eh4 is a multi-path route, i.e., \( \{eh3 \rightarrow sw1 \rightarrow sw5 \rightarrow sw7 \rightarrow sw2 \rightarrow eh4, eh3 \rightarrow sw1 \rightarrow sw5 \rightarrow sw6 \rightarrow sw7 \rightarrow sw2 \rightarrow eh4\} \), where each path would forward 50 percent of the traffic for eh3 -> eh4. The ALTO-PV service cannot reveal the traffic split of the multi-path route for eh3 -> eh4, or the bandwidth sharing for both flows on link sw5 -> sw6.

o Assume the network has a PCE server, through which the application can reserve bandwidth for both flows. Before the application makes the reservation request, the application queries the ALTO server to get the bandwidth capacity region of both flows, which it wants to use to decide how much bandwidth to reserve for each flow. Suppose when the ALTO server receives a query, the network

Figure 1: A Dumbbell Network Topology
only has two precomputed routes for both flows: eh1 -> sw1 -> sw5
-> sw6 -> sw7 -> sw2 -> eh2, and eh3 -> sw1 -> sw5 -> sw6 -> sw7
-> sw2 -> eh4. Through the ALTO-PV service, the application
receives the information that the total bandwidth it can reserve
for both flows cannot exceed 100 Mbps. However, one important
feature of the PCE server is that it can compute the route for
reservation request on-demand, and hence it can find routes with a
larger bandwidth. In this example, if the application submits a
request to reserve 100 Mbps bandwidth for each flow, the PCE
server can compute two on-demand routes, i.e., eh1 -> sw1 -> sw5
-> sw6 -> sw7 -> sw2 -> eh2 and eh3 -> sw1 -> sw5 -> sw7 -> sw2
-> eh4, and still return a success signal to the application. This
shows that the ALTO-PV service cannot encode the capacity region
for flows who use on-demand routing.

5. Overview of the Unified Representation Extension

Although different patches and extensions can be introduced to
address the aforementioned insufficiencies of the ALTO-PV service, it
is desirable to design a service that provides a generic solution
that can encode different types of resource sharing for a set of
flows. To this end, this document introduces the ALTO Unified
Representation (ALTO-UR) service.

5.1. Basic idea

The basic idea of the ALTO-UR service is to use mathematical
programming constraints to represent the capacity region for a set of
flows. Different from linear inequalities used in the ALTO-PV
service, mathematical programming constraints can represent a much
wider range of resource information. To illustrate the
expressiveness of mathematical programming constraints, we revisit
the examples in Figure 1.

Assume the route of eh1 -> eh2 is eh1 -> sw1 -> sw5 -> sw6 -> sw7
-> sw2 -> eh2, and the route of eh3 -> eh4 is eh3 -> sw1 -> sw5 -> sw7
-> sw2 -> eh4. Denote the SRLG of flow eh1 -> eh2 as f1:SRLG, and
that of flow eh3->eh4 as f2:SRLG. Then the SRLG of both flows can be
represented as

\[ f1:SRLG \cap f2:SRLG = \{2, 3, 4\} \]

Assume the route of eh1 -> eh2 is eh1 -> sw1 -> sw5 -> sw6 -> sw7
-> sw2 -> eh2, and the route of eh3 -> eh4 is a multi-path route, i.e.,
(eh3 -> sw1 -> sw5 -> sw7 -> sw2 -> eh4, eh3 -> sw1 -> sw5 -> sw6
-> sw7 -> sw2 -> eh4), where each path would forward 50 percent of the
traffic for eh3 -> eh4. Denote the available bandwidth of eh1 -> eh2

as f1-bw and those of eh3 -> eh4 along two paths as f2-bw-p1 and f2-bw-p2. The bandwidth sharing of two flows can be represented as:

\[
\begin{align*}
    f1:bw &\leq 100 \text{ Mbps, for } (sw1, sw5), (sw7, sw2) \\
    f2:bw:p1 + f2:bw:p2 &\leq 100 \text{ Mbps, for } (sw3, sw5), (sw7, sw4) \\
    f1:bw + f2:bw:p1 &\leq 100 \text{ Mbps, for } (sw5, sw6), (sw6, sw7) \\
    f2:bw:p2 &\leq 100 \text{ Mbps, for } (sw5, sw7) \\
    f2:bw:p1 &= f2:bw:p2
\end{align*}
\]

Assume the routes for both flows are computed on demand and each flow can only use a single path. For eh1 -> eh2, use f1-bw-p1 and f1-bw-p2 to represent the available bandwidth of routes eh1 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh2 and eh1 -> sw1 -> sw5 -> sw7 -> sw2 -> eh2, respectively. For eh3 -> eh4, use f2-bw-p1 and f2-bw-p2 to represent the available bandwidth of routes eh3 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh4 and eh3 -> sw1 -> sw5 -> sw7 -> sw2 -> eh4, respectively. The bandwidth capacity region of both flows can be represented as:

\[
\begin{align*}
    f1:bw:p1 + f1:bw:p2 &\leq 100 \text{ Mbps, for } (sw1, sw5), (sw7, sw2) \\
    f2:bw:p1 + f2:bw:p2 &\leq 100 \text{ Mbps, for } (sw3, sw5), (sw7, sw4) \\
    f1:bw:p1 + f2:bw:p1 &\leq 100 \text{ Mbps, for } (sw5, sw6), (sw6, sw7) \\
    f1:bw:p2 + f2:bw:p2 &\leq 100 \text{ Mbps, for } (sw5, sw7) \\
    f1:bw:p1 &= 0 \text{ or } f1:bw:p2 = 0 \\
    f2:bw:p1 &= 0 \text{ or } f2:bw:p2 = 0
\end{align*}
\]

The next few subsections present the approaches adopted by the ALTO unified representation extension.

5.2. New Cost Type to Encode Mathematical Programming Variables

This document introduces the unified representation cost type, with the following cost mode and cost metric.

5.2.1. Cost Mode: array

The cost mode of the notation cost type is "array", which is defined in [DRAFT-PV]. The values are arrays of JSONValue. The specific type of each element in the array depends on the cost metric.

5.2.2. Cost Metric: variable-list

This document specifies a new cost metric called "variable-list". This cost metric indicates that the cost value is a list of variables that will be used in mathematical programming constraints.
5.3. New Entity Domain to Provide Mathematical Programming Constraints

This document adopts the property map defined in [DRAFT-UP] to encode the properties of abstract network elements. A new domain "cstr" (short for constraint) is registered in the property map. Each entity in the "cstr" domain has an identifier of an CSTR. Each CSTR has one property, which represents the semantics of this constraint, e.g., a "bw-cstr" property indicates that this constraint represents the bandwidth sharing among flows. This property is provided in information resources called "Property Map Resource" and "Filtered Property Map Resource". The "Filtered Property Map" resource which supports the "cstr" domain is used to encode the properties of cstr entities, and it is called a cstr Property Map in this document.

5.4. Multipart Response to Provide the Unified Representation

To ensure the consistency between the unified representation cost map and the corresponding CSTR property map, this document adopts the design of [DRAFT-PV] to allow a response to contain both the unified representation in a filtered cost map and the associated CSTR property map.

5.5. Designing New Interfaces for More Flexible Queries

Current ALTO protocol and its extensions allow applications to query resource information by specifying IP addresses of endpoints and simple filters. However, with the emerging of new networking architecture (e.g., software defined networking and network function virtualization) and the fine-grained resource requirement of applications (e.g., link-disjoint paths and endpoint precedence), applications need a more flexible interface to specify queries of resource information.

[DRAFT-FCS] proposes a flexible flow query extension service to allow applications to specify query entities based on flexible matching conditions (e.g., TCP/IP 5-tuple) instead of IP addresses only. However, it still does not allow an application to specify more fine-grained resource requirements. For example, a multi-domain service function chaining application may want to get the endpoint cost information of a chain of endpoints in the order of firewall, load balancer and data analyzer. The endpoint cost information of a chain of endpoints not in this order is of no interest to the application and should not be returned to the application.

As such, this document makes an initial proposal to design new interfaces for application to express fine-grained requirements of resource in the query. In addition to allowing the ALTO client to specify endpoints using flexible matching conditions (e.g., TCP/IP 5
tuple), one key idea in this proposal is to use a resource filter design.

Specifically, two types of resource properties are defined. The first type is value property, which are typical quality of services metrics for a given flow. Examples of the value property include the bandwidth, delay, loss rate and so on. The second type is set property. Examples of such a property include the forwarding and processing devices used by a flow (denoted as nodes), the physical links used by a flow (denoted as links) and the shared risk link group (SRLG) of all the devices and links used by a flow.

With these two types of resource properties, the atomic resource requirement predicates in resource filters are the comparison expressions on value properties and set properties. Resource requirement predicates can be composited using conjunction, disjunction and negation. The ALTO client can send resource query with composed resource requirement predicates to ALTO server, which only returns the information of resources that satisfy such predicates to ALTO client.

The details of this new interface will be fully specified in the next version of this document.

6. Example

6.1. Protocol Extension

To allow the ALTO client to query and receive the mathematical programming constraints for a set of flows, the Filtered Cost Map and Endpoint Cost Service of the ALTO protocol need to be extended. The current design adopted in this document uses a similar approach as the ALTO-PV extension does in [DRAFT-PV]: (1) extending the FilteredCostMapCapabilities object with a new member "property-map" and (2) using a multipart service to send both the unified representation cost map and the CSTR property map together. This design is illustrated in the next few subsections.

However, for the ALTO-UR service, this is still an early stage design. As a major next step for ALTO-UR service, other design options are being investigated with the aim of enabling better modularity and extensibility, and the protocol extension will be updated in the next version accordingly.
6.2. Workflow

A typical workflow of an ALTO client using the unified representation extension is as follows:


2. Look for the resource of the Cost Map Service which contains the unified representation cost type and get the resource ID of the dependent cstr property map.

3. Check whether the capabilities of the property map includes the desired "prop-types".

4. Send a unified-representation request which accepts "multipart/related" media type following "application/alto-costmap+json" or "application/endpointcost+json"

6.3. Information Resource Directory Example

An example of an Information Resource Directory is as follows. In this example, filtered cost map "cost-map-ur" supports the unified-representation extension. The property map "propmap-cstr" support two properties, "bw-cstr" and "srlg-cstr", representing bandwidth constraint and SRLG constraint, respectively.
6.4. Cost Map Service Example

The following is an example of the cost map service in the ALTO-UR extension. In the returned cost map, flow 1 has two bandwidth variables, f1:bw:p1 and f2:bw:p2, and one SRLG variable, f1:srlg. And flow 2 has one bandwidth variable, f2:bw, and one SRLG variable, f2:srlg. Four mathematical programming constraints are returned in the property map. The first three are bandwidth sharing constraints, and the fourth is an SRLG constraint.

POST /costmap/pv HTTP/1.1
Host: alto.example.com
Accept: multipart/related, application/alto-costmap+json, application/alto-propmap+json, application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-costmapfilter+json

```json
{
    "cost-type": {
        "cost-mode": "array",
        "cost-metric": "variable-list"
    },
    "pids": {
        "srcs": [ "PID1" ],
        "dsts": [ "PID2", "PID3" ]
    }
}
```

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=42

--42
Content-Type: application/alto-costmap+json

```json
{
    "meta": {
        "dependent-vtags": [
            {
                "resource-id": "default-network-map",
                "tag": "75ed013b3cb58f896e839582504f622838ce670f"
            }
        ],
        "cost-type": {
            "cost-mode": "array",
            "cost-metric": "variable-list"
        },
        "cost-map": {
            "PID1": {
                "PID2": [ "f1:bw:p1", "bw:p2", "f1:srlg" ]
            },
            "PID3": [ "f2:bw:p1", "f2:srlg" ]
        }
    }
}
```

--42
Content-Type: application/alto-propmap+json

```json
{
    "Xiang, et al.          Expires September 12, 2019              
    [Page 11]

```
"property-map": {
  "cstr:001": { "bw-cstr": "[0][0] add [0][1] leq 100"},
  "cstr:002": { "bw-cstr": "[0][0] eq [0][1]"},
  "cstr:003": { "bw-cstr": "[1][0] add [0][0] leq 100"},
  "cstr:004": { "srlg-cstr": "[0][2] intersect [1][1] eq {2, 3, 4}"},
}
}

7. Security and Privacy Considerations

The unified representation extension may expose more private
information to applications than the ALTO base protocol does.
However, as shown in the motivating example of providing SRLG
information for a set of flows, this extension has the capability of
exposing less private information than the ALTO-PV extension does,
while having a better expressiveness on providing fine-grained
resource information to applications. A systematic study on the
security and privacy issues of the ALTO-UR extension is one of the
major next steps.

8. References

8.1. Normative References

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

8.2. Informative References

[DRAFT-FCS]
Zhang, J., Gao, K., Wang, J., Xiang, Q., and Y. Yang,
"ALTO Extension: Flow-based Cost Query", 2017,

[DRAFT-NFCHAIN]
Perez, D. and C. Rothenberg, "ALTO-based Broker-assisted
Multi-domain Orchestration", 2018,

[DRAFT-PV]
Bernstein, G., Lee, Y., Roome, W., Scharf, M., and Y.
Yang, "ALTO Extension: Abstract Path Vector as a Cost


Authors’ Addresses

Qiao Xiang
Yale University
51 Prospect Street
New Haven, CT
USA

Email: qiao.xiang@cs.yale.edu

Franck Le
IBM
Thomas J. Watson Research Center
Yorktown Heights, NY
USA

Email: fle@us.ibm.com
Many ALTO use cases involve multiple ALTO information resources like different network maps, cost maps and property maps to achieve their own specific goals. To make the ALTO client query them one by one is not only inefficient but also error-prone. The inconsistent responses can be performed because of the unstable communication environment, and finally conduct the unexpected traffic optimization. Furthermore, some ALTO information resources may have correlation, which means one’s input parameters may depend on another one’s response. To address those issues, some advanced query schema is required. This document proposes an ALTO extension to support the multiple ALTO resources query in the single request using the HTTP multipart message and the existing JSON query languages.
Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(https://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction ................................................. 3
2. Terminologies ............................................. 4
   2.1. Resource Query Entry ................................. 4
   2.2. Resource Response Entry .............................. 4
   2.3. Resource Response Entry Body ...................... 4
3. Use Cases ................................................ 4
   3.1. Simple Batch Query ................................. 4
   3.2. Properties Constrained Query ...................... 5
   3.3. Path Vector Query ................................. 5
4. Requirements .............................................. 5
5. Design Space of Multipart Resource in ALTO .................. 6
6. Multipart Query Resource .................................. 7
   6.1. Media Type ........................................ 7
   6.2. HTTP Method ....................................... 7
   6.3. Capabilities ....................................... 7
   6.4. Accept Input Parameters ......................... 7
   6.5. Uses .............................................. 8
   6.6. Response ......................................... 8
7. Protocol Errors ........................................... 8
   7.1. Partial Error ..................................... 8
   7.2. Entire Error ..................................... 9
8. Incremental Update Integration ............................. 9
9. Examples .................................................. 10
   9.1. IRD Example ..................................... 10
   9.2. Example 1: Simple Batch Query .................. 12
   9.3. Example 2: Properties Constrained Query ....... 13
   9.4. Example 3: Path Vector Query .................. 16
10. Compatibility ........................................... 17
   10.1. Compatibility with Legacy ALTO Clients/Servers ... 17
   10.2. Compatibility with Existing Protocol Extensions .. 18
11. Misc Considerations ...................................... 18
1. Introduction

Application-Layer Traffic Optimization (ALTO) protocol [RFC7285] and its extensions already define several types of information resources, like Network Map, Cost Map and Property Map, to expose useful network informations to applications. However, many applications do not only use a single information resource to perform their traffic optimization. Retrieving multiple ALTO information resources is very common in many ALTO use cases.

Using the current ALTO framework defined in [RFC7285], the ALTO client can only query multiple ALTO information resources one by one. It is not only inefficient but also error-prone. Because of the network delay between different requests and the frequent change of ALTO information resources, the responses received by the ALTO client may be inconsistent.

Further more, some ALTO information resources have known dependencies, which means the ALTO client may need one’s response to decide another one’s query input parameters.

To be summarized, we need the multipart query service for three reasons:

- Clients may want to query multiple ALTO information resources in a single request to reduce the time consumption.
- Clients may want to query multiple ALTO resources consistently, which means the server should guarantee the responses of all resources are generated at the same time.
- Some use cases need to query multiple ALTO resources with a joint relationship.
This document defines a new ALTO services for: (1) querying multiple ALTO resources in a single request/response, and (2) supporting general-purpose JSON query languages to resolve the relational query.

2. Terminologies

Besides the terms defined in [RFC2045], [RFC2046], [RFC2387], and [RFC7285], this document also uses the following additional terms:

2.1. Resource Query Entry

A Resource Query Entry indicates the ResourceQuery object (see Section 6.4) for an individual resource in the accept input parameters of the Multipart Query resource.

2.2. Resource Response Entry

A Resource Response Entry indicates the entry of an individual part of the multipart response message, including the MIME headers and the body content.

2.3. Resource Response Entry Body

A Resource Response Entry Body indicates the body content of a Resource Response Entry.

3. Use Cases

The following use cases can benefit from the multipart query service.

3.1. Simple Batch Query

The simplest use case is to query a batch of ALTO resources in a single request.

Although the ALTO client can perform ALTO requests for multiple times, it is not only inefficient but also inconsistent.

For example, the ALTO server provides a network map resource A and a dependent cost map resource B. Both resources may change frequently. Assume the ALTO client queries the network map first, and it gets the revision A1. When the client queries the cost map, the network map may be already changed from A1 to A2, and the client receives cost map B2 which depends on A2 not A1. So the responded cost map B2 is not consistent with the previous network map A1.

This case requires the ALTO server to provide a way for the ALTO client to query multiple ALTO resources in a single transaction.
3.2. Properties Constrained Query

Beyond the simple batch query, there are also some another use cases requiring a new service for relational query. For example, Some clients may need to query an endpoint property map first, and find endpoints with some properties fitting some conditions. And then they query the endpoint cost of these endpoints.

In this case, the endpoint cost query depends on the result of the property map query. Although the ALTO client can cache the whole property map in its local storage, it is still not efficient and may conduct the consistency issue if the property map changes frequently. So it requires a new service to provide multiple dependent resources efficiently and consistently.

A general multipart query service benefits the ALTO client in two aspects:

- It allows the ALTO client to specify the boolean test to reduce the transmission of the useless data from the ALTO server.
- It compounds multiple ALTO information resources in a single response to reduce the communication times. Thus, the transmission latency can be reduced.

3.3. Path Vector Query

Another use case requiring the multiple resource query is the relational query between the on-demand generated resources. A straightforward example is the path vector query demonstrated in [I-D.ietf-alto-path-vector].

[I-D.ietf-alto-path-vector] introduces an extension of ALTO to provide path vector information by cost map and unified property map [I-D.ietf-alto-unified-props-new]. The client using path vector extension will usually query cost map and a dynamically generated property map sequentially. It is even hard to cache the full data of resources, because both the cost map and the property map are on-demand generated by the query input here. Thus, the only way to reduce the time consumption is to compound the two resources.

4. Requirements

From the use cases described in Section 3, there are three additional requirements for ALTO protocol:
MPQ-Req1: The ALTO protocol SHOULD allow the client to query multiple ALTO resources in a single request, and return the result in a single response.

It is the basic requirement to provide the query for the compound resources. Even simple cases can benefit if this requirement is realized.

MPQ-Req2: The ALTO protocol SHOULD provide general filter schema for any ALTO resources.

Current filter schema in ALTO protocol only supports the simple boolean test of numerical comparison. And the boolean filtered query is only supported by the cost map and the endpoint cost resource. It is not enough for the general cases. Even simple property map may require more general filter schema.

MPQ-Req3: The ALTO protocol SHOULD support relational query for multiple joint resources.

Some ALTO resources are relational and cannot be used individually. The path vector query is such an example. In these use cases, the support of relational query for multiple joint resources is very helpful.

5. Design Space of Multipart Resource in ALTO

This document discusses the solution of how to apply "multipart/*" (see [RFC2045] and [RFC2046]) response to the ALTO protocol.

There are three cases applying Multipart response to ALTO:

- **Multipart Request and Multipart Response**: In this case, an ALTO client can start a single request using Multipart encoding to query a batch of resources.

- **Single Request and Fixed-Layout Multipart Response**: In this case, an ALTO server receives a non-Multipart request, e.g., the filtered costmap request or the endpoint cost request, but returns a Multipart response. The ALTO server MUST export the layout of the Multipart response in the IRD. A special example can be found in [I-D.ietf-alto-path-vector].

- **Single Request and Flexible-Layout Multipart Response**: This case extends the previous case to allows the Multipart response with flexible layout. The ALTO server receives the unified query request and generate the layout of the response based on the request. The ALTO client can even use general-purpose query.
language like XQuery [W3CXQUERY] and JSONiq [JSONIQ] for general query process and relational joint query.

The application about Multipart request to the single object response is out of the scope of this document.

6. Multipart Query Resource

6.1. Media Type

"multipart/related" [RFC2387].

6.2. HTTP Method

An ALTO Multipart Query resource is requested using the HTTP POST method.

6.3. Capabilities

The capabilities are defined by an object of type MultipartQueryCapabilities:

object {
    JSONString query-langs<0..*>;
} MultipartQueryCapabilities;

where "query-langs" is an array of JSONString to indicate which query languages are supported by this resource.

6.4. Accept Input Parameters

The input parameters for a Multipart Query request are supplied in the entity body of the POST request. This document specifies the input parameters with a data format indicated by the media type "application/alto-multipartquery+alto", which is a JSON object of type ReqMultipartQuery:

object {
    ResourceQuery resources<1..*>;
    [JsonString query-lang;]
} ReqMultipartQuery;

object {
    JsonString resource-id;
    [JsonValue input;]
} ResourceQuery;

with fields:
resources: List of ResourceQuery objects for which resources are to be queried and how to query them. Each ResourceQuery object MUST include the "resource-id" field to indicate which resource is to be queried. If the queried resource requires the POST method, the "input" field MUST be specified. The value of the "input" field MUST be either a JSONObject or a JSONArray. When its value is a JSONObject, its format MUST be as the accept input parameters of its resource. When its value is a JSONArray, it MUST be a program written in the query language specified by the "query-lang" field.

query-lang: Optional. The value of the "query-lang" field MUST be one of values in the "query-langs" capability. If this field is not specified in the request, the ALTO client SHOULD NOT use any query language in the "input" field.

6.5. Uses

An array with the resource ID(s) of resource(s) which this multipart query resource can compound. The used resource can be any available ALTO resources except for the multipart query resource. If the "uses" field is not specified, all the available ALTO resources can be queried except for the multipart query resource.

6.6. Response

The response of multipart query resource is a multipart message. Each part of this multipart message is the response of a queried resource in the request.

7. Protocol Errors

At the top level, the request of ALTO Multipart Query resource may conduct two types of errors: Partial Error and Entire Error.

7.1. Partial Error

The Partial Error only occurs when the value of the "resource-id" field or the "input" field is invalid.

When the Partial Error occurs, the ALTO server MUST still return the response in the media type "multipart/related". For the resource query entry with an error, the ALTO server MUST specify the "Content-Type" of its resource response entry as "application/alto-error+json", and include the ALTO error message in its resource response entry body. For the resource query entry without any error, the ALTO server MUST perform its query request normally.
The value of the "resource-id" field is invalid when this resource id is not defined by the Information Resource Directory. In this case, the ALTO server MUST return the E_INVALID_FIELD_VALUE error.

The validation of each "input" field of the multipart query input parameters depends on the queried resource:

- If the "input" field of the multipart query input parameters is neither a JSONObject nor a JSONString, the ALTO server SHOULD return the E_INVALID_FIELD_TYPE error, unless a future protocol extension supports the non-JSONObject input parameters.

- If the "input" field of the multipart query input parameters is a JSONObject, the ALTO server MUST validate the value using its queried resource and return the corresponding error if it has.

- If the "input" field of the multipart query input parameters is a JSONString:
  * If the "query-lang" is not specified, the ALTO server MUST return the E_INVALID_FIELD_TYPE error.
  * If the "query-lang" is specified, the ALTO server MUST execute this JSONString as a program written in the "query-lang". If the execution failed, the ALTO server MUST return the E_INVALID_FIELD_VALUE error. If the execution succeed but the result fails to pass the validation of the queried resource, the ALTO server MUST return the E_INVALID_FIELD_VALUE error and attach the error message returned by the queried resource into the "message" field of the ALTO error message.

The syntax error is an Entire Error.

7.2. Entire Error

Any other invalid request will conduct the Entire Error.

When the Entire Error occurs, the ALTO server MUST return the error response in the media type "application/alto-error+json" instead of "multipart/related". The process of the Entire Error is as defined in Section 8.5 of [RFC7285].

8. Incremental Update Integration

This document defines a compatible incremental update process for Multipart Query resource with [I-D.ietf-alto-incr-update-sse].
An ALTO server’s IRD can export an Update Stream service defined in [I-D.ietf-alto-incr-update-sse] including the Resource ID of a Multipart Query resource in the "uses" field. When an ALTO client subscribe the incremental update for this Multipart Query resource, the ALTO server sends the whole Multipart response message back at the first data update message. Then the ALTO server subscribe all nodes in this multipart resource tree automatically. Once data updated later, the ALTO server publishes the update for each node individually.

9. Examples

9.1. IRD Example

Assume the root IRD is like the following:

```
{
  "meta": {
    "path-vector": {
      "cost-mode": "array",
      "cost-metric": "ane-path"
    },
    "num-routingcost": {
      "cost-mode": "numerical",
      "cost-metric": "routingcost"
    },
    "num-hopcount": {
      "cost-mode": "numerical",
      "cost-metric": "hopcount"
    }
  },
  "resources": {
    "my-default-networkmap": {
      "uri": "http://alto.example.com/networkmap",
      "media-type": "application/alto-networkmap+json"
    },
    "my-default-costmap": {
      "uri": "http://alto.example.com/costmap",
      "media-type": "application/alto-costmap+json",
      "capabilities": {
        "cost-type-names": [ "num-routingcost" ]
      },
      "uses": [ "my-default-networkmap" ]
    },
    "my-filtered-costmap": {
      "uri": "http://alto.example.com/costmap/filtered",
      "media-type": "application/alto-costmap+json",
      "accepts": "application/alto-costmapfilter+json".
    }
  }
}
```
"capabilities": {
  "cost-type-names": [ "num-hopcount" ]
},
"uses": [ "my-default-networkmap" ]
},
"endpoint-path-vector": {
  "uri": "http://alto.example.com/endpointcost",
  "media-type": "application/alto-endpointcost+json",
  "accepts": "application/alto-endpointcostparams+json",
  "capabilities": {
    "cost-constraints": true,
    "cost-type-names": [ "path-vector" ]
  },
  "property-map": "propmap-availbw"
},
"propmap-availbw-delay": {
  "uri": "http://alto.example.com/propmap/availbw",
  "media-type": "application/alto-propmap+json",
  "accepts": "application/alto-propmapparams+json",
  "capabilities": {
    "domain-types": [ "ane" ],
    "prop-types": [ "availbw" ]
  }
},
"propmap-location": {
  "uri": "http://alto.example.com/propmap/location",
  "media-type": "application/alto-propmap+json",
  "accepts": "application/alto-propmapparams+json",
  "capabilities": {
    "domain-types": [ "pid" ],
    "prop-types": [ "country", "state" ]
  }
},
"multipart-query": {
  "uri": "http://alto.example.com/multipart",
  "media-type": "multipart/related",
  "accepts": "application/alto-multipartquery+json",
  "capabilities": {
    "query-langs": [ "xquery", "jsoniq" ]
  }
}
9.2. Example 1: Simple Batch Query

POST /multipart HTTP/1.1
Host: alto.example.com
Accept: multipart/related, application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-multipartquery+json

{
  "resources": [
  {
    "resource-id": "my-default-networkmap"
  },
  {
    "resource-id": "my-default-costmap"
  }
  ]
}

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=simple-batch-query

--simple-batch-query
Content-Type: application/alto-networkmap+json

{
  "meta": {
    "vtag": {
      "resource-id": "my-default-networkmap",
      "tag": "75ed013b3cb58f896e839582504f622838ce670f"
    }
  },
  "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"
      ]
    }
  }
}
--simple-batch-query
Content-Type: application/alto-costmap+json

{
  "meta": {
    "dependent-vtags": [
      {
        "resource-id": "my-default-networkmap",
        "tag": "75ed013b3cb58f896e839582504f622838ce670f"
      }
    ],
    "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 }
  }
}

9.3. Example 2: Properties Constrained Query

NOTE: In this example, we use the "\" block to express the raw string with unescaped characters like \n and \". It is not valid HTTP body, but only used to better present. When the request is sent to the ALTO server, the "\" block should be escaped.
POST /multipart HTTP/1.1
Host: alto.example.com
Accept: multipart/related, application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-multipartquery+json

{  
    "query-lang": "jsoniq",
    "resources": [
        {
            "resource-id": "propmap-location"
        },
        {
            "resource-id": "my-default-costmap",
            "input": `  
               let $propmap := collection("propmap-location")
                    .("property-map")
               return {
                   "cost-type": {
                       "cost-mode": "numerical",
                       "cost-metric": "hopcount"
                   },
                   "pids": {
                       "srcs": [  
                           for $pid in keys($propmap)  
                           where $propmap.$pid.country eq "US"
                           return substring-after($pid, "PID:")
                       ],
                       "dsts": [  
                           for $pid in keys($propmap)  
                           where $propmap.$pid.country eq "CA"
                           return substring-after($pid, "PID:")
                       ]
                   }  
               }`
        }
    ]
}

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=prop-const-query

--prop-const-query
Content-Type: application/alto-propmap+json

{  

Zhang                  Expires September 26, 2019              [Page 14]
"property-map": {  
  "pid:PID1": {  
    "country": "US",  
    "state": "CA"  
  },  
  "pid:PID2": {  
    "country": "US",  
    "state": "CT"  
  },  
  "pid:PID3": {  
    "country": "CA",  
    "state": "QC"  
  },  
  "pid:PID4": {  
    "country": "CA",  
    "state": "NT"  
  },  
  "pid:PID5": {  
    "country": "FR"  
  }  
}  

--prop-const-query  
Content-Type: application/alto-costmap+json  

{  
  "meta": {  
    "cost-type": {  
      "cost-mode": "numerical",  
      "cost-metric": "hopcount"  
    }  
  },  
  "cost-map": {  
    "PID1": {  
      "PID3": 5,  
      "PID4": 7  
    },  
    "PID2": {  
      "PID3": 8,  
      "PID4": 4  
    }  
  }  
}
9.4. Example 3: Path Vector Query

POST /multipart HTTP/1.1
Host: alto.example.com
Accept: multipart/related, application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-multipartquery+json

{
  "query-lang": "jsoniq",
  "resources": [
    {
      "resource-id": "endpoint-path-vector",
      "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" ]
        }
      }
    },
    {
      "resource-id": "propmap-availbw",
      "input": '
      let $propmap := collection("endpoint-path-vector")
                   .("endpoint-cost-map")
      return {
        "entities": [
          distinct-values(flatten(
            for $src in keys($propmap)
            let $dsts := $propmap.$src
            return flatten(
              for $dst in keys($dsts)
              return $dsts.$dst
            )))
        ],
        "properties": [ "availbw" ]
      }' 
    }
  ]
}
HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=path-vector-query

--path-vector-query
Content-Type: application/alto-endpointcost+json
{
    "meta": {
        "cost-type": {
            "cost-mode": "array",
            "cost-metric": "ane-path"
        }
    },
    "endpoint-cost-map": {
        "ipv4:192.0.2.2": {
            "ipv4:192.0.2.89": [ "ane:L001", "ane:L003", "ane:L004" ],
            "ipv4:203.0.113.45": [ "ane:L001", "ane:L004", "ane:L005" ],
            "ipv6:2001:db8::10": [ "ane:L001", "ane:L005", "ane:L007" ]
        }
    }
}

--path-vector-query
Content-Type: application/alto-propmap+json
{
    "property-map": {
        "ane:L001": { "availbw": 50 },
        "ane:L003": { "availbw": 48 },
        "ane:L004": { "availbw": 55 },
        "ane:L005": { "availbw": 60 },
        "ane:L007": { "availbw": 35 }
    }
}

10. Compatibility
10.1. Compatibility with Legacy ALTO Clients/Servers

The multipart query service is a new ALTO service using the new media type. So the legacy ALTO client cannot identify this service from the IRD of the ALTO server supporting it. And the legacy ALTO server also cannot interpret the request of a multipart query service sent by the ALTO client.
10.2. Compatibility with Existing Protocol Extensions

The multipart query service can use any ALTO resources exchanging JSON data in request/response mechanism. So all the known ALTO extensions like ALTO Calendar [I-D.ietf-alto-cost-calendar], Multi-Cost [RFC8189] and the Path Vector [I-D.ietf-alto-path-vector] extension, which does not change the request/response mechanism, are compatible with the multipart query service.

11. Misc Considerations

11.1. Support Incremental Update

Because the response body entry of the multipart query resource is not a single JSON object, it may not be compatible with the current incremental update representation used in [I-D.ietf-alto-incr-update-sse].

11.2. Anonymous Resources

Some use cases may need the server generates "anonymous" ALTO resources for the on-demand information. The "anonymous" ALTO resources usually cannot appear alone but need to bind with some "non-anonymous" ALTO resources.

12. Security Considerations

Allow the ALTO clients to upload the query language script may not be safe. The code injection and many potential attacks can be conducted. The security issue should be discussed and considered.

To avoid the attacks like the code injection, this document recommends the following approaches:

Database Isolation: Some clients may attempt to access the secure database inside the server. Isolate the data into the different databases can reduce the risk of the information leak.

Application Container Isolation: Attackers may inject harmful code into the input query programs to attempt to access the system control. To avoid this, each query process is recommended to be isolated using the application container.

Resource Limit: Even attackers cannot get the permission to crack the data or the system, they can still inject some heavy-load programs to consume the server resources. Thus, limiting the memory usage and execution time of each query process is highly recommended.
13. IANA Considerations

13.1. application/alto-* Media Types

This document registers an additional ALTO media type, listed in Table 1.

<table>
<thead>
<tr>
<th>Type</th>
<th>Subtype</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>alto-multipartquery+json</td>
<td>Section 6.4</td>
</tr>
</tbody>
</table>

Table 1: Additional ALTO Media Type.

Type name: application

Subtype name: This document registers multiple subtypes, as listed in Table 1.

Required parameters: n/a

Optional parameters: n/a

Encoding considerations: Encoding considerations are identical to those specified for the "application/json" media type. See [RFC8259].

Security considerations: Security considerations related to the generation and consumption of ALTO Protocol messages are discussed in Section 15 of [RFC7285].

Interoperability considerations: This document specifies formats of conforming messages and the interpretation thereof.

Published specification: This document is the specification for these media types; see Table 1 for the section documenting each media type.

Applications that use this media type: ALTO servers and ALTO clients either stand alone or are embedded within other applications.

Additional information:

Magic number(s): n/a

File extension(s): This document uses the mime type to refer to protocol messages and thus does not require a file extension.
14. Acknowledgements

15. References

15.1. Normative References


15.2. Informative References

[1-D.ietf-alto-cost-calendar]

[1-D.ietf-alto-incr-update-sse]

[1-D.ietf-alto-path-vector]

[1-D.ietf-alto-unified-props-new]

[JSONIQ]


[W3CXQUERY]

Appendix A. Figures

TODO: Put additional figures here if we have.
Author’s Address

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

Email: jingxuan.n.zhang@gmail.com