J. Zhang
Tongji University
K. Gao
Sichuan University
J. Wang

Q. Xiang
Y.R. Yang
Yale University
15 March 2020

ALTO Extension: Flow-based Cost Query
draft-gao-alto-fcs-07

Abstract

ALTO cost maps and endpoint cost services map a source-destination pair into a cost value. However, current filter specifications, which define the set of source-destination pairs in an ALTO query, have two limitations: 1) Only very limited address types are supported (IPv4 and IPv6), which is not sufficient to uniquely identify a flow in networks with fine-grained routing, such as the emerging Software Defined Networks; 2) The base ALTO protocol only defines filters enumerating all sources and all destinations, leading to redundant information in the response; 3) Cannot distinguish transmission types of flows in the query, which makes the server hard to respond the accurate resource consumption. To address these three issues, this document extends the base ALTO protocol with a more fine-grained filter type which allows ALTO clients to select only the concerned source-destination pairs and announce the flow-specific information like data transmission type, and a more expressive address space which allows ALTO clients to make queries beyond the limited IP addresses.

Status of This Memo

Table of Contents

1. Introduction

Application-Layer Traffic Optimization (ALTO) protocol [RFC7285] defines several cost query services, like Filtered Cost Map and Endpoint Cost Service, to allow applications to query path costs. Generally, an ALTO cost query service can be regarded as a function transforming a given subset of a specific query space into a network view abstract, where the subset is specified by a PID filter or an endpoint filter. However, the current specification has some limitations.

First, in the base ALTO protocol [RFC7285], the endpoint filter only contains the source and destination IP addresses. In practice, both Internet Service Providers (ISP) and local network administrators may conduct policy-based routing, e.g., P2P traffic may be constrained and has a smaller bandwidth than HTTP traffic. Also, web services with different QoS requirements may be hosted on the same machine with the same IP address but different paths with different QoS metrics.

Second, in the base ALTO protocol [RFC7285], the query space is defined by a source list and a destination list. For a query with N sources and M destinations, the response contains N*M entries. While such a query schema is well suited for peer-to-peer (P2P) applications where files of the same seed are stored on all hosts, it may lead to a lot of redundancy in use cases such as modern data analytics systems where replicas of the same dataset are stored on only a small subset of servers. Consider a system where the number of replicas is 3 (the default in HDFS), jointly scheduling N concurrent transfers only needs a maximum of 3N entries but the base ALTO protocol may return up to N^2 entries.

Third, in the base ALTO protocol [RFC7285], the query does not distinguish among the different transmission types like unicast and multicast. For some use cases like the multi-flow scheduling demonstrated by [I-D.ietf-alto-path-vector], the data transmission between endpoints could be beyond unicast. And in those cases, different transmission types may affect the network resource consumption. If applications can receive the path costs distinguishing the different transmission types, it can help applications perform their data transmission decision better.

Thus, we conclude that the following additional requirements (AR) MUST be satisfied to allow ALTO clients make more accurate and efficient cost queries.

AR-1: The ALTO server SHOULD allow the ALTO client to specify accurate query space in cost query services.

The base ALTO protocol only includes IPv4 and IPv6 addresses as endpoint address types, which may not be sufficient to accurately identify an endpoint with emerging flow-based routing mechanisms. ALTO clients MAY suffer from suboptimal decisions because of such inaccuracy. Thus, the ALTO protocol SHOULD be extended so that clients are able to specify accurate query space, i.e., using more fine-grained endpoint address types.

AR-2: The ALTO server SHOULD allow the ALTO client to specify only the essential query space.

Existing PIDFilter (see Sec 11.3.2.3 in [RFC7285]) and EndpointFilter (see Sec 11.5.1.3 in [RFC7285]) represent the cross-product of sources and destinations, and can introduce a lot of redundancy in certain use cases. This limitation greatly harms the scalability of the ALTO protocol. Thus, the ALTO protocol SHOULD be extended so that ALTO clients are able to specify only the essential cost query space, i.e., the concerned source-destination pairs.

AR-3: The ALTO server SHOULD allow the ALTO client to specify different data transmission types for transimissions in the query space.

The input parameters of existing ALTO cost query services only allow the ALTO client to specify the queried transmissions by sources and destinations. The transmission between each source and destination will always be considered as the unicast. This limitaiton may make the ALTO client lose the accurate available resources. Thus, the ALTO protocol SHOULD be extended so that ALTO clients are able to speicfy different transmission types.

In this document, we describe an ALTO extension specifying flow-based cost queries. The rest of this document is organized as follows. Section 3 introduces several new address types that extend the query space of ALTO cost services. Section 5 describes the extended schema on Filtered Cost Map (FCM) and Endpoint Cost Service (ECS) to support cost queries of arbitrary source-destination combinations with the optional flow-specific information. Section 6 and Section 7 discuss security and IANA considerations.

1.1. Requirement 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].

1.2. Terminology

This document uses the same terms as defined in [RFC7285] and [RFC8189] with the following additional term:

1.2.1. Flow

In this document, a flow refers to all communications between two endpoints. A flow is "valid" if and only if there CAN be valid communications between the two endpoints, which oftentimes requires that that two endpoint addresses have compatible address types.

1.2.2. Data Transmission Type

This document use the term "Data Transmission Type" or "Transmission Type" to indicate how an application sends the network flows. See Section 4.2.

2. Overview of Approaches

This section presents a non-normative overview of the extension to support flow-based cost query. It assumes the readers are familiar with Filtered Cost Map and Endpoint Cost Service defined in [RFC7285] and their extensions defined in [RFC8189].

2.1. Extended Endpoint Address

To allow ALTO clients specify accurate query space in cost query services (AR-1), this document defines several new endpoint address types. An endpoint address with a new type is referred to as an extended endpoint address.

Since the address types of both the source and the destination correspond to the same network flow, they MUST NOT conflict. This document defines an address type conflict table to indicate conflicts. If some source and destination address types in a query conflict with each others, an ALTO server MUST return the corresponding error.

2.2. Flow-based Filter

To allow ALTO clients specify only the essential query space in cost query services (AR-2), both PIDFilter and EndpointFilter in the base protocol MUST be extended. The extended filters are referred to as flow-based filters.

A straight-forward way of satisfying AR-2 is to have an ALTO client list all its concerned flows. Despite its simplicity, it MAY be too large in size, especially when many flows have common sources or common destinations in the query. Also from the implementation's perspective, it cannot reuse the functionality to parse a PIDFilter/EndpointFilter.

Thus, the flow-based filters defined in this document allow ALTO clients to include multiple PIDFilter/EndpointFilter objects in the same query. Apparently, if we replace each PIDFilter/EndpointFilter of N sources and M destinations with NM filters that have exactly one source and destination, the two representations refer to the same set of flows. As a result, one can aggregate flows with common sources or destinations in one PIDFilter/EndpointFilter object without introducing redundant flows.

From the implementation's perspective, one MAY reuse an ALTO library which parses PIDFilter/EndpointFilter and/or converts them into a set of source-destination pairs.

2.3. Flow-specific Announcement

Some properties are only related to the transmission and cannot be encoded into endpoints, e.g., the data transmission type of a flow. However, these properties may help the ALTO client get more accurate costs.

To allow the ALTO client to specify these flow-specific properties (AR-3), this document introduces an extensible field in the flow-based filter. The ALTO client can announce the flow-specific information in this field. An announcement, whose type is encoded as a FlowSpecAnnounceType (Section 4.1), can represent transmission type, equal cost multipath assumption and other kinds of flow-specific information.

This document adopts an extensible design for this announcement field. Although only the data transmission type is defined in this document, other announcement properties can be defined in future documents and are not in the scope of this document.

3. Extended Endpoint Address

This document registers new address types and defines the corresponding formats for endpoint addresses of each new address type.

3.1. Address Type

The new AddressType identifiers defined in this document are as follows:

eth: An endpoint address with type "eth" is the address of an Ethernet interface. It is used to uniquely identify an endpoint in the data link layer.

domain: An endpoint address with type "domain" is the domain name of a web service. It is used to uniquely identify a web service which MAY be translated to one or more IPv4 address(es).

domain6: An endpoint address with type "domain6" is the domain name of a web service. It is used to uniquely identify a web service which MAY be translated to one or more IPv6 address(es).

tcp: An endpoint address with type "tcp" is the address of a TCP socket. It is used to uniquely identify an IPv4 TCP socket in the transport layer.

tcp6: An endpoint address with type "tcp6" is the address of a TCP socket. It is used to uniquely identify an IPv6 TCP socket in the transport layer.

udp: An endpoint address with type "udp" is the address of a UDP socket. It is used to uniquely identify an IPv4 UDP socket in the transport layer.

udp6: An endpoint address with type "udp6" is the address of a UDP socket. It is used to uniquely identify an IPv6 UDP socket in the transport layer.

3.2. Endpoint Address

This document defines EndpointAddr when AddressType is in Section 7.1.

3.2.1. MAC Address

An Endpoint Address of type "eth" is encoded as a MAC address, whose format is encoded as specified by either format EUI-48 in [EUI48] or EUI-64 in [EUI64].

3.2.2. Internet Domain Name

An Endpoint Address of type "domain" or "domain6" is encoded as a domain name in the Internet, as specified in Section 11 of [RFC2181]. It MUST have at least one corresponding A ("domain") or AAAA ("domain6") record in the DNS.

3.2.3. IPv4 Socket Address

An Endpoint Address of type "tcp" or "udp" is encoded as an IPv4 socket address. It is encoded as a string of the format Host:Port with the ":" character as a separator. The Host component of an IPv4 socket address is encoded as specified by either an IPv4 address (see Section 10.4.3.1 of [RFC7285]) or an IPv4-compatible domain name (see Section 3.2.2). The Port component of an IPv4 socket address is encoded as an integer between 1 and 65535.

3.2.4. IPv6 Socket Address

An Endpoint Address of type "tcp6" or "udp6" is encoded as an IPv6 socket address. It is also encoded as a string of the format Host:Port with the ":" character as a separator. The Host component of an IPv6 socket address is encoded as specified by either an IPv6 address (see Section 10.4.3.2 of [RFC7285]) enclosed in the "[" and "]" characters as recommended in [RFC2732] or an IPv6-compatible domain name (see Section 3.2.2). The Port component of IPv6 socket address is encoded as an integer between 1 and 65535.

3.3. Address Type Compatibility

In practice, a flow with endpoint addresses with different types MAY NOT be valid. For example, a source endpoint with an IPv4 address CANNOT establish a network connection with a destination endpoint with an IPv6 address. Neither can a source with a TCP socket address and a destination with a UDP socket address.

Thus, to explicitly define the compatibility between AddressType identifiers, every ALTO AddressType identifier MUST provide a list of AddressType identifiers that are compatible with it in the "ALTO Address Type Compatibility Registry" Section 7.2. For all sources and destinations in a PIDFilter/EndpointFilter, if the AddressType identifiers of a given pair DO NOT appear in the ALTO Address Type Compatibility Registry, an ALTO server MUST return an ALTO error response with the error code "E_INVALID_FIELD_VALUE" with optional information to help diagnose the incompatibility.

3.4. Examples

Some valid endpoint addresses are demonstrated as follows:

```
"eth:98-e0-d9-9c-df-81"
"domain:www.example.com"
"tcp:198.51.100.34:5123"
"udp6:[2000::1:2345:6789:abcd]:8080"
```

4. Flow-specific Announcement

Each flow-specific announcement refers to a property of the traffic between a set of source and destination pairs. The interpretation of a property, however, depends on the meaning, i.e., the type, of the property. This document uses flow-specific announcement type as an indicator of the "type" information, and requires the flow-specific announcement types be registered in an IANA registry called "ALTO Flow-specific Announcement Registry".

4.1. Flow-specific Announcement Type

It is encoded as a JSONString and has the same format as a PID Name defined in Section 10.1 in [RFC7285]. The type FlowSpecAnnounceType is used in this document to indicate a string of that format.

4.2. Data Transmission Type Announcement

This document defines a flow-specific announcement called the data transmission type. The type of this announcement is indicated by the string "transmission-type", and the value of this announcement is a JSONString of either "unicast", "anycast", "broadcast" or "multicast".

5. Extended Cost Query Filters

This section describes extensions to [RFC7285] and [RFC8189] to support flow-based cost queries.

This document uses the notation rules specified in Section 8.2 of [RFC7285] and also the notation rules for optional fields in Section 4 of [RFC8189].

5.1. Filtered Cost Map Extension

This document extends the Filtered Cost Map as defined in Section 11.3.2 of [RFC7285] and Section 4.1 of [RFC8189], by adding a new capability and new input parameters.

The media type, HTTP method, and "uses" specifications (described in Sections 11.3.2.1, 11.3.2.2, and 11.3.2.5 of [RFC7285], respectively) are not changed.

The format of the response is the same as defined in Section 4.1.3 of [RFC8189]. But this document recommends how to generate the response based on the extended input parameters.

5.1.1. Capabilities

The Filtered Cost Map capabilities are extended with two additional members:

* flow-based-filter

* flow-spec-announce

The capability "flow-based-filter" indicates whether this resource supports flow-based cost queries, and the capability "flow-spec-announce" indicates which flow-specific announcements are supported. The FilteredCostMapCapabilities object in Section 4.1.1 of [RFC8189] is extended as follows:

```
object {
  JSONString cost-type-names<1..*>;
  [JSONBool cost-constraints;]
  [JSONNumber max-cost-types;]
  [JSONString testable-cost-type-names<1..*>;]
  [JSONBool flow-based-filter;]
  [FlowSpecAnnounceType flow-spec-announce<1..*>;]
} FilteredCostMapCapabilities;
```

cost-type-names and cost-constraints: As defined in Section 11.3.2.4 of [RFC7285].

max-cost-types and testable-cost-type-names: As defined in Section 4.1.1 of [RFC8189].

flow-based-filter: If true, an ALTO Server allows a field "pid-flows" to be included in the requests. If not present, this field MUST be interpreted as if it is false.

flow-spec-announce: It MUST NOT be present if "flow-based-filter" is not true. If present, the value is the an array of supported flow-specific announcement types.

5.1.2. Accept Input Parameters

The ReqFilteredCostMap object in Section 4.1.2 of [RFC8189] is extended as follows:

```
object {
  [CostType cost-type;]
  [CostType multi-cost-types<1..*>;]
  [CostType testable-cost-types<1..*>;]
  [JSONString constraints<0..*>;]
  [JSONString or-constraints<1..*><1..*>;]
  [PIDFilter     pids;]
  [ExtPIDFilter  pid-flows<1..*>;]
} ReqFilteredCostMap;

object {
  [FlowSpecAnnounceMap flow-spec-announce;]
} ExtPIDFilter : PIDFilter;

object-map {
  FlowSpecAnnounceType -> JSONValue;
} FlowSpecAnnounceMap;
```

cost-type, multi-cost-types, testable-cost-types, constraints, or-constraints: As defined in Section 4.1.2 of [RFC8189].

pids: As defined in Section 11.3.2.3 of [RFC7285].

pid-flows: Defined as a list of ExtPIDFilter objects. The ALTO server MUST interpret PID pairs appearing in multiple ExtPIDFilter objects as if they appeared only once. If the capability "flow-spec-announce" is present, the "flow-spec-announce" input parameter can be specified. The value of this field is of type FlowSpecAnnounceMap, which maps a FlowSpecAnnounceType to its corresponding value. The interpretation of the value MUST follow the definition of the flow-specific announcement in the ALTO Flow-specific Announcement Registry.

An ALTO client MUST include either "pids" or "pid-flows" in a query but MUST NOT include both at the same time.

5.2. Response

This document does not change the format of the response entity. But the ALTO server responds the request with "pid-flows" filter as follows:

The ALTO server MUST include the path costs of pairs in each ExtPIDFilter in the "pid-flows" filter. If the "flow-spec-announce" field is specified in some ExtPIDFilter, the path costs for flows in this ExtPIDFilter SHOULD respond the flow-specific information announced by this field.

5.3. Endpoint Cost Service Extension

This document extends the Endpoint Cost Service as defined in Section 11.5.1 of [RFC7285] and Section 4.2 of [RFC8189], by adding a new capability and input parameters.

The media type, HTTP method, and "uses" specifications (described in Sections 11.5.1.1, 11.5.1.2, and 11.5.1.5 of [RFC7285], respectively) are unchanged.

The format of the response is the same as defined in Section 4.2.3 of [RFC8189]. But this document recommends how to generate the response based on the extended input parameters.

5.3.1. Capabilities

The extension to EndpointCostCapabilities includes three additional members:

* flow-based-filter

* address-types

* flow-spec-announce

Only if the capability "flow-based-filter" is present and its value is "true", the ALTO server supports the flow-based extension for this endpoint cost service. The capability "address-types" indicates which endpoint address types are supported by this resource, it MUST NOT be specified if "flow-based-filter" is absent or the value is false. The capability "flow-spec-announce" indicates which flow-specific announcements are supported, just like it works in the Filtered Cost Map resource.

```
object {
  [JSONBool    flow-based-filter;]
  [JSONString  address-types<0..*>;]
  [FlowSpecAnnounceType flow-spec-announce<1..*>;]
} EndpointCostCapabilities : FilteredCostMapCapabilities;
```

flow-based-filter: If true, an ALTO Server MUST accept field "endpoint-flows" in the requests. If not present, this field MUST be interpreted as if it is specified false.

address-types: Defines a list of AddressType identifiers encoded as a JSONArray of JSONString. All AddressType identifiers MUST be registered in the "ALTO Address Type Registry" (see Section 14.4 of [RFC7285]). An ALTO server SHOULD NOT claim "ipv4" and "ipv6" in this field explicitly, because they are supported by default. If not present, this field MUST be interpreted as if it is an empty array, i.e., the ALTO server only supports the default "ipv4" and "ipv6" address types.

flow-spec-announce: It MUST NOT be present if "flow-based-filter" is not true. If present, the value is the an array of supported flow-specific announcement types.

5.3.2. Accept Input Parameters

The ReqEndpointCostMap object in Section 4.2.2 of [RFC8189] is extended as follows:

```
object {
  [CostType cost-type;]
  [CostType multi-cost-types<1..*>;]
  [CostType testable-cost-types<1..*>;]
  [JSONString constraints<0..*>;]
  [JSONString or-constraints<1..*><1..*>;]
  [EndpointFilter    endpoints;]
  [ExtEndpointFilter endpoint-flows<1..*>;]
} ReqEndpointCostMap;

object {
  [FlowSpecAnnounceMap flow-spec-announce;]
} ExtEndpointFilter : EndpiontFilter;
```

cost-type, multi-cost-types, testable-cost-types, constraints, or-constraints:
As defined in Section 4.1.2 of [RFC8189].

endpoints: As defined in Section 11.5.1.3 of [RFC7285].

endpoint-flows: Defined as a list of ExtEndpointFilter objects. The ALTO server MUST interpret endpoint pairs appearing in multiple ExtEndpointFilter objects as if they appeared only once. If the capability "flow-spec-announce" is present, the "flow-spec-announce" input parameter can be specified. The interpretation of this field is the same as in Section 5.1.2.

If the AddressType of the source and destination in the same EndpointFilter do not conform the compatibility rule defined in Table 1 of Section 7.1, an ALTO server MUST return an ALTO error response with the error code "E_INVALID_FIELD_VALUE".

An ALTO client MUST specify either "endpoints" or "endpoint-flows", but MUST NOT specify both in the same query.

5.4. Response

This document does not change the format of the response entity. But the ALTO server responds the request with "pid-flows" filter as follows:

The ALTO server MUST include the path costs of pairs in each ExtPIDFilter in the "pid-flows" filter. If the "flow-spec-announce" field is specified in some ExtPIDFilter, the path costs for flows in this ExtPIDFilter SHOULD respond the flow-specific information announced by this field. Especially, if "transmission-type" is specified as "multicast", the ALTO server SHOULD expose all the destination address as a multicast group address, and append the shared trees to the multicast destination addresses into the response if possible.

5.5. Examples

5.5.1. Information Resource Directory

The following is an example of IRD with relevant resources of the ALTO server. It provides a default network map, a property map of "ane" domain, a filtered cost map and two endpoint cost resources. All of three cost query resources (filtered cost map and endpoint cost resources) support "flow-based-filter". One endpoint cost resource support "flow-spec-announce" and the compound query extension defined in I-D.ietf-alto-path-vector.

Examples followed this section use the same IRD in this document.

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

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: application/alto-directory+json

{
  "meta" : {
    "default-alto-network-map" : "my-default-network-map",
    "cost-types" : {
      "num-hopcount" : {
        "cost-mode" : "numerial",
        "cost-metric" : "hopcount"},
      "num-routingcost" : {
        "cost-mode" : "numerial",
        "cost-metric" : "routingcost"},
      "ord-routingcost" : {
        "cost-mode" : "ordinal",
        "cost-metric" : "routingcost"},
      "path-vector" : {
        "cost-mode" : "array",
        "cost-metric" : "ane-path"}
    },
    .....
    Other ALTO cost types as described in RFC7285
    .....
  },
  "resources" : {
```

```
        "my-default-network-map" : {
          "uri" : "http://alto.example.com/networkmap",
          "media-type" : "application/alto-networkmap+json"
        },
        "flow-based-cost-map" : {
          "uri" : "http://alto.example.com/costmap/multi/filtered",
          "media-type" : "application/alto-costmap+json",
          "accepts" : "application/alto-costmapfilter+json",
          "uses" : [ "my-default-network-map" ],
          "capabilities" : {
            "max-cost-types" : 2,
            "flow-based-filter" : true,
            "cost-type-names" : [ "num-hopcount",
                                  "num-routingcost" ]
          }
        },
        "flow-based-endpoint-cost" : {
          "uri" : "http://alto.example.com/endpointcost/lookup",
          "media-type" : "application/alto-endpointcost+json",
          "accepts" : "application/alto-endpointcostparams+json",
          "capabilities" : {
            "address-types": ["tcp", "udp"],
            "flow-based-filter" : true,
            "cost-type-names" : [ "ord-routingcost",
                                  "num-routingcost" ]
          }
        },
        "path-vector-endpoint-cost" : {
          "uri" : "http://alto.example.com/pathvector/lookup",
          "media-type": "multipart/related;
                         type=application/alto-costmap+json",
          "accepts" : "application/alto-endpointcostparams+json",
          "capabilities" : {
            "address-types": ["tcp", "tcp6"],
            "flow-based-filter" : true,
            "flow-spec-announce" : [ "transmission-type" ]
            "cost-type-names" : [ "path-vector" ],
            "ane-property-names": [ "maxresbw" ],
          }
        }
      }
    }
```

5.5.2. Flow-based Filtered Cost Map Example

This example shows how an ALTO client requests a filtered cost map using the "pid-flows" filter. In this case, the ALTO client receives a sparse cost map, which cuts 50% useless cost values from the full mesh.

```
POST /costmap/multi/filtered HTTP/1.1
Host: alto.example.com
Accept: application/alto-costmap+json,
        application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-costmapfilter+json

{
  "cost-type": {
    "cost-mode": "numerical",
    "cost-metric": "routingcost"
  },
  "pid-flows": [
    { "srcs": ["PID1"], "dsts": ["PID2", "PID3"] },
    { "srcs": ["PID3"], "dsts": ["PID4"] }
  ]
}

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: application/alto-costmap+json

{
  "meta": {
    "dependent-vtags": [
      {
        "resource-id": "my-default-network-map",
        "tag": "75ed013b3cb58f896e839582504f622838ce670f"
      }
    ],
    "cost-type": {
      "cost-mode": "numerical",
      "cost-metric": "hopcount"
    }
  },
  "cost-map": {
    "PID1": { "PID2": 6, "PID3": 2 },
    "PID3": { "PID4": 1 }
  }
}
```

5.5.3. Flow-based Endpoint Cost Service Example #1

This example shows how the ALTO client requests endpoint cost using "flow-based-filter" and extended endpoint addresses. In this case, the ALTO client specifies tcp socket address to get more accurate path cost.

```
POST /endpointcost/lookup HTTP/1.1
Host: alto.example.com
Accept: application/alto-endpointcost+json,
        application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-endpointcostparams+json

{
  "cost-type": {
    "cost-mode": "numerical",
    "cost-metric": "hopcount"
  },
  "endpoint-flows": [
    { "srcs": ["ipv4:192.0.2.2"],
      "dsts": ["ipv4:192.0.2.89", "tcp:cdn1.example.com:21"] },
    { "srcs": ["tcp:203.0.113.45:54321"],
      "dsts": ["tcp:cdn1.example.com:21"] }
  ]
}

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: application/alto-endpointcost+json

{
  "meta": {
    "cost-type": {
      "cost-mode": "numerical",
      "cost-metric": "routingcost"
    }
  },
  "endpoint-cost-map": {
    "ipv4:192.0.2.2": {
      "ipv4:192.0.2.89": 100,
      "tcp:cdn1.example.com:21": 20
    },
    "tcp:203.0.113.45:54321": {
      "tcp:cdn1.example.com:21": 80
    }
  }
}
```

5.5.4. Flow-based Endpoint Cost Service Example #2

This example shows the integration of the path vector extension and the flow-based query. And in this example, the ALTO client specifies the flow from "tcp6:203.0.113.45:54321" to "tcp6:group1.example.com:21" is multicast. So the ALTO server will expose the destination IP as a multicast group IP, and find the multicast destinations "fe80::40e:9594:da3d:34b" and "fe80::826:daff:feb8:1bb". Then the ALTO server will append the cost for the shared tree into the "endpoint-cost-map".

```
POST /pathvector/lookup 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": "array",
    "cost-metric": "ane-path"
  },
  "endpoint-flows": [
    { "srcs": ["ipv4:192.0.2.2"],
      "dsts": ["tcp:192.0.2.89:21",
               "tcp:cdn1.example.com:21"] },
    { "srcs": ["tcp6:203.0.113.45:54321"],
      "dsts": ["tcp6:group1.example.com:21"],
      "flow-spec-announce": {
        "transmission-type": "multicast" } }
  ],
  "ane-property-names": ["maxresbw"]
}

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=example;
              type=application/alto-endpointcost+json

--example
Resource-Id: ecs
Content-Type: application/alto-endpointcost+json

{
  "meta": {
    "vtags": {
```

```
          "resource-id": "path-vector-endpoint-cost.ecs",
          "tag": "bb6bb72eafe8f9bdc4f335c7ed3b10822a391cef"
        },
        "cost-type": {
          "cost-mode": "array",
          "cost-metric": "ane-path"
        }
      },
      "endpoint-cost-map": {
        "ipv4:192.0.2.2": {
          "tcp:192.0.2.89:21": [ "ane:S1", "ane:D1" ],
          "tcp:cdn1.example.com:21": [ "ane:S1", "ane:D2", "ane:D3" ]
        },
        "tcp6:203.0.113.45:54321": {
          "tcp6:group1.example.com:21": [ "ane:S2", "ane:D3" ]
        },
        "tcp6:group1.example.com:21": {
          "tcp6:[fe80::40e:9594:da3d:34b]:21": [ "ane:G1" ],
          "tcp6:[fe80::826:daff:feb8:1bb]:21": [ "ane:G2" ],
        }
      }
    }

    --example
    Resource-Id: propmap
    Content-Type: application/alto-propmap+json

    {
      "meta": {
        "dependent-vtags": [
          {
            "resource-id": "path-vector-endpoint-cost.ecs",
            "tag": "bb6bb72eafe8f9bdc4f335c7ed3b10822a391cef"
          }
        ]
      },
      "property-map": {
        "ane:S1": { "maxresbw": 100 },
        "ane:S2": { "maxresbw": 100 },
        "ane:D1": { "maxresbw": 150 },
        "ane:D2": { "maxresbw": 80 },
        "ane:D3": { "maxresbw": 150 },
        "ane:G1": { "maxresbw": 100 },
        "ane:G2": { "maxresbw": 100 }
      }
    }
```

6. Security Considerations

As discussed in Section 15.4 of [RFC7285], an ALTO server or a third party who is able to intercept the flow-based cost query messages MAY store and process the obtained information in order to analyze user behaviors and communication patterns. Since flow-based cost queries MAY potentially provide more accurate information, an ALTO client should be cognizant about the trade-off between redundancy and privacy.

7. IANA Considerations

This document defines new address types to be registered to an existing ALTO registry, and a new registry for their compatible address types.

7.1. ALTO Address Type Registry

This document defines several new address types to be registered to "ALTO Address Type Registry", listed in Table 1.

Identifier	Address Encoding	Prefix Encoding	Mapping to/from IPv4/v6
eth	See Section 3.2.1	None	Mapping to/from IPv4 by [RFC0903] and [RFC0826]; Mapping to/from IPv6 by [RFC3122] and [RFC4861]
domain	See Section 3.2.2	None	Mapping to/from IPv4 by [RFC1034]
domain6	See Section 3.2.2	None	Mapping to/from IPv6 by [RFC3596]
tcp	See Section 3.2.3	None	No mapping
tcp6	See Section 3.2.4	None	No mapping
upd	See Section 3.2.3	None	No mapping
udp6	See Section 3.2.4	None	No mapping

Table 1: ALTO Address Type Registry

7.2. ALTO Address Type Compatibility Registry

This document proposes to create a new registry called "ALTO Address Type Compatibility Registry", whose purpose is stated in Section 3.3.

The compatible address type identifiers of the ones registered in the ALTO Address Type Registry are listed in Table 2.

Identifier	Compatible Identifiers
eth	ipv4, ipv6
domain	eth, ipv4
domain6	eth, ipv6
tcp	eth, ipv4, domain
tcp6	eth, ipv6, domain6
udp	eth, ipv4, domain
udp6	eth, ipv6, domain6

Table 2: ALTO Address Type Compatibility Registry

The entry of an address type identifier SHOULD only include the identifiers registered before it. The compatibility between address types are bidirectional. For example, although "eth" does not register "tcp" as its compatible identifier, an ALTO server MUST recognize them as compatible because "eth" is registered as a compatible identifier of "tcp".

Any new ALTO address type identifier registered after this document MUST register their compatible identifiers in this registry simultaneously.

7.3. ALTO Flow-specific Announcement Registry

This document proposes to create a new registry called "ALTO Flow-specific Announcement Registry", whose purpose is stated in Section 4. An initial registry entry is also documented as shown in Table 3.

Type	Interpretation
transmission-type	See Section 4.2

Table 3: ALTO Flow-specific Announcement Registry

8. References

8.1. Normative References

[EUI48] IEEE, ., "Guidelines for use of a 48-bit Extended Unique Identifier (EUI-48)", 2012.

[EUI64] IEEE, ., "Guidelines for use of a 64-bit Extended Unique Identifier (EUI-64)", 2012.

[I-D.ietf-alto-path-vector] Gao, K., Randriamasy, S., Yang, Y., and J. Zhang, "ALTO Extension: Path Vector", Work in Progress, Internet-Draft, draft-ietf-alto-path-vector-10, 9 March 2020, <http://www.ietf.org/internet-drafts/draft-ietf-alto-path-vector-10.txt>.

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

[RFC8189] Randriamasy, S., Roome, W., and N. Schwan, "Multi-Cost Application-Layer Traffic Optimization (ALTO)", RFC 8189, DOI 10.17487/RFC8189, October 2017, <https://www.rfc-editor.org/info/rfc8189>.

8.2. Informative References

[RFC0826] Plummer, D., "An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware", STD 37, RFC 826, DOI 10.17487/RFC0826, November 1982, <https://www.rfc-editor.org/info/rfc826>.

[RFC0903] Finlayson, R., Mann, T., Mogul, J.C., and M. Theimer, "A Reverse Address Resolution Protocol", STD 38, RFC 903, DOI 10.17487/RFC0903, June 1984, <https://www.rfc-editor.org/info/rfc903>.

[RFC1034] Mockapetris, P.V., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

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

DOI 10.17487/RFC2119, March 1997,
<https://www.rfc-editor.org/info/rfc2119>.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <https://www.rfc-editor.org/info/rfc2181>.

[RFC2732] Hinden, R., Carpenter, B., and L. Masinter, "Format for Literal IPv6 Addresses in URL's", RFC 2732, DOI 10.17487/RFC2732, December 1999, <https://www.rfc-editor.org/info/rfc2732>.

[RFC3122] Conta, A., "Extensions to IPv6 Neighbor Discovery for Inverse Discovery Specification", RFC 3122, DOI 10.17487/RFC3122, June 2001, <https://www.rfc-editor.org/info/rfc3122>.

[RFC3596] Thomson, S., Huitema, C., Ksinant, V., and M. Souissi, "DNS Extensions to Support IP Version 6", STD 88, RFC 3596, DOI 10.17487/RFC3596, October 2003, <https://www.rfc-editor.org/info/rfc3596>.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <https://www.rfc-editor.org/info/rfc4861>.

Appendix A. Acknowledgment

The authors would like to thank Dawn Chen, Haizhou Du, Sabine Randriamasy and Wendy Roome for their fruitful discussions and feedback on this document. Shawn Lin also gave substantial review feedback and suggestions on the protocol design.

Appendix B. Change Logs

Note to Editor: Please remove this section prior to publication.

This section records the change logs of the draft updates.

Changes since -06 versions:

* Clarify the type of "flow-spec-announcement" in both the request and the response

* Modify examples to be compatible with the latest path vector document

* Add a new registry for flow-specific announcements

* Fix some typos

Changes since -05 versions:

* Add flow-specific information announcement in the flow-based filter.

* Modify examples and add descriptions to Make them clear.

* Rename the address type "Domain Name" to "Internet Domain Name" to distinguish it with the "Domain Name" in the unified properties draft.

Changes since older versions:

Changes since -04 revision:

* Improve the clarity of the document by explicitly stating the problems.

* Keep only "flow" in the terminology section.

* Move section 6 "Advanced Flow-based Query" out of this document.

* Change "ALTO Address Type Conflicts Registry" to "ALTO Address Type Compatibility Registry".

Since -03 revision:

* Remove some irrelevant content from the draft.

* Improve the description of the new endpoint address type identifier registry. And add a new registry to declare the conflicting address type identifiers.

Since -02 revision:

* Change "EndpointURI" to "AddressType::EndpointAddr" for consistency.

* Replace "Cost Confidence" by "Cost Statistics" for compatibility.

Since -01 revision:

* Define the basic flow-based query extensions for Filtered Cost Map and Endpoint Cost service. The basic flow-based query is downward

compatible with the legacy ALTO service. It does not introduce any new media types.

* Move the service of media-type "application/alto-flowcost+json" to the advanced flow-based query extension. It will ask ALTO server to support the new media type.

Since -00 revision:

* Change the schema of "pid-flows" and "endpoint-flows" fields from pair list to pair mesh list.

Authors' Addresses

Jingxuan Jensen Zhang
China
201804
Shanghai
4800 Caoan Road
Tongji University

Email: jingxuan.n.zhang@gmail.com

Kai Gao
China
610000
Chengdu
No.24 South Section 1, Yihuan Road
Sichuan University

Email: kaigao@scu.edu.cn

Junzhuo Wang

Qiao Xiang
Yale University
51 Prospect Street
New Haven, CT
United States of America

Email: qiao.xiang@cs.yale.edu

Yang Richard Yang
Yale University

51 Prospect Street
New Haven, CT
United States of America

Email: yry@cs.yale.edu

ALTO WG K. Gao
Internet-Draft Tsinghua University
Intended status: Standards Track X. Wang
Expires: September 3, 2018 Tongji University
Q. Xiang
Tongji/Yale University
C. Gu
Tongji University
Y. Yang
Yale University
G. Chen
Huawei
March 2, 2018

Compressing ALTO Path Vectors
draft-gao-alto-routing-state-abstraction-08.txt

Abstract

The path vector extension [I-D.ietf-alto-path-vector] has extended the base ALTO protocol [RFC7285] with the ability to represent a more detailed view of the network which contains not only end-to-end costs but also information about shared bottlenecks.

However, the view computed by straw man algorithms can contain redundant information and result in unnecessary communication overhead. The situation gets even worse when certain ALTO extensions are enabled, for example, the incremental update extension [I-D.ietf-alto-incr-update-sse] which continuously pushes data changes to ALTO clients. Redundant information can trigger unnecessary updates.

In this document, several algorithms are described which can effectively reduce the redundancy in the network view while still providing the same information as in the original path vectors.

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 will expire on September 3, 2018.

Copyright Notice

Table of Contents

1. Introduction

The path vector extension [I-D.ietf-alto-path-vector] has extended the base ALTO protocol [RFC7285] with the ability to present more complex network views than the simple abstraction used by Cost Map or Endpoint Cost Service. ALTO clients can query more sophisticated information such as shared bottlenecks, and schedule their flows properly to avoid congestion and to better utilize network resources.

Meanwhile, the extension itself does not specify how an ALTO server should respond to a path-vector query. A straw man approach, as in the context of Software Defined Networking (SDN) where network providers have a global view, can compute the path vectors by retrieving the paths for all requested flows and returning the links on those paths as abstract network elements. However, this approach has several drawbacks:

- The resultant network view may lead to privacy leaks. Since the paths constitute a sub-graph of the global network topology, they may contain sensitive information without further processing.

- The resultant network view may contain redundant information. The path vector information is primarily used to avoid network bottlenecks. Thus, if a link cannot become the bottleneck, as demonstrated in Section 4, it is considered as redundant. Redundant links not only increase the communication overhead of the path vector extension, but also trigger false-positive data change events when the incremental update extension [I-D.ietf-alto-incr-update-sse] is activated.

To overcome these limitations, this document describes equivalent transformation algorithms that identify redundant abstract network elements and reduce them as much as possible. The algorithm can be integrated with any implementation of the path vector extension as a post-processing step. As the name suggests, this algorithm conducts equivalent transformations on the original path vectors, removes redundant information and obtains a more compact view.

This document is a supplement to the path vector extension and can be optionally turned on and off without affecting the correctness of responses. A crucial part of the equivalent transformation algorithm is how to find redundant abstract network elements. By tuning the redundancy check algorithm, one can make different trade-off decisions between efficiency and privacy. A reference implementation of redundancy check algorithm is also described in this document.

This document is organized as follows. Section 4 gives a concrete example to demonstrate the importance of compressing path vectors. The compression algorithms are specified in Section 5 and Section 6 discusses how one can use these algorithms on existing path vector responses. Finally, Section 7 and Section 8 discuss security and IANA considerations.

2. Changes Since Version -03, -04, -05, -06 and -07

In early versions of this draft, a lot of contents are shared with the path vector draft. From version -04, the authors have adjusted the structure and target this document as a supplement of the path vector extension with

o practical compression algorithms which can effectively reduce the leaked information and the communication overhead; and

o detailed instructions on how an original path vector response can be processed by these algorithms.

The -06 version fixed some minor issues in -04 and -05. The -07 version has focused on improving the clarity of the algorithms with more examples. The -08 version has improved the overall quality of the draft, especially the clarity of the algorithms using simpler symbols.

3. Terminology

This document uses the same terms as in [I-D.ietf-alto-path-vector].

4. Compressing Path Vectors

We use the example shown in Figure 1 to demonstrate the importance of compressing path vectors. The network has 6 switches (sw1 to sw6) forming a dumbbell topology where switches sw1/sw3 provide access on the left hand side, s2/s4 provide access on the right hand side, and sw5/sw6 form the backbone. End hosts eh1 to eh4 are connected to access switches sw1 to sw4 respectively. Assume that the bandwidth of each link is 100 Mbps, and that the network is abstracted with 4 PIDs each representing a host at one access switch.

```
PID1 +-----+                                           +-----+  PID2
eh1__|     |_                                      ____|     |__eh2
     | sw1 | \   +------+        +------+ /    | sw2 |
     +-----+  \  |      |        |      |/     +-----+
               \_| sw5  +---------+ sw6  |
PID3 +-----+   / |      |        |      |\     +-----+  PID4
eh3__|     |__/  +------+        +------+ \____|     |__eh4
     | sw3 |                                       | sw4 |
     +-----+                                       +-----+
```

Figure 1: Raw Network Topology

Link	Description
link1	sw1 <==> sw5
link2	sw2 <==> sw6
link3	sw3 <==> sw5
link4	sw4 <==> sw6
link5	sw5 <==> sw6

Table 1: Description of the Links

Three cases are identified when path vectors can be further compressed and an example is provided for each case.

4.1. Equivalent Aggregation

Consider an application which schedules the traffic consisting of two flows, eh1 -> eh2 and eh3 -> eh4. The application can query the path vectors and a straw man implementation will return all 5 links (abstract network elements) as shown in Figure 2.

```
path vectors:
    eh1: [ eh2: [ane:l1, ane:l5, ane:l2]]
    eh3: [ eh4: [ane:l3, ane:l5, ane:l4]]

abstract network element property map:
    ane:l1 : 100 Mbps
    ane:l2 : 100 Mbps
    ane:l3 : 100 Mbps
    ane:l4 : 100 Mbps
    ane:l5 : 100 Mbps
```

Figure 2: Path Vectors Returned by a Straw Man Implementation

The resultant path vectors represent the following linear constraints on the available bandwidth for the two flows:

```
bw(eh1->eh2)                <= 100 Mbps (ane:l1)
bw(eh1->eh2)                <= 100 Mbps (ane:l2)
               bw(eh3->eh4) <= 100 Mbps (ane:l3)
               bw(eh3->eh4) <= 100 Mbps (ane:l4)
bw(eh1->eh2) + bw(eh3->eh4) <= 100 Mbps (ane:l5)
```

Figure 3: Linear Constraints Represented by the Path Vectors

It can be seen that the constraints of ane:l1 and ane:l2 are exactly the same, and so are those of ane:l3 and ane:l4. Intuitively, we can replace ane:l1 and ane:l2 with a new abstract network element ane:1, and similarly replace ane:l3 and ane:l4 with ane:2. The new path vectors are shown in Figure 4.

```
path vectors:
    eh1: [ eh2: [ane:1, ane:l5]]
    eh3: [ eh4: [ane:2, ane:l5]]

abstract network element property map:
    ane:1  : 100 Mbps
    ane:2  : 100 Mbps
    ane:l5 : 100 Mbps
```

Figure 4: Path Vectors after Merging ane:l1/ane:l2 and ane:l3/ane:l4

4.2. Redundant Network Elements

Consider the same case as in Section 4.1. Taking a deeper look at Figure 3, one can conclude that constraints of ane:1 (ane:l1/ane:l2) and ane:2 (ane:l3/ane:l4) can be implicitly derived from that of ane:l5. Thus, these constraints are considered _redundant_ and the path vectors in Figure 4 can be further reduced. We replace ane:l5 with a new ane:3 and the new path vectors are shown in Figure 5.

```
path vectors:
    eh1: [ eh2: [ane:3]]
    eh3: [ eh4: [ane:3]]

abstract network element property map:
    ane:3 : 100 Mbps
```

Figure 5: Path Vectors after Removing Redundant Elements

It is clear that the new path vectors (Figure 5) are much more compact than the original path vectors (Figure 2) but they contain just as much information. Meanwhile, the application can hardly infer anything about the original topology with the compact path vectors.

4.3. Equivalent Decomposition

However, it is not always possible to directly remove all redundant network elements. For example, consider the case when both bandwidth and routingcost are requested, and the values are as shown in Figure 6. Note that we have changed the bandwidth for ane:l5 for demonstration purpose.

```
              path vectors:
                  eh1: [ eh2: [ane:l1, ane:l5, ane:l2]]
                  eh3: [ eh4: [ane:l3, ane:l5, ane:l4]]

              abstract network element property map:
                  ane:l1 : 100 Mbps, 1
                  ane:l2 : 100 Mbps, 2
                  ane:l3 : 100 Mbps, 1
                  ane:l4 : 100 Mbps, 1
                  ane:l5 : 200 Mbps, 1
```

Figure 6: Path Vectors Returned by a Straw Man Implementation

```
       bw(eh1->eh2)                <= 100 Mbps (ane:l1)
       bw(eh1->eh2)                <= 100 Mbps (ane:l2)
                      bw(eh3->eh4) <= 100 Mbps (ane:l3)
                      bw(eh3->eh4) <= 100 Mbps (ane:l4)
       bw(eh1->eh2) + bw(eh3->eh4) <= 200 Mbps (ane:l5)
```

Figure 7: Bandwidth Constraints in the Original Path Vectors

```
  rc(eh1->eh2) = rc(ane:l1) + rc(ane:l2) + rc(ane:l5) = 4
  rc(eh3->eh4) = rc(ane:l3) + rc(ane:l4) + rc(ane:l5) = 3
```

Figure 8: Routingcost Information in the Original Path Vectors

Figure 7 and Figure 8 demonstrate the bandwidth and routingcost information one can obtain from the original path vector. Again, ane:l1/ane:l2 and ane:l3/ane:l4 can still be aggregated in a similar way as in Figure 4 by setting the routingcost of ane:1 and ane:2 to 3 and 2 respectively. However, we cannot remove the redundant network element (ane:l5 in this case) directly because the resultant path vectors (Figure 9) would not provide the same routingcost information as in the original path vector.

```
path vectors:
    eh1: [ eh2: [ane:1]]
    eh3: [ eh4: [ane:2]]

abstract network element property map:
    ane:1  : 100 Mbps, 3
    ane:2  : 100 Mbps, 2
```

Figure 9: Path Vectors after Removing Redundant Network Element

A further observation is that since the bandwidth constraint of ane:l5 is redundant, it can be equally represented as two abstract network elements ane:a5 and ane:b5, as shown in Figure 10.

```
path vectors:
    eh1: [ eh2: [ane:1, ane:a5]]
    eh3: [ eh4: [ane:2, ane:b5]]

abstract network element property map:
    ane:1  : 100 Mbps, 3
    ane:2  : 100 Mbps, 2
    ane:a5 : 200 Mbps, 1
    ane:b5 : 200 Mbps, 1
```

Figure 10: Path Vectors after Decomposing ane:l5

Since ane:1/ane:a5 and ane:2/ane:b5 can be aggregated as ane:3 and ane:4 respectively, the final path vectors only contain two network elements, as shown in Figure 11.

```
path vectors:
    eh1: [ eh2: [ane:1]]
    eh3: [ eh4: [ane:2]]

abstract network element property map:
    ane:1  : 100 Mbps, 4
    ane:2  : 100 Mbps, 3
```

Figure 11: Path Vectors after Merging ane:1/ane:a5 and ane:2/ane:b5

One can verify that this path vector response has just the same information as in Figure 6 but contains much less contents.

5. Compression Algorithms

To provide a guideline on how path vectors MIGHT be compressed, this section describes the details of the algorithms for the three aforementioned cases:

1. Equivalent aggregation (EQUIV_AGGR), which compresses the original path vectors by aggregating the network elements with the same set of pairs as shown in Section 4.1;

2. Identification of redundant constraints (IS_REDUNDANT), which compresses the original path vectors by removing the network elements that provide only redundant information as shown in Section 4.2;

3. Equivalent decomposition (EQUIV_DECOMP), which compresses the original path vectors by decomposing redundant network elements to obtain the same end-to-end routing metrics as shown in Section 4.3.

5.1. Equivalent Aggregation

5.1.1. Parameters and Variables

The equivalent aggregation algorithm takes 3 parameters: the set of network elements "V", the set of relevant host pairs "P" and the set of metrics "M".

Set of network elements V: The set of network elements consists of all the network elements that exists in the original path vectors. The "i"-th network element in "V" is denoted as "v_i".

Set of relevant host pairs P: The "i"-th element in "P" is denoted as "p_i". It represents the set of (src, dst) pairs whose paths traverse "v_i" in the original path vectors.

Set of metrics M: The "i-th" element in "M" is denoted as "m_i". It represents the set of metrics associated with network element "v_i".

The output of the equivalent aggregation algorithm is a new set of network elements "V’", a new set of relevant host pairs "P’" and a new set of metrics "M’", i.e., "V’, P’, M’ = EQUIV_AGGR(V, P, M)".

5.1.2. Algorithm Description

1. Set "V’", "P’", "M’" to empty sets. Set "k" to 0. Go to step 2.

2. If "V" is empty, go to step 6. Otherwise, go to step 3.

3. Select an arbitrary element "v_i" from "V", remove "v_i" from "V" and go to step 4.

4. For any element "v_j" in "V", if "p_i = p_j", remove "v_j" from "V" and update "m_i" with "m_j", i.e., "m_i = UPDATE(m_i, m_j)" (which will be explained later). Go to step 5.

5. Increment "k" by 1, let "v'_k = v_i", "p'_k = p_i" and "m'_k = m_i". Go to step 2.

6. Return "V'", "P'", and "M'"

The process of update "m_i" with "m_j" depends on the metric types. For example, for routingcost and hopcount, the update is numerical addition, while for bandwidth, the update is calculating the minimum. The UPDATE function for some common metrics are listed in Table 2.

metric	UPDATE(x, y)	default
hopcount	x + y	0
routingcost	x + y	0
bandwidth	min(x, y)	+infinity
loss rate	1 - (1 - x) * (1 - y)	0

Table 2: UPDATE Function of Different Metrics

5.1.3. Example

Consider the path vectors in Figure 2 which can be represented as:

```
V     = { ane:l1, ane:l2, ane:l3, ane:l4, ane:l5 }

p_1   = { eh1->eh2 }
p_2   = { eh1->eh2 }
p_3   = { eh3->eh4 }
p_4   = { eh3->eh4 }
p_5   = { eh1->eh2, eh3->eh4 }

m_1   = 100 Mbps
m_2   = 100 Mbps
m_3   = 100 Mbps
m_4   = 100 Mbps
m_5   = 100 Mbps
```

As "p_1 = p_2" and "p_3 = p_4", the resultant attributes after the aggregation become:

```
V'    = { ane:1, ane:2, ane:l5 }

p'_1  = { eh1->eh2 } = p_1 = p_2
p'_2  = { eh3->eh4 } = p_3 = p_4
p'_3  = { eh1->eh2, eh3->eh4 } = p_5

m'_1  = 100 Mbps = UPDATE(m_1, m_2)
m'_2  = 100 Mbps = UPDATE(m_3, m_4)
m'_3  = 100 Mbps = m_5
```

5.2. Redundant Network Element Identification

5.2.1. Parameters and Variables

The redundant network element identification algorithm is based on the algorithm introduced by Telgen [TELGEN83]. It takes 3 parameters: the set of network elements "V", the set of relevant host pairs "P" and the set of available bandwidth values "B".

"V", "v_i", "P" and "p_i" are defined the same way as in Section 5.1.1.

Set of available bandwidth values B: The "i"-th element in "B" is denoted as "b_i". It represents the available bandwidth for network element "v_i".

The output of the IS_REDUNDANT function is a set of indices "R", which represents the indices of network elements whose bandwidth constraints are redundant, i.e., "R = IS_REDUNDANT(V, P, B)".

In addition to the parameters and output values, the algorithm also maintains the following variables:

Set of host pairs H: The "i"-th element of "H" is denoted as "h_i". It represents a (src, dst) pair ever appeared in the path vector query. "H" is the union of all "p_i" in "P".

Set of bandwidth constraints C: The "i"-th element of "C" is denoted as "c_i". It represents a linear bandwidth constraint on the flows between the end host pairs. The constraint "c_i" has the form of "a_i x <= b_i" where "a_i" is a row vector of 0-1 coefficients derived from "p_i", "x" is a column vector representing the bandwidth of all the host pairs, and "b_i" is the available bandwidth of "v_i".

5.2.2. Algorithm Description

1. The first step is to convert a network element to its bandwidth constraint "c_i". The bound "b_i" is directly obtained as the available bandwidth and the coefficients "a_i" are computed as:

```
        1  if h_j in p_i
a_ij =
        0  otherwise.
```

   Set "R" to an empty set. Go to step 2.

2. For each "i", solve the following linear programming problem:

```
            y_i = max a_i x
subject to:
    a_j x <= b_j, j = 1..|V|, i <> j
```

   Go to step 3.

3. For each "i", if "y_i <= b_i", "c_i" is redundant and we say "v_i" is redundant, "R = UNION(R, {i})". Go to step 4.

4. Return "R".

5.2.3. Example

Consider the path vectors in Figure 4 such that the input to the IS_REDUNDANT algorithm is as follows.

```
V     = { ane:1, ane:2, ane:l5 }

p_1   = { eh1->eh2 }
p_2   = { eh3->eh4 }
p_3   = { eh1->eh2, eh3->eh4 }

b_1   = 100 Mbps
b_2   = 100 Mbps
b_3   = 100 Mbps
```

With that information, one can follow the algorithm and get:

```
c_1:   x1       <= 100
c_2:        x2  <= 100
c_3:   x1 + x2  <= 100

y_1  = 100 Mbps <= b_1
y_2  = 100 Mbps <= b_2
y_3  = 200 Mbps >  b_3

R    = IS_REDUNDANT(V, P, B) = { 1, 2 }
```

5.3. Equivalent Decomposition

5.3.1. Parameters and Variables

The equivalent decomposition algorithm takes 4 parameters: the set of network elements "V", the set of relevant host pairs "P", the set of metrics "M" and the set of redundant network elements "R".

"V", "P" and "M" are as defined as in Section 5.1.1. If the "j"-th metric is bandwidth, we can construct the set of available bandwidth values "B" as "b_i = m_ij" and "R" is the output of the redundant network element identification procedure, i.e. "R = IS_REDUNDANT(V, P, B)". Otherwise, if bandwidth is not included in the metrics, "R" is {1, ..., |V|}.

The output of the function EQUIV_DECOMP is a new set of network elements "V’", a new set of relevant host pairs "P’", and a new set of metrics "M’", i.e., "V’, P’, M’ = EQUIV_DECOMP(V, P, M, R)".

5.3.2. Algorithm Description

1. Set "V’", "P’", "M’" to empty sets. Set "k" to 0. Go to step 2.

2. For each "i" such that "i" in "R", go to step 3. After processing each "i", go to step 7.

3. For each "j" such that "j <> i", go to step 4. After processing each "j", go to step 6.

4. If "p_j" is a subset of "p_i", go to step 5. Otherwise go to step 3.

5. Let "p_i = p_i \ p_j" and "m_j = UPDATE(m_j, m_i)". Go to step 3.

6. If "p_i" is not empty, increment "k" by 1 and let "v’_k = v_i", "p’_k = p_i" and "m’_k = m_i". Go to step 2.

7. For each "i" such that "i" is not in "R", go to step 8. After processing each "i", go to step 9.

8. Increment "k" by 1 and let "v'_k = v_i", "p'_k = p_i", "m'_k = m_i". Go to step 7.

9. Return "V'", "P'" and "M'".

5.3.3. Example

Consider the case in Section 4.3. Before the decomposition, the input to the algorithm is as follows:

```
V     = { ane:1, ane:2, ane:l5 }

p_1   = { eh1->eh2 }
p_2   = { eh3->eh4 }
p_3   = { eh1->eh2, eh3->eh4 }

m_1   = { bw: 100 Mbps, rc: 3 }
m_2   = { bw: 100 Mbps, rc: 2 }
m_3   = { bw: 200 Mbps, rc: 1 }

R     = { 3 }
```

Since there is only one element in "R", "v_i = ane:l5".

After the first iteration of steps 3-5 with "v_j = ane:1":

```
V     = { ane:1, ane:2, ane:l5 }

p_1   = { eh1->eh2 }
p_2   = { eh3->eh4 }
p_3   = { eh3->eh4 }

m_1   = { bw: 100 Mbps, rc: 4 }
m_2   = { bw: 100 Mbps, rc: 2 }
m_3   = { bw: 200 Mbps, rc: 1 }

V'    = { }
k     = 0
```

After the second iteration of steps 3-5 with "v_j = ane:2":

```
V     = { ane:1, ane:2, ane:l5 }

p_1   = { eh1->eh2 }
p_2   = { eh3->eh4 }
p_3   = { }

m_1   = { bw: 100 Mbps, rc: 4 }
m_2   = { bw: 100 Mbps, rc: 3 }
m_3   = { bw: 200 Mbps, rc: 1 }

V’    = { }
k     = 0
```

After step 6, since "p_3" is now empty, it just goes back to step 2. At step 2, since all indices in "R" has been processed, it goes to step 7.

After the first iteration of steps 7-8 with "i = 1":

```
V     = { ane:1, ane:2, ane:l5 }

p_1   = { eh1->eh2 }
p_2   = { eh3->eh4 }
p_3   = { }

m_1   = { bw: 100 Mbps, rc: 4 }
m_2   = { bw: 100 Mbps, rc: 3 }
m_3   = { bw: 200 Mbps, rc: 1 }

V’    = { ane:1 }
k     = 1

p’_1  = { eh1->eh2 } = p_1

m’_1  = { bw: 100 Mbps, rc: 4 } = m_1
```

After the second iteration of steps 7-8 with "i = 2":

```
V      = { ane:1, ane:2, ane:l5 }

p_1    = { eh1->eh2 }
p_2    = { eh3->eh4 }
p_3    = { }

m_1    = { bw: 100 Mbps, rc: 4 }
m_2    = { bw: 100 Mbps, rc: 3 }
m_3    = { bw: 200 Mbps, rc: 1 }

V'     = { ane:1, ane:2 }
k      = 2

p'_1   = { eh1->eh2 }
p'_2   = { eh3->eh4 } = p_2

m'_1   = { bw: 100 Mbps, rc: 4 }
m'_2   = { bw: 100 Mbps, rc: 3 } = m_2
```

So the final output of EQUIV_DECOMP is:

```
V'     = { ane:1, ane:2 }

p'_1   = { eh1->eh2 }
p'_2   = { eh3->eh4 }

m'_1   = { bw: 100 Mbps, rc: 4 }
m'_2   = { bw: 100 Mbps, rc: 3 }
```

5.4. Execution Order

As the examples demonstrate, the three algorithms MUST be executed in the same order as they are introduced, i.e., one MUST conduct "EQUIV_AGGR" before "IS_REDUNDANT" or "EQUIV_DECOMP", and conduct "IS_REDUNDANT" before "EQUIV_DECOMP". Otherwise, the results of the compressed path vectors MAY NOT be correct.

6. Encoding/Decoding Path Vectors

The three algorithms work mostly with network elements. Existing path vectors must be decoded before they can be passed on to the algorithms and the compressed results must be encoded as path vectors before they are sent to the clients. The decoding and encoding processes are specified as below.

6.1. Decoding Path Vectors

6.1.1. Parameters and Variables

The decoding algorithm DECODE takes a path vector response, which consists of the path vector part "PV" and the element property part "E".

Path vectors PV: The path vector part has a format of a CostMap (EndpointCostMap) where the cost value is a list of abstract network element names. We say a PID (endpoint address) "i" is IN "PV" if and only if there is an entry "i" in the cost-map (endpoint-cost-map), and denote the entry value as "PV[i]". Similarly, we say a PID (endpoint address) "j" is IN "PV[i]" if and only if there is an entry "j" in the DstCosts of "i", whose value is denoted as "PV[i][j]".

Element property map E: The element property map "E" maps an abstract network element name to its properties. We denote "E[n]" as the properties of element with name "n" and "E[n][pn]" as the value of property "pn".

The algorithm returns the set of elements "V", the set of relevant host pairs "P", the set of metrics "M" and the available bandwidth "B", as defined in Section 5.1.1 and Section 5.2.1. The algorithm uses a "SET" function which transforms a list into a set, and uses a "NAME" function which maps an integer in [1, K] to a unique property name where there are K properties in "E".

6.1.2. Algorithm Description

1. Set "V", "P", "M" and "B" to empty sets. Set "k" to 0. Go to step 2.

2. For each "i IN PV", go to step 3. After processing each "i", go to step 8.

3. For each "j IN PV[i]", go to step 4. After processing each "j", go to step 2.

4. For each "n" in "SET(PV[i][j])", go to step 5. After processing each "n", go to step 3.

5. If "n" is not in "V", go to step 6. Otherwise, go to step 7.

6. Increment "k" by 1 and let "v_k = n", "p_k = { i->j }". Go to step 4.

7. Find the index of "n" in "V" denoted as "a", let "p_a = UNION(p_a, {i->j})". Go to step 4.

8. For each "i" from 1 to |V|, go to step 9. After processing all "i", go to step 11.

9. For each "j" from 1 to K, go to step 10. After processing all "j", go back to step 8.

10. If "NAME(j) = 'availbw'", let "b_i = E[v_i][NAME(j)]". Let "m_ij = E[v_i][NAME(j)]".

11. Return "V", "P", "M" and "B".

6.1.3. Example

Consider the following example:

```
HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=example-2

--example-2
Content-Type: application/alto-endpointcost+json

{
  "meta": {
    "cost-types": [
      {"cost-mode": "array", "cost-metric": "ane-path"}
    ]
  }
  "endpoint-cost-map": {
    "ipv4:192.0.2.2": {
      "ipv4:192.0.2.89": [ "ane:L1", "ane:L3", "ane:L4" ],
      "ipv4:203.0.113.45": [ "ane:L1", "ane:L4", "ane:L5" ]
    }
  }
}
```

```
--example-2
Content-Type: application/alto-propmap+json

{
  "property-map": {
    "ane:L1": { "availbw": 50 },
    "ane:L3": { "availbw": 48 },
    "ane:L4": { "availbw": 55 },
    "ane:L5": { "availbw": 60 },
    "ane:L7": { "availbw": 35 }
  }
}

--example-2--
```

After the first iteration of Lines 2-5:

```
V       = { ane:L1, ane:L3, ane:L4 }

p_1     = { ipv4:192.0.2.2->ipv4:192.0.2.89 }
p_2     = { ipv4:192.0.2.2->ipv4:192.0.2.89 }
p_3     = { ipv4:192.0.2.2->ipv4:192.0.2.89 }.
```

After the second iteration of Lines 2-5:

```
V       = { ane:L1, ane:L3, ane:L4, ane:L5 }

p_1     = { ipv4:192.0.2.2->ipv4:192.0.2.89,
            ipv4:192.0.2.2->ipv4:203.0.113.45 }
p_2     = { ipv4:192.0.2.2->ipv4:192.0.2.89,
            ipv4:192.0.2.2->ipv4:203.0.113.45 }
p_3     = { ipv4:192.0.2.2->ipv4:192.0.2.89 }
p_4     = { ipv4:192.0.2.2->ipv4:203.0.113.45 }.
```

After the first iteration of Lines 6-9 with "i = 1":

```
m_1     = [50]

b_1     = 50
```

After all four iterations of Lines 6-9:

```
m_1    = [50]
m_2    = [48]
m_3    = [55]
m_4    = [60]

b_1    = 50
b_2    = 48
b_3    = 55
b_4    = 60
```

The decoded information can be passed on to "EQUIV_AGGR", "IS_REDUNDANT" and "EQUIV_DECOMP" for compression.

6.2. Encoding Path Vectors

6.2.1. Parameters and Variables

The algorithm ENCODE is the reverse process of DECODE. It takes the parameters "V", "P", "M" and constructs the path vector results.

The parameters are defined as in Section 5.1.1 and Section 5.2.1.

The algorithm also uses the NAME function in Section 6.1.1 which MUST return the same results in a paired ENCODE/DECODE process, and the "APPEND(L, e)" function which adds element "e" to list "L".

6.2.2. Algorithm Description

1. Set "PV={}", "E = {}". Go to step 2.

2. For each "v_i" in "V", go to step 3. If all "v_i" is processed, go to step XX.

3. For each "a->b" in "p_i", go to step 4. If all such "a->b" is processed, go to step 6.

4. If "a" is not in "PV", let "PV[a] = {}". Go to step 5.

5. If "b" is not in "PV[a]", let "PV[a][b] = [v_i]". Otherwise, let "PV[a][b] = APPEND(PV[a][b], v_i)". Go to step 2.

6. For each index "k" in [1, K], go to step 7. If all "k" is processed, go to step 1.

7. Set "E[v_i][NAME(k)] = m_ik". Go to step 6.

8. Return "PV" and "E".

6.2.3. Example

We consider the encoding of the decoded example in Section 6.1.3.

```
V      = { ane:L1, ane:L3, ane:L4, ane:L5 }

p_1    = { ipv4:192.0.2.2->ipv4:192.0.2.89,
             ipv4:192.0.2.2->ipv4:203.0.113.45 }
p_2    = { ipv4:192.0.2.2->ipv4:192.0.2.89,
             ipv4:192.0.2.2->ipv4:203.0.113.45 }
p_3    = { ipv4:192.0.2.2->ipv4:192.0.2.89 }
p_4    = { ipv4:192.0.2.2->ipv4:203.0.113.45 }

m_1    = [50]
m_2    = [48]
m_3    = [55]
m_4    = [60]
```

After the first iteration of steps 2-7:

```
PV[ipv4:192.0.2.2][ipv4:192.0.2.89  ] = [ane:L1]
PV[ipv4:192.0.2.2][ipv4:203.0.113.45] = [ane:L1]

E[ane:L1]["availbw"]                = 50
```

After the second iteration:

```
PV[ipv4:192.0.2.2][ipv4:192.0.2.89  ] = [ane:L1, ane:L3]
PV[ipv4:192.0.2.2][ipv4:203.0.113.45] = [ane:L1, ane:L3]

E[ane:L1]["availbw"]                = 50
E[ane:L3]["availbw"]                = 48
```

After the third iteration:

```
PV[ipv4:192.0.2.2][ipv4:192.0.2.89  ] = [ane:L1, ane:L3, ane:L4]
PV[ipv4:192.0.2.2][ipv4:203.0.113.45] = [ane:L1, ane:L3]

E[ane:L1]["availbw"]                = 50
E[ane:L3]["availbw"]                = 48
E[ane:L4]["availbw"]                = 55
```

After the fourth iteration:

```
PV[ipv4:192.0.2.2][ipv4:192.0.2.89  ] = [ane:L1, ane:L3, ane:L4]
PV[ipv4:192.0.2.2][ipv4:203.0.113.45] = [ane:L1, ane:L3, ane:L5]

E[ane:L1]["availbw"]                = 50
E[ane:L3]["availbw"]                = 48
E[ane:L4]["availbw"]                = 55
E[ane:L5]["availbw"]                = 60
```

Eventually, one can use the previous information to construct the endpoint cost service response.

```
HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=example-2

--example-2
Content-Type: application/alto-endpointcost+json

{
  "meta": {
    "cost-types": [
      {"cost-mode": "array", "cost-metric": "ane-path"}
    ]
  }
  "endpoint-cost-map": {
    "ipv4:192.0.2.2": {
      "ipv4:192.0.2.89": [ "ane:L1", "ane:L3", "ane:L4" ],
      "ipv4:203.0.113.45": [ "ane:L1", "ane:L4", "ane:L5" ]
    }
  }
}

--example-2
Content-Type: application/alto-propmap+json

{
  "property-map": {
    "ane:L1": { "availbw": 50 },
    "ane:L3": { "availbw": 48 },
    "ane:L4": { "availbw": 55 },
    "ane:L5": { "availbw": 60 },
  }
}

--example-2--
```

6.3. Compatibility

When the path vector extension is used with other extensions, such as [I-D.ietf-alto-cost-calendar] and [I-D.ietf-alto-multi-cost], the decoding and the encoding MUST only apply on the path vector part and leave the other attributes as they are.

Hence, this extension does not change the compatibility between the original path vector extension and other extensions.

7. Security Considerations

This document does not introduce any privacy or security issue on ALTO servers not already present in the base ALTO protocol or in the path vector extension.

The algorithms specified in this document can even help protect the privacy of network providers by conducting irreversible transformations on the original path vector.

8. IANA Considerations

This document does not define any new media type or introduce any new IANA consideration.

9. Acknowledgments

The authors would like to thank Dr. Qiao Xiang, Mr. Jingxuan Zhang (Tongji University), Prof. Jun Bi (Tsinghua University) and Dr. Andreas Voellmy (Yale University) for their early engagement and discussions.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

10.2. Informative References

[I-D.ietf-alto-cost-calendar]
Randriamasy, S., Yang, Y., Wu, Q., Lingli, D., and N. Schwan, "ALTO Cost Calendar", draft-ietf-alto-cost-calendar-01 (work in progress), February 2017.

[I-D.ietf-alto-incr-update-sse]
Roome, W. and Y. Yang, "ALTO Incremental Updates Using Server-Sent Events (SSE)", draft-ietf-alto-incr-update-sse-02 (work in progress), April 2016.

[I-D.ietf-alto-multi-cost]
Randriamasy, S., Roome, W., and N. Schwan, "Multi-Cost ALTO", draft-ietf-alto-multi-cost-10 (work in progress), April 2017.

[I-D.ietf-alto-path-vector]
Bernstein, G., Chen, S., Gao, K., Lee, Y., Roome, W., Scharf, M., Yang, Y., and J. Zhang, "ALTO Extension: Path Vector Cost Mode", draft-ietf-alto-path-vector-00 (work in progress), May 2017.

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

[TELGEN83]
Telgen, J., "Identifying Redundant Constraints and Implicit Equalities in Systems of Linear Constraints", Management Science , Volume 29, Issue 10, DOI 10.1287/mnsc.29.10.1209, 1983, <http://pubsonline.informs.org/doi/abs/10.1287/mnsc.29.10.1209>.

Authors' Addresses

Kai Gao
Tsinghua University
30 Shuangqinglu Street
Beijing 100084
China

Email: gaok12@mails.tsinghua.edu.cn

Xin (Tony) Wang
Tongji University
4800 CaoAn Road
Shanghai 210000
China

Email: xinwang2014@hotmail.com

Qiao Xiang
Tongji/Yale University
51 Prospect Street
New Haven, CT
USA

Email: qiao.xiang@cs.yale.edu

Chen Gu
Tongji University
4800 CaoAn Road
Shanghai 210000
China

Email: gc19931011jy@gmail.com

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

Email: yry@cs.yale.edu

G. Robert Chen
Huawei
Nanjing
China

Email: chenguohai@huawei.com

ALTO & CDNI WGs J. Seedorf
Internet-Draft HFT Stuttgart - Univ. of Applied Sciences
Intended status: Standards Track Y. Yang
Expires: 21 May 2021 Yale
K. Ma
Ericsson
J. Peterson
Neustar
J. Zhang
Tongji
17 November 2020

Content Delivery Network Interconnection (CDNI) Request Routing: CDNI Footprint and Capabilities Advertisement using ALTO
draft-ietf-alto-cdni-request-routing-alto-14

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. This document defines an FCI protocol using the Application-Layer Traffic Optimization (ALTO) protocol, following the guidelines defined in [RFC8008].

Requirements Language

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

Status of This Memo

This Internet-Draft will expire on 21 May 2021.

Copyright Notice

Table of Contents

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 the 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." At a high level, the scope of the CDNI Request Routing Interface, therefore, contains two main tasks: (1) determining if the dCDN (downstream CDN) is willing to accept a delegated content request, and (2) redirecting the content request coming from a uCDN (upstream CDN) to the proper entry point or entity in the dCDN.

Correspondingly, the request routing interface is broadly divided into two functionalities: (1) the CDNI Footprint & Capabilities Advertisement interface (FCI) defined in [RFC8008], and (2) the CDNI Request Routing Redirection interface (RI) defined in [RFC7975]. Since this document focuses on the first functionality (CDNI FCI), below is more details about it.

Specifically, CDNI FCI allows both an advertisement from a dCDN to a uCDN (push) and 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 defined 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 Advertisement Service".

There are multiple benefits in using ALTO as a transport protocol, as discussed 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 Advertisement Service", 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 new granularity to describe footprints. Section 5 introduces "Filtered CDNI Advertisement Service" to allow a uCDN to get footprints with given capabilities instead of getting the full resource, which can be large. Section 6 further shows another benefit of using ALTO: the ability to query footprint properties using ALTO unified properties. In this way, a uCDN can effectively fetch capabilities of footprints in which it is interested. IANA and security considerations are discussed in Section 7 and Section 8 respectively.

2. Background

The design of CDNI FCI transport using ALTO depends on the understanding of both FCI semantics and ALTO. Hence, this document starts with a non-normative review for both. The review uses the terminologies for CDNI as defined in [RFC6707], [RFC8006] and [RFC8008]; those for ALTO as defined in [RFC7285] and [I-D.ietf-alto-unified-props-new].

2.1. Semantics of FCI Advertisement

[RFC8008] (CDNI "Footprint and Capabilities Semantics") defines the semantics of CDNI FCI, provides guidance on what Footprint and Capabilities mean in a CDNI context, and specifies the requirements on the CDNI FCI transport protocol. The definitions in [RFC8008] depend on [RFC8006]. Below is a non-normative review of key related points of [RFC8008] and [RFC8006]. For detailed information and normative specification, the reader is referred to these two RFCs.

* Multiple types of mandatory-to-implement 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 or 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.

* Multiple initial types of capabilities are defined in [RFC8008] including (1) Delivery Protocol, (2) Acquisition Protocol, (3) Redirection Mode, (4) Capabilities related to CDNI Logging, and (5) Capabilities related to CDNI Metadata. They are required in all cases and therefore considered as mandatory-to-implement capabilities for all CDNI FCI implementations.

* Footprint and capabilities are defined together and cannot be interpreted independently from each other. Specifically, [RFC8008] integrates footprint and capabilities with an approach of "capabilities with footprint restrictions", by expressing capabilities on a per footprint basis.

* Specifically, for all 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.

* Given that a large part of Footprint and Capabilities Advertisement may 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 FCI. 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 (downstream CDN) to provide changes/updates regarding a footprint and/or capabilities that it has prior agreed to serve in a contract with a uCDN (upstream CDN). Hence, server push and incremental encoding will be necessary techniques.

2.2. ALTO Background and Benefits

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

Originally, ALTO was motivated by optimizing cross-ISP traffic generated by P2P applications [RFC5693]. However, ALTO can also be used for improving the request routing in CDNs. In particular, the CDNI problem statement [RFC6707] explicitly mentions ALTO as a candidate protocol for "actual algorithms for selection of CDN or Surrogate by Request-Routing systems".

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

* Application Layer-oriented: 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. This matches the need of CDNI: a uCDN wants to improve application layer CDN request routing by using information (provided by a dCDN) that the uCDN could not easily obtain otherwise. Hence, 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.

* Security: The identification between uCDNs and dCDNs is an important requirement. ALTO maps can be signed and hence provide inherent integrity protection. Please see Section 8.

* 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 provides extensive error-handling in the whole request and response process (see Section 8.5 of [RFC7285]). A CDNI FCI interface based on ALTO would inherit this this extensive error-handling framework. Please see Section 5.

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

* Filtered Map Service: The ALTO map filtering service would allow a uCDN to query only for parts of an ALTO map. For example, the ALTO filtered property map service can enable a uCDN to query properties of a part of footprints efficiently (see Section 6).

* Server-initiated Notifications and Incremental Updates: When the footprint or the capabilities of a dCDN change (i.e., unexpectedly from the perspective of a uCDN), server-initiated notifications would enable a dCDN to inform a uCDN about such changes directly. 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 recent network and cost map from the dCDN, and therefore redirect requests to the 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 [RFC8895].

* 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 a uCDN. This would enable a uCDN to determine whether a dCDN actually has the capabilities for a given 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 a uCDN for optimized dCDN 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 dCDN 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 Advertisement Service

The ALTO protocol is based on the ALTO Information Service Framework which consists of multiple 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 multiple initial services, e.g., the ALTO network map service and cost map service.

This document defines a new ALTO service called "CDNI Advertisement Service" which conveys JSON objects of media type "application/alto-cdni+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 Advertisement Service". Similar to other ALTO services, this document defines the ALTO information resource for the "CDNI Advertisement Service" as follows.

3.1. Media Type

The media type of the CDNI Advertisement resource is "application/alto-cdni+json".

3.2. HTTP Method

A CDNI Advertisement resource is requested using the HTTP GET method.

3.3. Accept Input Parameters

None.

3.4. Capabilities

None.

3.5. Uses

The "uses" field SHOULD NOT appear unless the CDNI Advertisement resource depends on other ALTO information resources. If the CDNI Advertisement resource has dependent resources, the resource IDs of its dependent resources MUST be included into the "uses" field. This document only defines one potential dependent resource for the CDNI Advertisement resource. See Section 4 for details of when and how to use it. Future documents may extend the CDNI Advertisement resource and allow other dependent resources.

3.6. Response

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

If a CDNI Advertisement response depends on other ALTO information resources, 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 its IRD entry.

The data component of an ALTO CDNI Advertisement response is named "cdni-advertisement", which is a JSON object of type CDNIAdvertisementData:

```
object {
  CDNIAdvertisementData cdni-advertisement;
} InfoResourceCDNIAdvertisement : ResponseEntityBase;

object {
  BaseAdvertisementObject capabilities-with-footprints<0..*>;
} CDNIAdvertisementData;
```

Specifically, a CDNIAdvertisementData object is a JSON object that includes only one property named "capabilities-with-footprints", 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, below is a non-normative specification of BaseAdvertisementObject. As mentioned above, the normative specification of BaseAdvertisementObject is in [RFC8008].

```
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

```
{
  "meta" : {...},
  "cdni-advertisement": {
    "capabilities-with-footprints": [
      {
        "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"
          ]
        },
        "footprints": [
          <Footprint objects>
        ]
      }
    ]
  }
}
```

EXAMPLE 2

```
{
  "meta" : {...},
  "cdni-advertisement": {
    "capabilities-with-footprints": [
      {
        "capability-type": "FCI.DeliveryProtocol",
        "capability-value": {
```

```
                "delivery-protocols": [
                  "https/1.1",
                  "http/1.1"
                ]
              },
              "footprints": [
                <Footprint objects>
              ]
            }
          ]
        }
      }
```

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

This document only requires the ALTO server to provide the initial FCI-specific CDNI Payload Types defined in [RFC8008] as the mandatory-to-implement CDNI capabilities. There may be other documents extending BaseAdvertisementObject and additional CDNI capabilities. They are outside the scope of this document. To support them, future documents can extend the specification defined in 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 related information resources (e.g., CDNI Advertisement resources), demonstrating a single, integrated environment.

Specifically, the IRD announces two network maps, one CDNI Advertisement resource without dependency, one CDNI Advertisement resource depending on a network map, one filtered CDNI Advertisement resource to be defined in Section 5, one property map including "cdni-capabilities" as its entity property, one filtered property map including "cdni-capabilities" and "pid" as its entity properties, and two update stream services (one for updating CDNI Advertisement resources, 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

HTTP/1.1 200 OK
Content-Length: 3571
Content-Type: application/alto-directory+json
```


```
{
  "meta" : {
    "default-alto-network-map": "my-default-network-map"
  },
  "resources": {
    "my-default-network-map": {
      "uri" : "https://alto.example.com/networkmap",
      "media-type" : "application/alto-networkmap+json"
    },
    "my-eu-netmap" : {
      "uri" : "https://alto.example.com/myeunetmap",
      "media-type" : "application/alto-networkmap+json"
    },
    "my-default-cdnifci": {
      "uri" : "https://alto.example.com/cdnifci",
      "media-type": "application/alto-cdni+json"
    },
    "my-cdnifci-with-pid-footprints": {
      "uri" : "https://alto.example.com/networkcdnifci",
      "media-type" : "application/alto-cdni+json",
      "uses" : [ "my-eu-netmap" ]
    },
    "my-filtered-cdnifci" : {
      "uri" : "https://alto.example.com/cdnifci/filtered",
      "media-type" : "application/alto-cdni+json",
      "accepts" : "application/alto-cdnifilter+json"
    },
    "cdnifci-property-map" : {
      "uri" : "https://alto.example.com/propmap/full/cdnifci",
      "media-type" : "application/alto-propmap+json",
      "uses": [ "my-default-cdni" ],
      "capabilities" : {
        "mappings": {
          "ipv4": [ "my-default-cdni.cdni-capabilities" ],
          "ipv6": [ "my-default-cdni.cdni-capabilities" ],
          "countrycode": [
            "my-default-cdni.cdni-capabilities" ],
          "asn": [ "my-default-cdni.cdni-capabilities" ]
        }
      }
```

```
        },
        "filtered-cdnifci-property-map" : {
          "uri" : "https://alto.example.com/propmap/lookup/cdnifci-pid",
          "media-type" : "application/alto-propmap+json",
          "accepts" : "application/alto-propmapparams+json",
          "uses": [ "my-default-cdni", "my-default-network-map" ],
          "capabilities" : {
            "mappings": {
              "ipv4": [ "my-default-cdni.cdni-capabilities",
                        "my-default-network-map.pid" ],
              "ipv6": [ "my-default-cdni.cdni-capabilities",
                        "my-default-network-map.pid" ],
              "countrycode": [
                "my-default-cdni.cdni-capabilities" ],
              "asn": [ "my-default-cdni.cdni-capabilities" ]
            }
          }
        },
        "update-my-cdni-fci" : {
          "uri": "https:///alto.example.com/updates/cdnifci",
          "media-type" : "text/event-stream",
          "accepts" : "application/alto-updatestreamparams+json",
          "uses" : [
            "my-default-network-map",
            "my-eu-netmap",
            "my-default-cdnifci",
            "my-filtered-cdnifci",
            "my-cdnifci-with-pid-footprints"
          ],
          "capabilities" : {
            "incremental-change-media-types" : {
             "my-default-network-map" : "application/json-patch+json",
             "my-eu-netmap" : "application/json-patch+json",
             "my-default-cdnifci" :
             "application/merge-patch+json,application/json-patch+json",
             "my-filtered-cdnifci" :
             "application/merge-patch+json,application/json-patch+json",
             "my-cdnifci-with-pid-footprints" :
             "application/merge-patch+json,application/json-patch+json"
            }
          }
        },
        "update-my-props": {
          "uri" : "https://alto.example.com/updates/properties",
          "media-type" : "text/event-stream",
          "uses" : [
            "cdnifci-property-map",
            "filtered-cdnifci-property-map"
```

```
          ],
          "capabilities" : {
            "incremental-change-media-types": {
             "cdnifci-property-map" :
             "application/merge-patch+json,application/json-patch+json",
             "filtered-cdnifci-property-map":
             "application/merge-patch+json,application/json-patch+json"
            }
          }
        }
      }
    }
```


3.7.2. Basic Example

This basic example demonstrates a simple CDNI Advertisement resource, which does not depend on other resources. There are three BaseAdvertisementObjects in this resource 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 /cdnifci HTTP/1.1
Host: alto.example.com
Accept: application/alto-cdni+json,
        application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 1235
Content-Type: application/alto-cdni+json
```


```
{
  "meta" : {
    "vtag": {
      "resource-id": "my-default-cdnifci",
      "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
    }
  },
  "cdni-advertisement": {
    "capabilities-with-footprints": [
      {
        "capability-type": "FCI.DeliveryProtocol",
        "capability-value": {
          "delivery-protocols": [
            "http/1.1"
          ]
        },
```

```
            "footprints": [
              {
                "footprint-type": "ipv4cidr",
                "footprint-value": [ "192.0.2.0/24" ]
              }
            ]
          },
          {
            "capability-type": "FCI.DeliveryProtocol",
            "capability-value": {
              "delivery-protocols": [
                "https/1.1",
                "http/1.1"
              ]
            },
            "footprints": [
              {
                "footprint-type": "ipv4cidr",
                "footprint-value": [ "198.51.100.0/24" ]
              }
            ]
          },
          {
            "capability-type": "FCI.AcquisitionProtocol",
            "capability-value": {
              "acquisition-protocols": [
                "https/1.1"
              ]
            },
            "footprints": [
              {
                "footprint-type": "ipv4cidr",
                "footprint-value": [ "203.0.113.0/24" ]
              }
            ]
          }
        ]
      }
    }
```


3.7.3. Incremental Updates Example

A benefit of using ALTO to provide CDNI Advertisement resources is that such resources 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 updates for "my-default-cdnifci", and the ALTO server returns the "control-uri" followed by the full CDNI Advertisement response. Then when there is a change in the delivery-protocols in that http/1.1 is removed (from [http/1.1, https/1.1] to only https/1.1) due to maintenance of the https/1.1 clusters, the ALTO server regenerates the new CDNI Advertisement resource and pushes the full replacement to the ALTO client. Later on, the ALTO server notifies the ALTO client that "192.0.2.0/24" is added into the "ipv4" footprint object for delivery-protocol https/1.1 by sending the change encoded by JSON patch to the ALTO client.

```
POST /updates/cdnifci HTTP/1.1
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: 92

{ "add": {
    "my-cdnifci-stream": {
      "resource-id": "my-default-cdnifci"
    }
  }
}

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

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

event: application/alto-cdni+json,my-cdnifci-stream
data: { ... full CDNI Advertisement resource ... }

event: application/alto-cdni+json,my-cdnifci-stream
data: {
data:   "meta": {
data:     "vtag": {
data:       "tag": "dasdfa10ce8b059740bddsfasd8eb1d47853716"
data:     }
data:   },
data:   "cdni-advertisement": {
data:     "capabilities": [
data:       {
data:         "capability-type": "FCI.DeliveryProtocol",
```

```
data:         "capability-value": {
data:           "delivery-protocols": [
data:             "https/1.1"
data:           ]
data:         },
data:         "footprints": [
data:           { "footprint-type": "ipv4cidr",
data:             "footprint-value": [ "203.0.113.0/24" ]
data:           }
data:         ]
data:       },
data:       { ... other CDNI advertisement object ... }
data:     ]
data:   }
data: }

event: application/json-patch+json,my-cdnifci-stream
data: [
data:   { "op": "replace",
data:     "path": "/meta/vtag/tag",
data:     "value": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
data:   },
data:   { "op": "add",
data:     "path": "/cdni-advertisement/capabilities-with-footprints
/0/footprints/0/footprint-value/-",
data:     "value": "192.0.2.0/24"
data:   }
data: ]
```

4. CDNI Advertisement Service using ALTO Network Map

4.1. Network Map Footprint Type: altopid

The ALTO protocol defines a concept called PID to represent a group of IPv4 or IPv6 addresses which can be applied the same management policy. The PID is an alternative to the pre-defined CDNI footprint types (i.e., ipv4cidr, ipv6cidr, asn, and countrycode).

To leverage this concept, this document defines a new CDNI Footprint Type called "altopid". A CDNI Advertisement resource can depend on an ALTO network map resource and use "altopid" footprints to compress its CDNI Footprint Payload.

Specifically, the "altopid" 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 Advertisement resource using "altopid" footprints depends on a network map. For such a CDNI Advertisement resource, the resource id of its dependent network map MUST be included in the "uses" field of its IRD entry, and the "dependent-vtag" field with a reference to this network map MUST be included in its response (see the example in Section 4.2.3).

4.2. Examples

4.2.1. IRD Example

The examples below use the same IRD given in Section 3.7.1.

4.2.2. ALTO Network Map for CDNI Advertisement Example

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

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

HTTP/1.1 200 OK
Content-Length: 309
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": [ "203.0.113.0/24" ]
    }
  }
}
```

4.2.3. ALTO PID Footprints in CDNI Advertisement

This example shows a CDNI Advertisement resource that depends on a network map described in Section 4.2.2.

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

HTTP/1.1 200 OK
Content-Length: 738
Content-Type: application/alto-cdni+json
```


```
{
  "meta" : {
    "dependent-vtags" : [
      {
        "resource-id": "my-eu-netmap",
        "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"
      }
    ]
  },
  "cdni-advertisement": {
    "capabilities-with-footprints": [
      { "capability-type": "FCI.DeliveryProtocol",
        "capability-value": [ "https/1.1" ],
        "footprints": [
          { "footprint-type": "altopid",
            "footprint-value": [ "south-france" ]
          }
        ]
      },
      { "capability-type": "FCI.AcquisitionProtocol",
        "capability-value": [ "https/1.1" ],
        "footprints": [
          { "footprint-type": "altopid",
            "footprint-value": [ "germany", "south-france" ]
          }
        ]
      }
    ]
  }
}
```

4.2.4. Incremental Updates Example

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

```
POST /updates/cdnifci HTTP/1.1
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: 183

{ "add": {
    "my-eu-netmap-stream": {
      "resource-id": "my-eu-netmap"
    },
    "my-netmap-cdnifci-stream": {
      "resource-id": "my-cdnifci-with-pid-footprints"
    }
  }
}

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

event: application/alto-updatestreamcontrol+json
data: {"control-uri":
data: "https://alto.example.com/updates/streams/3141592653590"}

event: application/alto-networkmap+json,my-eu-netmap-stream
data: { ... full Network Map of my-eu-netmap ... }

event: application/alto-cdnifci+json,my-netmap-cdnifci-stream
data: { ... full CDNI Advertisement resource ... }

event: application/json-patch+json,my-netmap-cdnifci-stream
data: [
data:   { "op": "replace",
data:     "path": "/meta/vtag/tag",
data:     "value": "dasdfa10ce8b059740bddsfasd8eb1d47853716"
data:   },
data:   { "op": "add",
data:     "path":
```

```
     data:     "/cdni-advertisement/capabilities-with-footprints
     /0/footprints/0/footprint-value/-",
     data:     "value": "germany"
     data:   }
     data: ]

     event: application/json-patch+json,my-netmap-cdnifci-stream
     data: [
     data:   { "op": "replace",
     data:     "path": "/meta/vtag/tag",
     data:     "value": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
     data:   },
     data:   { "op": "remove",
     data:     "path":
     data:     "/cdni-advertisement/capabilities-with-footprints
     /1/footprints/0/footprint-value/1"
     data:   }
     data: ]
```

5. Filtered CDNI Advertisement using CDNI Capabilities

Section 3 and Section 4 describe CDNI Advertisement Service which can be used to enable a uCDN to get capabilities with footprint restrictions from dCDNs. However, since always getting full CDNI Advertisement resources from dCDNs is inefficient, this document introduces a new service named "Filtered CDNI Advertisement Service", to allow a client to filter a CDNI Advertisement resource using a client-given set of CDNI capabilities. For each entry of the CDNI Advertisement response, an entry will only be returned to the client if it contains at least one of the client given CDNI capabilities. The relationship between a filtered CDNI Advertisement resource and a CDNI Advertisement resource is similar to the relationship between a filtered network/cost map and a network/cost map.

5.1. Media Type

A filtered CDNI Advertisement resource uses the same media type defined for the CDNI Advertisement resource in Section 3.1.

5.2. HTTP Method

A filtered CDNI Advertisement resource is requested using the HTTP POST method.

5.3. Accept Input Parameters

The input parameters for a filtered CDNI Advertisement resource 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-cdnifilter+json" which is a JSON object of type ReqFilteredCDNIAdvertisement, where:

```
object {
    JSONString capability-type;
    JSONValue capability-value;
} CDNICapability;

object {
    [CDNIFCICapability cdni-capabilities<0..*>;]
} ReqFilteredCDNIAdvertisement;
```

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 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 Advertisement resource. The ALTO server MUST interpret entries appearing in a list multiple times as if they appeared only once. If the ALTO server does not define any footprints for a CDNI capability, it MUST omit this capability from the response.

5.4. Capabilities

None.

5.5. Uses

Same to the "uses" field of the CDNI Advertisement resource (see Section 3.5).

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 Advertisement request is invalid if:

* 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 a 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 capability.

The ALTO server returns a filtered CDNI Advertisement resource for a valid request. The format of a filtered CDNI Advertisement resource is the same as a full CDNI Advertisement resource (See Section 3.6.)

The returned CDNI Advertisement resource 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 Advertisement resource from which the filtered CDNI Advertisement resource 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

The examples below use the same IRD example as in Section 3.7.1.

5.7.2. Basic Example

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

```
POST /cdnifci/filtered HTTP/1.1
HOST: alto.example.com
Accept: application/alto-cdni+json
Content-Type: application/cdnifilter+json
Content-Length: 176

{
  "cdni-capabilities": [
    {
      "capability-type": "FCI.DeliveryProtocol",
      "capability-value": {
        "delivery-protocols": [ "https/1.1" ]
      }
    }
  ]
}

HTTP/1.1 200 OK
Content-Length: 571
Content-Type: application/alto-cdni+json

{
  "meta" : {
    "vtag": {
      "resource-id": "my-filtered-cdnifci",
      "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
    }
  },
  "cdni-advertisement": {
    "capabilities-with-footprints": [
      {
        "capability-type": "FCI.DeliveryProtocol",
        "capability-value": {
          "delivery-protocols": [
            "https/1.1",
            "http/1.1"
          ]
        },
```

```
             "footprints": [
               {
                 "footprint-type": "ipv4cidr",
                 "footprint-value": [ "198.51.100.0/24" ]
               }
             ]
           }
         ]
       }
     }
```


5.7.3. Incremental Updates Example

In this example, the ALTO client only cares about the updates of one advertisement object for delivery protocol capability whose value includes "https/1.1". So it adds its limitation of capabilities in "input" field of the POST request.

```
POST /updates/cdnifci HTTP/1.1
Host: fcialtoupdate.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: 346
```


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


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

```
event: application/alto-updatestreamcontrol+json
data: {"control-uri":
data: "https://alto.example.com/updates/streams/3141592653590"}

event: application/alto-cdni+json,my-filtered-fci-stream
data: { ... filtered CDNI Advertisement resource ... }

event: application/json-patch+json,my-filtered-fci-stream
data: [
data:   {
data:     "op": "replace",
data:     "path": "/meta/vtag/tag",
data:     "value": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
data:   },
data:   { "op": "add",
data:     "/cdni-advertisement/capabilities-with-footprints
/0/footprints/0/footprint-value/-",
data:     "value": "192.0.2.0/24"
data:   }
data: ]
```


6. Query Footprint Properties using ALTO Property Map Service

Besides the requirement of retrieving footprints of given capabilities, another common requirement for uCDN is to query CDNI capabilities of given footprints.

Considering each footprint as an entity with properties including CDNI capabilities, a natural way to satisfy this requirement is to use the ALTO property map as defined in [I-D.ietf-alto-unified-props-new]. This section describes how ALTO clients look up properties for individual footprints. First, it describes how to represent footprint objects as entities in the ALTO property map. Then it describes how to represent footprint capabilities as entity properties in the ALTO property map. Finally, it provides examples of the full property map and the filtered property map supporting CDNI capabilities, and their incremental updates.

6.1. Representing Footprint Objects as 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", "countrycode", and "altopid"). Considering each ALTO entity defined in [I-D.ietf-alto-unified-props-new] also has two properties: entity domain type and domain-specific identifier, a straightforward approach to represent a footprint as an ALTO entity is to represent its footprint-type as an entity domain type, and its footprint value as a domain-specific identifier.

Each existing footprint type can be represented as an entity domain type as follows:

* According to [I-D.ietf-alto-unified-props-new], "ipv4" and "ipv6" are two predefined entity domain types, which can be used to represent "ipv4cidr" and "ipv6cidr" footprints respectively.

* "pid" is also a predefined entity domain type, which can be used to represent "altopid" footprints. Note that "pid" is a resource-specific entity domain. To represent an "altopid" footprint, the specifying information resource of the corresponding "pid" entity domain MUST be the dependent network map used by the CDNI Advertisement resource providing this "altopid" footprint.

* However, no existing entity domain type can represent "asn" and "countrycode" footprints. To represent footprint-type "asn" and "countrycode", this document registers two new domains in Section 7 in addition to the ones in [I-D.ietf-alto-unified-props-new].

Here is an example of representing a footprint object of "ipv4cidr" type as a set of "ipv4" entities in the ALTO property map. The representation of the footprint object of "ipv6cidr" type is similar.

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

6.1.1. ASN Domain

The ASN domain associates property values with Autonomous Systems in the Internet.

6.1.1.1. Entity Domain Type

asn

6.1.1.2. Domain-Specific Entity Identifiers

The entity identifier of an entity in an asn domain is encoded as a string consisting of the characters "as" (in lowercase) followed by the Autonomous System Number [RFC6793].

6.1.1.3. Hierarchy and Inheritance

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

6.1.2. COUNTRYCODE Domain

The COUNTRYCODE domain associates property values with countries.

6.1.2.1. Entity Domain Type

countrycode

6.1.2.2. Domain-Specific Entity Identifiers

The entity identifier of an entity in a 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. Representing CDNI Capabilities as Property Map Entity Properties

This document defines a new entity property type called "cdni-capabilities". An ALTO server can provide a property map resource mapping the "cdni-capablities" entity property type for a CDNI Advertisement resource that it provides to an "ipv4", "ipv6", "asn" or "countrycode" entity domain.

6.2.1. Defining Information Resource Media Type for Property Type cdni-capabilities

The entity property type "cdni-capabilities" allows to define resource-specific entity properties. When resource-specific entity properties are defined with entity property type "cdni-capabilities", the defining information resource for a "cdni-capabilities" property MUST be a CDNI Advertisement resource provided by the ALTO server. The media type of the defining information resource for a "cdni-capabilities" property is therefore:

application/alto-cdni+json

6.2.2. Intended Semantics of Property Type cdni-capabilities

A "cdni-capabilities" property for an entity is to indicate all the CDNI capabilities that a corresponding CDNI Advertisement resource provides for the footprint represented by this entity. Thus, the value of a "cdni-capabilities" property MUST be a JSON array. Each element in a "cdni-capabilities" property MUST be an JSON object as format of CDNICapability (see Section 5.3). The value of a "cdni-capabilities" property for an "ipv4", "ipv6", "asn", "countrycode" or "altopid" entity MUST include all the CDNICapability objects that are provided by the defining CDNI Advertisement resource and the represented footprint object of this entity are in their footprint restrictions.

6.3. Examples

6.3.1. IRD Example

The examples use the same IRD example given by Section 3.7.1.

6.3.2. Property Map Example

This example shows a full property map in which entities are footprints and entities' property is "cdni-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: 1522
Content-Type: application/alto-propmap+json
```


```
{
  "property-map": {
    "meta": {
      "dependent-vtags": [
        { "resource-id": "my-default-cdnifci",
          "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf62"}
      ]
    },
    "countrycode:us": {
      "my-default-cdnifci.cdni-capabilities": [
        { "capability-type": "FCI.DeliveryProtocol",
          "capability-value": {
```

```
                "delivery-protocols": ["http/1.1"]}}]
        },
        "ipv4:192.0.2.0/24": {
          "my-default-cdnifci.cdni-capabilities": [
            { "capability-type": "FCI.DeliveryProtocol",
              "capability-value": {
                "delivery-protocols": ["http/1.1"]}}]
        },
        "ipv4:198.51.100.0/24": {
          "my-default-cdnifci.cdni-capabilities": [
            { "capability-type": "FCI.DeliveryProtocol",
              "capability-value": {
                "delivery-protocols": ["https/1.1", "http/1.1"]}}]
        },
        "ipv4:203.0.113.0/24": {
          "my-default-cdnifci.cdni-capabilities": [
            { "capability-type": "FCI.AcquisitionProtocol",
              "capability-value": {
                "acquisition-protocols": ["http/1.1"]}}]
        },
        "ipv6:2001:db8::/32": {
          "my-default-cdnifci.cdni-capabilities": [
            { "capability-type": "FCI.DeliveryProtocol",
              "capability-value": {
                "delivery-protocols": ["http/1.1"]}}]
        },
        "asn:as64496": {
          "my-default-cdnifci.cdni-capabilities": [
            { "capability-type": "FCI.DeliveryProtocol",
              "capability-value": {
                "delivery-protocols": ["https/1.1", "http/1.1"]}}]
        }
      }
    }
```


6.3.3. Filtered Property Map Example

This example uses the filtered property map service to get "pid" and "cdni-capabilities" properties for two footprints "ipv4:192.0.2.0/24" and "ipv6:2001:db8::/32".

```
      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: 181

      {
        "entities": [
          "ipv4:192.0.2.0/24",
          "ipv6:2001:db8::/32"
        ],
        "properties": [ "my-default-cdnifci.cdni-capabilities",
                        "my-default-networkmap.pid" ]
      }

    HTTP/1.1 200 OK
    Content-Length: 796
    Content-Type: application/alto-propmap+json

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

6.3.4. Incremental Updates Example

In this example, the client is interested in updates for the properties "cdni-capabilities" and "pid" of two footprints "ipv4:192.0.2.0/24" and "countrycode:fr".

```
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: 337

{ "add": {
    "fci-propmap-stream": {
      "resource-id": "filtered-cdnifci-property-map",
      "input": {
        "properties": [ "my-default-cdnifci.cdni-capabilities",
                        "my-default-networkmap.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: "https://alto.example.com/updates/streams/1414213562373"}

event: application/alto-cdni+json,fci-propmap-stream
data: { ... filtered property map ... }

event: application/merge-patch+json,fci-propmap-stream
data: {
data:   "property-map": {
data:     "meta": {
data:       "dependent-vtags": [
data:         { "resource-id": "my-default-cdnifci",
data:           "tag": "2beeac8ee23c3dd1e98a73fd30df80ece9fa5627"},
data:         { "resource-id": "my-default-networkmap",
data:           "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf63"}
data:       ]
data:     },
```

```
    data:     "ipv4:192.0.2.0/24": {
    data:       "my-default-cdnifci.cdni-capabilities": [
    data:         { "capability-type": "FCI.DeliveryProtocol",
    data:           "capability-value": {
    data:             "delivery-protocols": ["http/1.1", "https/1.1"]}}]
    data:     }
    data:   }
    data: }

    event: application/json-patch+json,fci-propmap-stream
    data: [
    data:   { "op": "replace",
    data:     "path": "/meta/dependent-vtags/0/tag",
    data:     "value": "61b23185a50dc7b334577507e8f00ff8c3b409e4"
    data:   },
    data:   { "op": "replace",
    data:     "path":
    data:     "/property-map/countrycode:fr/my-default-networkmap.pid",
    data:     "value": "pid5"
    data:   }
    data: ]
```


7. IANA Considerations

7.1. application/alto-* Media Types

This document registers two additional ALTO media types, listed in Table 1.

Type	Subtype	Specification
application	alto-cdni+json	Section 3
application	alto-cdnifilter+json	Section 5

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: 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).

7.2. CDNI Metadata Footprint Type Registry

As proposed in Section 7.2 of [RFC8006], "CDNI Metadata Footprint Types" registry is requested. A new footprint type is to be registered, listed in Table 2.

Footprint Type	Description	Specification
altopid	A list of PID names	Section 4 of RFCthis

Table 2: CDNI Metadata Footprint Type

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

7.3. ALTO Entity Domain Type Registry

As proposed in Section 11.2 of [I-D.ietf-alto-unified-props-new], "ALTO Entity Domain Type Registry" is requested. Two new entity domain types are to be registered, listed in Table 3.

Identifier	Entity Address Encoding	Hierarchy & Inheritance	Media Type of Defining Resource
asn	See Section 6.1.1.2 of RFCthis	None	None
countrycode	See Section 6.1.2.2 of RFCthis	None	None

Table 3: Additional ALTO Entity Domain Types

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

7.4. ALTO Entity Property Type Registry

As proposed in Section 11.3 of [I-D.ietf-alto-unified-props-new], "ALTO Entity Property Type Registry" is required. A new entity property type is to be registered, listed in Table 4.

Identifier	Intended Semantics	Media Type of Defining Resource
cdni-capabilities	Section 6.2 of RFCthis	application/alto-cdni+json

Table 4: Additional ALTO Entity Property Type

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

8. Security Considerations

As an extension of the base ALTO protocol ([RFC7285]), this document fits into the architecture of the base protocol. And hence Security Considerations of the base protocol (Section 15 of [RFC7285]) fully apply when this extension is provided by an ALTO server.

In the context of CDNI Advertisement, additional security considerations should be included as follows:

* For authenticity and integrity of ALTO information, an attacker may disguise itself as an ALTO server for a dCDN, and provide false capabilities and footprints to a uCDN using the CDNI Advertisement service. Such false information may lead a uCDN to (1) select an incorrect dCDN to serve user requests, or (2) skip uCDNs in good conditions.

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

* For confidentiality and privacy of ALTO information, footprint properties integrated with ALTO unified property may expose network location identifiers (e.g., IP addresses or fine-grained PIDs).

* For availability of ALTO services, an attacker may conduct service degradation attacks using services defined in this document to disable ALTO services of a network. It may request potentially large, full CDNI Advertisement resources from an ALTO server in a dCDN continuously, to consume the bandwidth resources of that ALTO server. It may also query filtered property map services with many smaller individual footprints, to consume the computation resources of the ALTO server.

Although protection strategies as described in Section 15 of [RFC7285] should be applied to address aforementioned security considerations, one additional information leakage risk introduced by this document could not be addressed by these strategies. In particular, if a dCDN signs agreements with multiple uCDNs without any isolation, this dCDN may disclose extra information of one uCDN to another one. In that case, one uCDN may redirect requests which should not have to be served by this dCDN to it.

To reduce the risk, a dCDN should isolate full/filtered CDNI Advertisement resources for different uCDNs. It could consider generating URIs of different full/filtered CDNI Advertisement resources by hashing its company ID, a uCDN’s company ID as well as their agreements. A dCDN should avoid exposing all full/filtered CDNI Advertisement resources in one of its IRDs.

9. Acknowledgments

The authors thank Matt Caulfield, Danny Alex Lachos Perez, Daryl Malas and Sanjay Mishra for their timely reviews and invaluable comments.

Jan Seedorf has been 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.

This document has also been supported by the Coordination Support Action entitled 'Supporting European Experts Presence in lnternational Standardisation Activities in ICT' ("StandlCT.eu") funded by the European Commission under the Horizon 2020 Programme with Grant Agreement no. 780439. 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 European Commission.

10. Contributors

Mr. Xiao Shawn Lin is an author of an early version of this document, with many contributions.

11. References

11.1. Normative References

[ISO3166-1]
The International Organization for Standardization, "Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes", ISO 3166-1:2013, 2013.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6793] Vohra, Q. and E. Chen, "BGP Support for Four-Octet Autonomous System (AS) Number Space", RFC 6793, DOI 10.17487/RFC6793, December 2012, <https://www.rfc-editor.org/info/rfc6793>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

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

[RFC8006] Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "Content Delivery Network Interconnection (CDNI) Metadata", RFC 8006, DOI 10.17487/RFC8006, December 2016, <https://www.rfc-editor.org/info/rfc8006>.

[RFC8008] Seedorf, J., Peterson, J., Previdi, S., van Brandenburg, R., and K. Ma, "Content Delivery Network Interconnection (CDNI) Request Routing: Footprint and Capabilities Semantics", RFC 8008, DOI 10.17487/RFC8008, December 2016, <https://www.rfc-editor.org/info/rfc8008>.

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

11.2. Informative References

[I-D.ietf-alto-path-vector] Gao, K., Lee, Y., Randriamasy, S., Yang, Y., and J. Zhang, "ALTO Extension: Path Vector", Work in Progress, Internet-Draft, draft-ietf-alto-path-vector-12, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-alto-path-vector-12.txt>.

[I-D.ietf-alto-unified-props-new] Roome, W., Randriamasy, S., Yang, Y., Zhang, J., and K. Gao, "Unified properties for the ALTO protocol", Work in Progress, Internet-Draft, draft-ietf-alto-unified-props-new-13, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-alto-unified-props-new-13.txt>.

[RFC5693] Seedorf, J. and E. Burger, "Application-Layer Traffic Optimization (ALTO) Problem Statement", RFC 5693, DOI 10.17487/RFC5693, October 2009, <https://www.rfc-editor.org/info/rfc5693>.

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <https://www.rfc-editor.org/info/rfc6707>.

[RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <https://www.rfc-editor.org/info/rfc7336>.

[RFC7975] Niven-Jenkins, B., Ed. and R. van Brandenburg, Ed., "Request Routing Redirection Interface for Content Delivery Network (CDN) Interconnection", RFC 7975, DOI 10.17487/RFC7975, October 2016, <https://www.rfc-editor.org/info/rfc7975>.

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

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
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.ietf@gmail.com

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

Email: jon.peterson@neustar.biz

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

Email: jingxuan.zhang@tongji.edu.cn

ALTO WG W. Roome
Internet-Draft Nokia Bell Labs
Intended status: Standards Track Y. Yang
Expires: September 21, 2020 Yale University
March 20, 2020

ALTO Incremental Updates Using Server-Sent Events (SSE)
draft-ietf-alto-incr-update-sse-22

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

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.

Copyright Notice

Table of Contents

1. Introduction

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 small, in that if only a small section of an information resource changes, the ALTO server can send just the changes; and (2) updates can be immediate, in that the ALTO server can send updates as soon as they are available.

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 ALTO endpoint property and endpoint cost services. 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 11.

The rest of this document is organized as follows. Section 3 gives background on the basic techniques used in this design: (1) JSON merge patch and JSON patch to allow incremental update; and (2) Server-Sent Events (SSE) [SSE] to allow server push. With the background, Section 4 gives a non-normative overview of the design. Section 5 defines individual messages in an update stream. Section 6 defines the update stream service; Section 7 defines the stream control service; Section 8 gives several examples to illustrate the two types of services. Section 9 describes operation and processing considerations by both ALTO servers and clients; Section 15 discusses a design feature that is not supported; Section 10 discusses security issues; Section 11 and Section 12 review the requirements for future ALTO services to use SSE and IANA considerations, respectively.

2. Terms

Besides the terminologies as defined in [RFC7285], this document also uses additional terminologies defined as follows:

Update Stream: A reliable, in-order HTTP/1.x compatible 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: This document refers 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 Information Resource Directory (IRD).

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

Data Update Message: An update message that 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 data update message for a resource that encodes the content of the resource in its original ALTO encoding.

Incremental Change: An data update message that 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: A service that provides an HTTP URI so that the ALTO client of an update stream can use it to send stream control requests to the ALTO server on the addition or removal of resources receiving update messages from the 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. (Note that the Stream Control Service in ALTO has no association with the similarly named Stream Control Transmission Protocol [RFC4960].)

Stream Control: A shorthand for stream control service.

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

Substream-ID: An ALTO client can assign a unique substream-id when requesting the addition of a resource receiving update messages from an update stream. The server puts the substream-id in each update event for that resource. Substream-id allows a client to use one update stream to receive updates to multiple requests for the same resource (i.e., with the same resource-id in an ALTO IRD), for example, for a POST-mode resource with different input parameters.

Data-ID: A subfield of the 'event' field of [SSE] to identify the ALTO data (object) to be updated. For an ALTO resource returning a multipart response, the data-id to identify the data (object) is the substream-id, in addition to the content-id of the object in the multipart response. The data-id of a single part response is just the substream-id.

Control Update Message: An update message for the update stream server to notify the ALTO client of related control information of the update stream. A control update message may be triggered by an internal event at the server, such as server overloading and hence the update stream server will no longer send updates for an information resource, or as a result of a client sending a request through the stream control service. 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.

3. Background

The design requires two basic techniques: encoding of incremental changes and server push. For incremental changes, existing techniques include JSON merge patch and JSON patch; this design uses both. For server push, existing techniques include HTTP/2 and [SSE]; this design adopts some design features of HTTP/2 but uses [SSE] as the basic server-push design. The rest of this section gives a non-normative summary of JSON merge patch, JSON patch, HTTP/2 and [SSE].

3.1. Incremental Encoding: JSON Merge Patch

To avoid always sending complete data, a server needs mechanisms to encode incremental changes, and JSON merge patch is one mechanism. [RFC7396] defines the encoding of incremental changes (called JSON merge patch objects) to be used by the HTTP PATCH method [RFC5789]. This document adopts from [RFC7396] only the JSON merge patch object encoding and does not use the HTTP PATCH method, as the updates are sent as events, instead of HTTP methods; also the updates are server-to-client in the updates, and PATCH semantics is more for client-to-server. Below is a non-normative summary of JSON merge patch objects; see [RFC7396] for the normative definition.

3.1.1. JSON Merge Patch Encoding

Informally, a JSON merge patch message consists of a JSON merge patch object (referred to as a patch in [RFC7396]), which defines how to transform one JSON value into another using a recursive merge patch algorithm. Specifically, the patch is computed by treating two JSON values (first one being the original, and the second being the updated) 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 patch 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. Hence, in the patch, null as the value of a name/value pair will delete the element with "name" in the original JSON value. The patch does not have an entry for any leaf that has the same value in both versions. See the MergePatch pseudocode at the beginning of Section 2 of [RFC7396] for the formal specification of how to apply a given patch. 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.

3.1.2. 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, the sections below show the application of JSON merge patch to two key ALTO messages.

3.1.2.1. 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:

```
     {
       "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 "203.0.113.0/25" and an ipv6 prefix "2001:db8:8000::/33" to "PID1", (2) deletes "PID2", and (3) assigns a new "tag" to the network map:

```
     {
       "meta" : {
         "vtag" : {
           "tag" : "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
         }
       },
       "network-map": {
         "PID1" : {
           "ipv4" : [ "192.0.2.0/24", "198.51.100.0/25",
                      "203.0.113.0/25" ],
           "ipv6" : [ "2001:db8:8000::/33" ]
         },
         "PID2" : null
       }
     }
```

Applying the JSON merge patch update to the initial network map is equivalent to the following ALTO network map:

```
{
  "meta" : {
    "vtag": {
      "resource-id" : "my-network-map",
      "tag" : "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
    }
  },
  "network-map" : {
    "PID1" : {
      "ipv4" : [ "192.0.2.0/24", "198.51.100.0/25",
                 "203.0.113.0/25" ],
      "ipv6" : [ "2001:db8:8000::/33" ]
    },
    "PID3" : {
      "ipv4" : [ "0.0.0.0/0" ],
      "ipv6" : [ "::/0" ]
    }
  }
}
```


3.1.2.2. 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:


```
{
  "meta" : {
    "dependent-vtags" : [
      {"resource-id": "my-network-map",
       "tag": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
      }
    ],
    "cost-type" : {
      "cost-mode"  : "numerical",
      "cost-metric": "routingcost"
    },
    "vtag": {
      "resource-id" : "my-cost-map",
      "tag" : "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"
    }
  },
  "cost-map" : {
    "PID1": { "PID1": 1,  "PID2": 5,  "PID3": 10 },
    "PID2": { "PID1": 5,  "PID2": 1,  "PID3": 15 },
    "PID3": { "PID1": 20, "PID2": 15  }
  }
}
```

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:

```
{
  "meta" : {
    "dependent-vtags" : [
      {"resource-id": "my-network-map",
       "tag": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
      }
    ],
    "cost-type" : {
      "cost-mode"  : "numerical",
      "cost-metric": "routingcost"
    },
    "vtag": {
      "resource-id": "my-cost-map",
      "tag": "c0ce023b8678a7b9ec00324673b98e54656d1f6d"
    }
  },
  "cost-map" : {
    "PID1": { "PID1": 1,  "PID2": 9,  "PID3": 10 },
    "PID2": { "PID1": 5,  "PID2": 1,  "PID3": 15 },
    "PID3": {             "PID2": 15, "PID3": 1  }
  }
}
```

3.2. Incremental Encoding: JSON Patch

3.2.1. 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 3.1.2.1. 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 3.1.1) 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. See [RFC6902] for the normative definition.

3.2.2. JSON Patch ALTO Messages

To provide both examples of JSON patch and a demonstration of the difference between JSON patch and JSON merge patch, the sections below show the application of JSON patch to the same updates shown in Section 3.1.2.

3.2.2.1. JSON Patch Network Map Messages

First consider the same update as in Section 3.1.2.1 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": "203.0.113.0/25"
  }
  {
    "op": "add",
    "path": "/network-map/PID1/ipv6",
    "value": ["2001:db8:8000::/33"]
  },
  {
    "op": "remove",
    "path": "/network-map/PID2"
  }
]
```


3.2.2.2. 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 3.1.2.2, the encoding using JSON patch is:

```
[
  {
    "op": "replace",
    "path": "/meta/vtag/tag",
    "value": "c0ce023b8678a7b9ec00324673b98e54656d1f6d"
  },
  {
    "op": "replace",
    "path": "/cost-map/PID1/PID2",
    "value": 9
  },
  {
    "op": "remove",
    "path": "/cost-map/PID3/PID1"
  },
  {
    "op": "replace",
    "path": "/cost-map/PID3/PID3",
    "value": 1
  }
]
```


3.3. Multiplexing and Server Push: HTTP/2

HTTP/2 ([RFC7540]) provides two related features: multiplexing and server push. In particular, HTTP/2 allows a client and a server to multiplex multiple HTTP requests and responses over a single TCP connection. The requests and responses can be interleaved on a block (frame) by block (frame) basis, by indicating the requests and responses in HTTP/2 messages, avoiding the head-of-line blocking problem encountered with HTTP/1.1. To achieve the same goal, this design introduces substream-id to allow a client to receive updates to multiple resources. HTTP/2 also provides a Server Push facility, to allow a server to send asynchronous updates.

Despite the two features of HTTP/2, this design chooses an HTTP/1.x-compatible design for the simplicity of HTTP/1.x. An HTTP/2 based design may more likely need to be implemented using a more complex HTTP/2 client library. In such a case, 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. An HTTP/2 client library may 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. Further, it is more likely that an HTTP/2 based design may encounter issues with a proxy between the client and the server, in that 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 this design leaves an HTTP/2 based design as a future work and focuses on ALTO updates on HTTP/1.x and [SSE].

3.4. Server Push: Server-Sent Event

Server-Sent Events (SSE) is a technique which can work with HTTP/1.1. The following is a non-normative summary of SSE; see [SSE] for its normative definition.

SSE 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. Overview of Approach and High-level Protocol Message Flow

With the preceding background, this section now gives a non-normative overview of the update mechanisms and message flow to be defined in later sections of this document. Figure 2 gives the main components and overall message flow.

```
    ------------------------------------------------------------------
   |                                                                  |
   |          +-------+         +-------+ 1. init request  +------+   |
   |          |       |         |       | <-------------   |      |   |
   |          |       |         |       | ------------->   |      |   |
   | 3.add/   |       |         |       | 1'. control uri  |      |   |
   | remove   |       |         |       |                  |      |   |
   | resource |Stream |         |Update |                  |      |   |
     -------->|Control| private |Stream | 2a. data update  |Client| --
              |Server |<------->|Server | messages         |      |
     -------- |       |         |       | -------------->  |      | <-
   | response |       |         |       | -------------->  |      |   |
   |          |       |         |       | 2b.control update|      |   |
   |          +-------+         +-------+ messages         +------+   |
   |                                                                  |
    ------------------------------------------------------------------
```

Figure 2: ALTO SSE Architecture and Message Flow.

4.1. Update Stream Service Message Flow

The building block of the update mechanism defined in this document is the update stream service (defined in Section 6), where each update stream service is a POST-mode service that provides update streams.

Note that the lines of the format "** ... **" are used to describe message flows in this section and the following sections.

** Initial request: client -> update server **:

When an ALTO client requests an update stream service, the ALTO client establishes a persistent connection to the update stream server and submits an initial update-stream request (defined in Section 6.5), creating an update stream. This initial request creating the update stream is labeled "1. init request" in Figure 2.

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. Also, to avoid creating too many update streams, this design allows an ALTO client to use one update stream to receive updates to multiple requests. In particular, the client may request to receive updates for the same resource but with different parameters for a POST-mode resource, in addition to being able to consolidate updates for multiple resources into a single stream. The updates for each request is called a substream, and hence, the update server needs an identifier to indicate the substream when sending an update. To achieve this goal, the client assigns a unique substream-id when requesting updates to a resource in an update stream, and the server puts the substream-id in each update.

** Data updates: update server -> client **:

The objective of an update stream is to continuously push to an ALTO client data value changes to a set of resources, where the set of resources is specified by the ALTO client's requests. This document refers to messages sending such data-value changes as data update messages (defined in Section 5.2). Although an update stream may update one or more requests, each data update message updates only one request 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. Since the transport is a HTTP/1.x compatible design, data update messages are delivered reliably and in order, and the lossless, sequential delivery of its messages allows the server to know the exact state of the client to compute the correct incremental updates. Figure 2 shows examples of data update messages (labeled "2a. data update messages")in the overall message flow.

** Control updates: update server -> client **:

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 document allows the update stream server to send control update messages (defined in Section 5.3) in addition to data update messages to the ALTO client. Figure 2 shows that both data updates and control updates can be sent by the server to the client (labeled "2b. control update messages").

4.2. Stream Control Service Message Flow

** Stream control: client -> stream control server **:

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 (defined in Section 7). 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 (labeled "1': control uri" in Figure 2).

The ALTO client can then use the URI to ask the stream control server specified in the URI to request 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.

4.3. Service Announcement and Management Message Flow

** Service announcements: IRD server -> client **:

An update 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.

** Service management (server) **:

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.

5. Update Messages: Data Update and Control Update Messages

This section defines the format of update messages sent from the server to the client. It first defines the generic structure of update messages (Section 5.1). It then defines the details of the data update messages (Section 5.2) and the control update messages (Section 5.3). These messages will be used in the next two sections to define the Update Stream Service (Section 6) and the Stream Control Service (Section 7).

5.1. Generic ALTO Update Message Structure

Both data update and control update messages from the server to the client have the same basic structure: each message includes a data field to provide data information, which is typically a JSON object; and an event field preceding the data field, to specify the media type indicating the encoding of the data field.

A data update message needs additional information to identify the ALTO data (object) to which the update message applies. To be generic, this document use a data-id to identify the ALTO data (object) to be updated; see below.

Hence, the event field of ALTO update message can include two sub-fields (media-type and data-id), where the two sub-fields are separated by a comma (',', U+002C):

```
      media-type [ ',' data-id ]
```

According to Section 4.2 of [RFC6838], the comma character is not allowed in a media-type name. So there is no ambiguous when decoding of the two sub-fields.

Note that an update message does not use the SSE "id" field.

5.2. ALTO Data Update Message

A data update message is sent when a monitored resource changes. As discussed in the preceding section, the event field of a data update message includes two sub-fields: 'media-type' and 'data-id'.

The 'media-type' sub-field depends on whether the data update is a complete specification of the identified data, or an incremental patch (e.g., a JSON merge patch or JSON patch), if possible, describing the changes from the last version of the data. This document refers to these as full replacement and incremental change, respectively. The encoding of a full replacement is defined by its defining document (e.g., network and cost map messages by [RFC7285]), and uses the media type defined in that document. The encoding of JSON merge patch is defined by [RFC7396], with the media type "application/merge-patch+json"; the encoding of JSON patch is defined by [RFC6902], with media type "application/json-patch+json".

The `data-id` sub-field identifies the ALTO data to which the data update message applies.

First consider the case that the resource containing only a single JSON object. For example, since an ALTO client can request data updates for both a cost map resource (object) and its dependent network map resource (object) in the same update stream, to distinguish the updates, the client assigns a substream-id for each resource receiving data updates. Substream-ids MUST be unique within an update stream, but need not be globally unique. A substream-id is encoded as a JSON string with the same format as that of the type ResourceID (Section 10.2 of [RFC7285]). The type SubstreamID is used in this document to indicate a string of this format. The substream-id of a single JSON object is the 'data-id'.

As an example, assume that the ALTO client assigns substream-id "1" in its request to receive updates to the network map; and substream-id "2" to the cost map. Then the substream-ids are the data-ids indicating which objects will be updated. 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.

Next consider the case that a resource may include multiple JSON objects. This document considers the case that a resource may contain multiple components (parts) and they are encoded using the media type "multipart/related" [RFC2387]. Each part of this multipart response MUST be an HTTP message including a Content-ID header and a JSON object body. Each component requiring the update stream service (defined in Section 6) MUST be identified by a unique Content-ID to be defined in its defining document.

For a resource using the media type "multipart/related", the ‘data-id‘ sub-field MUST be the concatenation of the substream-id, the ’.’ separator (U+002E) and the unique Content-ID in order.

5.3. 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;]
   [SubstreamID     started<1..*>;]
   [SubstreamID     stopped<1..*>;]
   [String          description;]
} UpdateStreamControlEvent;
```

control-uri: the URI providing stream control for this update stream (see Section 7). The server sends a control update message notifying the client of the control-uri. This control update message notifying the control-uri will be sent once and MUST be the first event in an update stream. If the URI value 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 substream-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 substream-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 6.7.1) or an internal server event.

description: a non-normative, human-readable 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. There can be multiple reasons which trigger a "stopped" event; see above. The intention of this field is to provide a human-readable text for the developer and/or the administrator to diagnose potential problems.

6. Update Stream Service

An update stream service returns a stream of update messages, as defined in Section 5. 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. An IRD entry defining an update stream service MUST define the media type, HTTP method, and capabilities & uses as follows.

6.1. Media Type

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

6.2. HTTP Method

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

6.3. Capabilities

The capabilities are defined as an object of type UpdateStreamCapabilities:

```
  object {
    IncrementalUpdateMediaTypes incremental-change-media-types;
    Boolean                     support-stream-control;
  } UpdateStreamCapabilities;

  object-map {
     ResourceID -> String;
  } IncrementalUpdateMediaTypes;
```

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 supported media types of the incremental change separated by commas. 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 5, 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-types to encode incremental changes for a resource, the update stream server MUST 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 3.1.1 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 MUST choose JSON patch over JSON merge patch, if JSON patch is indicated as a capability of the update stream server; If the the server does not support JSON patch to handle such a case, the server then need to send a full replacement.

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 use the stream control specified in this document; else, the update stream server may use other mechanisms to provide the same functionality as stream control.

6.4. 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.

6.5. Request: 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:

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

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

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

add: specifies the resources (and the parameters for the resources) for which the ALTO client wants updates. In the scope of the same update stream, the ALTO client MUST assign a substream-id that is unique in the scope of the update stream (Section 5.2) for each entry, and use those substream-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 6.4). 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 6.7.1. 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 this substream. If the "incremental-changes" field is "true", the update stream server MAY send incremental changes for this substream. In this case, the client MUST support all incremental methods from the set announced in the server's capabilities for this resource; see Section 6.3 for server's announcement of potential incremental methods. If a client does not support all incremental methods from the set announced in the server's capabilities, the client can set "incremental-changes" to "false", and the update stream server then MUST NOT send incremental changes for that substream. The default value for "incremental-changes" is "true", so to suppress incremental changes, the ALTO client MUST explicitly set "incremental-changes" to "false". An alternative design of incremental-changes control is a more fine-grained control, by allowing a client to select a subset of incremental methods from the set announced in the server's capabilities. But this alternative design is not adopted in this document, because it adds complexity to the server, which is more likely to be the bottleneck. 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 the resource expects.

remove: it is used in update stream control requests (Section 7), 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 6.6.

6.6. 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: 131
Content-Type: application/alto-error+json
Connection: Closed

{
    "meta":{
        "code":  "E_INVALID_FIELD_VALUE",
        "field": "add/my-network-map/resource-id",
        "value": "my-networkmap/#"
    }
}
```

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

- o 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.

- o 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 the full path of the "resource-id" field 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 (i.e., TCP connection) without sending any events.

- o 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 5 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.

6.7. Additional Requirements on Update Stream Service

6.7.1. Event Sequence Requirements

- The first event MUST be a control update message with the URI of the update stream control service (see Section 7) for this update stream. Note that the value of the control-uri can be "null", indicating that there is no control stream service.

- As soon as possible after the ALTO client initiates the connection, the update stream server checks the "tag" field for each added update request. If the "tag" field is not specified in an added update request, the update stream server MUST first send a full replacement for the request. If the the "tag" field is specified, the client can accept incremental changes, and the server can compute an incremental update based on the "tag" (the server needs to ensure that for a POST resource with input, the "tag" should indicate the correct result for different inputs), the update stream server MAY omit the initial full replacement.

- If this update stream provides updates for resource-ids 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 7), the ALTO client MUST send a stream control request. The update stream server MUST send a control update message whose "stopped" field has the substream-ids of all stopped resources.

6.7.2. Cross-Stream Consistency Requirements

If multiple ALTO clients create multiple update streams from the same update stream resource, and with the same update request parameters (i.e., same resource, same input), 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 two different 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.

6.7.3. Multipart Update Requirements

This design allows any valid media type for full replacement. Hence, it supports ALTO resources using multipart to contain multiple JSON objects. This realizes the push benefit, but not the incremental encoding benefit of SSE.

JSON patch and merge patch provide the incremental encoding benefit but can be applied to only a single JSON object. If an update stream service supports a resource providing a multipart media type, which we refer to as a multipart resource, then the update stream service needs to handle the issue that the message of a full multipart resource can include multiple JSON objects. To address the issue, when an update stream service specifies that it supports JSON patch or merge patch incremental updates for a multipart resource, the service MUST ensure that (1) each part of a multipart message is a single JSON object, (2) each part is specified by a static content-id in the initial full message, (3) each data update event applies to only one part; and (4) each data update specifies substream-id.content-id as the `event` field of the event, to identify the part to be updated.

6.8. 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. Note that although TCP also provides a Keep-alive function, the interval between TCP Keep-alive messages can depend on the OS configuration and varies. The preceding recommended SSE keep-alive allows the SSE client to detect the status of the update stream server with more certainty.

7. Stream Control Service

A 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 5.3).

Each stream control request is an individual HTTP request. The ALTO client MAY send multiple stream control requests to the stream control server using the same HTTP connection.

7.1. URI

The URI for a 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 relative control URI reference [RFC3986] (for example, a URI reference 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 there is an internal mechanism to map a stream control URI to the unique update stream instance to be controlled. For example, the update stream service may assign a unique, internal stream id to each update stream instance. However, the exact mechanism is left to the update stream service provider.

To prevent an attacker from forging a stream control URI and sending bogus requests to disrupt other update streams, the service should consider two security issues. First, if http, not https, is used, the stream control URI can be exposed to an on-path attacker. To address this issue, in a setting where the path from the server to the client can traverse such an attacker, the server SHOULD use https. Second, even without direct exposure, an off-path attacker may guess valid stream control URIs. To address this issue, the server SHOULD choose stream control URIs with enough randomness, to make guessing difficult; the server SHOULD introduce mechanisms that detect repeated guesses indicating an attack (e.g., keeping track of the number of failed stream control attempts); please see https://www.w3.org/TR/capability-urls/ .

7.2. Media Type

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

7.3. HTTP Method

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

7.4. IRD Capabilities & Uses

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

7.5. Request: Accept Input Parameters

A stream control service accepts the same input media type and input parameters as the update stream service (Section 6.5). The only difference is that a stream control service also accepts the "remove" field.

If specified, the "remove" field is an array of substream-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. The ALTO client MUST assign a unique substream-id to each additional resource. Substream-ids MUST be unique over the lifetime of this update stream: an ALTO client MUST NOT reuse a previously removed substream-id. The processing of an "add" resource is the same as discussed in Section 6.5 and Section 6.7.

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 7.6.

7.6. 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 6.6. Detailed error code and error information are specified as below.

- o If the "add" request does not satisfy the requirements in Section 6.5, the stream control server MUST return the ALTO error message defined in Section 6.6.

- o If any substream-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 an array of the invalid substream-ids. Thus it is illegal to "add" and "remove" the same substream-id in the same request. However, it is legal to remove a substream-id twice. To support the preceding checking, the update stream server MUST keep track of previously-used-but-now-closed substream-ids.

- o If any substream-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 an array of invalid substream-ids.

- o If the request has a non-empty "add" field and a "remove" field with an empty list of substream-ids (to replace all active resources with a new set, the client MUST explicitly enumerate the substream-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 then 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 the modular design.

8. Examples

8.1. Example: IRD Announcing Update Stream Services

Below is an example IRD announcing three 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. The third, which is named "update-my-pv", provides updates to a non-standard ALTO service returning a multipart response.

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 9.3, the update stream provides updates for the second, but not the first.

This IRD also announces a non-standard ALTO service, which is named "my-pv". This service accepts an extended endpoint cost request as an input and returns a multipart response including an endpoint cost resource and a property map resource. This document does not rely on any other design details of this new service. In this document, the "my-pv" service is only used to illustrate how the update stream service provides updates to an ALTO resource returning a multipart response.


```
"my-network-map": {
  "uri": "https://alto.example.com/networkmap",
  "media-type": "application/alto-networkmap+json",
},
"my-routingcost-map": {
  "uri": "https://alto.example.com/costmap/routingcost",
  "media-type": "application/alto-costmap+json",
  "uses": ["my-networkmap"],
  "capabilities": {
```

```
          "cost-type-names": ["num-routingcost"]
        }
      },
      "my-hopcount-map": {
        "uri": "https://alto.example.com/costmap/hopcount",
        "media-type": "application/alto-costmap+json",
        "uses": ["my-networkmap"],
        "capabilities": {
          "cost-type-names": ["num-hopcount"]
        }
      },
      "my-filtered-cost-map": {
        "uri": "https://alto.example.com/costmap/filtered/constraints",
        "media-type": "application/alto-costmap+json",
        "accepts": "application/alto-costmapfilter+json",
        "uses": ["my-networkmap"],
        "capabilities": {
          "cost-type-names": ["num-routingcost", "num-hopcount"],
          "cost-constraints": true
        }
      },
      "my-simple-filtered-cost-map": {
        "uri": "https://alto.example.com/costmap/filtered/simple",
        "media-type": "application/alto-costmap+json",
        "accepts": "application/alto-costmapfilter+json",
        "uses": ["my-networkmap"],
        "capabilities": {
          "cost-type-names": ["num-routingcost", "num-hopcount"],
          "cost-constraints": false
        }
      },
      "my-props": {
        "uri": "https://alto.example.com/properties",
        "media-type": "application/alto-endpointprops+json",
        "accepts": "application/alto-endpointpropparams+json",
        "capabilities": {
          "prop-types": ["priv:ietf-bandwidth"]
        }
      },
      "my-pv": {
        "uri": "https://alto.example.com/endpointcost/pv",
        "media-type": "multipart/related;
                       type=application/alto-endpointcost+json",
        "accepts": "application/alto-endpointcostparams+json",
        "capabilities": {
          "cost-type-names": [ "path-vector" ],
          "ane-properties": [ "maxresbw", "persistent-entities" ]
        }
```

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

8.2. Example: Simple Network and Cost Map Updates

Given the update streams announced in the preceding example IRD, the section below shows 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 substream-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: 155

{ "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: "https://alto.example.com/updates/streams/3141592653589"}

event: application/alto-networkmap+json,my-network-map
data: {
data:   "meta" : {
data:     "vtag": {
data:       "resource-id" : "my-network-map",
data:         "tag" : "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
data:       }
data:     },
data:     "network-map" : {
```

```
     data:       "PID1" : {
     data:         "ipv4" : [ "192.0.2.0/24", "198.51.100.0/25" ]
     data:       },
     data:       "PID2" : {
     data:         "ipv4" : [ "198.51.100.128/25" ]
     data:       },
     data:       "PID3" : {
     data:         "ipv4" : [ "0.0.0.0/0" ],
     data:         "ipv6" : [ "::/0" ]
     data:       }
     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:     }],
     data:     "cost-type" : {
     data:       "cost-mode"  : "numerical",
     data:       "cost-metric": "routingcost"
     data:     },
     data:     "vtag": {
     data:       "resource-id" : "my-routingcost-map",
     data:       "tag" : "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"
     data:     }
     data:   },
     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  }
     data:   }
     data: }
```

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

As another example, the following represents a change to the network map: an ipv4 prefix "203.0.113.0/25" 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": "203.0.113.0/25"
        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  }
        data:   }
        data: }
```

8.3. 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 substream-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: 244

{ "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: "https://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 ... }

   (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:

```
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: 24

{
  "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 substream-id and the stream control server will return an error response to the ALTO client.

```
         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: 31
         {
           "remove": [ "properties" ]
         }

         HTTP/1.1 400 Bad Request
         Content-Length: 89
         Content-Type: application/alto-error+json

         {
           "meta":{
           "code": "E_INVALID_FIELD_VALUE",
           "field": "remove",
           "value": "properties"
         }
```

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: 17

{
  "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:

```
event: application/alto-updatestreamcontrol+json
data: {
data:   "stopped": ["net", "routing"]
data: }

    (server closes stream)
```

8.4. 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
Content-Length: 511
```


```
{ "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: "https://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 "priv:ietf-bandwidth" property and the "priv:ietf-load" 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: 448

{ "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: 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:  } }

   (pause)

event: application/merge-patch+json,props-3
data: { "endpoint-properties":
data:   {"ipv4:198.51.100.5" : {"priv:ietf-bandwidth": "15"}}
data: }

   (pause)

event: application/merge-patch+json,props-2
data: { "endpoint-properties":
data:   {"ipv6:2001:db8:100::2" : {"priv:ietf-load": "9"}}
data: }

   (pause)

event: application/merge-patch+json,props-4
data: { "endpoint-properties":
data:   {"ipv6:2001:db8:100::4" : {"priv:ietf-load": "3"}}
data: }
```

8.5. Example: Multipart Message Updates

This example shows how an ALTO client can request a non-standard ALTO service returning a multipart response. The update stream server immediately sends full replacements of the multipart response. After that, the update stream server sends data update messages for the individual parts of the response as the ALTO data (object) in each part changes.

```
POST /updates/pv HTTP/1.1
Host: alto.example.com
Accept: text/event-stream
Content-Type: application/alto-updatestreamparams+json
Content-Length: 382
```


```
{
  "add": {
    "ecspvsub1": {
      "resource-id": "my-pv",
      "input": {
        "cost-type": {
          "cost-mode": "array",
          "cost-metric": "ane-path"
        },
        "endpoints": {
          "srcs": [ "ipv4:192.0.2.2" ],
          "dsts": [ "ipv4:192.0.2.89", "ipv4:203.0.113.45" ]
        },
        "ane-properties": [ "maxresbw", "persistent-entities" ]
      }
    }
  }
}
```

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

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

event: multipart/related;boundary=example-pv;
       type=application/alto-endpointcost+json,ecspvsub1
data: --example-pv
data: Content-ID: ecsmap
data: Content-Type: application/alto-endpointcost+json
data:
data: { ... data (object) of an endpoint cost map ... }
data: --example-pv
data: Content-ID: propmap
data: Content-Type: application/alto-propmap+json
data:
data: { ... data (object) of a property map ... }
data: --example-pv--

   (pause)

event: application/merge-patch+json,ecspvsub1.ecsmap
data: { ... merge patch for updates of ecspvsub1.ecsmap ... }

event: application/merge-patch+json,ecspvsub1.propmap
data: { ... merge patch for updates of ecspvsub1.propmap ... }
```

9. Operation and Processing Considerations

9.1. Considerations for Choosing Data Update Messages

The update stream server should be cognizant of the effects of its update schedule, which includes both the choice of timing (i.e., when/what to trigger an update) and the choice of message format (i.e., given an update, send a full replacement or an incremental change). In particular, the update schedule can have effects on both the overhead and the freshness of information. To minimize overhead, the server may choose to batch a sequence of updates for resources that frequently change, by sending cumulative updates or a full replacement after a while. The update stream server should be cognizant that batching reduces the freshness of information. The server should also consider the effect of such delays on client behaviors (see below on client timeout on waiting for updates of dependent resources).

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; There is no advantage 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.

9.2. 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. The following paragraphs 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 6.7.1).

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 cost maps using the network map become invalid because they are inconsistent with the current network map; hence, the ALTO client MUST mark each such dependent cost map as temporarily invalid and MUST NOT use that each such cost map until the ALTO client receives a cost map update message indicating that it is based on the new network map version tag.

The update stream server SHOULD send updates for dependent resources (i.e., the cost maps in the preceding example) in a timely fashion. However, if the ALTO client does not receive the expected updates, a simple recovery method is that the ALTO client closes the update stream connection, discards the dependent resources, and reestablishes 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.

9.3. 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 SHOULD NOT 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 having to handle such a complicated behavior by offering update streams only for filtered cost maps which do not allow constraint tests.

9.4. 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.

9.5. Considerations for SSE Text Formatting and Processing

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 a full cost map as a single events, and a cost map may involve megabytes, if not tens of megabytes, of text. This has implications that the ALTO client and the update stream server may consider.

First, some SSE client libraries read all data for an event into memory, and then present it to the client as a character array. However, a client 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.

Second, an SSE client library may use a low level, generic socket read library that stores each line of an event data, just in case the higher level parser may need the line delimiters as part of the protocol formatting. A server sending a complete cost map as a single line may then generate a multi-megabyte data "line", and such a long line may then require complex memory management at the client. It is RECOMMENDED that an update stream server limit the lengths of data lines.

Third, an SSE server may use a library which may put line breaks in places that would have semantic consequences for the ALTO updates;

see Section 11. The update stream server implementation MUST ensure that no line breaks are introduced to change the semantics.

10. Security Considerations

The Security Considerations (Section 15 of [RFC7285]) of the base protocol fully apply to this extension. 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 additional services (addition of update streams and stream control URIs) provided by this extension extend the attack surface described in Section 15.1.1 of [RFC7285]. Below we discuss the additional risks and their remedies.

10.1. 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 substream-ids to one update stream.

To avoid these attacks on the update stream server, the server SHOULD choose to limit the number of active streams and reject new requests when that threshold is reached. An update stream server SHOULD also choose to limit the number of active substream-ids on any given stream, or limit the total number of substream-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".

It is important to note that the preceding approach are not the only possibilities. For example, it may be possible for the update stream server to use somewhat more clever logic involving IP reputation, rate-limiting, and compartmentalizing the overall threshold into smaller thresholds that apply to subsets of potential clients.

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.

10.2. 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.

10.3. Stream Control: Spoofed Control Requests and Information Breakdown

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.

In general, as the architecture allows the possibility for the update stream server and the stream control server to be different entities, the additional risks should be evaluated and remedied. For example, the private communication path between the servers may be attacked, resulting in a risk of communications breakdown between them, as well as invalid or spoofed messages claiming to be on that private communications path. Proper security mechanisms, including confidentiality, authenticity, and integrity mechanisms should be considered.

11. 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 9.3.

12. IANA Considerations

This document defines two new media-types, "application/alto-updatestreamparams+json", as described in Section 6.5, and "application/alto-updatestreamcontrol+json", as described in Section 5.3. All other media-types used in this document have already been registered, either for ALTO, JSON merge patch, or JSON patch.

12.1. application/alto-updatestreamparams+json Media Type

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 [RFC8259].

Security considerations: Security considerations relating to the generation and consumption of ALTO Protocol messages are discussed in Section 10 of [RFCthis] and Section 15 of [RFC7285].

Interoperability considerations: [RFCthis] specifies format of conforming messages and the interpretation thereof.

Published specification: Section 6.5 of [RFCthis].

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

Fragment identifier considerations: n/a

Additional information:

Magic number(s): n/a

File extension(s): [RFCthis] 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).

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

12.2. application/alto-updatestreamcontrol+json Media Type

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 [RFC8259].

Security considerations: Security considerations relating to the generation and consumption of ALTO Protocol messages are discussed in Section 10 of [RFCthis] and Section 15 of [RFC7285].

Interoperability considerations: [RFCthis] specifies format of conforming messages and the interpretation thereof.

Published specification: Section 5.3 of [RFCthis].

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

Fragment identifier considerations: n/a

Additional information:

Magic number(s): n/a

File extension(s): [RFCthis] 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).

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

13. Contributors

Section 2, Section 5.1, Section 5.2 and Section 8.5 of this document are based on contributions from Jingxuan Jensen Zhang, and he is considered an author.

14. 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.

15. Appendix: Design Decision: 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 is very rare. The 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.

16. References

16.1. Normative References

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

[RFC2387] Levinson, E., "The MIME Multipart/Related Content-type", RFC 2387, BCP 14, August 1998.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

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

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

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

[RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, October 2014.

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

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[SSE] Hickson, I., "Server-Sent Events (W3C)", W3C Recommendation 03 February 2015, February 2015.

16.2. Informative References

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

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

[RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, May 2015.

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

ALTO K. Gao
Internet-Draft Sichuan University
Intended status: Standards Track Y. Lee
Expires: 24 May 2021 Samsung
S. Randriamasy
Nokia Bell Labs
Y.R. Yang
Yale University
J. Zhang
Tongji University
20 November 2020

ALTO Extension: Path Vector
draft-ietf-alto-path-vector-13

Abstract

This document is an extension to the base Application-Layer Traffic Optimization (ALTO) protocol. It extends the ALTO Cost Map service and ALTO Property Map service so that the application can decide which endpoint(s) to connect based on not only numerical/ordinal cost values but also details of the paths. This is useful for applications whose performance is impacted by specified components of a network on the end-to-end paths, e.g., they may infer that several paths share common links and prevent traffic bottlenecks by avoiding such paths. This extension introduces a new abstraction called Abstract Network Element (ANE) to represent these components and encodes a network path as a vector of ANEs. Thus, it provides a more complete but still abstract graph representation of the underlying network(s) for informed traffic optimization among endpoints.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Network performance metrics are crucial to the Quality of Experience (QoE) of today's applications. The ALTO protocol allows Internet Service Providers (ISPs) to provide guidance, such as topological distance between different end hosts, to overlay applications. Thus, the overlay applications can potentially improve the QoE by better orchestrating their traffic to utilize the resources in the underlying network infrastructure.

Existing ALTO Cost Map and Endpoint Cost Service provide only cost information on an end-to-end path defined by its <source, destination> endpoints: The base protocol [RFC7285] allows the services to expose the topological distances of end-to-end paths, while various extensions have been proposed to extend the capability of these services, e.g., to express other performance metrics [I-D.ietf-alto-performance-metrics], to query multiple costs simultaneously [RFC8189], and to obtain the time-varying values [I-D.ietf-alto-cost-calendar].

While the existing extensions are sufficient for many overlay applications, however, the QoE of some overlay applications depends not only on the cost information of end-to-end paths, but also on particular components of a network on the paths and their properties. For example, job completion time, which is an important QoE metric for a large-scale data analytics application, is impacted by shared bottleneck links inside the carrier network. We refer to such components of a network as Abstract Network Elements (ANE).

Predicting such information can be very complex without the help of the ISP [AAAI2019]. With proper guidance from the ISP, an overlay application may be able to schedule its traffic for better QoE. In the meantime, it may be helpful as well for ISPs if applications could avoid using bottlenecks or challenging the network with poorly scheduled traffic.

Despite the benefits, ISPs are not likely to expose details on their network paths: first for the sake of confidentiality, second because it may result in a huge volume and overhead, and last because it is difficult for ISPs to figure out what information and what details an application needs. Likewise, applications do not necessarily need all the network path details and are likely not able to understand them.

Therefore, it is beneficial for both parties if an ALTO server provides ALTO clients with an "abstract network state" that provides the necessary details to applications, while hiding the network complexity and confidential information. An "abstract network state" is a selected set of abstract representations of Abstract Network Elements traversed by the paths between <source, destination> pairs combined with properties of these Abstract Network Elements that are relevant to the overlay applications' QoE. Both an application via its ALTO client and the ISP via the ALTO server can achieve better confidentiality and resource utilization by appropriately abstracting relevant Abstract Network Elements. The pressure on the server scalability can also be reduced by combining Abstract Network Elements and their properties in a single response.

This document extends [RFC7285] to allow an ALTO server convey "abstract network state", for paths defined by their <source, destination> pairs. To this end, it introduces a new cost type called "Path Vector". A Path Vector is an array of identifiers that each identifies an Abstract Network Element, which can be associated with various properties. The associations between ANEs and their properties are encoded in an ALTO information resource called Unified Property Map, which is specified in [I-D.ietf-alto-unified-props-new].

For better confidentiality, this document aims to minimize information exposure. In particular, this document enables and recommends that first ANEs are constructed on demand, and second an ANE is only associated with properties that are requested by an ALTO client. A Path Vector response involves two ALTO Maps: the Cost Map that contains the Path Vector results and the up-to-date Unified Property Map that contains the properties requested for these ANEs. To enforce consistency and improve server scalability, this document uses the "multipart/related" message defined in [RFC2387] to return the two maps in a single response.

The rest of the document is organized as follows. Section 3 introduces the extra terminologies that are used in this document. Section 4 uses an illustrative example to introduce the additional requirements of the ALTO framework, and discusses potential use cases. Section 5 gives an overview of the protocol design. Section 6 and Section 7 specify the Path Vector extension to the ALTO IRD and the information resources, with some concrete examples presented in Section 8. Section 9 discusses the backward compatibility with the base protocol and existing extensions. Security and IANA considerations are discussed in Section 11 and Section 12 respectively.

2. Requirements Languages

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.

3. Terminology

NOTE: This document depends on the Unified Property Map extension [I-D.ietf-alto-unified-props-new] and should be processed after the Unified Property Map document.

This document extends the ALTO base protocol [RFC7285] and the Unified Property Map extension [I-D.ietf-alto-unified-props-new]. In addition to the terms defined in these documents, this document also uses the following additional terms:

* Abstract Network Element (ANE): An Abstract Network Element is an abstract representation for a component in a network that handle data packets and whose properties can potentially have an impact on the end-to-end performance of traffic. An ANE can be a physical device such as a router, a link or an interface, or an aggregation of devices such as a subnetwork, or a data center.

  The definition of Abstract Network Element is similar to Network Element defined in [RFC2216] in the sense that they both provide an abstract representation of particular components of a network. However, they have different criteria on how these particular components are selected. Specifically, Network Element requires the components to be potentially capable of exercising QoS control, while Abstract Network Element only requires the components to have an impact on the end-to-end performance.

* ANE Name: An ANE can be constructed either statically in advance or on demand based on the requested information. Thus, different ANEs may only be valid within a particular scope, either ephemeral or persistent. Within each scope, an ANE is uniquely identified by an ANE Name, as defined in Section 6.1. Note that an ALTO client must not assume ANEs in different scopes but with the same ANE Name refer to the same component(s) of the network.

* Path Vector: A Path Vector, or an ANE Path Vector, is a JSON array of ANE Names. It is a generalization of BGP path vector. While standard BGP path vector specifies a sequence of autonomous systems for a destination IP prefix, the Path Vector defined in this extension specifies a sequence of ANEs either for a source PID and a destination PID as in a cost map or for a source endpoint and a destination endpoint as in an endpoint cost map.

* Path Vector resource: A Path Vector resource refers to an ALTO resource which supports the extension defined in this document.

* Path Vector cost type: The Path Vector cost type is a special cost type, which is specified in Section 6.5. When this cost type is present in an IRD entry, it indicates that the information resource is a Path Vector resource. When this cost type is present in a Cost Map or an Endpoint Cost Map, it indicates each cost value must be interpreted as a Path Vector.

* Path Vector request: A Path Vector request refers to the POST message sent to an ALTO Path Vector resource.

* Path Vector response: A Path Vector response refers to the multipart/related message returned by a Path Vector resource.

4. Problem Statement

4.1. Design Requirements

This section gives an illustrative example of how an overlay application can benefit from the Path Vector extension.

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 can not 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 of link eh1 -> sw1 and link sw1 -> sw5 are 150 Mbps, and the bandwidth of the rest links are 100 Mbps.

```
                                 +------+
                                 |      |
                               --+ sw6  +--
                             /   |      |  \
       PID1 +-----+         /    +------+   \          +-----+  PID2
       eh1__|     |_       /                 \     ____|     |__eh2
   1.2.3.4  | sw1 | \   +--|---+         +---|--+ /    | sw2 |  2.3.4.5
            +-----+  \  |      |         |      |/     +-----+
                      \_| sw5  +---------+ sw7  |
       PID3 +-----+   / |      |         |      |\     +-----+  PID4
       eh3__|     |__/  +------+         +------+ \____|     |__eh4
   3.4.5.6  | sw3 |                                    | sw4 |  4.5.6.7
            +-----+                                    +-----+
```

Figure 1: Raw Network Topology

The single-node ALTO topology abstraction of the network is shown in Figure 2.

```
                             +----------------------+
                    {eh1}    |                      |     {eh2}
                    PID1     |                      |     PID2
                      +------+                      +------+
                             |                      |
                             |                      |
                    {eh3}    |                      |     {eh4}
                    PID3     |                      |     PID4
                      +------+                      +------+
                             |                      |
                             +----------------------+
```

Figure 2: Base Single-Node Topology Abstraction

Consider an application overlay (e.g., a large-scale data analytics system) which wants to optimize the total throughput of the traffic among a set of end host <source, destination> pairs, say eh1 -> eh2 and eh1 -> 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 the ALTO server that the bandwidth of eh1 -> eh2 and eh1 -> eh4 are both 100 Mbps. But this information is not enough to determine the optimal total throughput. Consider the following two cases:

* Case 1: If eh1 -> eh2 uses the path eh1 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh2 and eh1 -> eh4 uses path eh1 -> sw1 -> sw5 -> sw7 -> sw4 -> eh4, then the application will obtain 150 Mbps at most.

* Case 2: If eh1 -> eh2 uses the path eh1 -> sw1 -> sw5 -> sw7 -> sw2 -> eh2 and eh1 -> eh4 uses the path eh1 -> sw1 -> sw5 -> sw7 -> sw4 -> eh4, then the application will obtain only 100 Mbps at most.

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

* For eh1 -> eh2, the ALTO server must give more details which is critical for the overlay application to distinguish between Case 1 and Case 2 and to compute the optimal total throughput accordingly.

* The ALTO server must allow the client to distinguish the common ANE shared by eh1 -> eh2 and eh1 -> eh4, e.g., eh1 - sw1 and sw1 - sw5 in Case 1.

* The ALTO server must give details on the properties of the ANEs used by eh1 -> eh2 and eh1 -> eh4, e.g., the available bandwidth between eh1 - sw1, sw1 - sw5, sw5 - sw7, sw5 - sw6, sw6 - sw7, sw7 - sw2, sw7 - sw4, sw2 - eh2, sw4 - eh4 in Case 1.

In general, we can conclude that to support the multiple flow scheduling use case, the ALTO framework must be extended to satisfy the following additional requirements:

AR1: An ALTO server must provide essential information on ANEs on the path of a <source, destination> pair that are critical to the QoE of the overlay application.

AR2: An ALTO server must provide essential information on how the paths of different <source, destination> pairs share a common ANE.

AR3: An ALTO server must provide essential information on the properties associated to the ANEs.

The Path Vector extension defined in this document propose a solution to provide these details.

4.2. Use Cases

While the multiple flow scheduling problem is used to help identify the additional requirements, the Path Vector extension can be applied to a wide range of applications. This section highlights some real use cases that are reported.

4.2.1. Large-scale Data Analytics

One potential use case of the Path Vector extension is for large-scale data analytics such as [SENSE] and [LHC], where data of Gigabytes, Terabytes and even Petabytes are transferred. For these applications, the QoE is usually measured as the job completion time, which is related to the completion time of the slowest data transfer. With the Path Vector extension, an ALTO client can identify bottlenecks inside the network. Therefore, the overlay application can make optimal traffic distribution or resource reservation (i.e., proportional to the size of the transferred data), leading to optimal job completion time and network resource utilization.

4.2.2. Context-aware Data Transfer

It is getting important to know the capabilities of various ANEs between two end hosts, especially in the mobile environment. With the Path Vector extension, an ALTO client may query the "network context" information, i.e., whether the two hosts are connected to the access network through a wireless link or a wire, and the capabilities of the access network. Thus, the client may use different data transfer mechanisms, or even deploy different 5G User Plane Functions (UPF) [I-D.ietf-dmm-5g-uplane-analysis] to optimize the data transfer.

4.2.3. CDN and Service Edge

A growing trend in today’s applications is to bring storage and computation closer to the end user for better QoE, such as Content Delivery Network (CDN), AR/VR, and cloud gaming, as reported in various documents ([I-D.contreras-alto-service-edge], [I-D.huang-alto-mowie-for-network-aware-app], and [I-D.yang-alto-deliver-functions-over-networks]).

With the Path Vector extension, an ALTO server can selectively reveal the CDNs and service edges that reside along the paths between different end hosts, together with their properties such as available Service Level Agreement (SLA) plans. Otherwise, the ALTO client may have to make multiple queries and potentially with the complete list of CDNs and/or service edges. While both approaches offer the same information, making multiple queries introduces larger delay and more overhead on both the ALTO server and the ALTO client.

5. Path Vector Extension: Overview

This section gives a non-normative overview of the Path Vector extension. It is assumed that readers are familiar with both the base protocol [RFC7285] and the Unified Property Map extension [I-D.ietf-alto-unified-props-new].

To satisfies the additional requirements, this extension:

1. introduces Abstract Network Element (ANE) as the abstraction of components in a network whose properties may have an impact on the end-to-end performance of the traffic handled by those component,

2. extends the Cost Map and Endpoint Cost Service to convey the ANEs traversed by the path of a <source, destination> pair as Path Vectors,

3. uses the Unified Property Map to convey the association between the ANEs and their properties.

Thus, an ALTO client can learn about the ANEs that are critical to the QoE of a <source, destination> pair by investigating the corresponding Path Vector value (AR1), identify common ANEs if an ANE appears in the Path Vectors of multiple <source, destination> pairs (AR2), and retrieve the properties of the ANEs by searching the Unified Property Map (AR3).

5.1. Abstract Network Element

This extension introduces Abstract Network Element (ANE) as an indirect and network-agnostic way to specify a component or an aggregation of components of a network whose properties have an impact on the end-to-end performance for traffic between a source and a destination.

When an ANE is defined by the ALTO server, it MUST be assigned an identifier, i.e., string of type ANEName as specified in Section 6.1, and a set of associated properties.

5.1.1. ANE Domain

In this extension, the associations between ANE and the properties are conveyed in a Unified Property Map. Thus, they must follow the mechanisms specified in the [I-D.ietf-alto-unified-props-new].

Specifically, this document defines a new entity domain called "ane" as specified in Section 6.2 and defines two initial properties for the "ane" domain.

5.1.2. Ephemeral ANE and Persistent ANE

For different requests, there can be different ways of grouping components of a network and assigning ANEs. For example, an ALTO server may define an ANE for each aggregated bottleneck link between the sources and destinations specified in the request. As the aggregated bottleneck links vary for different combinations of sources and destinations, the ANEs are ephemeral and are no longer valid after the request completes. Thus, the scope of ephemeral ANEs are limited to the corresponding Path Vector response.

While ephemeral ANEs returned by a Path Vector response do not exist beyond that response, some of them may represent entities that are persistent and defined in a standalone Property Map. Indeed, it may be useful for clients to occasionally query properties on persistent entities, without caring about the path that traverses them. Persistent entities have a persistent ID that is registered in a Property Map, together with their properties.

5.1.3. Property Filtering

Resource-constrained ALTO clients may benefit from the filtering of Path Vector query results at the ALTO server, as an ALTO client may only require a subset of the available properties.

Specifically, the available properties for a given resource are announced in the Information Resource Directory as a new capability called "ane-property-names". The selected properties are specified in a filter called "ane-property-names" in the request body, and the response MUST include and only include the selected properties for the ANEs in the response.

The "ane-property-names" capability for Cost Map and for Endpoint Cost Service are specified in Section 7.1.4 and Section 7.2.4 respectively. The "ane-property-names" filter for Cost Map and Endpoint Cost Service are specified in Section 7.1.3 and Section 7.2.3 accordingly.

5.2. Path Vector Cost Type

For an ALTO client to correctly interpret the Path Vector, this extension specifies a new cost type called the Path Vector cost type, which must be included both in the Information Resource Directory and the ALTO Cost Map or Endpoint Cost Map so that an ALTO client can correctly interpret the cost values.

The Path Vector cost type must convey both the interpretation and semantics in the "cost-mode" and "cost-metric" respectively. Unfortunately, a single "cost-mode" value cannot fully specify the interpretation of a Path Vector, which is a compound data type. For example, in programming languages such as Java, a Path Vector will have the type of JSONArray[ANEName].

Instead of extending the "type system" of ALTO, this document takes a simple and backward compatible approach. Specifically, the "cost-mode" of the Path Vector cost type is "array", which indicates the value is a JSON array. Then, an ALTO client must check the value of the "cost-metric". If the value is "ane-path", meaning the JSON array should be further interpreted as a path of ANENames.

The Path Vector cost type is specified in Section 6.5.

5.3. Multipart Path Vector Response

For a basic ALTO information resource, a response contains only one type of ALTO resources, e.g., Network Map, Cost Map, or Property Map. Thus, only one round of communication is required: An ALTO client sends a request to an ALTO server, and the ALTO server returns a response, as shown in Figure 3.

```
ALTO client                              ALTO server
     |-------------- Request ---------------->|
     |<------------- Response ----------------|
```

Figure 3: A Typical ALTO Request and Response

The Path Vector extension, on the other hand, involves two types of information resources: Path Vectors conveyed in a Cost Map or an Endpoint Cost Map, and ANE properties conveyed in a Unified Property Map. Instead of two consecutive message exchanges, the Path Vector extension enforces one round of communication. Specifically, the ALTO client MUST include the source and destination pairs and the requested ANE properties in a single request, and the ALTO server MUST return a single response containing both the Path Vectors and properties associated with the ANEs in the Path Vectors, as shown in Figure 4. Since the two parts are bundled together in one response message, their orders are interchangeable. See Section 7.1.6 and Section 7.2.6 for details.

```
ALTO client                              ALTO server
     |------------- PV Request -------------->|
     |<----- PV Response (Cost Map Part) -----|
     |<--- PV Response (Property Map Part) ---|
```

Figure 4: The Path Vector Extension Request and Response

This design is based on the following considerations:

1. Since ANEs may be constructed on demand, and potentially based on the requested properties (See Section 5.1 for more details). If sources and destinations are not in the same request as the properties, an ALTO server either cannot construct ANEs on-demand, or must wait until both requests are received.

2. As ANEs may be constructed on demand, mappings of each ANE to its underlying network devices and resources can be specific to the request. In order to respond to the Property Map request correctly, an ALTO server must store the mapping of each Path Vector request until the client fully retrieves the property information. The "stateful" behavior may substantially harm the server scalability and potentially lead to Denial-of-Service attacks.

One approach to realize the one-round communication is to define a new media type to contain both objects, but this violates modular design. This document follows the standard-conforming usage of "multipart/related" media type defined in [RFC2387] to elegantly combine the objects. Path Vectors are encoded as a Cost Map or an Endpoint Cost Map, and the Property Map is encoded as a Unified Propert Map. They are encapsulated as parts of a multipart message. The modular composition allows ALTO servers and clients to reuse the data models of the existing information resources. Specifically, this document addresses the following practical issues using "multipart/related".

5.3.1. Identifying the Media Type of the Root Object

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-endpointcost+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.

To address this issue, this document uses the "type" parameter to indicate the root object of a multipart/related message. For a Cost Map resource, the "media-type" in the IRD entry must be "multipart/related" with the parameter "type=application/alto-costmap+json"; for an Endpoint Cost Service, the parameter must be "type=application/alto-endpointcost+json".

5.3.2. References to Part Messages

The ALTO SSE extension (see [I-D.ietf-alto-incr-update-sse]) uses "client-id" to demultiplex push updates. However, "client-id" is provided for each request, which introduces ambiguity when applying SSE to a Path Vector resource.

To address this issue, an ALTO server must assign a unique identifier to each part of the "multipart/related" response message. This identifier, referred to as a Part Resource ID (See Section 6.6 for details), must be present in the part message's "Resource-Id" header. The MIME part header must also contain the "Content-Type" header, whose value is the media type of the part (e.g., "application/alto-costmap+json", "application/alto-endpointcost+json", or "application/alto-propmap+json").

If an ALTO server provides incremental updates for this Path Vector resource, it must generate incremental updates for each part separately. The client-id must have the following format:

```
pv-client-id '.' part-resource-id
```

where pv-client-id is the client-id assigned to the Path Vector request, and part-resource-id is the "Resource-Id" header value of the part. The media-type must match the "Content-Type" of the part.

The same problem applies to the part messages as well. The two parts must contain a version tag, which SHOULD contain a unique Resource ID. This document requires the resource-id in a Version Tag to have the following format:

```
pv-resource-id '.' part-resource-id
```

where pv-resource-id is the resource ID of the Path Vector resource in the IRD entry, and the part-resource-id has the same value as the "Resource-Id" header of the part.

5.3.3. Order and Completeness of Part Messages

According to [RFC2387], the Path Vector part, whose media type is the same as the "type" parameter of the multipart response message, is the root object. Thus, it is the element the application processes first. Even though the "start" parameter allows it to be placed anywhere in the part sequence, it is RECOMMENDED that the parts arrive in the same order as they are processed, i.e., the Path Vector part is always put as the first part, followed by the property map part. It is also RECOMMENDED that when doing so, an ALTO server SHOULD NOT set the "start" parameter, which implies the first part is the root object.

A complete and valid response MUST include both the Path Vector part and the Property Map part in the multipart message.

6. Specification: Basic Data Types

6.1. ANE Name

An ANE Name is encoded as a JSON string with the same format as that of the type PIDName (Section 10.1 of [RFC7285]).

The type ANEName is used in this document to indicate a string of this format.

6.2. ANE Domain

The ANE domain associates property values with the Abstract Network Elements in a Property Map. Accordingly, the ANE domain always depends on a Property Map.

6.2.1. Entity Domain Type

ane

6.2.2. Domain-Specific Entity Identifier

The entity identifiers are the ANE Names in the associated Property Map.

6.2.3. Hierarchy and Inheritance

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

6.2.4. Media Type of Defining Resource

When resource specific domains are defined with entities of domain type "ane", the defining resource for entity domain type "pid" MUST be a Property Map. The media type of defining resources for the "ane" domain is:

application/alto-propmap+json

Specifically, for ephemeral ANEs that appear in a Path Vector response, their entity domain names MUST be ".ane" and the defining resource of these ANEs is the Property Map part of the multipart response. Meanwhile, for persistent ANEs whose entity domain name has the format of "PROPMAP.ane" where PROPMAP is the name of a Property Map resource, PROPMAP is the defining resource of these ANEs. Persistent entities are "persistent" because standalone queries can be made by an ALTO client to their defining resources when the connection to the Path Vector service is closed.

For example, the defining resource of ".ane:NET1" is the Property Map part that contains this identifier, i.e., the ANE entity ".ane:NET1" is self-defined. The defining resource of "dc-props.ane:DC1" is the Property Map with the resource ID "dc-props".

6.3. ANE Property Name

An ANE Property Name is encoded as a JSON string with the same format as that of Entity Property Name (Section 5.2.2 of [I-D.ietf-alto-unified-props-new]).

6.4. Initial ANE Property Types

In this document, two initial ANE property types are specified, "max-reservable-bandwidth" and "persistent-entity-id".

Note that the two property types defined in this document do not depend on any information resource, so their ResourceID part must be empty.

```
                                        ----- L1
                                       /
           PID1   +---------------+ 10 Gbps +----------+    PID3
    1.2.3.0/24+---+ +-----------+ +---------+          +---+3.4.5.0/24
                  | |   MEC1    | |         |          |
                  | +-----------+ |   +-----+          |
           PID2   |               |   |     +----------+
    2.3.4.0/24+---+               |   |         NET3
                  |               |   | 15 Gbps
                  |               |   |        \
                  +---------------+   |         -------- L2
                        NET1          |
                               +---------------+
                               | +-----------+ |   PID4
                               | |   MEC2    | +---+4.5.6.0/24
                               | +-----------+ |
                               +---------------+
                                     NET2
```

Figure 5: Examples of ANE Properties

In this document, Figure 5 is used to illustrate the use of the two initial ANE property types. There are 3 sub-networks (NET1, NET2 and NET3) and two interconnection links (L1 and L2). It is assumed that each sub-network has sufficiently large bandwidth to be reserved.

6.4.1. New ANE Property Type: Maximum Reservable Bandwidth

Identifier: "max-reservable-bandwidth"

Intended Semantics: The maximum reservable bandwidth property stands for the maximum bandwidth that can be reserved for all the traffic that traverses an ANE. The value MUST be encoded as a non-negative numerical cost value as defined in Section 6.1.2.1 of [RFC7285] and the unit is bit per second. If this property is requested but not present in an ANE, it MUST be interpreted as that the ANE does not support bandwidth reservation.

Security Considerations: ALTO entity properties expose information to ALTO clients. ALTO service providers should be made aware of the security ramifications related to the exposure of an entity property.

To illustrate the use of "max-reservable-bandwidth", consider the network in Figure 5. An ALTO server can create an ANE for each interconnection link, where the initial value for "max-reservable-bandwidth" is the link capacity.

6.4.2. New ANE Property Type: Persistent Entity ID

Identifier: "persistent-entity-id"

Intended Semantics: The persistent entity ID property is the entity identifier of the persistent ANE which an ephemeral ANE presents (See Section 5.1.2 for details). The value of this property is encoded with the format defined in Section 5.1.3 of [I-D.ietf-alto-unified-props-new]. In this format, the entity ID combines:

* a defining information resource for the ANE on which a "persistent-entity-id" is queried, which is the property map defining the ANE as a persistent entity, together with the properties

* the persistent name of the ANE in this property map

With this format, the client has all the needed information for further standalone query properties on the persistent ANE.

Security Considerations: ALTO entity properties expose information to ALTO clients. ALTO service providers should be made aware of the security ramifications related to the exposure of an entity property.

To illustrate the use of "persistent-entity-id", consider the network in Figure 5. Assume the ALTO server has a Property Map resource called "mec-props" that defines persistent ANEs "MEC1" and "MEC2" that represent the corresponding mobile edge computing (MEC) clusters. Since MEC1 is associated with NET1, the "persistent-entity-id" of the ephemeral ANE ".ane:NET1" is the persistent entity id "mec-props.ane:MEC1".

6.5. Path Vector Cost Type

This document defines a new cost type, which is referred to as the "Path Vector" cost type. An ALTO server MUST offer this cost type if it supports the Path Vector extension.

6.5.1. Cost Metric: ane-path

The cost metric "ane-path" indicates the value of such a cost type conveys an array of ANE names, where each ANE name uniquely represents an ANE traversed by traffic from a source to a destination.

An ALTO client MUST interpret the Path Vector as if the traffic between a source and a destination logically traverses the ANEs in the same order as they appear in the Path Vector. However, under certain scenarios where the traversal order is not crucial, an ALTO server implementation may choose to not follow strictly the physical traversal order and may even obfuscate the order intentionally, for security and performance considerations. For example, in the multi-flow bandwidth reservation use case as introduced in Section 4, only the available bandwidth of the shared bottleneck link is crucial, and the ALTO server may change the order of links appearing in the Path Vector response.

6.5.2. Cost Mode: array

The cost mode "array" indicates that every cost value in a Cost Map or an Endpoint Cost Map MUST be interpreted as a JSON array object.

Note that this cost mode only requires the cost value to be a JSON array of JSONValue. However, an ALTO server that enables this extension MUST return a JSON array of ANEName (Section 6.1) when the cost metric is "ane-path".

6.6. Part Resource ID

A Part Resource ID is encoded as a JSON string with the same format as that of the type ResourceID (Section 10.2 of [RFC7285]).

Even though the client-id assigned to a Path Vector request and the Part Resource ID MAY contain up to 64 characters by their own definition, their concatenation (see Section 5.3.2) MUST also conform to the same length constraint. The same requirement applies to the resource ID of the Path Vector resource, too. Thus, it is RECOMMENDED to limit the length of resource ID and client ID related to a Path Vector resource to 31 characters.

7. Specification: Service Extensions

7.1. 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.

7.1.1. Media Type

The media type of the multipart filtered cost map resource is "multipart/related;type=application/alto-costmap+json".

7.1.2. HTTP Method

The multipart filtered cost map is requested using the HTTP POST method.

7.1.3. Accept Input Parameters

The input parameters of the multipart filtered cost map are supplied in the body of an HTTP POST request. This document extends the input parameters to a filtered cost map with a data format indicated by the media type "application/alto-costmapfilter+json", which is a JSON object of type PVReqFilteredCostMap, where:

```
object {
  [EntityPropertyName ane-property-names<0..*>;]
} PVReqFilteredCostMap : ReqFilteredCostMap;
```

with fields:

ane-property-names: A list of properties that are associated with the ANEs. Each property in this list MUST match one of the supported ANE properties indicated in the resource's "ane-property-names" capability. If the field is NOT present, it MUST be interpreted as an empty list, indicating that the ALTO server MUST NOT return any property in the Unified Property part.

Example: Consider the network in Figure 1. If an ALTO client wants to query the "max-reservable-bandwidth" between PID1 and PID2, it can submit the following request.

```
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": "array",
    "cost-metric": "ane-path"
  },
  "pids": {
    "srcs": [ "PID1" ],
    "dsts": [ "PID2" ]
  },
  "ane-property-names": [ "max-reservable-bandwidth" ]
}
```

7.1.4. Capabilities

The multipart filtered cost map resource extends the capabilities defined in Section 11.3.2.4 of [RFC7285]. The capabilities are defined by a JSON object of type PVFilteredCostMapCapabilities:

```
object {
  [EntityPropertyName ane-property-names<0..*>;]
} PVFilteredCostMapCapabilities : FilteredCostMapCapabilities;
```

with fields:

cost-type-names: The "cost-type-names" field MUST only include the Path Vector cost type, unless explicitly documented by a future extension. This also implies that the Path Vector cost type MUST be defined in the "cost-types" of the Information Resource Directory's "meta" field.

cost-constraints: If the "cost-type-names" field includes the Path Vector cost type, "cost-constraints" field MUST be "false" or not present unless specifically instructed by a future document.

testable-cost-type-names: If the "cost-type-names" field includes the Path Vector cost type, the Path Vector cost type MUST NOT be included in the "testable-cost-type-names" field unless specifically instructed by a future document.

ane-property-names: Defines a list of ANE properties that can be returned. If the field is NOT present, it MUST be interpreted as an empty list, indicating the ALTO server cannot provide any ANE property.

7.1.5. Uses

This member MUST include the resource ID of the network map based on which the PIDs are defined. If this resource supports "persistent-entity-id", it MUST also include the defining resources of persistent ANEs that may appear in the response.

7.1.6. 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 "Content-Type" header of the response MUST be "multipart/related" as defined by [RFC2387] with the following parameters:

type: The type parameter MUST be "application/alto-costmap+json". Note that [RFC2387] permits both parameters with and without the double quotes.

start: The start parameter is as defined in [RFC2387]. If present, it MUST have the same value as the "Resource-Id" header of the Path Vector part.

boundary: The boundary parameter is as defined in [RFC2387].

The body of the response MUST consist of two parts:

* The Path Vector part MUST include "Resource-Id" and "Content-Type" in its header. The value of "Resource-Id" MUST has the format of a Part Resource ID. The "Content-Type" MUST be "application/alto-costmap+json".

  The body of the Path Vector 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 follow the format in Section 5.3.2. 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 filtered cost map resource in IRD.

* The Unified Property Map part MUST also include "Resource-Id" and "Content-Type" in its header. The value of "Resource-Id" has the format of a Part Resource ID. The "Content-Type" MUST be "application/alto-propmap+json".

  The body of the Unified Property Map 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 of VersionTag objects as defined by Section 10.3 of [RFC7285]. The "vtag" of the Path Vector part MUST be included in the "dependent-vtags". If "persistent-entity-id" is requested, the version tags of the dependent resources that MAY expose the entities in the response MUST also be included. The PropertyMapData has one member for each ANEName that appears in the Path Vector part, which is an entity identifier belonging to the self-defined entity domain as defined in Section 5.1.2.3 of [I-D.ietf-alto-unified-props-new]. The EntityProps has one member for each property requested by an ALTO client if applicable.

If the "start" parameter is not present, the Path Vector part MUST be the first part in the multipart response. If any part is NOT present, the client MUST discard the received information and send another request if necessary.

Example: Consider the network in Figure 1. The response of the example request in Section 7.1.3 is as follows, where "ANE1" represents the aggregation of all the switches in the network.

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

--example-1
Resource-Id: costmap
Content-Type: application/alto-costmap+json

{
  "meta": {
    "vtag": {
      "resource-id": "filtered-cost-map-pv.costmap",
      "tag": "d827f484cb66ce6df6b5077cb8562b0a"
    },
    "dependent-vtags": [
      {
        "resource-id": "my-default-networkmap",
        "tag": "75ed013b3cb58f896e839582504f6228"
      }
    ],
    "cost-type": { "cost-mode": "array", "cost-metric": "ane-path" }
  },
  "cost-map": {
    "PID1": { "PID2": ["ANE1"] }
  }
}
--example-1
Resource-Id: propmap
Content-Type: application/alto-propmap+json

{
  "meta": {
    "dependent-vtags": [
      {
        "resource-id": "filtered-cost-map-pv.costmap",
        "tag": "d827f484cb66ce6df6b5077cb8562b0a"
      }
    ]
  },
  "property-map": {
    ".ane:ANE1": { "max-reservable-bandwidth": 100000000 }
  }
}
```

7.2. 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.

7.2.1. Media Type

The media type of the multipart endpoint cost resource is "multipart/related;type=application/alto-endpointcost+json".

7.2.2. HTTP Method

The multipart endpoint cost resource is requested using the HTTP POST method.

7.2.3. Accept Input Parameters

The input parameters of the multipart endpoint cost resource are supplied in the body of an HTTP POST request. This document extends the input parameters to an endpoint cost map with a data format indicated by the media type "application/alto-endpointcostparams+json", which is a JSON object of type PVEndpointCostParams, where

```
object {
  [EntityPropertyName ane-property-names<0..*>;]
} PVReqEndpointcost : ReqEndpointcost;
```

with fields:

ane-property-names: This document defines the "ane-property-names" in PVReqEndpointcost as the same as in PVReqFilteredCostMap. See Section 7.1.3.

Example: Consider the network in Figure 1. If an ALTO client wants to query the "max-reservable-bandwidth" between eh1 and eh2, it can submit the following request.

```
POST /ecs/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": "array",
    "cost-metric": "ane-path"
  },
  "endpoints": {
    "srcs": [ "ipv4:1.2.3.4" ],
    "dsts": [ "ipv4:2.3.4.5" ]
  },
  "ane-property-names": [ "max-reservable-bandwidth" ]
}
```

7.2.4. Capabilities

The capabilities of the multipart endpoint cost resource are defined by a JSON object of type PVEndpointcostCapabilities, which is defined as the same as PVFilteredCostMapCapabilities. See Section 7.1.4.

7.2.5. Uses

If this resource supports "persistent-entity-id", it MUST also include the defining resources of persistent ANEs that may appear in the response.

7.2.6. 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 "Content-Type" header of the response MUST be "multipart/related" as defined by [RFC7285] with the following parameters:

type: The type parameter MUST be "application/alto-endpointcost+json".

start: The start parameter is as defined in Section 7.1.6.

boundary: The boundary parameter is as defined in [RFC2387].

The body MUST consist of two parts:

* The Path Vector part MUST include "Resource-Id" and "Content-Type" in its header. The value of "Resource-Id" MUST has the format of a Part Resource ID. The "Content-Type" MUST be "application/alto-endpointcost+json".

  The body of the Path Vector 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 follow the format in Section 5.3.2.

* The Unified Property Map part MUST also include "Resource-Id" and "Content-Type" in its header. The value of "Resource-Id" MUST has the format of a Part Resource ID. The "Content-Type" MUST be "application/alto-propmap+json".

  The body of the Unified Property Map 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 of VersionTag objects as defined by Section 10.3 of [RFC7285]. The "vtag" of the Path Vector part MUST be included in the "dependent-vtags". If "persistent-entity-id" is requested, the version tags of the dependent resources that MAY expose the entities in the response MUST also be included. The PropertyMapData has one member for each ANEName that appears in the Path Vector part, which is an entity identifier belonging to the self-defined entity domain as defined in Section 5.1.2.3 of [I-D.ietf-alto-unified-props-new]. The EntityProps has one member for each property requested by the ALTO client if applicable.

If the "start" parameter is not present, the Path Vector part MUST be the first part in the multipart response. If any part is NOT present, the client MUST discard the received information and send another request if necessary.

Example: Consider the network in Figure 1. The response of the example request in Section 7.2.3 is as follows.

```
HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=example-1;
              type=application/alto-endpointcost+json

--example-1
Resource-Id: ecs
Content-Type: application/alto-endpointcost+json

{
  "meta": {
    "vtag": {
      "resource-id": "ecs-pv.costmap",
      "tag": "d827f484cb66ce6df6b5077cb8562b0a"
    },
    "dependent-vtags": [
      {
        "resource-id": "my-default-networkmap",
        "tag": "75ed013b3cb58f896e839582504f6228"
      }
    ],
    "cost-type": { "cost-mode": "array", "cost-metric": "ane-path" }
  },
  "cost-map": {
    "ipv4:1.2.3.4": { "ipv4:2.3.4.5": ["ANE1"] }
  }
}
--example-1
Resource-Id: propmap
Content-Type: application/alto-propmap+json

{
  "meta": {
    "dependent-vtags": [
      {
        "resource-id": "ecs-pv.costmap",
        "tag": "d827f484cb66ce6df6b5077cb8562b0a"
      }
    ]
  },
  "property-map": {
    ".ane:ANE1": { "max-reservable-bandwidth": 100000000 }
  }
}
```

8. Examples

This section lists some examples of Path Vector queries and the corresponding responses. Some long lines are truncated for better readability.

8.1. Example: Information Resource Directory

To give a comprehensive example of the Path Vector extension, we consider the network in Figure 5. The example ALTO server provides the following information resources:

* "my-default-networkmap": A Network Map resource which contains the PIDs in the network.

* "filtered-cost-map-pv": A Multipart Filtered Cost Map resource for Path Vector, which exposes the "max-reservable-bandwidth" property for the PIDs in "my-default-networkmap".

* "ane-props": A filtered Unified Property resource that exposes the information for persistent ANEs in the network.

* "endpoint-cost-pv": A Multipart Endpoint Cost Service for Path Vector, which exposes the "max-reservable-bandwidth" and the "persistent-entity-id" properties.

* "update-pv": An Update Stream service, which provides the incremental update service for the "endpoint-cost-pv" service.

Below is the Information Resource Directory of the example ALTO server. To enable the Path Vector extension, the "path-vector" cost type (Section 6.5) is defined in the "cost-types" of the "meta" field, and is included in the "cost-type-names" of resources "filetered-cost-map-pv" and "endpoint-cost-pv".


```
{
  "meta": {
    "cost-types": {
      "path-vector": {
        "cost-mode": "array",
        "cost-metric": "ane-path"
      }
    }
  },
  "resources": {
    "my-default-networkmap": {
      "uri" : "https://alto.example.com/networkmap",
      "media-type" : "application/alto-networkmap+json"
```

```
        },
        "filtered-cost-map-pv": {
          "uri": "https://alto.example.com/costmap/pv",
          "media-type": "multipart/related;
                         type=application/alto-costmap+json",
          "accepts": "application/alto-costmapfilter+json",
          "capabilities": {
            "cost-type-names": [ "path-vector" ],
            "ane-property-names": [ "max-reservable-bandwidth" ]
          },
          "uses": [ "my-default-networkmap" ]
        },
        "ane-props": {
          "uri": "https://alto.example.com/ane-props",
          "media-type": "application/alto-propmap+json",
          "accepts": "application/alto-propmapparams+json",
          "capabilities": {
            "mappings": {
              ".ane": [ "cpu" ]
            }
          }
        },
        "endpoint-cost-pv": {
          "uri": "https://alto.exmaple.com/endpointcost/pv",
          "media-type": "multipart/related;
                         type=application/alto-endpointcost+json",
          "accepts": "application/alto-endpointcostparams+json",
          "capabilities": {
            "cost-type-names": [ "path-vector" ],
            "ane-property-names": [
              "max-reservable-bandwidth", "persistent-entity-id"
            ]
          },
          "uses": [ "ane-props" ]
        },
        "update-pv": {
          "uri": "https://alto.example.com/updates/pv",
          "media-type": "text/event-stream",
          "uses": [ "endpoint-cost-pv" ],
          "accepts": "application/alto-updatestreamparams+json",
          "capabilities": {
            "support-stream-control": true
          }
        }
      }
    }
```

8.2. Example: Multipart Filtered Cost Map

The following examples demonstrate the request to the "filtered-cost-map-pv" resource and the corresponding response.

The request uses the "path-vector" cost type in the "cost-type" field. The "ane-property-names" field is missing, indicating that the client only requests for the Path Vector but not the ANE properties.

The response consists of two parts. The first part returns the array of ANEName for each source and destination pair. There are two ANEs, where "L1" represents the interconnection link L1, and "L2" represents the interconnection link L2.

The second part returns an empty Property Map. Note that the ANE entries are omitted since they have no properties (See Section 3.1 of [I-D.ietf-alto-unified-props-new]).

```
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": "array",
    "cost-metric": "ane-path"
  },
  "pids": {
    "srcs": [ "PID1" ],
    "dsts": [ "PID3", "PID4" ]
  }
}

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

--example-1
Resource-Id: costmap
Content-Type: application/alto-costmap+json

{
  "meta": {
```

```
        "vtag": {
          "resource-id": "filtered-cost-map-pv.costmap",
          "tag": "d827f484cb66ce6df6b5077cb8562b0a"
        },
        "dependent-vtags": [
          {
            "resource-id": "my-default-networkmap",
            "tag": "75ed013b3cb58f896e839582504f6228"
          }
        ],
        "cost-type": {
          "cost-mode": "array",
          "cost-metric": "ane-path"
        }
      },
      "cost-map": {
        "PID1": {
          "PID3": [ "L1" ],
          "PID4": [ "L1", "L2" ]
        }
      }
    }
    --example-1
    Resource-Id: propmap
    Content-Type: application/alto-propmap+json

    {
      "meta": {
        "dependent-vtags": [
          {
            "resource-id": "filtered-cost-map-pv.costmap",
            "tag": "d827f484cb66ce6df6b5077cb8562b0a"
          }
        ]
      },
      "property-map": {
      }
    }
```


8.3. Example: Multipart Endpoint Cost Resource

The following examples demonstrate the request to the "endpoint-cost-pv" resource and the corresponding response.

The request uses the path vector cost type in the "cost-type" field, and queries the Maximum Reservable Bandwidth ANE property and the Persistent Entity property.

The response consists of two parts. The first part returns the array of ANEName for each valid source and destination pair, where "NET1" represent sub-network NET1, and "AGGR" is the aggregation of L1 and NET3.

The second part returns the requested properties of ANEs. Since NET1 has sufficient bandwidth, it sets the "max-reservable-bandwidth" to a sufficiently large number. It also represents a persistent ANE defined in the "ane-props" resource, identified by "ane-props.ane:datacenter1". The aggregated "max-reservable-bandwidth" of ane:AGGR is constrained by the link capacity of L1. The "persistent-entity-id" property is omitted as both L1 and NET3 do not represent any persistent entity.

```
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": "array",
    "cost-metric": "ane-path"
  },
  "endpoints": {
    "srcs": [ "ipv4:1.2.3.4", "ipv4:2.3.4.5" ],
    "dsts": [ "ipv4:3.4.5.6" ]
  },
  "ane-property-names": [
    "max-reservable-bandwidth",
    "persistent-entity-id"
  ]
}

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

--example-2
Resource-Id: ecs
Content-Type: application/alto-endpointcost+json

{
  "meta": {
```

```
      "vtags": {
        "resource-id": "endpoint-cost-pv.ecs",
        "tag": "bb6bb72eafe8f9bdc4f335c7ed3b10822a391cef"
      },
      "cost-type": {
        "cost-mode": "array",
        "cost-metric": "ane-path"
      }
    },
    "endpoint-cost-map": {
      "ipv4:1.2.3.4": {
        "ipv4:3.4.5.6":   [ "NET1", "AGGR" ]
      },
      "ipv4:2.3.4.5": {
        "ipv4:3.4.5.6":   [ "NET1", "AGGR" ]
      }
    }
  }
  --example-2
  Resource-Id: propmap
  Content-Type: application/alto-propmap+json

  {
    "meta": {
      "dependent-vtags": [
        {
          "resource-id": "endpoint-cost-pv.ecs",
          "tag": "bb6bb72eafe8f9bdc4f335c7ed3b10822a391cef"
        },
        {
          "resource-id": "ane-props",
          "tag": "bf3c8c1819d2421c9a95a9d02af557a3"
        }
      ]
    },
    "property-map": {
      ".ane:NET1": {
        "max-reservable-bandwidth": 50000000000,
        "persistent-entity-id": "ane-props.ane:datacenter1",
      },
      ".ane:AGGR": {
        "max-reservable-bandwidth": 10000000000
      }
    }
  }
```


After the client obtains "ane-props.ane:datacenter1", it can query the "ane-props" resource to get the properties of the persistent ANE.

8.4. Example: Incremental Updates

In this example, an ALTO client subscribes to 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-side process defined in [I-D.ietf-alto-incr-update-sse], the ALTO server will send the "control-uri" first using Server-Sent Event (SSE), followed by 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": "https://alto.example.com/updates/streams/123"}

event: multipart/related;boundary=example-3;
       type=application/alto-endpointcost+json,ecspvsub1
data: --example-3
data: Resource-ID: ecsmap
data: Content-Type: application/alto-endpointcost+json
data:
data: <endpoint-cost-map-entry>
data: --example-3
data: Resource-ID: propmap
data: Content-Type: application/alto-propmap+json
data:
data: <property-map-entry>
data: --example-3--
```

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

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

event: application/merge-patch+json, ecspvsub1.propmap
data: <Merge patch for property-map-update>
```

9. Compatibility

9.1. Compatibility with Legacy ALTO Clients/Servers

The multipart filtered cost map resource and the multipart endpoint cost resource has no backward compatibility issue with legacy 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.

9.2. Compatibility with Multi-Cost Extension

This document does not specify how to integrate the Path Vector cost type with the multi-cost extension [RFC8189]. While it is not RECOMMENDED to put the Path Vector cost type with other cost types in a single query, there is no compatibility issue.

9.3. Compatibility with Incremental Update

The extension specified in this document is NOT compatible with the original incremental update extension [I-D.ietf-alto-incr-update-sse]. A legacy ALTO client CANNOT recognize the compound client-id, and a legacy ALTO server MAY use the same client-id for updates of both parts.

ALTO clients and servers MUST follow the specifications given in this document to support incremental updates for a Path Vector resource.

9.4. Compatibility with Cost Calendar

The extension specified in this document is compatible with the Cost Calendar extension [I-D.ietf-alto-cost-calendar]. When used together with the Cost Calendar extension, the cost value between a source and a destination is an array of path vectors, where the k-th path vector refers to the abstract network paths traversed in the k-th time interval by traffic from the source to the destination.

When used with time-varying properties, e.g., maximum reservable bandwidth (maxresbw), a property of a single ANE may also have different values in different time intervals. In this case, if such an ANE has different property values in two time intervals, it MUST be treated as two different ANEs, i.e., with different entity identifiers. However, if it has the same property values in two time intervals, it MAY use the same identifier.

This rule allows the Path Vector extension to represent both changes of ANEs and changes of the ANEs' properties in a uniform way. The Path Vector part is calendared in a compatible way, and the Property Map part is not affected by the calendar extension.

The two extensions combined together can provide the historical network correlation information for a set of source and destination pairs. A network broker or client may use this information to derive other resource requirements such as Time-Block-Maximum Bandwidth, Bandwidth-Sliding-Window, and Time-Bandwidth-Product (TBP) (See [SENSE] for details).

10. General Discussions

10.1. 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 syntax can only be used to test scalar cost types, and cannot easily express constraints on complex cost types, e.g., the Path Vector cost type defined in this document.

In practice, developing a language for general-purpose boolean tests can be complex and is likely to be a duplicated work. Thus, it is worth looking into the direction of integrating existing well-developed query languages, e.g., XQuery and JSONiq, or their subset with ALTO.

Filtering the Path Vector results or developing a more sophisticated filtering mechanism is beyond the scope of this document.

10.2. 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.

11. 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 fine-grained internal network structures than the base protocol, 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 the requested properties are still accurate, for example, by using minimal feasible region compression algorithms [TON2019] or obfuscation protocols [SC2018][JSAC2019].

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, for example, if an ALTO server implementation dynamically computes the Path Vectors for each requests. 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. Also, an ALTO server may consider using optimizations such as precomputation-and-projection mechanisms [JSAC2019].

12. IANA Considerations

12.1. ALTO Entity Domain Type Registry

This document registers a new entry to the ALTO Domain Entity Type Registry, as instructed by Section 12.2 of [I-D.ietf-alto-unified-props-new]. The new entry is as shown below in Table 1.

Identifier	Entity Address Encoding	Hierarchy & Inheritance
ane	See Section 6.2.2	None

Table 1: ALTO Entity Domain Type Registry

Identifier: See Section 6.2.1.

Entity Identifier Encoding: See Section 6.2.2.

Hierarchy: None

Inheritance: None

Media Type of Defining Resource: See Section 6.2.4.

Security Considerations: In some usage scenarios, ANE addresses 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 ANEs will be made aware of how (or if) the addressing scheme relates to private information and network proximity, in further iterations of this document.

12.2. ALTO Entity Property Type Registry

Two initial entries are registered to the ALTO Domain "ane" in the "ALTO Entity Property Type Registry", as instructed by Section 12.3 of [I-D.ietf-alto-unified-props-new]. The two new entries are shown below in Table 2.

Identifier	Intended Semantics
max-reservable-bandwidth	See Section 6.4.1
persistent-entity-id	See Section 6.4.2

Table 2: Initial Entries for ane Domain in the ALTO Entity Property Types Registry

13. Acknowledgments

The authors would like to thank discussions with Andreas Voellmy, Erran Li, Haibin Song, 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.

14. References

14.1. Normative References

[I-D.ietf-alto-cost-calendar]
Randriamasy, S., Yang, Y., WU, Q., Lingli, D., and N. Schwan, "Application-Layer Traffic Optimization (ALTO) Cost Calendar", Work in Progress, Internet-Draft, draft-ietf-alto-cost-calendar-21, 17 March 2020, <http://www.ietf.org/internet-drafts/draft-ietf-alto-cost-calendar-21.txt>.

[I-D.ietf-alto-incr-update-sse]
Roome, W. and Y. Yang, "ALTO Incremental Updates Using Server-Sent Events (SSE)", Work in Progress, Internet-Draft, draft-ietf-alto-incr-update-sse-22, 20 March 2020, <http://www.ietf.org/internet-drafts/draft-ietf-alto-incr-update-sse-22.txt>.

[I-D.ietf-alto-unified-props-new]
Roome, W., Randriamasy, S., Yang, Y., Zhang, J., and K. Gao, "Unified properties for the ALTO protocol", Work in Progress, Internet-Draft, draft-ietf-alto-unified-props-new-14, 17 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-alto-unified-props-new-14.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2216] Shenker, S. and J. Wroclawski, "Network Element Service Specification Template", RFC 2216, DOI 10.17487/RFC2216, September 1997, <https://www.rfc-editor.org/info/rfc2216>.

[RFC2387] Levinson, E., "The MIME Multipart/Related Content-type", RFC 2387, DOI 10.17487/RFC2387, August 1998, <https://www.rfc-editor.org/info/rfc2387>.

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

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

[RFC8189] Randriamasy, S., Roome, W., and N. Schwan, "Multi-Cost Application-Layer Traffic Optimization (ALTO)", RFC 8189, DOI 10.17487/RFC8189, October 2017, <https://www.rfc-editor.org/info/rfc8189>.

14.2. Informative References

[AAAI2019] Xiang, Q., Yu, H., Aspnes, J., Le, F., Kong, L., and Y.R. Yang, "Optimizing in the dark: Learning an optimal solution through a simple request interface", Proceedings of the AAAI Conference on Artificial Intelligence 33, 1674-1681 , 2019.

[I-D.contreras-alto-service-edge] Contreras, L., Perez, D., and C. Rothenberg, "Use of ALTO for Determining Service Edge", Work in Progress, Internet-Draft, draft-contreras-alto-service-edge-02, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-contreras-alto-service-edge-02.txt>.

[I-D.huang-alto-mowie-for-network-aware-app]
Huang, W., Zhang, Y., Yang, R., Xiong, C., Lei, Y., Han, Y., and G. Li, "MoWIE for Network Aware Application", Work in Progress, Internet-Draft, draft-huang-alto-mowie-for-network-aware-app-01, 13 July 2020, <http://www.ietf.org/internet-drafts/draft-huang-alto-mowie-for-network-aware-app-01.txt>.

[I-D.ietf-alto-performance-metrics]
WU, Q., Yang, Y., Lee, Y., Dhody, D., Randriamasy, S., and L. Contreras, "ALTO Performance Cost Metrics", Work in Progress, Internet-Draft, draft-ietf-alto-performance-metrics-12, 13 July 2020, <http://www.ietf.org/internet-drafts/draft-ietf-alto-performance-metrics-12.txt>.

[I-D.ietf-dmm-5g-uplane-analysis]
Homma, S., Miyasaka, T., Matsushima, S., and D. Voyer, "User Plane Protocol and Architectural Analysis on 3GPP 5G System", Work in Progress, Internet-Draft, draft-ietf-dmm-5g-uplane-analysis-04, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-dmm-5g-uplane-analysis-04.txt>.

[I-D.yang-alto-deliver-functions-over-networks]
Yang, S., Cui, L., Xu, M., Yang, Y., and R. Huang, "Delivering Functions over Networks: Traffic and Performance Optimization for Edge Computing using ALTO", Work in Progress, Internet-Draft, draft-yang-alto-deliver-functions-over-networks-01, 13 July 2020, <http://www.ietf.org/internet-drafts/draft-yang-alto-deliver-functions-over-networks-01.txt>.

[JSAC2019] Xiang, Q., Zhang, J., Wang, X., Liu, Y., Guok, C., Le, F., MacAuley, J., Newman, H., and Y.R. Yang, "Toward Fine-Grained, Privacy-Preserving, Efficient Multi-Domain Network Resource Discovery", IEEE/ACM IEEE Journal on Selected Areas of Communication 37(8): 1924-1940, 2019.

[LHC] "CERN - LHC", 2019, <https://atlas.cern/tags/lhc>.

[SC2018] Xiang, Q., Zhang, J., Wang, X., Liu, Y., Guok, C., Le, F., MacAuley, J., Newman, H., and Y.R. Yang, "Fine-grained, multi-domain network resource abstraction as a fundamental primitive to enable high-performance, collaborative data sciences", Proceedings of the Super Computing 2018, 5:1-5:13 , 2019.

[SENSE] "Services - SENSE", 2019, <http://sense.es.net/services>.

[TON2019] Gao, K., Xiang, Q., Wang, X., Yang, Y.R., and J. Bi, "An objective-driven on-demand network abstraction for adaptive applications", IEEE/ACM Transactions on Networking (TON) Vol 27, no. 2 (2019): 805-818., 2019.

Appendix A. Changes since -12

Revision -13

* changes the abstract based on the chairs' reviews

* integrates Richard's responds to WGLC reviews

Appendix B. Changes since -11

Revision -12

* clarifies the definition of ANEs in a similar way as how Network Elements is defined in [RFC2216]

* restructures several paragraphs that are not clear (Sec 3, Path Vector bullet, Sec 4.2, Sec 5.1.3, Sec 6.2.4, Sec 6.4.2, Sec 9.3)

* uses "ALTO Entity Domain Type Registry"

Appendix C. Changes since -10

Revision -11

* replaces "part" with "components" in the abstract;

* identifies additional requirements (AR) derived from the flow scheduling example, and introduces how the extension addresses the additional requirements

* fixes the inconsistent use of "start" parameter in multipart responses;

* specifies explicitly how to handle "cost-constraints";

* uses the latest IANA registration mechanism defined in [I-D.ietf-alto-unified-props-new];

* renames "persistent-entities" to "persistent-entity-id";

* makes "application/alto-propmap+json" as the media type of defining resources for the "ane" domain;

* updates the examples;

* adds the discussion on ephemeral and persistent ANEs.

Appendix D. Changes since -09

Revision -10

* revises the introduction which

  - extends the scope where the PV extension can be applied beyond the "path correlation" information

* brings back the capacity region use case to better illustrate the problem

* revises the overview to explain and defend the concepts and decision choices

* fixes inconsistent terms, typos

Appendix E. Changes since -08

This revision

* fixes a few spelling errors

* emphasizes that abstract network elements can be generated on demand in both introduction and motivating use cases

Appendix F. Changes Since Version -06

* We emphasize the importance of the path vector extension in two aspects:

  1. It expands the problem space that can be solved by ALTO, from preferences of network paths to correlations of network paths.

  2. It is motivated by new usage scenarios from both application's and network's perspectives.

* More use cases are included, in addition to the original capacity region use case.

* We add more discussions to fully explore the design space of the path vector extension and justify our design decisions, including the concept of abstract network element, cost type (reverted to -05), newer capabilities and the multipart message.

* Fix the incremental update process to be compatible with SSE -16 draft, which uses client-id instead of resource-id to demultiplex updates.

* Register an additional ANE property (i.e., persistent-entities) to cover all use cases mentioned in the draft.

Authors' Addresses

Kai Gao
Sichuan University
No.24 South Section 1, Yihuan Road
Chengdu
610000
China

Email: kaigao@scu.edu.cn

Young Lee
Samsung
South Korea

Email: younglee.tx@gmail.com

Sabine Randriamasy
Nokia Bell Labs
Route de Villejust
91460 Nozay
France

Email: sabine.randriamasy@nokia-bell-labs.com

Yang Richard Yang
Yale University
51 Prospect Street
New Haven, CT
United States of America

Email: yry@cs.yale.edu

Jingxuan Jensen Zhang
Tongji University
4800 Caoan Road
Shanghai

201804
China

Email: jingxuan.n.zhang@gmail.com

ALTO WG W. Roome
Internet-Draft S. Randriamasy
Intended status: Standards Track Nokia Bell Labs
Expires: 31 May 2021 Y. Yang
Yale University
J. Zhang
Tongji University
K. Gao
Sichuan University
27 November 2020

ALTO extension: Entity Property Maps
draft-ietf-alto-unified-props-new-15

Abstract

This document extends the base Application-Layer Traffic Optimization (ALTO) Protocol by generalizing the concept of "endpoint properties" to generic types of entities, and by presenting those properties as maps, similar to the network and cost maps in the base ALTO protocol. The protocol is extended in two major directions. First, from endpoints restricted to IP addresses to entities covering a wider and extensible set of objects; second, from properties on specific endpoints to entire entity property maps. These extensions introduce additional features allowing entities and property values to be specific to a given information resource. This is made possible by a generic and flexible design of entity and property types.

Table of Contents

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 three limitations.

First, the EPS allows properties to be associated with only endpoints that are identified by individual communication addresses like IPv4 and IPv6 addresses. It is reasonable to think that collections of endpoints, as defined by CIDRs [RFC4632] or PIDs, may also have properties. Furthermore, recent ALTO use cases show that properties of entities such as network flows [RFC7011] and routing elements [RFC7921] are also useful. Such cases are documented in [draft-gao-alto-fcs]. The current EPS however is restricted to individual endpoints and cannot be applied to those entities.

Second, the EPS only allows endpoints identified by global communication addresses. However, an endpoint address may be a local IP address or an anycast IP address which is also not globally unique. Additionally, an entity such as a PID may have an identifier that is not globally unique. That is, a same PID identifier may be used in multiple network maps, while in each network map, this PID identifier points to a different set of addresses. For example, PID "mypid10" may be defined in "netmap1" and "netmap2" while in each network map, "mypid10" covers a different set of addresses.

Third, 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 process cost lookups without querying the ALTO server. [RFC7285] does not define a similar 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. However, in practice, it is highly unlikely that properties will be defined for every endpoint address. It is much more likely that properties may be defined for only a subset of endpoint addresses, and the specification of properties uses an aggregation representation to allow enumeration. This is particularly true if blocks of endpoint addresses with a common prefix (e.g., a CIDR) have the same value for a property. Entities in other domains may very well allow aggregated representation and hence be enumerable as well.

To address the three limitations, this document specifies a protocol extension for defining and retrieving ALTO properties:

* The first limitation is addressed by introducing a generic concept called ALTO Entity, which generalizes an endpoint and may represent a PID, a network element, a cell in a cellular network, an abstracted network element as defined in [I-D.ietf-alto-path-vector], or other physical or logical objects involved in a network topology. Each entity is included in a collection called an ALTO Entity Domain. Since each ALTO Entity Domain includes only one type of entities, each Entity Domain can be classified by the type of entities in it.

* The second limitation is addressed by using resource-specific entity domains. A resource-specific entity domain contains entities that are defined and identified with respect to a given ALTO information resource, which provides scoping. For example, an entity domain containing PIDs is identified with respect to the network map in which these PIDs are defined. Likewise, an entity domain containing local IP addresses may be defined with respect to a local network map.

* The third limitation is addressed by defining two new types of ALTO information resources: Property Map, detailed in Section 7 and Filtered Property Map, detailed in Section 8. The former is a GET-mode resource that returns the property values for all entities in one or more entity domains, and is analogous to a network map or a cost map in [RFC7285]. The latter is a POST-mode resource that returns the values for sets of properties and entities requested by the client, and is analogous to a filtered network map or a filtered cost map.

The protocol extension defined in this document is augmentable. New entity domain types can be defined without revising the specification defined in this document. Similarly, new cost metrics and new endpoint properties can be defined in other documents without revising the protocol specification defined in [RFC7285].

1.1. Terminology

This document uses the following terms and abbreviations, that will be further defined in the document. While this document introduces the feature "entity property map", it will use both the term "property map" and "entity property map" to refer to this feature.

* 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.

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

* EPM: An abbreviation for entity property map

* FPM: An abbreviation for filtered property map.

* EPS: An abbreviation for Endpoint Property Service.

2. 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.

3. Basic Features of the Entity Property Map Extension

This section gives a high-level overview of the basic features involved in ALTO Entity Property Maps. It assumes the reader is familiar with the ALTO protocol [RFC7285]. The purpose of this extension is to allow conveying properties on objects that extend ALTO Endpoints and are called ALTO Entities, or entities for short.

The features introduced in this section can be used as standalone. However, in some cases, these features may depend on particular information resources and need to be defined with respect to them. To this end, Section 4 introduces additional features that extend the ones presented in the present section.

The Entity Property Maps extension described in this document introduces a number of features that are summarized in Appendix A, where Table 4 lists the features and references the sections in this document that give a high-level and normative description thereof.

3.1. Entity

The concept of an ALTO Entity generalizes the concept of an ALTO Endpoint defined in Section 2.1 of [RFC7285]. An entity is an object that can be an endpoint that is defined 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 an object that is not even related to any network address. Thus, whereas all endpoints are entities, not all entities are endpoints.

Examples of entities are:

* an ALTO endpoint, defined in [RFC7285], that represents an application or a host identified by a communication address (e.g., an IPv4 or IPv6 address) in a network,

* a PID, defined in [RFC7285], that has a provider defined human-readable identifier specified by an ALTO network map, which maps a PID to a set of ipv4 and ipv6 addresses,

* an autonomous system (AS), that has an AS number (ASN) as its identifier and maps to a set of ipv4 and ipv6 addresses,

* a country with a code as specified in [ISO3166-1], to which applications such as CDN providers associate properties and capabilities,

* a TCP/IP network flow, that is identified by a TCP/IP 5-Tuple specifying its source and destination addresses and port numbers and the utilized protocol,

* a routing element, that is specified in [RFC7921] and is associated with routing capabilities information,

* an abstract network element, that represents an abstraction of a network part such as a routable network node, one or more links, a network domain or their aggregation.

3.2. Entity Domain

An entity domain defines a set of entities of the same semantic type. An entity domain is characterized by its type and identified by its name.

In this document, an entity must be owned by exactly one entity domain name. An entity identifier must point to exactly one entity. If two entities in two different entity domains refer to the same physical or logical object, they are treated as different entities. For example, if an object has both an IPv4 and an IPv6 address, these two addresses will be treated as two entities, defined respectively in the "ipv4" and "ipv6" entity domains.

3.2.1. Entity Domain Type

The type of an entity domain type defines the semantics of a type of entity. Entity domain types can be defined in different documents. For example: the present document defines entity domain types "ipv4", "ipv6" and "pid" in sections Section 6.1 and Section 6.2. The entity domain type "ane", that defines Abstract Network Elements (ANEs), is introduced in [I-D.ietf-alto-path-vector]. The entity domain type that defines country codes is introduced in [draft-ietf-alto-cdni-request-routing-alto]. An entity domain type is expected to be registered at the IANA, as specified in section Section 12.2.2 and similarly to an ALTO address type.

3.2.2. Entity Domain Name

The name of an entity domain is defined in the scope of an ALTO server. An entity domain name can sometimes be identical to the name of its relevant entity domain type. This is the case when the entities of a domain have an identifier that points to the same object throughout all the information resources of the Server that provide entity properties for this domain. For example, a domain of type "ipv4" containing entities identified by a public IPv4 address can be named "ipv4" because its entities are uniquely identified by all the Server resources.

In some cases, a domain type and domain name must be different. Indeed, for some domain types, entities are defined relatively to a given information resource. As a consequence, entities in such domains may be defined in a resource handling this domain type but not in other resources handling this same domain type. Moreover, across different ALTO information resources handling a domain type, an entity identifier may point to different objects. This is the case for entities of domain type "pid". A PID is defined relatively to a network map. For example: an entity "mypid10" of domain type "pid" may be defined in a given network map resource and be undefined in other network maps, or may even map to a different set of endpoint addresses. In this case, naming an entity domain only by its type "pid" does not guarantee that its entities are owned by exactly one entity domain name. Section 4.2 and Section 5.1.2 of this document describe how a domain is uniquely identified by a name that associates the domain type and the related information resource.

3.3. Entity Property Type

An entity property defines a property of an entity. This is similar to the endpoint property defined in Section 7.1 of [RFC7285]. An entity property can convey either network-aware or network-agnostic information. Similarly to an entity domain, an entity property is characterized by its type and identified by its name. An entity property type is expected to be registered at the IANA, as specified in section Section 12.3.

Below are some examples with real and fictitious entity domain and property names:

* an entity in the "ipv4" domain type may have a property whose value is an Autonomous System (AS) number indicating the AS that owns this IPv4 address and another property named "countrycode" indicating a country code mapping to this address,

* an entity identified by its country code in the entity domain type "countrycode", defined in [draft-ietf-alto-cdni-request-routing-alto] may have a property indicating what delivery protocol is used by a CDN,

* an entity in the "netmap1.pid" domain may have a property that indicates the central geographical location of the endpoints it includes.

It should be noted that some identifiers may be used for both an entity domain type and a property type. For example:

* the identifier "countrycode" may point to both the entity domain type "countrycode" and the fictitious property type "countrycode".

* the identifier "pid" may point to both the entity domain type "pid" and the property type "pid".

Likewise, a same identifier may point to both a domain name and a property name. For example: the identifier "netmap10.pid" may point to either the domain defined by the PIDs of network map "netmap10" or to a property that returns, for an entity defined by its IPv4 address, the PID of netmap10 that contains this entity. Such cases will be further explained in Section 4.

3.4. New information resource and media type: ALTO Property Map

This document introduces a new ALTO information resource named Property Map. An ALTO property map provides a set of properties on one or more sets of entities. A property may apply to different entity domain types and names. For example, an ALTO property map may define the "ASN" property for both "ipv4" and "ipv6" entity domains.

The present extension also introduces a new media type.

This document uses the same definition of an information resource as Section 9.1 of [RFC7285]. ALTO uses media types to uniquely indicate the data format used to encode the content to be transmitted between an ALTO server and an ALTO client in the HTTP entity body. In the present case, an ALTO property map resource is defined by the media type "application/alto-propmap+json".

A Property Map can be queried as a GET-mode resource, thus conveying values of all property values on all entities indicated in its capabilities. It can also be queried as a POST-mode resource, thus conveying a selection of properties on a selection of entities.

4. Advanced Features of the Entity Property Map Extension

4.1. Entity Identifier and Entity Domain Name

In [RFC7285], an endpoint has an identifier that is explicitly associated with the "ipv4" or "ipv6" address domain. Examples are "ipv4:192.0.2.14" and "ipv6:2001:db8::12".

In this document, an entity must be owned by exactly one entity domain name and an entity identifier must point to exactly one entity. To ensure this, an entity identifier is explicitly attached to the name of its entity domain and an entity domain type characterizes the semantics and identifier format of its entities.

The encoding format of an entity identifier is further specified in Section 5.1.3 of this document.

For instance:

* if an entity is an endpoint with example routable IPv4 address "192.0.2.14", its identifier is associated with domain name "ipv4" and is "ipv4:192.0.2.14",

* if an entity is a PID named "mypid10" in network map resource "netmap2", its identifier is associated with domain name "netmap2.pid" and is "netmap2.pid:mypid10".

4.2. Resource-Specific Entity Domain Name

Some entities are defined and identified uniquely and globally. This is the case for instance when entities are endpoints that are identified by a routable IPv4 or IPv6 address. The entity domain for such entities can be globally defined and named "ipv4" or "ipv6". Those entity domains are called resource-agnostic entity domains in this document, as they are not associated with any specific ALTO information resources.

Some other entities and entity types are only defined relatively to a given information resource. This is the case for entities of domain type "pid", that can only be understood with respect to the network map where they are defined. For example, a PID named "mypid10" may be defined to represent a set S1 of IP addresses in a network map resource named "netmap1". Another network map "netmap2" may use the same name "mypid10" and define it to represent another set S2 of IP addresses. The identifier "pid:mypid10" may thus point to different objects because the information on the originating information resource is lost.

To solve this ambiguity, the present extension introduces the concept of resources-specific entity domain. This concept applies to domain types where entities are defined relatively to a given information resource. It can also apply to entity domains that are defined locally, such as local networks of objects identified with a local IPv4 address.

In such cases, an entity domain type is explicitly associated with an identifier of the information resource where these entities are defined. Such an information resource is referred to as the "specific information resource". Using a resource-aware entity domain name, an ALTO property map can unambiguously identify distinct entity domains of the same type, on which entity properties may be queried. Examples of resource-specific entity domain names may look like: "netmap1.pid" or "netmap2.pid". Thus, a name association such as "netmap1.pid:mypid10" and "netmap2.pid:mypid10" allows to distinguish the two abovementioned PIDs that are both named "mypid10" but in two different resources, "netmap1" and "netmap2".

An information resource is defined in the scope of an ALTO Server and so is an entity domain name. The format of a resource-specific entity domain name is further specified in Section 5.1.2.

4.3. Resource-Specific Entity Property Value

Like entity domains, some types of properties are defined relatively to an information resource. That is, an entity may have a property of a given type, whose values are associated to different information resources.

For example, suppose entity "192.0.2.34" defined in the "ipv4" domain has a property of type "pid", whose value is the PID to which address "192.0.2.34" is attached in a network map. The mapping of network addresses to PIDs is specific to a network map and probably different from one network map resource to another one. So that if a property "pid" is defined for entity "192.0.2.34" in two different network maps "netmap1" and "netmap2", the value for this property will likely be a different value in "netmap1" and "netmap2".

To support information resource dependent property values, this document uses the same approach as in Section 10.8.1 of [RFC7285] entitled "Resource-Specific Endpoint Properties". When a property value depends on a given information resource, the name of this property must be explicitly associated with the information resource that defines it.

For example, the property "pid" queried on entity "ipv4:192.0.2.34" and defined in both "netmap1" and "netmap2", can be named "netmap1.pid" and "netmap2.pid". This allows a Client to get a property of the same type but defined in different information resources with a single query. Specifications on the property name format are provided in Section 5.2.

4.4. Entity Hierarchy and Property Inheritance

For some domain types, entities can be grouped in a set and be defined by the identifier of this set. This is the case for domain types "ipv4" and "ipv6", where individual Internet addresses can be grouped in blocks. When a same property value applies to a whole set, a Server can define a property for the identifier of this set instead of enumerating all the entities and their properties. This allows a substantial reduction of transmission payload both for the Server and the Client. For example, all the entities included in the set defined by the address block "ipv6:2001:db8::1/64" share the same properties and values defined for this block.

Additionally, entity sets sometimes are related by inclusion, hierarchy or other relations. This allows defining inheritance rules for entity properties that propagate properties among related entity sets. The Server and the Client can use these inheritance rules for further payload savings. Entity hierarchy and property inheritance rules are specified in the documents that define the applicable domain types. The present document defines these rules for the "ipv4" and "ipv6" domain types.

This document introduces, for applicable domain types, "Entity Property Inheritance rules", with the following concepts: Entity Hierarchy, Property Inheritance and Property Value Unicity. A detailed specification of entity hierarchy and property inheritance rules is provided in Section 5.1.4.

4.4.1. Entity Hierarchy

An entity domain may allow using a single identifier to identify a set of individual entities. For example, a CIDR block can be used to identify a set of IPv4 or IPv6 entities. A CIDR block is called a hierarchical entity identifier, as it can reflect inclusion relations among entity sets. For example, the CIDR "ipv4:192.0.1.0/24" includes all the individual ipv4 entities identified by the CIDR "ipv4:192.0.1.0/26".

4.4.2. Property Inheritance

A property may be defined for a hierarchical entity identifier, while it may be undefined for individual entities covered by this identifier. In this case, these individual entities inherit the property value defined for the identifier that covers them. For example, suppose a property map defines the ASN property only for the hierarchical entity identifier "ipv4:192.0.1.0/24" but not for individual entities in this block. Suppose also that inheritance rules are specified for CIDR blocks in the "ipv4" domain type. When receiving this property map, a Client can infer that entity "ipv4:192.0.1.1" inherits the "ASN" property value of block "ipv4:192.0.1.0/24" because the address "ipv4:192.0.1.1" is included by the CIDR block "ipv4:192.0.1.0/24".

Property value inheritance rules also apply among entity sets. A property map may define values for an entity set belonging to a hierarchy but not for "sub" sets that are covered by this set identifier. In this case, inheritance rules must specify how entities in "sub" sets inherit property values from their "super" set. For instance, if the "ASN" property is defined only for the entity set identified by address block "ipv4:192.0.1.0/24", the entity set identified by "ipv4:192.0.1.0/30" and thus included in the former set, may inherit the "ASN" property values from set "ipv4:192.0.1.0/24".

4.4.3. Property Value Unicity

The inheritance rules must ensure that an entity belonging to a hierarchical set of entities inherits no more than one property value, for the sake of consistency. Indeed, a property map may define a property on a hierarchy of entity sets that inherit property values from one or more including sets in the upper levels. On the other hand, a property value, defined at a lower level of the hierarchy may be different from the value defined at an upper level. In such a case, a set in the lower level of the hierarchy may potentially end up with different values. This may be the case for address blocs with increasing prefix length, on which a property value gets increasingly accurate and thus may differ. For example, a fictitious property such as "geo-location" or "average transfer volume" may be defined at a progressively finer grain for entity sets defined with progressively longer CIDR prefixes. It seems more interesting to have property values of progressively higher accuracy. A unicity rule, applied to the entity domain type must specify an arbitration rule among the different property values for an entity. An example illustrating the need for such rules is provided in Section 6.1.3.

4.5. Supported Properties on Entity Domains in Property Map Capabilities

A property type is not necessarily applicable to any domain type, or an ALTO Server may just not provide a property on all applicable domains. For instance, a property type reflecting link bandwidth is likely not defined on entities of a domain of type "country-code". Therefore an ALTO server providing Property Maps needs to specify the properties that can be queried on the different entity domains it supports.

This document explains how the Information Resources Directory (IRD) capabilities of a Property Map resource unambiguously expose what properties a Client can query on a given entity domain.

* a field named "mappings" lists the names of the entity domains supported by the Property Map,

* for each listed entity domain, a list of the names of the applicable properties is provided.

An example is provided in Section 10.3. The "mappings" field associates entity domains and properties that can be resource-agnostic or resource-specific. This allows a Client to formulate compact and unambiguous entity property queries, possibly relating to one or more information resources. In particular:

* it avoids a Client to query a property on entity domains on which it is not defined,

* it allows a Client to query, for an entity E, values for a property P that are defined in several information resources,

* it allows a Client to query a property P on entities that are defined in several information resources.

Further specifications are provided in Section 7.4.

4.6. Defining Information Resource for Resource-Specific Entity Domains

A Client willing to query properties on entities belonging to a domain needs to know how to retrieve these entities. To this end, the Client can look up the "mappings" field exposed in IRD capabilities of a property map, see Section 4.5. This field, in its keys, exposes all the entity domains supported by the property map. The syntax of the entity domain identifier specified in Section 5.1.2 allows the client to infer whether the entity domain is resource-specific or not. The Client can extract, if applicable, the identifier of the specific resource, query the resource and retrieve the entities. For example:

* an entity domain named "netmap1.ipv4" includes the IPv4 addresses that appear in the "ipv4" field of the endpoint address group of each PID in the network map "netmap1", and that cannot be recognized outside "netmap1" because, for instance, these are local non-routable addresses,

* an entity domain named "netmap1.pid" includes the PIDs listed in network map "netmap1".

* an entity domain named "ipv4" is resource-agnostic and covers all the routable IPv4 addresses.

Besides, it is also necessary to inform a Client about which associations of specific resources and entity domain types are allowed, because it is not possible to prevent a Server from exposing inappropriate associations. An informed Client will just ignore inappropriate associations exposed by a Server and avoid error-prone transactions with the Server.

For example, the association "costmap3.pid" is not allowed for the following reason: although a cost map exposes PID identifiers, it does not define them, that is, the set of addresses included in this PID. Neither does a cost map list all the PIDs on which properties can be queried, because a cost map only exposes PID pairs on which a queried cost type is defined. Therefore, the resource "costmap3" does not enable a Client to extract information on the existing PID entities or on the addresses they contain.

Instead, the cost map uses a network map, where all the PIDs used in a cost map are defined together with the addresses contained by the PIDs. This network map is qualified in this document as the Defining Information Resource for the entity domain of type "pid" and this concept is explained in Section 4.6.1.

4.6.1. Defining Information Resource and Media Type

For the reasons explained in the previous section, this document introduces the concept of defining information resource and media type.

A defining information resource for an entity domain D is the information resource where entities of D are defined. That is, all the information on the entities of D can be retrieved in this resource. This concept applies to resource-specific domains. This is useful for entity domain types that are by essence domain-specific, such as "pid" and "ane" domain types. It is also useful for resource-specific entity domains constructed from resource-agnostic domain types, such as network map specific domains of local IPv4 addresses.

The defining information resource of an entity domain D has the following specificities:

* it has an entry in the IRD,

* it defines the entities of D,

* it does not use another information resource that defines these entities,

* it defines and exposes entity identifiers that are all persistent.

* its media type is unique and equal to the one that is specified for the defining information resource of an entity domain type.

A fundamental attribute of a defining information resource is its media type. There is a unique association between an entity domain type and the media type of its defining information resource. When an entity domain type allows associations with defining information resources, the document that defines this entity domain type specifies the media type of the potential defining information resource. Likewise, the IANA registration of an entity domain type also specifies the media type of potential defining information resources.

When the Client wants to use a resource-specific entity domain, it needs to be cognizant of the media-type of its defining information resource. If the Server exposes resources a resource specific entity domain with a non-compliant media type for the domain type, the Client can avoid transaction errors by ignoring them.

4.6.2. Examples of defining information resources media-types

Here are some examples of specific information resource types associated to entity domain types and their media type.

* For entity domain type "pid": the media type of the specific resource is "application/alto-networkmap+json", because PIDs are defined in network map resources.

* For entity domain types "ipv4" and "ipv6": the media type of the specific resource is "application/alto-networkmap+json", because IPv4 and IPv6 addresses covered by the Server are defined in network map resources.

* For entities of domain type "ane": [I-D.ietf-alto-path-vector] defines entities named "ANE", where ANE stands for Abstracted Network Element, and the entity domain type "ane". When an ANE has a persistent identifier, say, "entity-4", the latter is provided by the Server as a value of the "persistent-entity-id" property of the ANE. Further properties may be queried on an ANE with a persistent entity ID. These properties are available from a persistent property map, that defines properties on a specific "ane" domain. Together with the persistent identifier, the Server also provides the property map resource identifier where the "ane" domain containing "entity-4" is defined. The definition of the "ane" entity domain containing "entity-4" is thus specific to the property map. Therefore, for entities of domain type "ane" that have a persitent identifier, the media type of the specific information resource is "application/alto-propmap+json".

4.7. Defining Information Resource for Resource-Specific Property Values

As explained in Section 4.3, a property type may take values that are resource specific. This is the case for property type "pid", whose values are by essence defined relatively to a specific network map. The PID value returned for an IPv4 address is specific to the network map defining the PID and may differ from one network map to another one. Property values may be specific to different types of information resources. For example: the value for property "pid" is specific to a network map. The value for property type "cdnifci-capab" is specific to the information resource "cdnifci-map", defined in [draft-ietf-alto-cdni-request-routing-alto], while network maps do not define property "fci-capability" for ipv4 addresses and a cdnifci-map does not define "pid" values for IPv4 addresses.

Thus, similarly to resource specific entity domains, the Client needs to be cognizant of appropriate associations of information resource and property types.

4.7.1. Examples of defining resources media-types for properties

Here are some examples of specific information resources types associated to entity property types and their media type.

* For property type "pid": the media type of the specific resource is "application/alto-networkmap+json", because PIDs are defined in network map resources.

* For property type "cdni-fci-capability": the media type of the specific resource is "application/alto-cdnifci+json"

5. Protocol Specification: Basic Data Types

5.1. Entity Domain

5.1.1. Entity Domain Type

An entity domain has a type, which is uniquely identified by a string that MUST be no more than 64 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). The '.' separator MUST NOT be used unless specifically indicated in a further extension document.

For example, the strings "ipv4", "ipv6", and "pid" are valid entity domain types. "ipv4.anycast" and "pid.local" are invalid.

The type EntityDomainType is used in this document to denote a JSON string meeting the preceding requirements.

An entity domain type defines the semantics of a type of entity, independently of any specifying resource. Each entity domain type MUST be registered with the IANA. The format of the entity identifiers (see Section 5.1.3) in that type of entity domain, as well as any hierarchical or inheritance rules (see Section 5.1.4) for those entities, MUST be specified at the same time.

5.1.2. Entity Domain Name

As said in Section 3.2 when introducing entity domains, an entity domain is characterized by its type and identified by its name.

This document distinguishes three categories of entity domains: resource-specific entity domains, resource-agnostic entity domains and self-defined entity domains. Their entity domain names are constructed as specified in the following sub-sections.

Each entity domain is identified by a unique entity domain name which is a string of the following format:

```
EntityDomainName ::= [ [ ResourceID ] '.' ] EntityDomainType
```

Where the presence and construction of component:

"[[ResourceID] '.']"

depends on the category of entity domain.

Note that the '.' separator is not allowed in EntityDomainType and hence there is no ambiguity on whether an entity domain name refers to a resource-agnostic entity domain or a resource-specific entity domain.

Note also that the resource ID format defined in Section 10.1 of [RFC7285] specifies that: "the '.' separator is reserved for future use and MUST NOT be used unless specifically indicated in this document, or an extension document". The present extension keeps the format specification of [RFC7285], hence the '.' separator MUST NOT be used in an information resource ID.

5.1.2.1. Resource-specific Entity Domain

A resource-specific entity domain is identified by an entity domain name constructed as follows. It MUST start with a resource ID using the ResourceID type defined in Section 10.2 of [RFC7285], followed by the '.' separator (U+002E), followed by a string of the type EntityDomainType specified in Section 5.1.1.

For example, if an ALTO server provides two network maps "netmap-1" and "netmap-2", these network maps can define two resource-specific domains of type "pid", respectively identified by "netmap-1.pid" and "netmap-2.pid".

5.1.2.2. Resource-agnostic Entity Domain

A resource-agnostic entity domain contains entities that are identified independently of any information resource. Hence, the identifier of a resource-agnostic entity domain is simply the identifier of its entity domain type. For example, "ipv4" and "ipv6" identify the two resource-agnostic Internet address entity domains defined in Section 6.1.

5.1.2.3. Self-defined Entity Domain

A property map can define properties on entities that are specific to a unique information resource, which is the property map itself. This may be the case when an ALTO Server provides properties on a set of entities that are defined only in this property map, are not relevant to another one and do not depend on another specific resource.

For example: a specialised property map may define a domain of type "ane", defined in [I-D.ietf-alto-path-vector], that contains a set of ANEs representing data centers, that each have a persistent identifier and are relevant only to this property map.

In this case, the entity domain is qualified as "self-defined". The identifier of a self-defined entity domain can be of the format:

    EntityDomainName ::= .EntityDomainType

where '.' indicates that the entity domain only exists within the property map resource using it.

A self-defined entity domain can be viewed as a particular case of resource-specific entity domain, where the specific resource is the current resource that uses this entity domain. In that case, for the sake of simplification, the component "ResourceID" SHOULD be omitted in its entity domain name.

5.1.3. Entity Identifier

Entities in an entity domain are identified by entity identifiers (EntityID) of the following format:

EntityID ::= EntityDomainName ':' DomainTypeSpecificEntityID

Examples from the Internet address entity domains include individual IP addresses such as "net1.ipv4:192.0.2.14" and "net1.ipv6:2001:db8::12", as well as address blocks such as "net1.ipv4:192.0.2.0/26" and "net1.ipv6:2001:db8::1/48".

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

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

Note that two entity identifiers with different valid textual representations may refer to the same entity, for a given entity domain. For example, the strings "net1.ipv6:2001:db8::1" and "net1.ipv6:2001:db8:0:0:0:0:0:1" refer to the same entity in the "ipv6" entity domain.

5.1.4. Hierarchy and Inheritance

To simplify the representation, some types of entity domains allow the ALTO Client and Server to use a hierarchical entity identifier format to represent a block of individual entities. For instance, in an IPv4 domain "net1.ipv4", a CIDR "net1.ipv4:192.0.2.0/26" covers 64 individual IPv4 entities. In this case, the corresponding property inheritance rule MUST be defined for the entity domain type. The hierarchy and inheritance rule MUST have no ambiguity.

5.2. Entity Property

Each entity property has a type to indicate the encoding and the semantics of the value of this entity property, and has a name to identify it.

5.2.1. Entity Property Type

The type EntityPropertyType is used in this document to indicate a string denoting an entity property type. The string MUST be no more than 32 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), or the low line ('_', U+005F). Note that the '.' separator is not allowed because it is reserved to separate an entity property type and an information resource identifier when an entity property is resource-specific.

Each entity property type MUST be registered with the IANA. The intended semantics of the entity property type MUST be specified at the same time.

Identifiers prefixed with "priv:" are reserved for Private Use [RFC5226] without a need to register with IANA. All other identifiers for entity property types appearing in an HTTP request or response with an "application/alto-*" media type MUST be registered in the "ALTO Entity Property Type Registry", defined in Section 12.3. For an entity property identifier with the "priv:" prefix, an additional string (e.g., company identifier or random string) MUST follow the prefix to reduce potential collisions, that is, the string "priv:" alone is not a valid endpoint property identifier.

To distinguish from the endpoint property type, the entity property type has the following characteristics:

* Some entity property types are applicable to entities in particular entity domain types only. For example, the property type "pid" is applicable to entities in the entity domain types "ipv4" or "ipv6" while is not applicable to entities in an entity domain of type "pid".

* The intended semantics of the value of an entity property may also depend on the entity domain type. For example, suppose that a property named "geo-location" is defined as the coordinates of a point, encoded as: "latitude longitude [altitude]." When applied to an entity that represents a specific host computer, identified by an address in an entity domain of type "ipv4" or "ipv6", the "geo-location" property would define the host's location. However, when applied to an entity in a "pid" domain type, the property would indicate the location of the center of all hosts in this "pid" entity.

5.2.2. Entity Property Name

Each entity property is identified by an entity property name, which is a string of the following format:

```
EntityPropertyName ::= [ ResourceID ] '.' EntityPropertyType
```

Similar to the endpoint property type defined in Section 10.8 of [RFC7285], each entity property may be defined by either the property map itself (self-defined) or some other specific information resource (resource-specific).

The entity property name of a resource-specific entity property starts with a string of the type ResourceID defined in [RFC7285], followed by the '.' separator (U+002E) and a EntityDomainType typed string. For example, the "pid" properties of an "ipv4" entity defined by two different maps "net-map-1" and "net-map-2" are identified by "net-map-1.pid" and "net-map-2.pid" respectively.

The specific information resource of an entity property may be the current information resource itself, that is, the property map defining the property. In that case, the ResourceID in the property name SHOULD be ignored. For example, the property name ".asn" applied to an entity identitifed by its IPv4 address, indicates the AS number of the AS that "owns" the entity, where the returned AS number is defined by the property map itself.

5.2.3. Format for Entity Property Value

[RFC7285] in Section 11.4.1.6, specifies that an implementation of the Endpoint Property Service specified in [RFC7285] SHOULD assume that the property value is a JSONString and fail to parse if it is not. This document extends the format of a property value by allowing it to be a JSONValue instead of just a JSONString.

6. Entity Domain Types Defined in this Document

This document requires the definition of each entity domain type MUST include (1) the entity domain type name and (2) domain-specific entity identifiers, and MAY include (3) hierarchy and inheritance semantics optionally. This document defines three initial entity domain types as follows.

6.1. Internet Address Domain Types

The document defines two entity domain types (IPv4 and IPv6) for Internet addresses. Both types are resource-agnostic entity domain types and hence define corresponding resource-agnostic entity domains as well. Since the two domains use the same hierarchy and inheritance semantics, we define the semantics together, instead of repeating for each.

6.1.1. Entity Domain Type: IPv4

6.1.1.1. Entity Domain Type Identifier

ipv4

6.1.1.2. Domain-Specific Entity Identifiers

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

6.1.2. Entity Domain Type: IPv6

6.1.2.1. Entity Domain Type Identifier

ipv6

6.1.2.2. Domain-Specific Entity Identifiers

Individual addresses are strings as specified by Section 4 of [RFC5952]; Hierarchical addresses are prefix-match strings as specified in Section 7 of [RFC5952]. To define 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.

6.1.3. Hierarchy and Inheritance of Internet Address 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 a hierarchical Internet address C which prefix-matches I, then the address I inherits the value of P defined for the hierarchical address C. If more than one such hierarchical addresses define a value for P, I inherits the value of P in the hierarchical address with the longest prefix. Note that this longest prefix rule ensures no multiple value inheritances, and hence no ambiguity.

Hierarchical addresses can also inherit properties: if a property P is not defined for the hierarchical address C, but is defined for another hierarchical address C’ which covers all IP addresses in C, and C’ has a shorter prefix length than C, then C MAY inherit the property from C’. If there are multiple such hierarchical addresses like C’, C MUST inherit from the hierarchical address having the longest prefix length.

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.

6.1.4. Defining Information Resource Media Type for domain types IPv4 and IPv6

Entity domain types "ipv4" and "ipv6" both allow to define resource specific entity domains. When resource specific domains are defined with entities of domain type "ipv4" or "ipv6", the defining information resource for an entity domain of type "ipv4" or "ipv6" MUST be a Network Map. The media type of a defining information resource is therefore:

application/alto-networkmap+json

6.2. Entity Domain Type: PID

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

6.2.1. Entity Domain Type Identifier

pid

6.2.2. Domain-Specific Entity Identifiers

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

6.2.3. Hierarchy and Inheritance

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

6.2.4. Defining Information Resource Media Type for Domain Type PID

The entity domain type "pid" allows to define resource specific entity domains. When resource specific domains are defined with entities of domain type "pid", the defining information resource for entity domain type "pid" MUST be a Network Map. The media type of a defining information resource is therefore:

application/alto-networkmap+json

6.2.5. 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 all the properties of a PID to the prefixes in that PID or only some of these properties.

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".

6.3. Internet Address Properties vs. PID Properties

Because the Internet address and PID domains relate to completely distinct domain types, the question may arise as to which entity domain type is the best for a property. In general, the Internet address domain types are RECOMMENDED for properties that are closely related to the Internet address, or are associated with, and inherited through, hierarchical addresses.

The PID domain type 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 Internet address domain types. On the other hand, a property that explains why a PID was formed, or how it relates to a provider's network, would best be associated with the PID domain type.

7. 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 10.4 gives an example of a property map request and its response.

7.1. Media Type

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

7.2. HTTP Method

The property map is requested using the HTTP GET method.

7.3. Accept Input Parameters

None.

7.4. Capabilities

The capabilities are defined by an object of type PropertyMapCapabilities:

```
object {
  EntityPropertyMapping mappings;
} PropertyMapCapabilities;

object-map {
  EntityDomainName -> EntityPropertyName<1..*>;
} EntityPropertyMapping
```

with fields:

mappings: A JSON object whose keys are names of entity domains and values are the supported entity properties of the corresponding entity domains.

7.5. Uses

The "uses" field of a property map resource in an IRD entry specifies dependent resources of this property map. It is an array of the resource ID(s) of the resource(s).

7.6. 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 {
  EntityPropertyName -> 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:

* If the entity is in a resource-specific entity domain, the ALTO server SHOULD only return self-defined properties and resource-specific properties which depend on the same resource as the entity does. The ALTO client SHOULD ignore the resource-specific property in this entity if their mapping is not registered in the ALTO Resource Entity Property Transfer Registry of the type of the corresponding resource.

* If the entity is in a shared entity domain, the ALTO server SHOULD return self-defined properties and all resource-specific properties defined for all resource-specific entities which have the same domain-specific entity identifier as this entity does.

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.

8. 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 10.5, Section 10.6, Section 10.7 and Section 10.8 give examples of filtered property map requests and responses.

8.1. Media Type

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

8.2. HTTP Method

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

8.3. Accept Input Parameters

The input parameters for a filtered property map request are supplied in the 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:

```
object {
  EntityID             entities<1..*>;
  EntityPropertyName   properties<1..*>;
} ReqFilteredPropertyMap;
```

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 8.4).

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 8.4). 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.

8.4. Capabilities

The capabilities are defined by an object of type PropertyMapCapabilities, as defined in Section 7.4.

8.5. Uses

Same to the "uses" field of the Property Map resource (see Section 7.5).

8.6. Filtered Property Map 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 in the following cases:

* An entity identifier in the "entities" field of the request is invalid if:

  - The domain of this entity is not defined in the "entity-domains" capability of this resource in the IRD,

  - The entity identifier is not valid for the entity domain.

  A valid entity identifier does never generate an error, even if the filtered property map resource does not define any properties for it.

  If an entity identifier in the "entities" field of 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 the provided invalid entity identifier.

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

  When a filtered property map resource does not define a value for a property requested on a particular entity, it is not an error. 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 7.6), except that:

* If the requested entities include entities in the shared entity domain, the "dependent-vtags" field in its "meta" field MUST include version tags of all dependent resources appearing in the "uses" field.

* If the requested entities only include entities in resource-specific entity domains, the "dependent-vtags" field in its "meta" field MUST include the version tags of the resources on which the requested resource-specific entity domains and the requested resource-specific properties are dependent on.

* The response only includes the entities and properties requested by the client. If an entity in the request is identified by a hierarchical identifier (e.g., a "ipv4" or "ipv6" prefix), the response MUST cover properties for all identifiers in this hierarchical identifier.

The filtered property map response MUST include all the inherited property values for the requested entities and all the entities which are able to inherit property values from the requested entities. 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.

* 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.

* If an entity address in the response is already covered by other entities addresses in the same response, it SHOULD be removed from the response, for the sake of compactness. 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.

8.7. Entity property type defined in this document

This document defines the entity property type "pid". This property type extends the ALTO Endpoint Property Type "pid" defined in section 7.1.1 of [RFC7285] as follows: the property has the same semantics and applies to IPv4 and IPv6 addresses; the difference is that the IPv4 and IPv6 addresses have evolved from the status of endpoints to the status of entities.

The defining information resource for property type MUST be a network map. This document requests a IANA registration for this property

8.7.1. Entity Property Type: pid

1. Identifier: pid

2. Semantics: the intended semantics are the same as in [RFC7285] for the ALTO Endpoint Property Type "pid"

3. Media type of defining information resource: application/alto-networkmap+json

4. Security considerations: for entity property type "pid" are the same as documented in [RFC7285] for the ALTO Endpoint Property Type "pid".

9. Impact on Legacy ALTO Servers and ALTO Clients

9.1. 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.

9.2. 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 on, while global property names have no such prefix. The property map and the filtered property map defined in this document define similar categories of entity properties. The difference is that entity property maps do not define "global" entity properties. Instead, they define "self-defined" entity properties as a special case of "resource-specific" entity properties, where the specific resource is the property map itself. This means that "self-defined" properties are defined within the scope of the property map.

9.3. 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 hierarchical entity identifiers, rather than just individual entity identifiers, which might change the semantics of a property.

10. Examples

10.1. Network Map

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

```
defaultpid:  ipv4:0.0.0.0/0  ipv6:::0/0
pid1:        ipv4:192.0.2.0/25
pid2:        ipv4:192.0.2.0/27
pid3:        ipv4:192.0.3.0/28
pid4:        ipv4:192.0.3.16/28
```

Figure 3: Example Default Network Map

And another simple alternative network map:

```
defaultpid:  ipv4:0.0.0.0/0  ipv6:::0/0
pid1:        ipv4:192.0.2.0/27
pid2:        ipv4:192.0.3.0/27
```

Figure 4: Example Alternative Network Map

10.2. 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:

	ISP	ASN	country	state
ipv4:192.0.2.0/23:	BitsRus	-	us	-
ipv4:192.0.2.0/28:	-	12345	-	NJ
ipv4:192.0.2.16/28:	-	12345	-	CT
ipv4:192.0.2.1:	-	-	-	PA
ipv4:192.0.3.0/28:	-	12346	-	TX
ipv4:192.0.3.16/28:	-	12346	-	MN

Figure 5: Example Property Values for Internet Address Domains

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

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

Figure 6: Example Property Values for Default Network Map's 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 "ipv4:192.0.2.1" 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.

Similar to the PID domain of the default network map, the examples in this section use the property "ASN" for the PID domain of the alternative network map with the following values:

```
                        ASN
pid:defaultpid:          -
pid:pid1:             12345
pid:pid2:             12346
```

Figure 7: Example Property Values for Alternative Network Map's PID Domain

10.3. Information Resource Directory (IRD)

The following IRD defines ALTO Server information resources that are relevant to the Entity Property Service. It provides two property maps: one for the "ISP" and "ASN" properties, and another one for the "country" and "state" properties. The server could have provided a single property map for all four properties, but does not, presumably because the organization that runs the ALTO server believes that a client is not necessarily interested in getting all four properties.

The server provides several filtered property maps. The first returns all four properties, and the second returns only 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 providing the "pid" property does have a "uses" capability for the default network map, because the default network map defines the values of the "pid" property.

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

The server provides another filtered Property map resource, named "ane-dc-property-map", that returns a fictitious properties named "storage-capacity", "ram" and "cpu" for ANEs that have a persistent identifier. The entity domein to which the ANEs belong is "self-defined" and valid only within the property map.


```
"meta" : {
  ...
  "default-alto-network-map" : "default-network-map"
},
"resources" : {
  "default-network-map" : {
    "uri" : "http://alto.example.com/networkmap/default",
    "media-type" : "application/alto-networkmap+json"
  },
  "alt-network-map" : {
    "uri" : "http://alto.example.com/networkmap/alt",
    "media-type" : "application/alto-networkmap+json"
  },
  .... property map resources ....
  "ia-property-map" : {
    "uri" : "http://alto.example.com/propmap/full/inet-ia",
    "media-type" : "application/alto-propmap+json",
    "uses": [ "default-network-map", "alt-network-map" ],
    "capabilities" : {
      "mappings": {
        "ipv4": [ ".ISP", ".ASN" ],
        "ipv6": [ ".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",
    "uses": [ "default-network-map", "alt-network-map" ],
    "capabilities" : {
      "mappings": {
        "ipv4": [ ".ISP", ".ASN", ".country", ".state" ],
```

```
              "ipv6": [ ".ISP", ".ASN", ".country", ".state" ]
            }
          }
        },
        "region-property-map": {
          "uri": "http://alto.example.com/propmap/lookup/region",
          "media-type": "application/alto-propmap+json",
          "accepts": "application/alto-propmapparams+json",
          "uses" : [ "default-network-map", "alt-network-map" ],
          "capabilities": {
            "mappings": {
              "default-network-map.pid": [ ".region" ],
              "alt-network-map.pid": [ ".ASN" ],
            }
          }
        },
        "ip-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", "alt-network-map" ],
          "capabilities" : {
            "mappings": {
              "ipv4": [ "default-network-map.pid",
                        "alt-network-map.pid" ],
              "ipv6": [ "default-network-map.pid",
                        "alt-network-map.pid" ]
            }
          }
        },
        "legacy-endpoint-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",
                             "alt-network-map.pid" ]
          }
        },
        "ane-dc-property-map": {
          "uri" : "http://alto.example.com/propmap/lookup/ane-dc",
          "media-type" : "application/alto-propmap+json",
          "accepts": "application/alto-propmapparams+json",
          "capabilities": {
            "mappings": {
              ".ane" : [ "storage-capacity", "ram", "cpu" ]
            }
          },
```

```
      }
    }
```

Figure 8: Example IRD

10.4. Full Property Map Example

The following example uses the properties and IRD defined above in Section 10.3 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

{
  "meta": {
    "dependent-vtags": [
      {"resource-id": "default-network-map",
       "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"},
      {"resource-id": "alt-network-map",
       "tag": "c0ce023b8678a7b9ec00324673b98e54656d1f6d"}
    ]
  },
  "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"}
  }
}
```

10.5. 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

{
  "meta": {
    "dependent-vtags": [
      {"resource-id": "default-network-map",
       "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"},
      {"resource-id": "alt-network-map",
       "tag": "c0ce023b8678a7b9ec00324673b98e54656d1f6d"}
    ]
  },
  "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"}
  }
}
```

10.6. 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 explicitly listed in the request. 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

{
  "meta": {
    "dependent-vtags": [
      {"resource-id": "default-network-map",
       "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"},
      {"resource-id": "alt-network-map",
       "tag": "c0ce023b8678a7b9ec00324673b98e54656d1f6d"}
    ]
  },
  "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"}
  }
}
```

10.7. Filtered Property Map Example #3

The following example uses the filtered property map resource to request the "default-network-map.pid" property and the "alt-network-map.pid" property for a set of IPv4 addresses and prefixes.

Note that the entity "ipv4:192.0.3.0/27" is decomposed into two entities "ipv4:192.0.3.0/28" and "ipv4:192.0.3.16/28", as they have different "default-network-map.pid" property values.

```
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",
                "ipv4:192.0.2.0/27",
                "ipv4:192.0.3.0/27" ],
  "properties" : [ "default-network-map.pid",
                   "alt-network-map.pid ]
}

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

{
  "meta": {
    "dependent-vtags": [
      {"resource-id": "default-network-map",
       "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"},
      {"resource-id": "alt-network-map",
       "tag": "c0ce023b8678a7b9ec00324673b98e54656d1f6d"}
    ]
  },
  "property-map": {
    "ipv4:192.0.2.128":   {"default-network-map.pid": "defaultpid",
                           "alt-network-map.pid": "defaultpid"},
    "ipv4:192.0.2.0/27":  {"default-network-map.pid": "pid2",
                           "alt-network-map.pid": "pid1"},
    "ipv4:192.0.3.0/28":  {"default-network-map.pid": "pid3",
                           "alt-network-map.pid": "pid2"},
    "ipv4:192.0.3.16/28": {"default-network-map.pid": "pid4",
                           "alt-network-map.pid": "pid2"}
  }
}
```

10.8. Filtered Property Map Example #4

Here is an example of using the filtered property map to query the regions for several PIDs in "default-network-map". The "region" property is specified as a "self-defined" property, i.e., the values of this property are defined by this property map resource.

```
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

{
  "entities" : ["default-network-map.pid:pid1",
                "default-network-map.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": {
    "default-network-map.pid:pid1": {
      ".region": "us-west"
    },
    "default-network-map.pid:pid2": {
      ".region": "us-east"
    }
  }
}
```

10.9. Filtered Property Map for ANEs Example #5

The following example uses the filtered property map resource "ane-dc-property-map" to request properties "storage-capacity" and "cpu" on several ANEs defined in this property map.

```
POST /propmap/lookup/ane-dc 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" : [".ane:dc21",
                ".ane:dc45.srv9",
                ".ane:dc6.srv-cluster8"],
  "properties" : [ "storage-capacity", "cpu"]
}

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

{
  "meta" : {
  },
  "property-map": {
    ".ane:dc21":
      {"storage-capacity" : 40000 Gbytes, "cpu" : 500 Cores},
    ".ane:dc45.srv9":
      {"storage-capacity" : 100 Gbytes, "cpu" : 20 Cores},
    ".ane:dc6.srv-cluster8":
      {"storage-capacity" : 6000 Gbytes, "cpu" : 100 Cores}
  }
}
```

11. 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 should then 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.

12. IANA Considerations

This document defines additional application/alto-* media types. It defines an ALTO Entity Domain Type Registry that extends the ALTO Address Type Registry defined in [RFC7285]. It also defines an ALTO Entity Property Type Registry that extends the ALTO endpoint property registry defined in [RFC7285].

12.1. application/alto-* Media Types

This document registers two additional ALTO media types, listed in Table 1.

Type	Subtype	Specification
application	alto-propmap+json	Section 7.1
application	alto-propmapparams+json	Section 8.3

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:
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).

12.2. ALTO Entity Domain Type Registry

This document requests IANA to create and maintain the "ALTO Entity Domain Type Registry", listed in Table 2.

Identifier	Entity Identifier Encoding	Hierarchy & Inheritance	Media Type of Defining Resource
ipv4	See Section 6.1.1	See Section 6.1.3	application/alto-networkmap+json
ipv6	See Section 6.1.2	See Section 6.1.3	application/alto-networkmap+json
pid	See Section 6.2	None	application/alto-networkmap+json

Table 2: ALTO Entity Domain Types

This registry serves two purposes. First, it ensures uniqueness of identifiers referring to ALTO entity domain types. Second, it states the requirements for allocated entity domain types.

12.2.1. Consistency Procedure between ALTO Address Type Registry and ALTO Entity Domain Type Registry

One potential issue of introducing the "ALTO Entity Domain Type Registry" is its relationship with the "ALTO Address Types Registry" already defined in Section 14.4 of [RFC7285]. In particular, the entity identifier of a type of an entity domain registered in the "ALTO Entity Domain Type 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 Type 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 type is registered in the ALTO Entity Domain Type Registry.

* If an ALTO entity domain type 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 type in a future document:

* Whether the ALTO Address Type Registry contains an address type that can be used as an identifier for the candidate entity domain type identifier. This has been done for the identifiers "ipv4" and "ipv6" of Table 2.

* Whether the candidate entity domain type identifier can potentially be an endpoint address type, as defined in Sections 2.1 and 2.2 of [RFC7285].

When a new ALTO entity domain type is registered, the consistency with the ALTO Address Type Registry MUST be ensured by the following procedure:

* Test: Do corresponding entity domain type identifiers match a known "network" address type?

  - If yes (e.g., cell, MAC or socket addresses):

    o Test: Is such an address type present in the ALTO Address Type Registry?

      + If yes: Set the new ALTO entity domain type identifier to be the found ALTO address type identifier.

      + If no: Define a new ALTO entity domain type identifier and use it to register a new address type in the ALTO Address Type Registry following Section 14.4 of [RFC7285].

    o Use the new ALTO entity domain type identifier to register a new ALTO entity domain type in the ALTO Entity Domain Type Registry following Section 12.2.2 of this document.

  - If no (e.g., pid name, ane name or country code): Proceed with the ALTO Entity Domain Type registration as described in Section 12.2.2.

12.2.2. ALTO Entity Domain Type Registration Process

New ALTO entity domain types are assigned after IETF Review [RFC5226] to ensure that proper documentation regarding the new ALTO entity domain types and their security considerations has been provided. RFCs defining new entity domain types SHOULD indicate how an entity in a registered type of 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 types follow the same procedure.

Registered ALTO entity domain type identifiers MUST conform to the syntactical requirements specified in Section 5.1.2. Identifiers are to be recorded and displayed as strings.

Requests to the IANA to add a new value to the Entity Domain Type registry MUST include the following information:

* Identifier: The name of the desired ALTO entity domain type.

* Entity Identifier Encoding: The procedure for encoding the identifier of an entity of the registered domain type as an EntityID (see Section 5.1.3). If corresponding entity identifiers of an entity domain type 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]. To define 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.

* Inheritance: If entities can inherit property values from other entities, the procedure for determining that inheritance.

* Media type of defining information resource: some entity domain types allow an entity domain name to be combined with an information resource name to define a resource-specific entity domain. Such an information resource is called "defining information resource". In this case, the authorized media type of the defining information resources MUST be specified in the document defining the entity domain type. When an entity domain type allows combinations with defining resources, this must be indicated and the conditions fully specified in the document. The defining information resource for an entity domain type is the one that:

- has an entry in the IRD,

- defines these entities,

- does not use another information resource that defines these entities,

- defines and exposes entity identifiers that are all persistent.

- has a unique media type equal to the one specified in the document defining the entity domain type.

* Knowing the media type of the defining information resource is useful when Servers indicate resource specific entity domains in the property map capabilities. Clients need to know if the combination of information resource type and entity domain type is allowed. See also, Section 4.6 and Section 5.1 for more information.

* Mapping to ALTO Address Type: A boolean value to indicate if the entity domain type can be mapped to the ALTO address type with the same identifier.

* 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 cognizant 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.

12.3. ALTO Entity Property Type Registry

This document requests IANA to create and maintain the "ALTO Entity Property Type Registry", listed in Table 3.

This registry extends the "ALTO Endpoint Property Type Registry", defined in [RFC7285], in that a property type is defined on one or more entity domains, rather than just on IPv4 and IPv6 Internet address domains. An entry in this registry is an ALTO entity property type defined in Section 5.2.1. Thus, a registered ALTO entity property type identifier MUST conform to the syntactical requirements specified in that section.

Identifier	Intended Semantics	Media Type of Defining Resource
pid	See Section 7.1.1 of [RFC7285]	application/alto-networkmap+json

Table 3: ALTO Entity Property Types.

Requests to the IANA to add a new value to the registry MUST include the following information:

* Identifier: The unique id for the desired ALTO entity property type. The format MUST be as defined in Section 5.2.1 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.

* Media type of defining information resource: when the property type allows values to be defined relatively to a given information resource, the latter is referred to as the "defining information resource", see also description in Section 4.7. The media type of the possibly used defining information resource MUST be indicated.

* Security Considerations: ALTO entity properties expose information to ALTO clients. ALTO service providers should be cognizant of the security ramifications related to the exposure of an entity property.

In security considerations, the request should also discuss the sensitivity of the information, and why it is required for ALTO-based operations. Regarding this discussion, the request SHOULD follow the recommendations of Section 14.3. ALTO Endpoint Property Type Registry in [RFC7285].

This document requests registration of the identifier "pid", listed in Table 3. Semantics for this property are documented in Section 7.1.1 of [RFC7285]. No security issues related to the exposure of a "pid" identifier are considered, as it is exposed with the Network Map Service defined and mandated in [RFC7285].

13. Acknowledgments

The authors would like to thank Dawn Chen (Tongji University), and Shenshen Chen (Tongji/Yale University) for their contributions to earlier drafts. Thank you also to Qiao Xiang (Yale University), Shawn Lin, Xin Wang and Vijay Gurbani (Vail systems) for fruitful discussions. Last, big thanks to Danny Perez (University of Campinas) and Luis Contreras (Telefonica) for their substantial review feedback and suggestions to improve this document.

14. References

14.1. Normative References

[ISO3166-1]
The International Organization for Standardization, "Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes, ISO 3166-1:2013", 2013.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4632] Fuller, V. and T. Li, "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan", BCP 122, RFC 4632, DOI 10.17487/RFC4632, August 2006, <https://www.rfc-editor.org/info/rfc4632>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <https://www.rfc-editor.org/info/rfc7011>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

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

[RFC7921] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", RFC 7921, DOI 10.17487/RFC7921, June 2016, <https://www.rfc-editor.org/info/rfc7921>.

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

14.2. Informative References

[draft-gao-alto-fcs]
Zhang, J., Gao, K., Wang, J., Xiang, Q., and Y. Yang, "ALTO Extension: Flow-based Cost Query", 13 September 2016.

[draft-ietf-alto-cdni-request-routing-alto]
Roome, J., Yang, Y., Ma, K., Peterson, J., and J. Zhang, "Content Delivery Network Interconnection (CDNI) Request Routing: CDNI Footprint and Capabilities Advertisement using ALTO (work in progress)", 2020.

[I-D.ietf-alto-path-vector]
Gao, K., Lee, Y., Randriamasy, S., Yang, Y., and J. Zhang, "ALTO Extension: Path Vector", Work in Progress, Internet-Draft, draft-ietf-alto-path-vector-09, 3 November 2019, <http://www.ietf.org/internet-drafts/draft-ietf-alto-path-vector-09.txt>.

Appendix A. Features introduced with the Entity Property Maps extension

The Entity Property Maps extension described in this document introduces a number of features that are summarized in table below. The first column provides the name of the feature. The second column provides the section number of this document that gives a high level description of the feature. The third column provides the section number of this document that gives a normative description relating to the feature, when applicable.

Feature	High-level description	Related normative description
Entity	Section 3.1	Section 5.1.3
Entity domain (ED)	Section 3.2	
Entity domain type	Section 3.2.1	Section 5.1.1
Entity domain name	Section 3.2.2	Section 5.1.2
Entity property (EP) type	Section 3.3	Section 5.2, Section 5.2.1, Section 5.2.2, Section 5.2.3
Entity property map	Section 3.4	Section 7, Section 8
Resource-specific ED name	Section 4.2	Section 5.1.2, Section 5.1.2.1
Resource-specific EP value	Section 4.3	Section 5.2.3
Entity Hierarchy and property inheritance	Section 4.4	Section 5.1.4
Defining information resource	Section 4.6, Section 4.7	Section 12.2.2, Section 12.3

Table 4: Features introduced with ALTO Entity Property Maps

Authors' Addresses

Wendy Roome
Nokia Bell Labs (Retired)
124 Burlington Rd
Murray Hill, NJ 07974
United States of America

Phone: +1-908-464-6975
Email: wendy@wdroome.com

Sabine Randriamasy
Nokia Bell Labs
Route de Villejust
91460 NOZAY
France

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

Y. Richard Yang
Yale University
51 Prospect Street
New Haven, CT 06511
United States of America

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

Kai Gao
Sichuan University
No.24 South Section 1, Yihuan Road
Chengdu
610000
China

Email: kaigao@scu.edu.cn

ALTO WG D. Lachos
Internet-Draft C. Rothenberg
Intended status: Informational Unicamp
Expires: January 14, 2021 July 13, 2020

ALTO-based Broker-assisted Multi-domain Orchestration
draft-lachosrothenberg-alto-brokermdo-04

Abstract

Evolving networking scenarios (e.g., 5G) demand new multiple administrative domain (aka multi-domain) orchestration models. This document proposes a new broker-plane approach working on top of per-domain management and orchestration functions to coordinate the delivery of a multi-operator End-to-End Network Service (E2ENS). This proposed design resorts to the Application-Layer Traffic Optimization (ALTO) protocol to offer topology and resource information from different administrative domains. The ALTO services with the proposed protocol extension offer aggregated views on various types of resources contributing to a more simple and scalable solution.

Table of Contents

1. Introduction

Envisioned 5G network architectures and related service models consider broader cooperation between stakeholders in order to provide flexible multi-operator multi-domain services. These multi-provider orchestration operations will require the information exchange across Multi-domain Orchestrators (MdOs). The key information to be exchanged between MdOs includes the abstract network topology, resource availability (e.g., CPUs, Memory, and Storage) and capability (e.g., supported network functions).

This document presents a federation networking paradigm where a broker-plane works on top of the management and orchestration plane to assist and coordinate the creation of an End-to-End Network Service (E2ENS) spanning over multi-operator multi-domain networks. Our design resorts to the Application-Layer Traffic Optimization (ALTO) protocol [RFC7285] to address the lack of abstractions to discover and adequately represent in confidentiality-preserving fashion the resource and topology information from different administrative domains. Moreover, this draft introduces an extension to the ALTO base protocol for inter-domain connectivity information discovery.

2. Terminology

We use the following definitions, as established in [ETSI-NFV-DEF]:

Administrative Domain: Collection of systems and networks operated by a single organization or administrative authority.

Network Function (NF): Functional block within a network infrastructure that has well-defined external interfaces and well-defined functional behaviour.

Network Functions Virtualisation (NFV): The principle of separating network functions from the hardware they run on by using virtual hardware abstraction.

NF Forwarding Graph: (NFFG): Graph of logical links connecting NF nodes for the purpose of describing traffic flow between these network functions.

Network Service Orchestration (NSO): Function responsible for network service lifecycle management.

Resource Orchestration (RO): Function responsible for global resource management governance.

Our proof of concept implementation follows the architectural proposal of the 5GEx project [H2020.5GEX]. Some additional 5GEx terms commonly used in this document are defined below:

Domain Orchestrator (DO): Performs Resource Orchestration and/or Service Orchestration within the same administrative domain.

Multi-domain Orchestrator (MdO): Coordinates resource and/or service orchestration at multi-domain level, where multi-domain may refer to multiple DOs or multiple administrative domains.

Resource Topology (RT): Functional module that is responsible for keeping an updated global view of the underlying infrastructure topology exposed by DOs.

Service Graph (SG): A high-level data model for defining flexible network services (including traffic steering primitives).

Service Access Point (SAP): A named/tagged port supporting stitching (service to service, domain to domain, etc.)

3. Scope

Envisioned 5G scenarios are expected to work not only with heterogeneous technologies but also across different network operators. Many ongoing standardization activities and research projects are addressing the multi-provider multi-domain orchestration challenges under different approaches.

For example, within the IETF, [RFC8459] proposes a hierarchical Service Function Chaining (SFC) for multiple domains under the same administrative entity, and the document "Hybrid Hierarchical Multi-Domain SFC [DRAFT-HHSFC] describes SFC crossing different domains owned by various organizations or by a single organization with administration partitions. In the NFVRG, the draft "Multi-domain Network Virtualization" [DRAFT-MD-VIRT] envisions a complete E2E logical network as stitching services offered by multiple domains from multiple providers. Another initiative is the ETSI Industry Specification Group for Network Functions Virtualization (ETSI NFV ISG), the document [ETSI-NFV-IFA028] reports different NFV MANO architectural approaches with use cases related to network services provided using multiple administrative domains.

In case of research projects, [H2020.5GEX] [H2020-5G-TRANSFORMER] seek to integrate multiple administrations and technologies through the collaboration between operations. Other studies are envisioned to use a centralized approach, where each domain advertises its capabilities to a federation layer which will act as a broker [VITAL][T-NOVA]. The proposed architecture in [ICAF] allows the creation of cloud services from different administrative domains, however, it is not related to the provisioning of NFV-based cross-domain network services.

All such proposals described above envision the potential introduction of new business model approaches, including federation models [PPP-5:2013] among administrative domains. In this context, this document considers each network operator involved in the community advertises its abstracted capabilities (e.g., software/hardware resources, physical/virtual network functions, etc.) to a broker (i.e., 3rd party). This broker, in its turn, provides or assists coordinate E2E network services spanning multi-domain networks.

4. Problem Statement and Challenges

The provision of a complete E2E network service requires chaining services provided by multiple network operators with multiple technologies. In this multi-domain environment, the orchestration process will require an advertisement mechanism through which individual domains can describe their capabilities, resources, and VNFs in an interoperable manner. Moreover, a discovery mechanism is also necessary so that source domains can obtain candidate domains (with the corresponding connectivity information) which can provide a part of the service and/or slice in an E2ENS requirement.

In order that the advertising and discovery process works in a proper way, a number of challenges can be identified:

Lack of Abstractions: Multiple vendors with heterogeneous technologies need an information model to adequately represent in confidentiality-preserving fashion the resource and topology information.

Scalability: Involves the distribution of topology and resource information in a peer-to-peer fashion (MdO-to-MdO). Multi-operator multi-domain environments where the information distribution is advertised in a peer-to-peer model scales linearly. It means more MdO interconnections one has, the more it "costs" to distribute.

Flexibility: Considers that a distributed approach does not allow domains without physical infrastructure (e.g., without BGP or BGP-LS) to advertise resource capabilities and networking resources. Such procedures consist in deploying and configuring physical peering points for these domains.

Complexity: Refers to the discovery mechanism to pre-select candidate domains, accounting for resources and capabilities, necessary for an E2E network service deployment. An intrinsic complexity exists in the process of assembling, logically organizing, and enabling abstraction views of different resources and capabilities in multi-domain scenarios.

5. Proposed Approach

The primary design goal for ALTO-based Broker-assisted Multi-domain Orchestration is to discover resource and topology information from different administrative domains involved in the federation, while also safeguarding the privacy and autonomy of every domain.

In the architectural proposal shown in Figure 1, a broker component is conceived to be working as coordinator of a set of MdOs. In particular, the broker-assisted design consists of the following key components: (i) Inter-domain Resource (IdR), (ii) Inter-domain Topology (IdT), and (iii) ALTO Server.

```
                              BROKER COMPONENT
              +--------------------------------------------+
              |                                            |
              |             +-----------------+            |
              |             |                 |            |
          XXXXXXXXXXXXXXXXXXX ALTO SERVER(s)  |            |
          X   |             |                 +            |
          X   |             +----------------+\            |
          X   |            /                   \           |
          X   |           /                     \          |
          X   |  +------+/+-------+     +---------------+  |
          X   |  | Inter-domain   |     | Inter-domain  |  |
          X   |  | Topology (IdT) |     | Resource (IdR)|  |
          X   |  +-^------^-------+     +---^-------^---+  |
          X   |    .      .                 *       *      |
          X   +----.------.-----------------*-------*------+
          X        .      .                 *       *
          X        .      .                 *       *
       +--X--------.---+*********************       *
       |               |  .                         *
       |               |  .............+------------*---+
       |     MdO1      |               |                |
       |               +<------------->+                +---+
       +---------------+               |      MdO2      |   |
                                       |                |   |
   Legend:                             +-+--------------+   |
   XXX ALTO Protocol                     |                  |
   ... BGP/BGP-LS/REST                   |      MdO(n)      |
   *** UNIFY/TOSCA/ETSI-NFV              +------------------+
```

Figure 1: Broker-assisted Multi-operator Network Architecture

5.1. Inter-domain Resource (IdR) Component

It creates a hierarchical database that contains inter-domain resource information such as resource availability (i.e., CPU, memory, and storage), Virtual Network Functions (VNFs) and Physical Network Functions (PNFs) supported and Service Access Points (SAPs) to access those resources. UNIFY [UNIFY.NFFG], TOSCA [TOSCA], ETSI-NFV [ETSI-NFV-MANO], among other data models can be used to create the interface between IdR and MdOs.

5.2. Inter-domain Topology (IdT) Component

A hierarchical TED (Traffic Engineering Database) that contains inter-domain network topology information including additional key parameters (e.g., throughput and latency of links). This information can be retrieved from each MdO through BGP-LS or REST interfaces.

5.3. ALTO Server Functionalities

The ALTO server component is the core of the broker layer. Multiple logically centralized ALTO servers use the information collected from IdR and IdT components to create and provide abstract maps with a simplified view, yet enough information about MdOs involved in the federation. This information includes domain-level topology, resource availability (i.e., CPU, memory, and storage), PNF/VNF capabilities, and SAPs.

As an ALTO client, each MdO sends ALTO service queries to the ALTO server. This server provides aggregated inter-domain information exposed as set ALTO base services defined in [RFC7285], e.g., Network Map, Cost Map and ALTO extension services, e.g., Property Map [DRAFT-PM], Multi-Cost Map [RFC8189], Path Vector [DRAFT-PV].

For example, when a source MdO receives a customer service request, it checks whether or not it can deliver the full service. If it is unable to do so, the MdO consumes from the ALTO Server the Property Map service to have a clear global view of the resource information offered by other MdOs. This information allows discovering which candidate MdOs may be contacted to deliver the remaining requirements of a requested end-to-end service deployment. The connectivity information among discovered MdOs can be retrieved by a Cost Map service, responding, for instance, a path vector with the AS-level topology distance between the source MdO and candidate MdOs.

5.4. Filtered Cost Map Extension

The ALTO server MUST provide connectivity information for every SG link in the SG path for an E2E requirement. This information is the AS-level topology distance in the form of path vector, and it includes all possible ways for each (source node, destination node) pair in the SG link.

In this section, we introduce a non-normative overview of the Filtered Cost Map defined in Section 6.1 of [DRAFT-PV] [1].

The specifications for the "Media Types", "HTTP method", "Capabilities" and "Uses" (described in Section 6.1 of [DRAFT-PV] [2]) are unchanged.

5.4.1. Accept Input Parameters

The ReqFilteredCostMap object in Section 6.1.2 of [DRAFT-PV] [3] is extended as follow:

```
object {
  NFFG sg;
} ReqFilteredCostMap;

object {
  JSONString nfs<1..*>;
  JSONString saps<1..*>;
  NextHops sg_links<1..*>;
  REQs reqs<1..*>;
} NFFG;

object {
  JSONNumber id;
  JSONString src-node;
  JSONString dst-node;
} NextHops;

object {
  JSONString id;
  JSONString src-node;
  JSONString dst-node;
  JSONNumber sg-path<1..*>;
} REQs;
```

sg: If present, the ALTO Server MUST allow the request input to include an SG with a formatted body as an NFFG object. An NFFG object contains NFs, SAPs, SG links representing logical connections between NFs, SAPs or both and E2E requirements as a list of ids of SG links.

It is worth noting that further versions of this draft will define a more elaborated NFFG object to support extended parameters such as monitoring parameters, resource requirements, etc.

5.4.2. Response

If the ALTO client includes the path vector cost mode in the "cost-type" or "multi-cost-types" field of the input parameter, the response for each SG link in each E2E requirement MUST be encoded as a JSONArray of JSONArrays of JSONStrings. Anyone of the sub-arrays indicates a potential candidate path calculated as the per-domain topological distance corresponding to the amount of traversing domains.

Moreover, as defined in Section 6.3.6 of [DRAFT-PV] [4], If an ALTO client sends a request of the media type "application/alto-costmapfilter+json" and accepts "multipart/related", the ALTO server MUST provide path vector information along with the associated Property Map information (e.g., entry points of the corresponding foreign domains), in the same body of the response.

Section 5.5.2 gives an example of the Filtered Cost Map query and the corresponding responses.

5.5. Examples of Message Exchange

This section list a couple of examples of the Property Map and Filtered Cost Map queries and the corresponding responses. These responses are based on the information in Table 1 and Table 2 of a use case implementation described in Appendix A.

5.5.1. Property Map Service

In this example, the ALTO client wants to retrieve the entire Property Map for PID entities with the "entry-point", "cpu", "mem", "storage", "port" and "nf" properties.

o HTTP Request:

```
GET /propmap/full/inet-ucmspn HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json
```

o HTTP Response:


```
HTTP/1.1 200 OK
Content-Length: ###
Content-Type: application/alto-propmap+json
{
  "property-map": {
      "pid:AS1": {
          "entry-point": [ "http://172.25.0.10:8888/escape" ],
          "cpu": [ "50.0" ],
          "mem": [ "60.0" ],
          "storage": [ "70.0" ],
          "port": [ "SAP1" ],
          "nf": [ "NF1", "NF3" ]
      },
      "pid:AS2": {
          "entry-point": [ "http://172.26.0.10:8888/escape" ],
          "cpu": [ "10.0" ],
          "mem": [ "20.0" ],
          "storage": [ "30.0" ],
          "nf": [ "NF2" ]
      },
      "pid:AS3": {
          "entry-point": [ "http://172.27.0.10:8888/escape" ],
          "cpu": [ "80.0" ],
          "mem": [ "90.0" ],
          "storage": [ "100.0" ],
          "port": [ "SAP2" ],
          "nf": [ "NF1", "NF3" ]
      }
  }
}
```


5.5.2. Filtered Cost Map Service

The following example uses the Filtered Cost Map service to request the path vector for a given E2E requirement. The SG request information in Table 2 is used to describe the service, and it is composed of three NFs (NF1, NF2, and NF3) and two SAPs (SAP1 and SAP2). Links connecting the NFs and SAPs ("sg_links" tag) are also included, followed by an E2E requirement ("reqs" tag) with information about the order in which NFs are traversed from SAP1 to SAP2.

Note that the request accepts "multipart/related" media type. This means the ALTO server will include associated property information in the same response.

o HTTP Request:

```
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

{
  "cost-type": {
    "cost-mode": "array",
    "cost-metric": "ane-path"
  },
  "sg": {
    "nfs": [ "NF1", "NF2", "NF3" ],
    "saps": [ "SAP1", "SAP2" ],
    "sg_links":[
      {
        "id": 2,
        "src-node": "SAP1",
        "dst-node": "NF1",

      },
      {
        "id": 2,
        "src-node": "NF1",
        "dst-node": "NF2",
      },
      {
        "id": 3,
        "src-node": "NF2",
        "dst-node": "NF3",
      },
      {
        "id": 4,
        "src-node": "NF3",
        "dst-node": "SAP2",
      }
```

```
          ],
          "reqs": [
            {
              "id": 1,
              "src-node": "SAP1",
              "dst-node": "SAP2",
              "sg-path": [ 1, 2, 3, 4 ]
            }
          ]
        }
      }
```


o HTTP Response: The ALTO server returns connectivity information for the E2E requirement provided by the ALTO Client request of the above example.

For each SG link in the E2E requirement (SAP1->NF1, NF1->NF2, NF2->NF3, NF3->SAP2), the ALTO server returns sub-arrays indicating potential candidate paths calculated as the AS-level topological distance corresponding to the amount of traversing domains.

Also, the response includes Property Map information for each element in the path vector. In this case, it is retrieved a Property Map with the "entry-point" property, i.e., the URL of the MdO entry point for the corresponding network.

```
HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=example

--example
Content-Type: application/alto-endpointcost+json
```


```
{
  "meta": {
    "cost-type": {
       "cost-mode": "array",
       "cost-metric": "ane-path"
     },
  },

  "cost-map": {
    "SAP1": {
      "SAP2": {
```

```
               "SAP1": {
                   "NF1": [
                     [ "AS1" ], [ "AS1", "AS2", "AS3" ]
                   ]
               },
               "NF1": {
                   "NF2": [
                     [ "AS1", "AS2" ], [ "AS3", "AS2" ]
                   ]
               },
               "NF2": {
                   "NF3": [
                     [ "AS2", "AS1" ], [ "AS2", "AS3" ]
                   ]
               },
               "NF3": {
                   "SAP2": [
                     [ "AS1", "AS2", "AS3" ], [ "AS3" ]
                   ]
               }
           }
         }
       }
     }

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

     {
       "property-map": {
         "pid:AS1": { "entry-point": "http://172.25.0.10:8888/escape" },
         "pid:AS2": { "entry-point": "http://172.26.0.10:8888/escape" },
         "pid:AS3": { "entry-point": "http://172.27.0.10:8888/escape" }
       }
     }

     --example--
```


6. Discussion

In this section, we analyze the benefits and open issues in our broker-assisted architecture.

6.1. Benefits

The broker-assisted orchestration has numerous benefits, such as:

- o Avoid the distribution of topology and resource information in a peer-to-peer fashion (MdO-to-MdO)
- o The (abstracted) information and offered resources, services are maintained in each local MdO.
- o Allow domains without physical infrastructure (hence without BGP or BGP-LS) to advertise their capabilities.
- o An ALTO-based privacy-preserving information model to provide computing, storage, and networking resource info.
- o An MdO discovery method to determine the underlying network graph and a potential set of paths before bilateral negotiation between MdOs is started.

6.2. Open Issues

Although the broker-assisted information exchange has several advantages, it also raises some questions which we try to answer from our lessons learned.

- o What kind of organization will manage and support the operation of a broker entity? If a broker is used to exchange information, then how does one ensure that the data delivered amongst the operators by this 3rd party has not been changed?
  - * The broker entity must be trusted by each operator since it stores and handles sensitive information. For example, future deployment of SDN at IXPs can be used as a trusted third-party platform to support rich business models between different operators [DRAFT-HHSFC].
- o In the case of peer-to-peer information exchange model, an MdO failure concerns only the domain where the failure occurs, other peers can perform the information exchange without any limitation. However, If any error occurs in the broker entity the information exchange among all involved ASes will be impacted. How avoid this single point of failure?
  - * The broker entity maintains a centralized database. Local restoration/replication options may be applied.

- o The MdO information exchange depends on the policies. Operators have a preference to share a different view about its compute and network resources towards different operators. For example, a detailed view for the operators that are belonging to same operator group and a high-level information towards the other operators. How is the fine-grained/coarse-grained information exchange handled?.

  * It requires much more complex database handling and information exchange with the MdOs depending on the policies.

7. IANA Considerations

This document includes no request to IANA.

8. Security Considerations

TBD.

9. Acknowledgments

This work is supported by the Innovation Center of Ericsson S.A., Brazil (grant agreement UNI.64).

Thank you to Robert Szabo (Ericsson Research, Hungary) for the contribution and substantial feedback and suggestions in this document.

Many thanks to Richard Yang, Dawn Chan, Jensen Zhang, Shawn Lin, Qiao Xiang, Sabine Randriamasy for their feedback on this draft.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://xml.resource.org/public/rfc/html/rfc2119.html>.

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

[RFC8189] Randriamasy, S., Roome, W., and N. Schwan, "Multi-Cost Application-Layer Traffic Optimization (ALTO)", RFC 8189, DOI 10.17487/RFC8189, October 2017, <https://www.rfc-editor.org/info/rfc8189>.

[RFC8459] Dolson, D., Homma, S., Lopez, D., and M. Boucadair, "Hierarchical Service Function Chaining (hSFC)", RFC 8459, DOI 10.17487/RFC8459, September 2018, <https://www.rfc-editor.org/info/rfc8459>.

10.2. Informative References

[DRAFT-HHSFC]
Li, G., Li, G., Xu, Q., Zhou, H., and B. Feng, "Hybrid Hierarchical Multi-Domain Service Function chaining", draft-li-sfc-hhsfc-08 (work in progress), March 2020.

[DRAFT-MD-VIRT]
Bernardos, C., Contreras, L., Vaishnavi, I., Szabo, R., Li, X., Paolucci, F., Sgambelluri, A., Martini, B., Valcarenghi, L., Landi, G., Andrushko, D., and A. Mourad, "Multi-domain Network Virtualization", draft-bernardos-nfvrg-multidomain-05 (work in progress), September 2018.

[DRAFT-PM]
Roome, W., Randriamasy, S., Yang, Y., Zhang, J., and K. Gao, "Unified Properties for the ALTO Protocol", draft-ietf-alto-unified-props-new-11 (work in progress), March 2020.

[DRAFT-PV]
Gao, K., Randriamasy, S., Yang, Y., and J. Zhang, "ALTO Extension: Path Vector", draft-ietf-alto-path-vector-10 (work in progress), March 2020.

[ETSI-NFV-DEF]
ETSI, "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV V1.3.1", Jan 2018, <https://docbox.etsi.org/isg/nfv/open/Publications_pdf/Specs-Reports/NFV%20003v1.3.1%20-%20GR%20-%20Terminology%20for%20Main%20Concepts%20in%20NFV.pdf>.

[ETSI-NFV-IFA028]
ETSI, "Report on architecture options to support multiple administrative domains V3.1.1", Jan 2018, <http://www.etsi.org/deliver/etsi_gr/NFV-IFA/001_099/028/03.01.01_60/gr_NFV-IFA028v030101p.pdf>.

[ETSI-NFV-MANO]
ETSI, "Network Functions Virtualisation (NFV) Management and Orchestration V1.1.1", Dec 2014, <http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf>.

[H2020-5G-TRANSFORMER]
H2020, "5G-Transformer -- 5G Mobile Transport Platform for Vertical", 2017, <http://5g-transformer.eu/>.

[H2020.5GEX]
Bernardos, C., Dugeon, O., Galis, A., Morris, D., Simon, C., and R. Szabo, "5G Exchange (5GEx)--Multi-domain Orchestration for Software Defined Infrastructures", focus vol. 4, no.5, p.2, 2015.

[H2020.5GEX.ESCAPE]
5GEx Project, "ESCAPE: Extensible Service ChAin Prototyping Environment", 2015, <https://github.com/5GExchange/escape>.

[ICAF] Demchenko, Y., Makkes, M., Strijkers, R., Ngo, C., and C. Laat, "Intercloud Architecture Framework for Heterogeneous Multi-Provider Cloud based Infrastructure Services Provisioning", International Journal of Next-Generation Computing vol. 4, no.2, 2013.

[PPP-5:2013]
5G-PPP, "Advanced 5G Network Infrastructure for the Future Internet", 2013, <https://5g-ppp.eu/wp-content/uploads/2014/02/Advanced-5G-Network-Infrastructure-PPP-in-H2020_Final_November-2013.pdf>.

[T-NOVA] FP7 project T-NOVA, "T-NOVA Project, Network Functions as a Service over Virtualised Infrastructures", 2014, <http://www.t-nova.eu/>.

[TELEFONICA.NET.TOPO]
Telefonica I+D, "Netphony-Topology", 2016, <https://github.com/telefonicaid/netphony-topology>.

[TOSCA] OASIS, "TOSCA: Topology and Orchestration Specification for Cloud Applications V1.0", 2013, <http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf>.

[UNIFY.NFFG]
UNIFY Deliverable D3.2a, "Network Function Forwarding Graph specification", 2015, <http://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/UNIGY_D3.2a_NFFG%20Specification.pdf>.

[VITAL] VITAL PROJECT H2020, "VITAL -- VIrtualized hybrid satellite-TerrestriAl systems for resilient and fLexible future networks", 2015, <http://www.ict-vital.eu/>.

10.3. URIs

[1] https://tools.ietf.org/html/draft-ietf-alto-path-vector-02#section-6.1

[2] https://tools.ietf.org/html/draft-ietf-alto-path-vector-02#section-6.1

[3] https://tools.ietf.org/html/draft-ietf-alto-path-vector-02#section-6.1.2

[4] https://tools.ietf.org/html/draft-ietf-alto-path-vector-02#section-6.3.6

Appendix A. Proof of Concept Use Case Implementation

A strawman use case scenario has been implemented following the architectural proposal of the 5GEx project [H2020.5GEX]. It refers to an E2ENS orchestration involving three administrative domains. For reproducibility purposes, all supporting codes are publicly available in our research group repository: https://intrig.dca.fee.unicamp.br:8865/intrig-unicamp/alto-based-broker-assisted-mdo

As shown in Figure 2, each administrative domain has an MdO (MdO-AS1, MdO-AS2, and MdO-AS3) to coordinate resource and/or service orchestration at multi-operator level via interface I2 APIs. For the orchestration within the same administrative domain, each MdO uses emulated DOs with emulated I3 interfaces, since no data-plane is present. DOs use static configuration files to load local information about resources (I3-RC) and topology (I3-RT). The different MdO components are based on existing open source tools such as ESCAPE [H2020.5GEX.ESCAPE] (Service/Resource Orchestrator) and Netphony-topology [TELEFONICA.NET.TOPO] (Resource Topology) and run in Docker containers on a single computer. Besides, MdOs expose I1 interfaces to the tenants who request services and/or slices which should follow a Network Function Forwarding Graph (NFFG) [UNIFY.NFFG] format.

In case of the broker layer, the IdR and IdT components use a UNIFY Virtualizer API [UNIFY.NFFG] (broker-based I2-RC API) and a REST API (broker-based I2-RT API) respectively, in order to create the hierarchical databases. Regarding the IdT, the administrative domain 2 is a transit provider so that the domain-level topology computed is: AS1-AS2-AS3. From the inter-domain information are created the two different ALTO Map Services: (i) Property Map and (ii) Cost Map.

```
                            +----------------------------------------+
                            |       +---------------+    BROKER LAYER|
                       XXXXXXXXXXXXXX  ALTO Server  |                |
                       X    |       +--------+----+-+                |
                       X    |               /      \                 |
                       X    | +-----------+/+--+  +-\-------------+  |
                       X    | | Inter-domain   |  | Inter-domain  |  |
       +-----------+   X    | | Topology (IdT) |  | Resource (IdR)|  |
       | Service   |   X    | +----------------+  +---------------+  |
       | Graph (SG)|   X    +---------^-^----------------^--^--------+
       | Request   |   X              * *    .............  .
       +-----+-----+   X              * *    . ..............
             |         X              * *    . .              MdO-AS3
           I1|         X              * *    . .      +---------------+
             |         X     MdO-AS1  * *    . .      |              |
       +-----|---------X-----------+  * *    . .      |     MdO-AS2  |
       |     |                     |  * *    . .  +---------------+ |
       | +---v-------------------+ |  * *    . .  | +-----------+ | |
       | |                       | |  * *    . .  | |           | | |
       | |  Network Service Orch.| |  * *    . .  | |    NSO    | | |
       | |  (NSO)                | |  * *    . .  | |           | | |
       | +-----------------------+ |  * *    . .  | +-----------+ | |
       |                           |  * *    . .  |               | |
       | +---------+               |  * *    . .  | +---+         | |
       | | Resource........................... .   | |   |         | |
       | | Topoloy |               |  * *      .......RT |         | |
       | | (RT)    | +-----------+ |  * *          | |   |         | |
       | +---------+ |Resource   | |  * *          | +---+   +---+ | |
       |             |Orch.      | |  * **********************   | | |
       |             |(RO)      ******            |         |RO | +-+
       |             +-----------+ |               |         |   | |
       |                           |<------------->|         +---+ |
       +---------------------------+      I2      +-----+---------+
                  /     \                                 |
               I3/       \                                |I3
       +-------+---+  +-----------+                +-----------+
       | Domain    |  | Domain    |                | Domain    |
       | Orch (DO) |  | Orch (DO) |                | Orch (DO) |
       +-----------+  +-----------+                +-----------+
                                            Legend:
                                            XXX ALTO Protocol
                                            ... broker-based I2-RT API
                                            *** broker-based I2-RC API
```

Figure 2: Broker-assisted 5GEx Info Exchange

The Property Map includes property values grouped by Autonomous System (AS). Such values are SAPs, NFs and the 5GEx Entry Point (e.g., the URL of the ESCAPE orchestrator). An example of the Property Map in our prototype is:

	Entry Point	Port SAP	Capabilities	CPU	MEM	Storage	...
AS1	http://...	SAP1	{NF1, NF3}	50	60	70	...
AS2	http://...	-	{NF2}	10	20	30	...
AS3	http://...	SAP2	{NF1, NF3}	80	90	100	...

Table 1: ALTO Property Map

The Cost Map defines a path vector as an array of ASes, representing the AS-level topological distance for a given E2ENS request. Path vector constraints (as described in the Multi-Cost Map [RFC8189]) can be applied to restricts the response to costs that satisfy a list of simple predicates.

Table 2 below shows a brief example of an SG request and its path vector response containing a list of potential providers to be traversed to deliver such service. Every AS path is computed from the inter-domain topology information in the IdT module. In our scenario, MdO-AS2 is a transit provider, so that the domain-level topology map is AS1<->AS2<->AS3.

Service Graph (SG) Request	Path(s) Vector
SAP1->NF1->NF2->NF3->SAP2	1:{AS1:SAP1->AS1:NF1->AS2:NF2->AS3:NF3->AS3:SAP2} 2:{AS1:SAP1->AS1:NF1->AS2:NF2->AS1:NF3->AS2->AS3:SAP2} 3:{AS1:SAP1->AS2->AS3:NF1->AS2:NF2->AS3:NF3-AS3:SAP2} 4:{AS1:SAP1->AS2->AS3:NF1->AS2:NF2->AS1:NF3->AS2->AS3:SAP2}

Table 2: ALTO Cost Map

Authors' Addresses

Danny Alex Lachos Perez
University of Campinas
Av. Albert Einstein 400
Campinas, Sao Paulo 13083-970
Brazil

Email: dlachosp@dca.fee.unicamp.br
URI: https://intrig.dca.fee.unicamp.br/danny-lachos/

Christian Esteve Rothenberg
University of Campinas
Av. Albert Einstein 400
Campinas, Sao Paulo 13083-970
Brazil

Email: chesteve@dca.fee.unicamp.br
URI: https://intrig.dca.fee.unicamp.br/christian/

ALTO S. Randriamasy
Internet-Draft Nokia-Bell-labs
Intended status: Experimental March 09, 2020
Expires: September 10, 2020

ALTO cellular addresses
draft-randriamasy-alto-cellular-adresses-03

Abstract

This draft proposes to use the cellular address format composed of elements as specified by 3GPP and called ECGI. ECGI stands for E-UTRAN Cell Global Identifier and is used in Public Land Mobile Networks based on E-UTRAN.

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].

Table of Contents

1. Introduction

Cellular networks are present in a number of use cases investigated in the ALTO WG and it is useful to specify a format for Cellular addresses. In these cases, Endpoints, PIDs and entities may be cells. In order to specify services such as Network Maps, Cost Maps, Endpoint property or Property Maps, it is necessary to order to specify an ALTO format for Cell addresses.

For the sake of efficiency, a preferred option is to use the cell identifier format as specified by 3GPP [TS 36.300] and called ECGI, as already proposed in [draft-rauschenbach-alto-wireless-access-00] and in other discussions. ECGI stands for E-UTRAN Cell Global Identifier and is used in Public Land Mobile Networks based on E-UTRAN, see [TS 36.331].

The purpose of this document is to be completed by the ALTO WG and in particular:

- Amend and finalize the specification for the ALTO Cell identifier format proposed in the present version,

- define a placeholder for this specification, identify related ALTO features and ALTO WG documents.

2. Relevant ALTO services and documents

Particular services and drafts where an ALTO address type for cellular networks is needed include:

o - Endpoint property service: extended to allow endpoints to be cells on which properties can be requested,

o - (Filtered) Cost Map Service: where PIDs can be cells within and among which cost values can be requested, see also[draft-randriamasy-alto-cost-context-01],

o - "Mobility Network Models in ALTO" defined in [draft-bertz-alto-mobilitynets] propose to identify network points of attachment (PoA) such as cells to PIDs.

o - "ALTO Performance Cost Metrics": being defined in [draft-ietf-alto-performance-metrics-01], they will be extended to performance costs in cellular networks,

o - "Extensible Property Maps for the ALTO Protocol", being defined in [I-D.ietf-alto-unified-props-new] are applicable to entities that may be cells which are identified by their addresses. In this document a domain identifier for cells will need to be accordingly defined, and the entity domain identifier "ecgi" is proposed.

3. Cell addresses, ALTO Address types and ALTO Entity Domain names

This section reflects ALTO WG discussions. The ALTO Address Type Registry is detailed in Section 14.4 of RFC7285 specifying the base ALTO protocol. It currently lists ALTO address types "ipv4" and "ipv6". These ALTO address types can be used in the Endpoint Property Service and the Endpoint Cost Service. They can also be used to list the endpoints covered by a PID. The ALTO base protocol however does not preclude other address types, See RFC7285, section 2.2 Endpoint Address.

The draft [draft-ietf-alto-unified-props-new] introduces and specifies two new information services called Property Map and Filtered Property Maps. They specify two media types, called "alto-propmap+json" and "alto-propmapparams+json". A Property Map exposes values of properties that are defined on Entities, where an Entity is an object that extends the scope of an Endpoint having an individual IP address to groups of Endpoints, PIDs, network elements abstracted from one or more network elements of arbitrary nature. An Entity has a unique address or name and is defined as belonging to a Domain that has a unique identifier. To this end, [draft-ietf-alto-unified-props-new] specifies an Entity Domain Registry. An entity can thus potentially be a cell.

Example entities are Endpoints with addresses in the ipv4 or ipv6 domain, or PIDs with a name in the "pid" domain.

The draft points out that that "Entity domains and property names are extensible. New 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 by the ALTO protocol."

As a consequence, RFC7285 and draft [draft-ietf-alto-unified-props-new] provide the background to allow Endpoints and Entities to be cells with well-specified addresses.

3.1. ALTO Address Type for cellular networks

Registering cellular addresses in the ALTO Address Type Registry allows conveying Endpoint Costs and Properties and (Filtered) Cost Maps with PIDs and Endpoints being cells.

RFC7285 specifies endpoint address formats for ipv4 and ipv6 and the purpose of this draft is to agree on a format for cellular endpoints. When a cell is mapped to a PID, say "MyCell3" the ALTO Cell Id will be used to specify the endpoints within this PID.

Whereas IP addresses are associated to domains ipv4 and ipv6, a Cell Id will be associated to the domain "ecgi".

The domain name "ecgi" stems from the term ECGI -- E-UTRAN Cell Global Identifier, defined in 3GPP.

3.2. ALTO Entity Domain for cellular networks

Applications may want to query (Filtered) Property Maps on Cellular Networks or on networks comprising cells. In which case cells would have to be identified as Entities with an entity address specific to a domain registered in ALTO Entity Domain specified in [I-D.ietf-alto-unified-props-new].

The domain "ecgi" is suitable for Entities as well.

3.3. Consistency between ALTO Entity Domain and ALTO Address Type

Actually, the cell address format proposed in section Section 4 is suitable for both Endpoints and Entities. Likewise, ipv4 and ipv6 addresses. Whereas cellular and IP Endpoint addresses can also be Entity addresses, an Entity is not necessarily an Endpoint. This is the case for instance for entities like PIDs or ANEs. Therefore there is a consistency issue to be solved, and this is done in the ALTO Entity Domain specification of the draft [draft-ietf-alto-unified-props-new].

4. Proposed format for ALTO cell identifiers

4.1. Endpoint address canonical string format

'ecgi:' MCC '.' MNC ':' ECI

Where:

- o MCC: Mobile Country Code, as assigned by ITU. A 3 digits decimal number without leading zeros.
- o MNC: Mobile Network Code, as assigned by National Authority. A 2-3 digits decimal number without leading zeros.
- o ECI: E-UTRAN Cell Identifier. A 7 digits lower-case hexadecimal number.

Example:

- o ecgi:940.978:1234abc
  - * MCC value 940 stands for country or geographical area "Wonderland"
  - * MNC value 978 stands for Network N1 in Wonderland and other networks in other countries
  - * A same MNC value, say 020 may be associated with several MCCs.
  - * Some MCCs have MNCs encoded with 2 digits and MNCs encoded with 3 digits.

4.2. ALTO Cell Id formats

Three formats are proposed:

- 'ecgi:' MCC
- 'ecgi': MCC '.' MNC
- 'ecgi:' MCC '.' MNC ':' ECI-MASK '/' MASK-LEN

where:
MASK-LEN is a decimal number.
ECI-MASK is a string of lower-case hex digits, of which all but the first MASK-LEN bits are zero.

Prefix ecgi:P-MCC.P-MNC:P-ECI/N matches ecgi:MCC.MNC:ECI iff
MCC == P-MCC, and
MNC == P-MNC, and
ECI has the same number of hex digits as P-ECI, and
the first N bits of ECI match those of P-ECI.

5. Examples

- ecgi:940
  * Matches every cell address with MCC 940.
- ecgi:940.978
  * Matches every cell address with MCC 940 and MNC 978.
- ecgi:940.978:1234800/18
  * Matches every cell address with MCC 940, MNC 978, and a 7-digit ECI that starts with the 18 bits 0x12348. Thus it matches ecgi:940.978:1234abc and ecgi:940.978:1234800, but does not match ecgi:940.978:1234d00.

6. IANA Considerations

This document extends: the ALTO Address Type Registry defined in section 14.4 of RFC7285 and the ALTO Domain Entity Registry defined in [I-D.ietf-alto-unified-props-new]. If the latter is considered a superset of the former, it seems consistent to register only a new Entity Domain named "ecgi". This requires that implementations not willing to use the (Filtered) Property Map Service and related Entities should still be cognizant of the ALTO Domain Entity Registry. Potential extensions are as as follows:

- The ALTO Address Type Registry defined in section 14.4 has an additional item with the following properties:

  * Identifier : ecgi

  * Address encoding: see section Section 4

  * Prefix encoding: TBC

  * Mapping to/from IPv4/v6: none

- The ALTO Domain Entity Registry has an additional element with the following properties:

  * Identifier : ecgi

  * Entity Address encoding: see section Section 4

  * Field to be updated as [I-D.ietf-alto-unified-props-new] progresses: indicating that this entity domain can also be used as an ALTO Address Type

  * Hierarchy and inheritance: TBC

7. Security Considerations

TBC

8. Acknowledgements

Great thanks to Wendy Roome who initiated this document and Qin Wu for discussions.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

9.2. Informative References

[I-D.ietf-alto-unified-props-new]
Roome, W., Randriamasy, S., Yang, Y., Zhang, J., and K. Gao, "Unified Properties for the ALTO Protocol", draft-ietf-alto-unified-props-new-10 (work in progress), November 2019.

Appendix A. An Appendix

Author's Address

Sabine Randriamasy
Nokia-Bell-labs
Route de Villejust
Nozay 91460
FRANCE

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

ALTO WG Q. Xiang
Internet-Draft Yale University
Intended status: Informational J. Zhang
Expires: January 14, 2021 Tongji/Yale University
F. Le
IBM
Y. Yang
Yale University
H. Newman
California Institute of Technology
July 13, 2020

Resource Orchestration for Multi-Domain, Exascale, Geo-Distributed Data Analytics

draft-xiang-alto-multidomain-analytics-05.txt

Abstract

As the data volume increases exponentially over time, data analytics is transiting from a single-domain network to a multi-domain, geo-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 network calls for a resource orchestration framework that emphasizes the performance predictability of data analytics jobs, the high utilization of resources, and the autonomy and privacy of member networks.

This document presents the design of Unicorn, a unified resource orchestration framework for multi-domain, geo-distributed data analytics, which uses the Application-Layer Traffic Optimization (ALTO) protocol as the key component for (1) allows member networks to provide accurate information on different types of resources; (2) keeps the private information of member networks; and (3) allows data analytics jobs to accurately describe their requirements of different types of resources. As a part of Unicorn, an ALTO extension for privacy-preserving interdomain information aggregation is also presented.

Status of This Memo

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 January 14, 2021.

Copyright Notice

Table of Contents

1. Introduction

This document describes the design of Unicorn, a unified resource orchestration framework for large-scale data analytics in multi-domain, geo-distributed networks. An important use case for such settings is the Large Hadron Collider (LHC) network, which consists of over 180 member networks all over the world, to support scientists to access multiple resources, e.g., computing, storage and networking resources, distributed in the member networks to conduct large-scale data analytics. With more and more data being generated and stored in different geo-distributed member networks, network architects and administrators are exploring different designs for efficient resource orchestration in multi-domain, geo-distributed networks.

The design presented in this document is based on the development and deployment experience of Unicorn in the CMS network, one of the largest scientific experiments in the LHC network. The primary requirements of resource orchestration in such a multi-domain, geo-distributed environment are the performance predictability of various data analytics jobs, the high utilization of different types of resources, and the autonomy and privacy of resource owners, i.e., member networks.

Pre-production development and extensive testing have shown that the Application-Layer Traffic Optimization Protocol [RFC7285] is well suited as a fundamental component in Unicorn for providing a generic representation that (1) allows different types of data analytics jobs to accurately describe their resource requirements and (2) allows member networks to provide accurate information on different types of resources they own and at the same time maintain their privacies.

This is in contrast with the state-of-the-art resource orchestration frameworks, such as HTCondor and Mesos, which either do not provide accurate networking information or expose all the private details of member networks. This document elaborates on the design requirements of resource orchestration in multi-domain, geo-distributed networks that lead to this design choice and presents the details of Unicorn, including an ALTO extension for privacy-preserving, interdomain information aggregation.

This document first gives an overview of the characteristics of multi-domain, geo-distributed data analytics. Then, the design requirements for resource orchestration under such settings are summarized. After reviewing existing designs and their limitations, this document gives the arguments for using ALTO as the generic representation for describing both resource requirements and the resource information and describes the design details of Unicorn. Finally, a privacy-preserving, interdomain extension of ALTO is presented.

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 -03

- o Add a discussion on an ongoing work to develop and deploy unified resource representation as an ALTO extension to expose heterogeneous resource information to application in Section 10.3.

4. Characteristics of Multi-Domain, Geo-Distributed Data Analytics

This section describes the characteristics of multi-domain, geo-distributed data analytics.

4.1. Dynamic Data Analytics Workload

In multi-domain, geo-distributed data analytics, extremely large amounts of data are generated and stored across different member networks. Authorized users from different organizations can access data and resources in member networks to conduct various data analytics jobs using various data analytics applications.

An data analytics application usually provides an automated process that decomposes a large data analytics job into a set of smaller tasks, whose dependencies are expressed as a directed acyclic graph (DAG). Tasks without any dependency can be executed in parallel to improve the efficiency of the data analytics job they belong to. This decomposition is highly user- and application-dependent.

Each task may have different requirements on different resources. For instance, task T1 may require dataset A in storage node X as input and 1 CPU as the computing resource, while task T2 may require dataset B in storage node Y as input and 2 CPUs as the computing resource. Furthermore, each task may require resources from different member networks. In the previous example, T1 may require its output to be stored in a storage node in another member network for the purpose of secure storage. The resource requirements of tasks are highly user- and application-dependent.

From the above description, it is observed that the workload of multi-domain, geo-distributed data analytics is highly dynamic, in terms of the number of users, the types of applications, the number of jobs, the decomposition of jobs and the resource requirements of tasks.

Though with such dynamism, it is the general consensus of users to expect performance predictability of their analytics jobs (TODO: add Mogul citation). Hence the resource orchestration for multi-domain, geo-distributed data analytics must be able to achieve efficient resource sharing among different data analytics jobs of different applications from different users. To this end, a generic representation of resource requirements for different tasks from different analytics applications must be chosen. Furthermore, to ensure maximal deployment, the resource orchestration framework must be independent of and compatible with data analytics applications.

4.2. Dynamic Resource Availability

In the multi-domain, geo-distributed data analytics network, different member networks belong to different administrative domains. Each member network has its own resource management policies and can choose to use different management software, such as HTCondor and Mesos.

Each member network provides different types of resources with different amounts. For example, transit networks such as ESNet and Internet2 provide high-bandwidth networking resources. In contrast, campus science networks provide abundant computation and storage resources, but may provide limited networking bandwidths. And some smaller science networks only provide limited computation and storage resources. The availability of the resources in each member network is subject to the autonomous control of the member network.

Furthermore, member networks are interconnected with high bandwidth-delay-product links, where state-of-the-art networking resource allocation mechanisms, such as TCP, become inefficient [XCP].

From the above description, it is observed that the resource availability of the multi-domain, geo-distributed data analytics network is also highly dynamic, subject to the types of member networks, the resources provided by member networks and the resource management policies and management software used by member networks.

Though with such dynamism, it is the general consensus of member networks that the resource orchestration for multi-domain, geo-distributed data analytics must achieve high utilization of different types of resources, following the autonomy and privacy of each member network. To this end, a generic representation of resource availabilities for different types of resources must be chosen. Such a representation must be accurate and at the same time maintain the privacy of member networks. Furthermore, to ensure maximal deployment, the resource orchestration framework must be independent of and compatible with the resource management systems used by member networks.

5. Design Requirements

This section summarizes the design requirements for resource orchestration for multi-domain, geo-distributed data analytics from the previous section.

- REQ1: Provide performance predictability for data analytics jobs.

- REQ2: Achieve the efficient resource sharing among data analytics jobs.

- REQ3: Achieve the high utilization of different types of resources in member networks.

- REQ4: Maintain the autonomy and privacy of member networks.

- REQ5: Provide compatibility with different data analytics applications and resource management systems to maximize the deployment.

6. Review of Resource Orchestration Designs for Data Analytics

This section provides an overview of three general types of resource orchestration designs for data analytics -- the centralized resource-graph-based orchestration, the centralized ClassAds-based orchestration and the distributed opportunistic orchestration. Then, the key reason why these designs are inadequate for multi-domain, geo-distributed data analytics is provided.

6.1. Centralized resource-graph-based orchestration

Systems such as Mesos [Mesos] and Borg [Borg] adopt a graph-based abstraction to represent the resource availability of computing clusters. Each node in the graph is a physical node representing computation or storage resources and each edge between a pair of nodes denotes the networking resource connecting two physical nodes. This design is inadequate for multi-domain, geo-distributed data analytics system because (1) it compromises the privacy of different member networks by revealing all the details of resources; and (2) the overhead to keep the resource availability graph up to date is too expensive due to the heterogeneity and dynamicity of resources from different member networks.

6.2. Centralized ClassAds-based orchestration

HTCondor [HTCondor] proposes a ClassAds programming model, which allows different resource owners to advertise their resource supply and the job owners to advertise the resource demand. However, this programming model does not support the accurate discovery of networking resources, but leave the orchestration of networking resources completely to TCP, which has been known to behave poorly in networks with high bandwidth-delay products [XCP].

6.3. Distributed opportunistic orchestration

Some systems, such as Apollo [Apollo] and Sparrow [Sparrow], use a distributed design. In this design, given a data analytics job, a small number of computing and storage nodes are randomly selected as candidates. Then a scheduling algorithm makes the decision to select the best pair of computing and storage nodes within this small set of candidates. Though it is shown in production that this design achieves a performance very close to the theoretical optimal resource allocation scheme, this design cannot be applied to multi-domain, geo-distributed data analytics because (1) the pool of computing and storage resources is much larger, and is distributed across the world, and (2) it is hard to distributively orchestrate networking resources in such a high bandwidth-delay product scenario.

6.4. Inadequacy of Existing Designs for Multi-Domain, Geo-Distributed Data Analytics

Applying the designs reviewed in the preceding subsections for multi-domain, geo-distributed data analytics only satisfies the design requirement of compatibility (REQ5), but leaves all the other requirements unfulfilled. The key reason is that they do not have an information model that simultaneously

- o allows member networks to provide accurate information on different types of resources, e.g., the computing, storage and networking resources, they own;
- o keeps the private information of member networks, such as physical topologies and policies, from the data analytics applications; and
- o allows data analytics jobs to accurately describe their requirements of different types of resources.

7. Unicorn Design

This section presents the design of the Unicorn framework. First, the motivations of using ALTO as the information model of resource orchestration for multi-domain, geo-distributed data analytics are reviewed. Then the architecture of Unicorn is provided.

7.1. Choosing ALTO as the Resource Information Model

As reviewed in the preceding section, the commonly used resource-graph-based information model and the ClassAds information model do not support the accurate, yet privacy-preserving resource discovery across different member networks. In contrast, the ALTO protocol uses abstract maps of networks to provide network information with the goal of modifying network resource consumption patterns while maintaining or improving application performance [RFC7285]. This document proposes the use of ALTO for providing information of different types of resources, e.g., computing, storage and networking resources. This design has the following advantages:

- o ALTO provides the network information based on abstract maps of a network. Additional services are built on top of the ALTO abstract maps to provide information of other types of resources, e.g., the computing and storage resources. These maps provide accurate information of different types of resources for the resource orchestration system to effectively utilize them for data analytics applications. For example, the ALTO Endpoint Property Service can provide information of computing nodes and storage nodes.
- o The ALTO abstract maps provide a simplified view of resources of member networks, instead of the full details of their resource availability. Thus ALTO allows member networks to keep their private information, such as physical topologies and policies, from the applications. For example, the ALTO Network Map service

provides a "one-big-switch" view that defines a grouping of network endpoints. This view hides the details of the underlying physical topology of the network and a network deploying the ALTO server has the autonomy to adopt any endpoint grouping algorithm.

- o ALTO uses a client-server model, in which applications can use ALTO clients to accurately describe their requirements of different types of resources and send these requirements to the ALTO servers to retrieve the accurate information of resources that suit their requirements. For example, the ALTO Multi-Cost service [RFC8189] allows an ALTO client to specify a logic set of tests in a query. Such tests are used by ALTO servers to filter out the information of unqualified resources from the response sent back to the ALTO client.

7.2. Architecture of Unicorn

This section describes the design details of Unicorn. Figure 1 presents the architecture of Unicorn for a multi-domain, geo-distributed data analytics system with N member networks. In particular, Unicorn consists of the following key components:

```
          .-------------.       .-------------.
          |Application 1|  ...  |Application N|
          '-------------'       '-------------'
                \                     /
 .- - - - - - - -\- - - - - - - - - -/- - - - - - - - - - - - - - -.
 |  Unicorn       \                 /                               |
 |                .-----------------------.                         |
 |                | Resource Orchestrator |   .----------------------.|
 |                |                       |   |Distributed Hash Table||
 |                |      .-----------.    |---|   of Computing and   ||
 |                |      |ALTO Client|    |   |  Storage Resources   ||
 |                |      '-----------'    |   '----------------------'|
 |                '-----------------------'                         |
 |             /            |             \                         |
 |            /             |              \                        |
 |   .-------------.  .-----------.   .-------------.               |
 |   |ALTO Server 1|  | Execution |   |ALTO Server M|               |
 |   '-------------'  |  Agents   |   '-------------'               |
 |          |         '-----------'          |                      |
 |          |          /          \          |                      |
 |  .----------------./            \ .----------------.             |
 |  |     Site 1     |      . .      |     Site N     |             |
 |  '----------------'               '----------------'             |
 '- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -'
```

Figure 1: Architecture of Unicorn.

- o ALTO Server: for each member network, one or more ALTO servers are deployed to provide accurate, yet privacy-preserving information of different types of resources owned by the corresponding network. Examples of such information include the link bandwidth between endpoints, the memory I/O bandwidth and the CPU utilization at computing endpoints and the storage space at storage endpoints. In addition to the basic ALTO services defined in [RFC7285], The ALTO servers in Unicorn also provide ALTO extension services such as the ALTO Multi-Cost Service [RFC8189], the ALTO Server-Sent Event Service [DRAFT-SSE] and the ALTO Multipart Cost Property Service [DRAFT-PV] to provide fine-grained resource information.

- o Distributed Hash Table (DHT) of Computing and Storage Resources: A DHT system is deployed across member networks to lookup the location of computing and storage resources. Compared with the current centralized lookup services in the CMS network, i.e., PhEDEx and HTCondor, a DHT system provides a significant performance improvement for discovering the locations of computing and storage resources in multi-domain, geo-distributed data analytics systems.

- Resource Orchestrator: The orchestrator is a shim layer between the data analytics jobs from different applications and the member networks. It contains an ALTO client that communicates with the ALTO servers at member networks to retrieve resource information. Given a set of data analytics jobs, the orchestrator adopts a three-phase discovery process, which will be elaborated in the next section, to find the accurate information of all the resources that can be used to execute these jobs. Then the orchestrator runs a customized resource allocation algorithm to compute the resource allocation decisions for these jobs, and send the decisions to the execution agents at corresponding member networks.

- Execution Agent: One or more execution agents are deployed at each member network. They take the resource allocation decisions from the resource orchestrator, and communicate with the underlying resource management system deployed at the corresponding member network to reserve the resources for the data analytics jobs and execute them.

7.2.1. Three-Phase Resource Discovery

The preceding subsection describes the architecture and the key components of Unicorn. One missing component is how to accurately discover the information of different types of resources for a set of data analytics jobs with the assistance of ALTO. This section presents the three-phase resource discovery design in Unicorn.

7.2.1.1. Phase 1: Endpoint Property Discovery

Figure 2 shows the procedure of the endpoint property discovery phase. Given a set of data analytics jobs, the resource orchestrator communicates with the DHT lookup system to find the locations, i.e., the endpoint addresses, of all candidate computing and storage resources. With such information, the ALTO client then issues Endpoint Property Service (EPS) queries to the ALTO servers deployed at member networks to discover the information of all candidate endpoints.

```
      .-----------------------.
      | Resource Orchestrator |  Jobs' Resource
      |                       |   Requirements    .-------------.
      |     .------------.    | ----------------> | DHT Lookup  |
      |     |ALTO Client|     | <---------------- |   System    |
      |     '------------'    |     Endpoint      '-------------'
      '---------| ^-----------'     Locations
         EPS    | | Endpoint
       Queries  | | Properties
                v |
          .-------------.
          |ALTO Servers |
          '-------------'
```

Figure 2: The Endpoint Property Discovery Phase.

7.2.1.2. Phase 2: Endpoint Path Discovery

Candidate computing and storage endpoints need to move data between them before, during and after the execution of a data analytics job. In multi-domain, geo-distributed data analytics, a pair of candidate endpoints may not be in the same member network. In this case, the orchestrator needs to find out the connectivity information between such a pair of candidate endpoints.

Figure 3 shows the procedure of the endpoint path discovery phase. Given a pair of candidate endpoints that are not in the same member network, the ALTO client in the orchestrator adopts an iterative process to find the interdomain connectivity information for this pair. It starts by issuing an ALTO Endpoint Cost Service query or an ALTO Flow-based Endpoint Cost Service [DRAFT-FCS] to the ALTO server of the member network where the source endpoint locates. The cost type of this query is a customized type called next-hop, with a customized cost mode tuple and a customized cost metric next-network.

```
                    .-----------------------.
                    | Resource Orchestrator |
                    |                       |
                    |      .-----------.    |
                    |      |ALTO Client|    |
                    |      '-----------'    |
                    '---------| ^-----------'
                    Customized| | Endpoint
                       ECS    | | Path
                     Queries  | | Segments
                              v |
                        .-------------.
                        |ALTO Servers |
                        '-------------'
```

Figure 3: The Endpoint Path Discovery Phase.

The ALTO server returns a 2-tuple, where the first element is the autonomous number (AS) of the next member network along the AS-path from the source endpoint to the destination endpoint, and the second element is the ingress of this next member network. In a member network, the ALTO server can get such information from the underlying interdomain routing protocol, e.g., BGP. Based on the received response, the ALTO client then issues a similar query to the ALTO server of the next member network. The process stops when the ALTO server of the member network where the destination endpoint locates receives such a query, who will return a null 2-tuple in response to notify the ALTO client. By the end of this process, the ALTO client can assemble a domain-path, in the form of a path vector of (ingress, AS), of this pair of candidate endpoints.

7.2.1.3. Phase 3: Resource State Abstraction Discovery

After the second phase, the resource orchestrator has the connectivity information of each candidate endpoint pair, i.e., the domain-path. Equivalently, for each member network, it knows the set of all candidate endpoint pairs that will enter this network. With this information, the resource orchestrator can communicate with the ALTO servers at member networks to discover the resource sharing between all the candidate endpoint pairs. In particular, Unicorn extends the routing state abstraction [DRAFT-RSA] to the more generic resource state abstraction to represent such resource sharing.

Figure 4 shows the procedure of the resource state abstraction discovery phase. For each member network, the ALTO client in the orchestrator sends an ALTO Multipart Cost Property Service query defined in [DRAFT-PV] by providing the set of candidate endpoint pairs as input. The cost type of this query is path vector. Upon receiving the query, the ALTO server in each member network computes an ALTO cost map and an ATLO property map to the ALTO client. These two maps represent a set of linear inequalities revealing the resource sharing among the set of candidate endpoint pairs in the member network.

```
._______________________.
| Resource Orchestrator |
|                       |
|     .___________.     |
|     |ALTO Client|     |
|     '___________'     |
'_________| ^___________'
 Multipart| | Resource
 Cost     | | State
 Property | | Abstraction
 Queries  v |
    ._____________.
    |ALTO Servers |
    '_____________'
```

Figure 4: The Resource State Abstraction Discovery Phase.

Unicorn provides two mechanisms for the ALTO servers to return the computed cost maps and property maps to the ALTO client. The first mechanism is to let each ALTO server independently sends its response to the ALTO client. The second mechanism is a privacy-preserving interdomain information aggregation process, in which the ALTO servers in all member networks use a secure multi-party computation (SMPC) protocol to collectively send the responses to the ALTO client without revealing the source of any entry, i.e., the linear in equality, in the cost maps and property maps.

The first mechanism has a higher security risk in that it exposes the bottleneck resource information of each member network. In contrast, the second mechanism provides a better protection of the private information of each member network. The details of the privacy-preserving interdomain information aggregation process will be presented in the next section.

After receiving the responses sent back from the ALTO servers from all the member networks, the orchestrator finishes the whole resource discovery process and collects the accurate information of different types of resources for data analytics jobs.

7.2.2. Proactive Full-Mesh Resource Discovery

To ensure the resource discovery process scales, a proactive full-mesh resource discovery component is developed. The main idea of this component consists in having the ALTO client periodically query ALTO servers at all sites to discover the resource state abstraction between every pair of source and destination sites. As such, when an application submits a resource discovery request, the ALTO client does not need to send any query to the ALTO servers. Instead, using the site-level bandwidth sharing information, the ALTO client can immediately perform projection operations to get the resource information for the request. This mechanism substantially improves the scalability of Unicorn.

7.3. Example

This subsection gives an example to illustrate the workflow of Unicorn. Figure 5 gives a topology of three member networks, 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.

```
                      .------------.
                      | Network B  |
.------------.    ingB|            |
| Network A  |--------|     d1     |
|            |        '------------'
|    s1      |
|            |        .------------.
|    s2      |--------| Network C  |
'------------'    ingC|            |
                      |     d2     |
                      '------------'
```

Figure 5: An Illustrating Example for Unicorn.

In the endpoint property discovery phase, the Unicorn resource orchestrator finds that s1 stores X and s2 stores Y, and that the locations of s1, s2, d1 and d2, from the DHT lookup system. It then issues EPS queries to network A, B and C, respectively, to 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).

In the endpoint path discovery phase, the ALTO client in the orchestrator iteratively issues Endpoint Cost Service (ECS) query to the ALTO servers in member networks, and finds that the domain-path for pair (s1, d2) is [(null, A), (ingB, B)] and the domain-path for pair (s2, d2) is [(null, A), (ingB, B)]. Hence both pairs will use the networking resources of network A, while only (s1, d1) will use network B and only (s2, d2) will use network C.

In the resource state abstraction discovery phase, the ALTO client in the orchestrator issues Multipart Cost Property Service queries to network A, B and C, respectively. Denote the available bandwidth that can be assigned to T1 as x1 and that to T2 as x2. Assume the linear inequalities computed by the three networks are:

```
A: x1 + x2 <= 10Mbps
B: x1 <= 3Mbps
C: x2 <= 3Mbps
```

If the ALTO servers use the first mechanism to directly return their resource information to ALTO client, respectively, each of them will send a cost map and a property map response encoding its own linear inequality to the ALTO client. In this way, the orchestrator gets the accurate information about networking resource sharing between (s1, d1) and (s2, d2). It then can invoke a resource allocation algorithm to allocate the resources to tasks T1 and T2. For example, if the goal is to maximize the minimal bandwidth of two tasks, the allocation decision will be to assign endpoints s1 and d1 to T1, with a bandwidth of 3Mbps, and assign endpoints s2 and d2 to T2, with a bandwidth of 3Mbps as well.

8. ALTO Extension: Privacy-Preserving Interdomain Information Aggregation for Resource Discovery

This section describes a customized ALTO extension in Unicorn that supports the privacy-preserving discovery of networking resource sharing among a set of candidate endpoint pairs.

8.1. Extension Specification

Figure 6 presents the workflow of the proposed ALTO extension. Assume a set of N member networks denoted as AS_1, AS_2, ... AS_N and the number of all candidate endpoint pairs is F. The interdomain information aggregation process works as follows:

```
                                .-----------------------.
                                | Resource Orchestrator |
                                |                       |
                                |     .-----------.     |
                                |     |ALTO Client|     |
                                |     '-----------'     |
                               >'-----------------------'<
                              /Disguised   /|\            \
           Multipart Cost    / Response     |              \
           Property Queries v               |               \
                   .--------.          .--------.          .--------.
                   |ALTO    |          |ALTO    |          |ALTO    |
                   |Server 1| <------->|Server 2|<--... -->|Server N|
                   '--------' Limitedly'--------'          '--------'
                              Shared Secret
```

Figure 6: The Privacy-Preserving Interdomain Resource Information Aggregation.

- Step 1: The ALTO client sends the Multipart Cost Property Service request to and a homomorphic public key k_p to each member network.

- Step 2: The ALTO server of each network AS_i computes its own set of linear inequalities A_i x <= b_i. Denote the size of this set as m_i.

- Step 3: The ALTO server of each network AS_i introduces m_i non-negative slack variables to transform its set of linear inequalities into a set of linear equations.

- Step 4: The ALTO servers of all member networks use a private matrix SMPC summation protocol to collectively compute k=m_1 + m_2 + ... + m_N + 1. The value k is known to all the member networks.

- Step 5: The ALTO servers of each network AS_i selects a random k-by-m_i matrix P_i, and computes the matrix P_iA_i and P_ib_i.

- Step 6: The ALTO server of each network then uses a few matrices, which are only shared with a couple of other networks, to further obfuscate P_iA_i and P_ib_i, and sends the obfuscated matrices to the ALTO client via symmetric encryption.

- Step 7: the ALTO client decrypts the received responses from all ALTO servers, and sums up the decrypted response to get a set of linear equations sum P_iA_i x = sum P_ib_i.

This process ensures that the networking resource capacity region derived from sum P_iA_i x = sum P_ib_i is the same as that derived from A_1 x <= b_1, A_2 x <= b_2, ... A_N x <= b_N. More importantly, the ALTO client has no knowledge on the information of network resource sharing of a single member network.

8.2. Example

This subsection uses the same example in Figure 5 to illustrate the privacy-preserving information aggregation process. The set of linear inequalities computed by each network is as follows:

```
A: x1 + x2 <= 10
B: x1 <= 3
C: x2 <= 3
```

Then the networks collectively compute k=1+1+1+1=4. And then introduces slack variables to transform the linear inequalities into linear equations:

```
A: x1 + x2 + x3           <= 10
B: x1           + x4      <= 3
C:      x2           + x5 <= 3
```

For each network, the random matrix it chooses as follows:

```
P_A: [11, 49, 95, 34]
P_B: [58, 22, 75, 25]
P_C: [50, 69, 89, 95]
```

After the obfuscating process in Step 5 and Step 6 in the previous subsection, the decrypted set of linear equations the ALTO client gets is

```
69  x1 + 61  x2 + 11 x3 + 58 x4_ + 50 x5 = 434
71  x1 + 118 x2 + 49 x3 + 22 x4_ + 69 x5 = 763
170 x1 + 184 x2 + 95 x3 + 75 x4_ + 89 x5 = 1442
59  x1 + 129 x2 + 34 x3 + 25 x4_ + 95 x5 = 700
```

Assume the goal is still to maximize the minimal bandwidth of two tasks, the allocation decision made using this set of linear equations will still be x1=3 and x2=3, i.e., assigning endpoints s1 and d1 to T1, with a bandwidth of 3 and assigning endpoints s2 and d2 to T2, with a bandwidth of 3 as well.

9. Implementation and Demonstration Experience

The authors build an ALTO server on top of the OpenDaylight Software Defined Network controller. The ALTO server collects the network state information from the OpenDaylight controller, e.g., topology, policy and traffic statistics, processes the collected information into resource abstraction, and sends the abstraction back to the ALTO client at the resource orchestrator.

In the past few years, the Unicorn framework has been deployed and demonstrated in small federation networks connecting Dallas, Texas, Los Angles, California, and Denver, Colorado at different conventions. For example, in 2018, the federation in the demonstration is composed of three member networks. Network 1 is in Dallas, Texas, and Network 2 and network 3 are in Los Angeles, California. Network 1 is connected to network 2 through a layer-2 WAN circuit with a 100 Gbps bandwidth, provisioned by several providers such as SCinet, CenturyLink and CENIC. Network 1 is a temporal science network in the CMS experiment, while network 2 and 3 are long-running CMS Tier-2 sites. In this federation, users need to reserve network resources to transfer large-scale science datasets (e.g., with a size of hundreds of PB) between networks.

The authors evaluate the accuracy and latency of the framework for discovering network resources in this network. During the evaluation, the framework accurately discovers the network resource information for a large amount of circuits reservation requests with a very low discovery latency. Specifically, for all the reservation requests, Unicorn always provides the accurate information of available bandwidth sharing in the network (i.e., a 100% accuracy), with an average discovery latency of 100 milliseconds, and a worst latency of less than 1 second. With the discovered network resource information, users can transmit large-scale science datasets at a speed up to 100 Gbps, (i.e., the theoretical maximal throughput).

10. Discussion

10.1. Discovering the Domain-Paths Using a New Interdomain Routing Protocol

The current design of the endpoint path discovery process in Unicorn assumes that the underlying interdomain routing protocol is the standard BGP, which only provides the path vector of ASes instead of the path vector of (ingress, AS) tuples needed by Unicorn. If a multi-domain, geo-distributed data analytics system uses an interdomain routing protocol that provides the path vector of (ingress, AS) pairs, the endpoint path discovery process in Unicorn can be simplified to only send queries to the ALTO server of the network where the source candidate endpoint locates.

10.2. Comparison of the Efficiency to Achieve Optimal Resource Orchestration using ALTO and without using ALTO

The authors of this draft conduct a systematic investigation to understand the efficiency differences to achieve optimal resource orchestration between using ALTO and without using ALTO. Specifically, the authors study the problem of computing the optimal resource reservation without using ALTO, but only using the simple reservation interface deployed in PCE-based network resource reservation systems (e.g., OSCARS). Given a client's request, a novel algorithm is developed to compute the optimal resource reservation within O(n^3) times of queries on the simple reservation interface, where n is the number of flows in the request. This is the best known algorithm for this problem. Although the asymptotic complexity appears efficient, the authors' experience to test this design shows that it often takes tens of thousand queries on the simple reservation interface to compute the optimal resource orchestration, leading to substantial orchestration latencies. This result has been published at AAAI 2019.

In contrast, by leveraging the capability of ALTO to expose accurate network information to the client, the Unicorn framework can compute the optimal resource reservation by querying an ALTO server only once, resulting in orders of magnitude improvement on resource orchestration latencies.

10.3. Current Ongoing Work: Unified Resource Representation as an ALTO extension

The authors of this draft are currently designing an ALTO extension which uses generic mathematic programming constraints as a unified resource representation of heterogeneous resources in the network. In particular, a SQL-style language is designed for an ALTO client to express its requirement on available resources. An SMT-based compiler is developed for the network, which translates a user's SQL resource query into a constraint programming model, finds feasible resources in the network, and returns such resource information to user in the compact representation expressed as generic mathematic programming constraints. More details about this extension are reported in a different draft titled "ALTO Extension: Unified Resource Representation". A paper providing preliminary evaluation results of this extension has been accepted by ACM SIGCOMM NAI 2020 Workshop.

11. Security Considerations

This document does not introduce any privacy or security issue not already present in the ALTO protocol.

12. IANA Considerations

This document does not define any new media type or introduce any new IANA consideration.

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

13.2. Informative References

[Apollo] Boutin, E., Ekanayake, J., Lin, W., Shi, B., Zhou, J., Qian, Z., Wu, M., and L. Zhou, "Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing", 2014, <https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-boutin_0.pdf>.

[Borg] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and J. Wilkes, "Large-scale cluster management at Google with Borg", 2015, <https://dl.acm.org/citation.cfm?id=2741964>.

[DRAFT-FCS] Zhang, J., Gao, K., Wang, J., Xiang, Q., and Y. Yang, "ALTO Extension: Flow-based Cost Query", 2017, <https://datatracker.ietf.org/doc/draft-gao-alto-fcs/>.

[DRAFT-PV] Bernstein, G., Lee, Y., Roome, W., Scharf, M., and Y. Yang, "ALTO Extension: Abstract Path Vector as a Cost Mode", 2015, <https://tools.ietf.org/html/draft-yang-alto-path-vector-01>.

[DRAFT-RSA] Gao, K., Wang, X., Xiang, Q., Gu, C., Yang, Y., and G. Chen, "A Recommendation for Compressing ALTO Path Vectors", 2017, <https://datatracker.ietf.org/doc/draft-gao-alto-routing-state-abstraction/>.

[DRAFT-SSE]
Roome, W. and Y. Yang, "ALTO Incremental Updates Using Server-Sent Events (SSE)", 2015, <https://datatracker.ietf.org/doc/draft-ietf-alto-incr-update-sse/>.

[HTCondor]
Thain, D., Tannenbaum, T., and M. Livny, "Distributed computing in practice: the Condor experience", 2005, <http://dl.acm.org/citation.cfm?id=1064336>.

[Mesos] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A., Katz, R., Shenker, S., and I. Stoica, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center", 2011, <http://static.usenix.org/events/nsdi11/tech/full_papers/Hindman_new.pdf>.

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

[RFC8189] Randriamasy, S., Roome, W., and N. Schwan, "Multi-Cost Application-Layer Traffic Optimization (ALTO)", RFC 8189, DOI 10.17487/RFC8189, October 2017, <https://www.rfc-editor.org/info/rfc8189>.

[Sparrow] Ousterhout, K., Wendell, P., Zaharia, M., and I. Stoica, "Sparrow: Distributed, Low Latency Scheduling", 2013, <https://dl.acm.org/citation.cfm?id=2522716>.

[XCP] Katabi, D., Handley, M., and C. Rohrs, "Internet Congestion Control for Future High Bandwidth-Delay Product Environments", 2002, <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.58.4783>.

Authors' Addresses

Qiao Xiang
Yale University
51 Prospect Street
New Haven, CT
USA

Email: qiao.xiang@cs.yale.edu

J. Jensen Zhang
Tongji/Yale University
51 Prospect Street
New Haven, CT
USA

Email: jingxuan.zhang@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

Harvey Newman
California Institute of Technology
1200 California Blvd.
Pasadena, CA
USA

Email: newman@hep.caltech.edu