ALTO WG
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

S. Chen
X. Lin
Tongji University
D. Lachos
Unicamp
Y. Yang
Tongji/Yale University
C. Rothenberg
Unicamp

July 2, 2018

ALTO Implementations and Use Cases: A Brief Survey
draft-chen-alto-survey-00

Abstract

This document provides a comprehensive survey of ALTO, including current ALTO implementations and ALTO used in the literature work. This document identifies possible challenges and future opportunities of ALTO.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 3
1.1. ALTO Background 3
1.2. Goals and Scope 3
1.3. Document Organization 3
2. ALTO Implementations 3
2.1. Services Implemented in Projects 3
2.2. ALTO in Software Defined Mobile Networks 4
2.3. Recognized Issues and Future Work 4
3. ALTO in Literature/Use Cases 5
3.1. Peer Selection 6
3.1.1. Peer Selection in P2P 6
3.1.2. Surrogate Selection in CDN 6
3.1.3. Downstream CDN Selection in CDNI 6
3.1.4. Cache Selection in Hybrid CDN-P2P System 6
3.1.5. Mobile Edge Caching 7
3.2. Path Selection 7
3.2.1. Path Selection in MPTS-AR 7
3.3. Resource Placement 8
3.3.1. Virtualized Service Function Chain Placement 8
3.3.2. Intelligent Virtual Machine Placement 8
3.3.3. Service Placement in IoT 8
3.4. Measure Results Interface 8
3.4.1. An Interface to Query on the LMAP measure results . . 8
4. Challenges and Research Opportunities 9
4.1. Possible New Address Format 9
4.2. Mistrust between entities 9
4.3. Bidirectional Network/Application collaboration 9
5. Acknowledgments 9
6. Security Considerations 9
7. References 9
7.1. Normative References 10

7.2. Informative References 10
Appendix A. Questionnaire 10
Authors' Addresses 12

1. Introduction

1.1. ALTO Background

The Application-Layer Traffic Optimization (ALTO) protocol [RFC7285] is a client/server protocol operating between an ALTO client and an ALTO server. An ALTO server provides network related information to ALTO clients so that ALTO clients can make better decisions. ALTO is initially proposed to help distributed applications (P2P and client/server used for file sharing, real-time communication, etc.) to choose a better resource from multiple replicas. Recently, as new use cases like Content Delivery Network (CDN), Software-Define Network (SDN) and Datacenter Network (DCN) in single and multi domain environments emerge to utilize ALTO, they will bring forth new requirements and challenges on ALTO protocol.

1.2. Goals and Scope

This document aims to provide a comprehensive survey of ALTO, it covers: 1) ALTO implementations in different companies/institutions; 2) Application scenarios of ALTO in literature work through a lot of paper reading. With the survey of above topics, we want to explore: 1) The system architecture of ALTO implementation and possible issues in realizing the system; 2) Current use cases and future uses cases of ALTO; 3) Possible ALTO extensions to support future use cases.

1.3. Document Organization

The document is organized as below:

In Section 2, we present the survey of ALTO implementation and its results. In Section 3, we collect ALTO use cases in literature work and classified them based on ??? criteria. Finally, based on the ALTO use cases in literature work, we will show possible ALTO extensions and research opportunities of ALTO in Section 4.

2. ALTO Implementations

2.1. Services Implemented in Projects

Abbreviations:

- NM: network map

- CM: cost map
- FM: filtered map
- MC: multi-cost
- CC: cost calendar
- PV: path vector
- SDMN: software defined mobile networks

2.2. ALTO in Software Defined Mobile Networks

ALTO in software defined mobile networks (SDMN) project is a sub-project of Celtic-Plus SIGMONA, and it is implemented by Budapest University of Technology and Economics.

As the elasticity and portability of network functions becomes reality, the significance of service endpoint selection increases. ALTO is a standard protocol, which can provide guidance in service endpoint selection. So ALTO is very suitable for this case.

This project has implemented network map and cost map, and it introduced two new interfaces namely ALTO client-to-redirect server interface and ALTO network-to-server interface.

For cost calculation, this project regarded the number of switches between end hosts as "num-routing" and the average utilization of the path in the last 5 seconds as "num-delay".

The ALTO server in this project is standalone, and the ALTO client in this project is embedded into a SDN controller.

2.3. Recognized Issues and Future Work

Extension of dynamic network and cost map information provision:

- From non-SDN network segments
- From multiple network controllers (merging information in ALTO server)

ALTO protocol applies abstract network and cost maps:

- Pros: privacy of topology information
- Cons: sometimes not enough fine grained

- o Rank aggregation: which method should we select
- o Support of non reactive packet forwarding methods

3. ALTO in Literature/Use Cases

We make a comprehensive survey of ALTO used in literature work. We collect *** ALTO related papers, study the use cases and we have make an initial classification of use cases. The taxonomy of all the use cases is below.

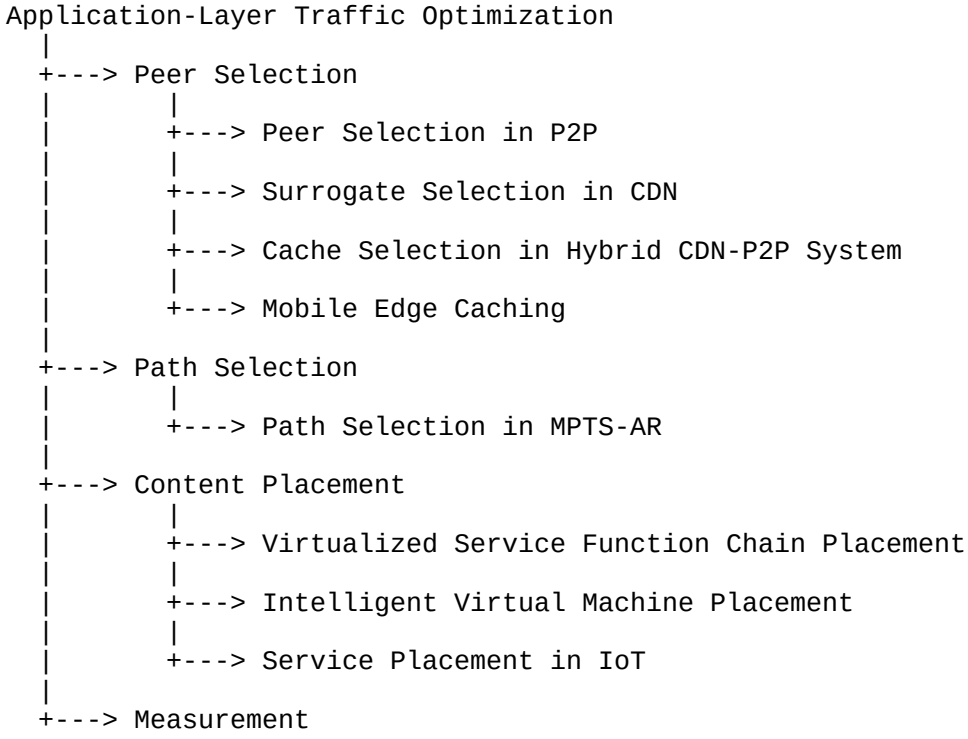

Figure 1: Taxonomy of ALTO Literature: Version A

the use cases are already mentioned in [RFC7971]. In the following sections, we will describe each use cases one by one. Some of the use cases are already mentioned in [RFC7971]. We classified all the related work into 4 categories and will describe each in the following.

3.1. Peer Selection

3.1.1. Peer Selection in P2P

3.1.1.1. Usage Scenario

Without little knowledge of the underlying network informtion, users in P2P system retrieve data from a random peer. Such a random peer selection mechanism have some disadvantages: 1. It causes an extra burden in the internet traffic. 2. It reduces the system performance.

3.1.2. Surrogate Selection in CDN

Content Delivery Networks (CDNs) have been used to deliver some Internet services such as video stream, websites or software updates because they provide numerous benefits including reduced delivery cost for cacheable content, improved QoS for end users and increased robustness of delivery.

The primary use case for ALTO in a CDN context is to improve the selection of a surrogate, cache or origin. By providing cost maps and network maps, ALTO can help a CDN to optimize selection of its surrogate, cache or origin.

3.1.3. Downstream CDN Selection in CDNI

Above section mainly talks about the surrogate selection in the CDN from a single administrative domain. However, some CDNs from different administrative domains may collaborate together to provide a broader coverage area. In such case, an upstream CDN needs to select the best downstream CDN For each content request it intends to redirect.

There are two tasks need to be done when an upstream CDN selects a downstream CDN. The first one is which dCDN is willing to accept a delegated request, and the second one is which dCDN is the most proper for redirecting the content request. For the first task, ALTO CDNI FCI Map can be used by uCDN to actively or passively gather dCDNs information such as footprints. For the second task, ALTO cost map can be used by uCDN to select the best surrogate with the smallest cost.

3.1.4. Cache Selection in Hybrid CDN-P2P System

Hybrid CDN-P2P system propose a two-tier topologies. The first level (CDN-L1) is composed by high capacity servers strategically placed in order to increase network backbone capacity. A second level (CDN-L0)

consists of Level-0 Entry Point (L0-EP). These node are equipped with some caching functionalities and placed geographically closer to end users. The L0-EP includes a standard HTTP proxy frontend used to optimally redirect the HTTP GET requests of the users to the best content location in CDN-L0 or CDN-L1. The decision logic module residing in L0-EP considers 3 factors (the availability of the content in the L0-EP local cache, the list of cache nodes that can contribute to retrieve the content and the locality awareness) to make its final decision.

The locality information here is provided through ALTO interface from the network monitors.

3.1.5. Mobile Edge Caching

3.1.5.1. Usage Scenario

Mobile Edge Caching is regarded to be possible to reduce the latency for users to retrieve content, thus improving the perveived Quality of Experience. The main technique in mobile edge caching is to equip edge network elements with cache capabilities. Popular contents can be stored at several caching servers at edge.

3.1.5.2. Applicability of ALTO

In the context of mobile edge caching, the information that a User Equipment (UE) tries to access, such as a document or a video, is referred to a s a Named Data Object (NDO). [MEC] proposes that ALTO can perform like a DNS server. It can be used to discover the caching servers that host the desired resource . To leverage ALTO for resource discovery, ALTO will regard each resource a PID. The IP addresses that a PID contains are the caching servers holding specific NDO. As response to the ALTO query, ALTO server will return a network map with a list of PIDs, each with IP addresses of cache servers. Such discovery mechanism allows the operator to inform the UE about all the cache servers that contain the required NDO in single operation. Second, the UE can use additional ALTO features (such as cost maps) to inquire which cache-server is preferred in terms of data routing costs (i.e., availble bandwidth, network load and others).

3.2. Path Selection

3.2.1. Path Selection in MPTS-AR

3.3. Resource Placement

3.3.1. Virtualized Service Function Chain Placement

3.3.2. Intelligent Virtual Machine Placement

3.3.2.1. Usage Scenario

The cloud management system can optimize resource placement by taking into account the network performance as well as the compute and storage resources.

3.3.2.2. Applicability of ALTO

3.3.2.3. Challenges and Opportunities

3.3.3. Service Placement in IoT

3.3.3.1. Usage Scenario

With the cutting-edge paradigms like Internet of Things and Smart Cities develop, new services have special requirements, one of which is low latency levels. A delayed reply could render to chaos for applications such as eHealth and public safety. One of the solutions is a smart service placement system that facilitates the location of services in the proper position according to specific needs. Locating the services just on wireless hop away from the users is not as simple as it sounds, since the service could be wrongfully placed in a congested location, or even farther from the users, which would lead to a greater latency. Thus, efficient service placement mechanisms are needed in order to take advantage of the current conditions of the network while minimizing the service latency.

3.3.3.2. Applicability of ALTO

3.3.3.3. Challenges of ALTO/ Opportunities of ALTO

3.4. Measure Results Interface

3.4.1. An Interface to Query on the LMAP measure results

In the context of LMAP, measurement results are made available to the public either at the finest granularity level, or in a very high level human-readable format. ALTO can provide an intermediate way to access large-scale network measurement result, and it can be used to tweak the aggregation-level of measurement results.

Controller sends instructions to measurement agent to do the measurement. And then measurement result will be transferred to the collector by report protocol. Just providing finest granularity level or providing a very high level human-readable format are not enough. ALTO can provide an intermediate way to access large-scale network measurement result, and it can be used to tweak the aggregation-level of measurement results.

4. Challenges and Research Opportunities

4.1. Possible New Address Format

Some scenarios may require new address format.

4.2. Mistrust between entities

Application developers and network operators are natural mistrust, with both parties reluctant to share what they consider sensitive information.

4.3. Bidirectional Network/Application collaboration

ALTO is originally designed to provide network information to applications so that applications can achieve better performance while the cost of the network can be reduced. Exposing non-critical information from application providers to network providers may potentially bring more benefits. A paper named "Bidirectional Network Collaboration Interface for CDNs and Clouds Services Traffic Optimization" describes this idea, and it proposes the constraint map which provides application needs and constraints to network providers.

5. Acknowledgments

...

6. Security Considerations

This draft is a survey of existing literature on ALTO implementations and use cases, it does not introduce any new security considerations to be taken into account beyond what is already discussed in each paper surveyed.

7. References

7.1. 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>.

[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>.

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

7.2. Informative References

[MEC] Poderys, Justas. and Matteo. Artuso, "Caching at the Mobile Edge: A Practical Implementation", 2018.

[SDN] Diego, Kreutz. and Ramos. Fernando MV, "Software-defined networking: A comprehensive survey", 2015.

[VPN] Scharf, Michael., "Dynamic VPN Optimization by ALTO Guidance", 2013.

Appendix A. Questionnaire

```
   Company/Organization Name:
   Project Name:
   Motivation:
   System Architecture:
   What map services that you have implemented:
       Map Service
           |
           +---> Cost Map
           |
           +---> Network Map
       Map Filtering Service
       Endpoint Cost Service
       Endpoint Property Service
       Unified Property Service
       Multi-cost
       Cost Calendar
       Path Vector

   Four Entites in ALTO Implementation
    |
    +---> Source of network informaton
    |
    +---> ALTO Server
    |     |
    |     +---> Who oeperates the ALTO server?
    |     |       |
    |     |       +---> network operator
    |     |       |
    |     |       +---> third parties
    |     |       |
    |     |       +---> user communities
    |     |
    |     +---> How ALTO server groups endpoints (PID)
    |     |
    |     +---> How ALTO server calculates costs
    |
    +---> ALTO client
    |      |
    |      +---> network management entity
    |      |
    |      +---> peer (endpoint) entity
    |
    +---> Information Consumer

    Main Benefits of Using ALTO (System and Service Performance)

    Recognized Issues
```

Authors' Addresses

Shiwei Dawn Chen
Tongji University
4800 Cao'an Road
Shanghai 201804
China

Email: dawn_chen_f@hotmail.com

X.S. Lin
Tongji University
4800 Cao'an Road
Shanghai 201804
China

Email: x.shawn.lin@gmail.com

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/

Y. Richard Yang
Tongji/Yale University
51 Prospect Street
New Haven, CT 06511
USA

Email: yry@cs.yale.edu

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 WG
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

J. Zhang
Tongji University
K. Gao
Tsinghua University
J. Wang
Tongji University
Q. Xiang
Tongji/Yale University
Y. Yang
Yale University

July 2, 2018

ALTO Extension: Flow-based Cost Query
draft-gao-alto-fcs-06.txt

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.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 3
2. Terminology 5
2.1. Flow 5
2.2. Data Transmission Type 5
3. Overview of Approaches 5
3.1. Extended Endpoint Address 5
3.2. Flow-based Filter 6
3.3. Flow-specific Announcement 6
4. Change Logs 7
5. Extended Endpoint Address 8
5.1. Address Type 8
5.2. Endpoint Address 9
5.2.1. MAC Address 9
5.2.2. Internet Domain Name 9
5.2.3. IPv4 Socket Address 9
5.2.4. IPv6 Socket Address 9
5.3. Address Type Compatibility 10
5.4. Examples 10
6. Extended Cost Query Filters 10

6.1. Filtered Cost Map Extension 10
6.1.1. Capabilities 11
6.1.2. Accept Input Parameters 11
6.2. Response 12
6.3. Endpoint Cost Service Extension 12
6.3.1. Capabilities 13
6.3.2. Accept Input Parameters 14
6.4. Response 15
6.5. Examples 15
6.5.1. Information Resource Directory 15
6.5.2. Flow-based Filtered Cost Map Example 17
6.5.3. Flow-based Endpoint Cost Service Example #1 18
6.5.4. Flow-based Endpoint Cost Service Example #2 19
7. Security Considerations 21
8. IANA Considerations 22
8.1. ALTO Address Type Registry 22
8.2. ALTO Address Type Compatibility Registry 22
9. Acknowledgment 23
10. References 23
10.1. Normative References 23
10.2. Informative References 24
Authors' Addresses 25

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, ALTO cost query services can be regarded as functions transforming a given subset of a specific query space into a network view abstract. However, the current specification has some limitations.

First, in the base ALTO protocol [RFC7285], the endpoint cost 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 and have the same IP address but different paths with different QoS metrics.

Second, in the base ALTO protocol [RFC7285], the query space is defined by the lists of sources and destinations. 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., with more fine-grained endpoint address types.

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

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 5 introduces several new address types that extend the query space of ALTO cost services. Section 6 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 7 and Section 8 discuss security and IANA considerations.

2. Terminology

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

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.

2.2. Data Transmission Type

This document use the term "Data Transmission Type" or "Transmission Type" to indicate the method of applications send the network flows. It can be unicast, broadcast or multicast.

3. 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].

3.1. 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, ALTO servers SHOULD return the corresponding error.

3.2. 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-1 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.

3.3. Flow-specific Announcement

Some informations are flow-specific and hard to be encoded into endpoints, e.g., the data transmission type of a flow. These informations may help the ALTO client get more accurate costs.

To allow the ALTO client to specify these informations (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. The announcement can be 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, more supported information in the announcement can be defined in the future documents. And how to interpret those informations depends on the implementation. It is not in the scope of this document.

4. Change Logs

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

This section records the change logs of the draft updates.

Changes since -05 revision:

- 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.

5. Extended Endpoint Address

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

5.1. 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.

5.2. Endpoint Address

This document defines EndpointAddr when AddressType is in Section 8.1.

5.2.1. 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].

5.2.2. 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.

5.2.3. 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 5.2.2). The Port component of an IPv4 socket address is encoded as an integer between 1 and 65535.

5.2.4. 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 or an IPv6-compatible domain name (see Section 5.2.2). The Port component of IPv6 socket address is encoded as an integer between 1 and 65535.

5.3. 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 8.2. 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.

5.4. 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"
```

6. 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 rule for optional fields in Section 4 of [RFC8189].

6.1. 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 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 unchanged.

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.

6.1.1. 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;]
  [JSONString 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 field. In this document, only "transmission-type" is defined.

6.1.2. 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 {
  [JSONObject flow-spec-announce;]
} ExtPIDFilter : PIDFilter;
```

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 a JSONObject. Each key of this JSONObject MUST be chosen from the list specified by the capability "flow-spec-announce", and the value of each key depends on the key itself.

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

6.2. 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.

6.3. 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.

6.3.1. 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..*>;]
      [JSONString  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 field. In this document, only "transmission-type" is defined.

6.3.2. 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 {
  [JSONObject 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 value of this field is a JSONObject. Each key of this JSONObject MUST be chosen from the list specified by the capability "flow-spec-announce", and the value of each key depends on the key itself.

If the AddressType of the source and destination in the same EndpointFilter do not conform the compatibility rule defined in Table 1 of Section 8.1, 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.

6.4. 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.

6.5. Examples

6.5.1. 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"
        },
        "propmap-availbw": {
          "uri": "http://alto.exmaple.com/propmap/ane-prop",
          "media-type": "application/alto-propmap+json",
          "accepts": "application/alto-propmapparams+json",
          "capabilities": {
            "domain-types": [ "ane" ],
            "prop-types": [ "availbw" ]
          },
          "uses": [ "path-vector-endpoint-cost" ]
        }
        "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" : "application/alto-endpointcost+json",
          "accepts" : "application/alto-endpointcostparams+json",
          "capabilities" : {
            "address-types": ["tcp", "tcp6"],
            "flow-based-filter" : true,
            "cost-type-names" : [ "path-vector" ],
            "flow-spec-announce" : [ "transmission-type" ],
            "dependent-property-map" : "propmap-availbw",
            "allow-compound-response" : true
          }
        }
      }
    }
```

6.5.2. 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 },
    "PID1": { "PID3": 2 },
    "PID3": { "PID4": 1 }
  }
}
```

6.5.3. 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
    }
  }
}
```

6.5.4. 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 /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": "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" } }
  ],
  "properties": ["availbw"]
}
```

```
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": {
      "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" ],
    }
  },
  "property-map": {
    "ane:S1": { "availbw": 100 },
    "ane:S2": { "availbw": 100 },
    "ane:D1": { "availbw": 150 },
    "ane:D2": { "availbw": 80 },
    "ane:D3": { "availbw": 150 },
    "ane:G1": { "availbw": 100 },
    "ane:G2": { "availbw": 100 }
  }
}
```

7. 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.

8. 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.

8.1. 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 5.2.1	None	Mapping to/from IPv4 by [RFC0903] and [RFC0826]; Mapping to/from IPv6 by [RFC3122] and [RFC4861]
domain	See Section 5.2.2	None	Mapping to/from IPv4 by [RFC1034]
domain6	See Section 5.2.2	None	Mapping to/from IPv6 by [RFC3596]
tcp	See Section 5.2.3	None	No mapping
tcp6	See Section 5.2.4	None	No mapping
upd	See Section 5.2.3	None	No mapping
udp6	See Section 5.2.4	None	No mapping

Table 1: ALTO Address Type Registry

8.2. 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 5.3.

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.

9. 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.

10. References

10.1. Normative 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., 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., "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>.

[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>.

[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>.

10.2. Informative References

[EUI48] IEEE, , "Guidelines for use of a 48-bit Extended Unique Identifier (EUI-48)", 2012, <http://standards.ieee.org/develop/regauth/tut/eui48.pdf>.

[EUI64] IEEE, , "Guidelines for use of a 64-bit Extended Unique Identifier (EUI-64)", November 2012, <http://standards.ieee.org/develop/regauth/tut/eui64.pdf>.

[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 Type", draft-ietf-alto-path-vector-03 (work in progress), March 2018.

[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>.

[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>.

Authors' Addresses

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

Email: jingxuan.n.zhang@gmail.com

Kai Gao
Tsinghua University
30 Shuangqinglu Street
Beijing 100084
China

Email: gaok12@mails.tsinghua.edu.cn

Junzhuo Austin Wang
Tongji University
4800 Cao'an Hwy, Jiading District
Shanghai
China

Email: wangjunzhuo200@gmail.com

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

Email: qiao.xiang@cs.yale.edu

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

Email: yry@cs.yale.edu

CDNI
Internet-Draft
Intended status: Standards Track
Expires: December 20, 2018

J. Seedorf
HFT Stuttgart - Univ. of Applied Sciences
Y. Yang
Tongji/Yale
K. Ma
Ericsson
J. Peterson
Neustar
X. Lin
Tongji

June 18, 2018

Content Delivery Network Interconnection (CDNI) Request Routing: CDNI Footprint and Capabilities Advertisement using ALTO draft-ietf-alto-cdni-request-routing-alto-03

Abstract

The Content Delivery Networks Interconnection (CDNI) WG is defining a set of protocols to inter-connect 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 is the CDNI Request Routing Footprint & Capabilities Advertisement interface (FCI) [RFC7336]. [RFC8008] has defined precisely the semantics of FCI and provided guidelines on the FCI protocol, but the exact protocol is explicitly outside the scope of that document. In this document, we define an FCI protocol using the Application-Layer Traffic Optimization (ALTO) protocol.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 20, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 3
2. Background 4
2.1. Semantics of FCI Advertisement 4
2.2. ALTO Background and Benefits 6
3. CDNI FCI Map 8
3.1. Media Type 9
3.2. HTTP Method 9
3.3. Accept Input Parameters 9
3.4. Capabilities 9
3.5. Uses 9
3.6. Response 9
3.7. Examples 10
3.7.1. IRD Example 10
3.7.2. Basic Example 12
3.7.3. Incremental Updates Example 13
4. CDNI FCI Map using ALTO Network Map 15
4.1. Introduce Footprint Type: altonetworkmap 15
4.2. Examples 15
4.2.1. IRD Example 15
4.2.2. ALTO Network Map for CDNI FCI Footprints Example . . 15
4.2.3. ALTO Network Map Footprints in CDNI FCI Map 16
4.2.4. Incremental Updates Example 17
5. Filtered CDNI FCI Map using Capabilities 19
5.1. Media Type 19
5.2. HTTP Method 19
5.3. Accept Input Parameters 19
5.4. Capabilities 20
5.5. Uses 20
5.6. Response 20
5.7. Examples 21
5.7.1. IRD Example 21

5.7.2. Basic Example 21
5.7.3. Incremental Updates Example 22
6. Query Footprint Properties using ALTO Unified Property Service 24
6.1. Representing Footprint Objects as Unified Property Map Entities 24
6.1.1. ASN Domain 25
6.1.2. COUNTRYCODE Domain 25
6.2. Examples 26
6.2.1. IRD Example 26
6.2.2. Property Map Example 26
6.2.3. Filtered Property Map Example 27
6.2.4. Incremental Updates Example 28
7. Protocol Errors 30
8. IANA Considerations 30
8.1. CDNI Metadata Footprint Type Registry 30
8.2. ALTO Entity Domain Registry 31
8.3. ALTO CDNI FCI Property Type Registry 31
9. Security Considerations 31
10. Acknowledgments 32
11. References 32
11.1. Normative References 32
11.2. Informative References 33
Authors' Addresses 33

1. Introduction

Many Network Service Providers (NSPs) are currently considering or have already started to deploy Content Delivery Networks (CDNs) within their networks. As a consequence of this development, there is a need for interconnecting these local CDNs. Content Delivery Networks Interconnection (CDNI) has the goal of standardizing protocols to enable such interconnection of CDNs [RFC6707].

The CDNI problem statement [RFC6707] defines four interfaces to be standardized within the IETF for CDN interconnection:

- CDNI Request Routing Interface
- CDNI Metadata Interface
- CDNI Logging Interface
- CDNI Control Interface

The main purpose of the CDNI Request Routing Interface is described in [RFC6707] as follows: "The CDNI Request Routing interface enables a Request Routing function in an Upstream CDN to query a Request

Routing function in a Downstream CDN to determine if the Downstream CDN is able (and willing) to accept the delegated Content Request. It also allows the Downstream CDN to control what should be returned to the User Agent in the redirection message by the upstream Request Routing function." On a high level, the scope of the CDNI Request Routing Interface, therefore, contains two main tasks:

- o determining if the downstream CDN (dCDN) is willing to accept a delegated content request;
- o redirecting the content request coming from an upstream CDN (uCDN) to the proper entry point or entity in the downstream CDN.

Correspondingly, the request routing interface is broadly divided into two functionalities:

- o CDNI Footprint & Capabilities Advertisement interface (FCI): the advertisement from a dCDN to a uCDN or a query from a uCDN to a dCDN for the uCDN to decide whether to redirect particular user requests to that dCDN;
- o CDNI Request Routing Redirection interface (RI): the synchronous operation of actually redirecting a user request.

This document focuses solely on CDNI FCI, with a goal to specify a new Application-Layer Traffic Optimization (ALTO) [RFC7285] service called "CDNI FCI Map Service", to transport and update CDNI FCI objects, which are defined in a separate document in [RFC8008] and to describe a mechanism for filtering CDNI FCI map on capabilities or footprints.

Throughout this document, we use the terminology for CDNI defined in [RFC6707] and [RFC8008].

2. Background

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

2.1. Semantics of FCI Advertisement

The CDNI document on "Footprint and Capabilities Semantics" [RFC8008] defines the semantics for the CDNI FCI. It thus provides guidance on what Footprint and Capabilities mean in a CDNI context and how a protocol solution should in principle look like. The definitions in [RFC8008] depend on [RFC8006]. Here we briefly summarize key related

points of [RFC8008] and [RFC8006]. For a detailed discussion, the reader is referred to the RFCs.

- Footprint and capabilities are tied together and cannot be interpreted independently from each other. In such cases, i.e. where capabilities must be expressed on a per footprint basis, it may be beneficial to combine footprint and capabilities advertisement. [RFC8008] integrates footprint and capabilities with an approach of "capabilities with footprint restrictions".

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

- Multiple types of footprints are defined in [RFC8006]:

  * List of ISO Country Codes

  * List of AS numbers

  * Set of IP-prefixes

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

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

a candidate if it covers the country of the request routing source. The CDNI specifications do not define how a given uCDN determines the country of the request routing source. Multiple footprint constraints are additive, i.e. the advertisement of different types of footprint narrows the dCDN candidacy cumulatively.

- The following capabilities are defined as "base" capabilities, i.e. ones that are needed in any case and therefore constitute mandatory capabilities to be supported by the CDNI FCI:
  - Delivery Protocol (e.g., HTTP vs. RTMP)
  - Acquisition Protocol (for acquiring content from a uCDN)
  - Redirection Mode (e.g., DNS Redirection vs. HTTP Redirection as discussed in [RFC7336])
  - Capabilities related to CDNI Logging (e.g., supported logging mechanisms)
  - Capabilities related to CDNI Metadata (e.g., authorization algorithms or support for proprietary vendor metadata)

2.2. ALTO Background and Benefits

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

Originally, ALTO was motivated by the huge amount of cross-ISP traffic generated by P2P applications [RFC5693]. Recently, however, ALTO is also being considered for improving the request routing in CDNs [I-D.jenkins-alto-cdn-use-cases]. In this context, it has also been proposed to use ALTO for selecting an entry-point in a downstream NSP’s network (see section 3.4 "CDN delivering Over-The-Top of a NSP’s network" in [I-D.jenkins-alto-cdn-use-cases]). Also, the CDNI problem statement explicitly mentions ALTO as a candidate protocol for "actual algorithms for selection of CDN or Surrogate by Request-Routing systems" [RFC6707].

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

- CDN request routing is done at the application layer. ALTO is a protocol specifically designed to improve application layer traffic (and application layer connections among hosts on the Internet) by providing additional information to applications that these applications could not easily retrieve themselves. For CDNI, this is exactly the case: a uCDN wants to improve application layer CDN request routing by using dedicated information (provided by a dCDN) that the uCDN could not easily obtain otherwise.

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

- Security: ALTO maps can be signed and hence provide inherent integrity protection (see Section 9).

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

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

- Filtered network map: The ALTO Map Filtering Service (see [RFC7285] for details) would allow a uCDN to query only for parts of an ALTO map.

- Server-initiated Notifications and Incremental Updates: In case the footprint or the capabilities of a downstream CDN change abruptly (i.e. unexpectedly from the perspective of an upstream CDN), server-initiated notifications would enable a dCDN to directly inform an upstream CDN about such changes. Consider the case where - due to failure - part of the footprint of the dCDN is not functioning, i.e. the CDN cannot serve content to such clients with reasonable QoS. Without server-initiated notifications, the uCDN might still use a very recent network and cost map from dCDN, and therefore redirect requests to dCDN which it cannot serve. Similarly, the possibility for incremental updates would enable

efficient conveyance of the aforementioned (or similar) status changes by the dCDN to the uCDN. The newest design of ALTO supports server pushed incremental updates [I-D.ietf-alto-incr-update-sse].

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

- Resource Availability on Hosts or Links: The capabilities on links (e.g. maximum bandwidth) or caches (e.g. average load) might be useful information for an upstream CDN for optimized downstream CDN selection. For instance, if a uCDN receives a streaming request for content with a certain bitrate, it needs to know if it is likely that a dCDN can fulfill such stringent application-level requirements (i.e. can be expected to have enough consistent bandwidth) before it redirects the request. In general, if ALTO could convey such information via new endpoint properties, it would enable more sophisticated means for downstream CDN selection with ALTO.

3. CDNI FCI Map

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

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

3.1. Media Type

The media type of the CDNI FCI Map is "application/alto-cdnifcimap+json".

3.2. HTTP Method

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

3.3. Accept Input Parameters

None.

3.4. Capabilities

None.

3.5. Uses

The resource ID of the resource based on which the CDNI FCI map will be defined. For example, if a CDNI FCI map depends on a network map, the resource ID of the network map MUST be included in "Uses" field. Please see Section 4 for details. If the CDNI FCI map does not depend on any other resources, "Uses" field MUST NOT appear.

3.6. Response

If a CDNI FCI map does not depend on other resources, the "meta" field of a CDNI FCI map response MUST include the "vtag" field defined in Section 10.3 of [RFC7285], which provides the version tag of the retrieved CDNI FCI map. If a CDNI FCI map response depends on a resource such a network map, it MUST include the "dependent-vtags" field, whose value is an array to indicate the version tag of the resource used, where the resource is specified in "uses" of the IRD. The current defined dependent resource is only network map, and the usage of it is described in Section 4. The data component of an ALTO CDNI FCI map response is named "cdni-fci-map", which is a JSON object of type CDNIFCIMapData:

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

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

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

The ALTO client MUST interpret footprints appearing multiple times as if they appeared only once. If no footprint restriction list is specified (or an empty list is specified), the ALTO client MUST understand that all footprint types are reset to "global" coverage.

Note: Further optimization of BaseAdvertisement objects to effectively provide the advertisement of capabilities with footprint restrictions is certainly possible, however, it is not necessary for the basic interconnection of CDNs. The note here is for completeness, however, the specifics of such mechanisms are outside the scope of this document.

3.7. Examples

3.7.1. IRD Example

Below is an example IRD announcing two network maps, one CDNI FCI map without dependency, one CDNI FCI map depending on a network map, one filtered CDNI FCI map, one unified property map including "cdni-fci-capabilities" as its entities' property, one filtered unified property map including "cdni-fci-capabilities" and "pid" as its entities' properties and two update stream services (one for updating CDNI FCI maps, and the other for updating property maps).

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

{
  "meta" : { ... },
  "resources": {
    "my-default-network-map": {
      "uri" : "http://alto.example.com/networkmap",
      "media-type" : "application/alto-networkmap+json"
    },
    "my-eu-netmap" : {
      "uri" : "http://alto.example.com/myeunetmap",
      "media-type" : "application/alto-networkmap+json"
    },
    "my-default-cdnifci-map": {
```

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

```
            "my-filtered-cdnifci-map" :
            "application/merge-patch+jso,application/json-patch+json",
            "my-cdnifci-map-with-network-map-footprints" :
            "application/merge-patch+json,application/json-patch+json"
          }
        }
      },
      "update-my-props": {
        "uri" : "http://alto.example.com/updates/properties",
        "media-type" : "text/event-stream",
        "uses" : [
          "cdnifci-property-map",
          "filtered-cdnifci-property-map"
        ],
        "capabilities" : {
          "incremental-change-media-types": {
            "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

In this example, we demonstrate a simple CDNI FCI map; this map does not depend on other resources.

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

```
    HTTP/1.1 200 OK
    Content-Length: XXX
    Content-Type: application/alto-cdnifcimap+json
    {
      "meta" : {
        "vtag": {
          "resource-id": "my-default-cdnifci-map",
          "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
        }
      },
      "cdni-fci-map": {
        "capabilities": [
          {
```

```
              "capability-type": "FCI.DeliveryProtocol",
              "capability-value": {
                "delivery-protocols": [
                  "http/1.1"
                ]
              },
              "footprints": [
                <Footprint objects>
              ]
            },
            {
              "capability-type": "FCI.DeliveryProtocol",
              "capability-value": {
                "delivery-protocols": [
                  "https/1.1",
                  "http/1.1"
                ]
              },
              "footprints": [
                <Footprint objects>
              ]
            },
            {
              "capability-type": "FCI.AcquisitionProtocol",
              "capability-value": {
                "acquisition-protocols": [
                  "https/1.1"
                ]
              },
              "footprints": [
                <Footprint objects>
              ]
            }
          ]
        }
      }
```

3.7.3. Incremental Updates Example

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

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

```
Content-Length: ###

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

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

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

event: application/alto-cdnifcimap+json,my-fci-stream
data: { ... full CDNI FCI map ... }

event: application/merge-patch+json,my-fci-stream
data: {
data:   "meta": {
data:     "vtag": {
data:       "tag": "dasdfa10ce8b059740bddsfasd8eb1d47853716"
data:     }
data:   },
data:   {
data:     "capability-type": "FCI.DeliveryProtocol",
data:     "capability-value": {
data:       "delivery-protocols": [
data:         "http/1.1"
data:       ]
data:     },
data:     "footprints": [
data:       <Footprint objects that are different from
data:        footprint objects in delivery-protocols http/1.1>
data:     ]
data:   }
data: }

event: application/json-patch+json,my-fci-stream
data: [
data:   {
data:     "op": "replace",
data:     "path": "/meta/vtag/tag",
data:     "value": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
data:   },
data:   { "op": "add",
```

```
data:     "path": "/cdni-fci-map/capabilities/0/footprints/-",
data:     "value": "ipv4:192.0.2.0/24"
data:   }
data: ]
```

4. CDNI FCI Map using ALTO Network Map

4.1. Introduce Footprint Type: altonetworkmap

In addition to the already defined CDNI footprint typs (e.g., ipv4cidr, ipv6cidr, asn, countrycode), ALTO network maps can be a type of FCI footprint. To enable such referencing to ALTO network maps, a new CDNI Footprint Type "altonetworkmap" is defined (see also Section 8.1).

All altonetworkmap entries MUST be of type PIDName (as defined in [RFC7285], where PIDName corresponds to a PID in the ALTO network map referenced by the resource ID of the network map listed in "dependent-vtags" field).

4.2. Examples

4.2.1. IRD Example

We use the same IRD example given by Section 3.7.1.

4.2.2. ALTO Network Map for CDNI FCI Footprints Example

Below is an example network map that is referenced by the CDNI FCI map example in Section 4.2.3

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

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

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

4.2.3. ALTO Network Map Footprints in CDNI FCI Map

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

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

```
HTTP/1.1 200 OK
Content-Length: 618
Content-Type: application/alto-cdnifcimap+json

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

4.2.4. Incremental Updates Example

In this example, the ALTO client is interested in changes of "my-cdnifci-map-with-network-map-footrprints". And we present two patches here. The first one of it is to change footprints of http/1.1 Delivery Protocol capability, and the second one is to remove "south-france" from the footprints of https/1.1 Delivery Protocol capability.

```
  POST /updates/cdnifcimaps HTTP/1.1
```

```
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

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

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

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

event: application/alto-cdnifcimap+json,my-fci-stream
data: { ... full CDNI FCI map ... }

event: application/merge-patch+json,my-fci-stream
data: {
data:   "meta": {
data:     "dependent-vtags" : [
data:      {
data:        "resource-id": "my-eu-netmap",
data:        "tag": "3ee2cb7e8d63d9fab71b9b34cbf764436315542e"
data:      }
data:     ],
data:     "vtag": {
data:       "tag": "dasdfa10ce8b059740bddsfasd8eb1d47853716"
data:     }
data:   },
data:   {
data:     "capability-type": "FCI.DeliveryProtocol",
data:     "capability-value": {
data:       "delivery-protocols": [
data:         "http/1.1"
data:       ]
data:     },
data:     "footprints": [
data:       <Footprint objects that are different from
data:        footprint objects in delivery-protocols http/1.1>
data:     ]
data:   }
data: }
```

```
event: application/json-patch+json,my-fci-stream
data: [
data:   {
data:     "op": "replace",
data:     "path": "/meta/vtag/tag",
data:     "value": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
data:   },
data:   { "op": "remove",
data:     "path": "/cdni-fci-map/capabilities/2/footprints/0/
data:       footprint-value/1",
data:   }
data: ]
```

5. Filtered CDNI FCI Map using Capabilities

This document defines a new service named "Filtered CDNI FCI Map Service". The semantic of a Filtered CDNI FCI Map is that given some capabilities, which footprints have at least one of these capabilities. And a filtered CDNI FCI map is a CDNI FCI map for which an ALTO client may supply additional capabilities to limit the scope of the resulting CDNI FCI map. The relationship between a filtered CDNI FCI map and a CDNI FCI Map is similar to the relationship between a filtered network/cost map and a network/cost map.

5.1. Media Type

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

5.2. HTTP Method

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

5.3. Accept Input Parameters

The input parameters for a filtered CDNI FCI map are supplied in the entity body of the POST request. This document specifies the input parameters with a data format indicated by the media type "application/alto-cdni-filter", which is a JSON object of type ReqFilteredCDNIFCIMap, where:

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

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

with fields:

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

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

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

5.4. Capabilities

None.

5.5. Uses

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

5.6. Response

The format is the same as an unfiltered CDNI FCI map. See Section 3.6 for the format.

The returned CDNI FCI map MUST contain only BaseAdvertisementObject objects whose CDNI capability object is the superset of one of CDNI capability object in "cdni-fci-capabilities". Specifically, that a CDNI capability object A is the superset of another CDNI capability

object B means that these two CDNI capability objects have the same capability type and mandatory properties in capability value of A MUST include mandatory properties in capability value of B semantically. For example, if a CDNI FCI capability in "cdni-fci-capabilities" is Delivery Protocol capability object with "http/1.1" in its field "delivery-protocols" and the original full CDNI FCI map has two CDNI FCI objects whose capabilities are Delivery Protocol capability objects with ["http/1.1"] and ["http/1.1", "https/1.1"] in their field "delivery-protocols" respectively, both of these two CDNI FCI objects MUST be returned. If the input parameters contain a CDNI capability object that is not currently defined, the ALTO server MUST behave as if the CDNI capability object did not appear in the input parameters.

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

5.7. Examples

5.7.1. IRD Example

We use the same IRD example by Section 3.7.1.

5.7.2. Basic Example

This example is filtering the full CDNI FCI map example in Section 3.7.2.

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

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

```
HTTP/1.1 200 OK
Content-Length: XXX
Content-Type: application/alto-cdnifcimap+json
{
  "meta" : {
    "vtag": {
      "resource-id": "my-default-cdnifci-map",
      "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
    }
  },
  "cdni-fci-map": {
    "capabilities": [
      {
        "capability-type": "FCI.DeliveryProtocol",
        "capability-value": {
          "delivery-protocols": [
            "http/1.1"
          ]
        },
        "footprints": [
          <Footprint objects>
        ]
      },
      {
        "capability-type": "FCI.DeliveryProtocol",
        "capability-value": {
          "delivery-protocols": [
            "https/1.1",
            "http/1.1"
          ]
        },
        "footprints": [
          <Footprint objects>
        ]
      }
    ]
  }
}
```

5.7.3. Incremental Updates Example

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

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

```
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

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

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

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

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

event: application/merge-patch+json,my-fci-stream
data: {
data:   "meta": {
data:     "vtag": {
data:       "tag": "dasdfa10ce8b059740bddsfasd8eb1d47853716"
data:     }
data:   },
data:   {
data:     "capability-type": "FCI.DeliveryProtocol",
data:     "capability-value": {
data:       "delivery-protocols": [
data:         "http/1.1"
data:       ]
data:     },
data:     "footprints": [
```

```
data:       <Footprint objects that are different from
data:        footprint objects in delivery-protocols http/1.1>
data:     ]
data:   }
data: }

event: application/json-patch+json,my-fci-stream
data: [
data:   {
data:     "op": "replace",
data:     "path": "/meta/vtag/tag",
data:     "value": "a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
data:   },
data:   { "op": "add",
data:     "path": "/cdni-fci-map/capabilities/0/footprints/-",
data:     "value": "ipv4:192.0.2.0/24"
data:   }
data: ]
```

6. Query Footprint Properties using ALTO Unified Property Service

In this section, we describe how ALTO clients look up properties for individual footprints. Our design decision here is to use ALTO unified property map service to query footprint properties because we do not want to introduce extra complexity and ALTO unified property map defined in [I-D.ietf-alto-unified-props-new] already meets the requirement. A footprint is a group of entities, and CDNI capabilities can be regarded as properties of a footprint. Unified property map is used to provide properties for collections of entities such as CIDRs or PIDs. So every footprint can be presented as a set of entities, and we will describe it in details in Section 6.1. In addition, two resource types Property Maps and Filtered Property Maps are already well-defined in [I-D.ietf-alto-unified-props-new].

A unified property map that includes "cdni-fci-capabilities" property registered in Section 8 builds the inverted index of a CDNI FCI map. The building process consists of two steps: firstly, each footprint object is represented as a set of unified property map entities in a domain; secondly, each unified property map entity is mapped into a list of property objects including CDNI capabilities.

6.1. Representing Footprint Objects as Unified Property Map Entities

A footprint object has two properties: footprint-type and footprint-value. A footprint-value is an array of footprint values conforming to the specification associated with the registered footprint type ("ipv4cidr", "ipv6cidr", "asn", and "countrycode"). Since each

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

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

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

6.1.1. ASN Domain

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

6.1.1.1. Domain Name

asn

6.1.1.2. Domain-Specific Entity Addresses

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

6.1.1.3. Hierarchy and Inheritance

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

6.1.2. COUNTRYCODE Domain

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

6.1.2.1. Domain Name

countrycode

6.1.2.2. Domain-Specific Entity Addresses

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

6.1.2.3. Hierarchy and Inheritance

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

6.2. Examples

6.2.1. IRD Example

We use the same IRD example given by Section 3.7.1.

6.2.2. Property Map Example

This example shows a full unified property map in which entities are footprints and entities' property is "cdni-fci-capabilities".

```
GET /propmap/full/cdnifci HTTP/1.1
HOST: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json
```

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

{
  "property-map": {
    "meta": {
      "dependent-vtags": [
        {"resource-id": "my-default-cdnifci-map",
         "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf62"}
      ]
    },
    "countrycode:us": {
      "cdni-fci-capabilities": [{"capability-type":,
                                 "capability-value":}]
    },
    "ipv4:192.0.2.0/24": {
      "cdni-fci-capabilities": [{"capability-type":,
                                 "capability-value":}]
    },
    "ipv4:198.51.100.0/24": {
      "cdni-fci-capabilities": [{"capability-type":,
                                 "capability-value":}]
    },
    "ipv6:2001:db8::/32": {
      "cdni-fci-capabilities": [{"capability-type":,
                                 "capability-value":}]
    },
    "asn:as64496": {
      "cdni-fci-capabilities": [{"capability-type":,
                                 "capability-value":}]
    }
  }
}
```

6.2.3. Filtered Property Map Example

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

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

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

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

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

6.2.4. Incremental Updates Example

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

```
POST /updates/properties HTTP/1.1
```

```
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

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

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

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

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

event: application/merge-patch+json,my-fci-stream
data: {
data:   "property-map":
data:   {
data:     "meta": {
data:       "dependent-vtags": [
data:         {"resource-id": "my-default-cdnifci-map",
data:          "tag": "2beeac8ee23c3dd1e98a73fd30df80ece9fa5627"},
data:         {"resource-id": "my-default-networkmap",
data:          "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf63"}
data:       ]
data:     },
data:     "ipv4:192.0.2.0/24":
data:     {
data:       "cdni-fci-capabilities":
data:         [{"capability-type":,"capability-value":}]
data:     }
data:   }
data: }
```

```
event: application/json-patch+json,my-fci-stream
data: {[
data: {
data:   { "op": "replace",
data:       "path": "/meta/dependent-vtags/0/tag",
data:       "value": "61b23185a50dc7b334577507e8f00ff8c3b409e4"
data:   },
data:   { "op": "replace",
data:       "path": "/property-map/ipv4:192.0.2.0/124/",
data:       "value": "pid5"
data:   }
data: }
data: ]}
```

7. Protocol Errors

Protocol errors are handled as specified in Section 8.5 of the ALTO protocol [RFC7285].

Here we explain the error-handling mechanism of filtered CDNI FCI map:

- E_SYNTAX covers all cases of syntax errors of filtered CDNI FCI map queries.

- When the syntax of queries is correct, there may be some errors in queries' semantics. Such Cases can be covered by E_INVALID_FIELD_VALUE:

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

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

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

The error-handling mechanism of query footprints is the same as the error-handling mechanism of ALTO Unified Property Map Service described in [I-D.ietf-alto-unified-props-new].

8. IANA Considerations

8.1. CDNI Metadata Footprint Type Registry

Footprint Type	Description	Specification
altonetworkmap	A list of PID-names	RFCthis

Table 1: CDNI Metadata Footprint Type

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

8.2. ALTO Entity Domain Registry

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

Identifier	Entity Address Encoding	Hierarchy & Inheritance
asn	See Section 6.1.1.2	None
countrycode	See Section 6.1.2.2	None

Table 2: ALTO Entity Domain

8.3. ALTO CDNI FCI Property Type Registry

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

Identifier	Intended Semantics
cdni-fci-capabilities	An array of CDNI FCI capability objects

Table 3: ALTO CDNI FCI Property Type

9. Security Considerations

One important security consideration is the proper authentication of advertisement information provided by a downstream CDN. The ALTO protocol provides a specification for a signature of ALTO information (see Section 15 of [RFC7285]. ALTO thus provides a proper mechanism for protecting the integrity of FCI information.

More Security Considerations will be discussed in a future version of this document.

10. Acknowledgments

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

Jan Seedorf is partially supported by the GreenICN project (GreenICN: Architecture and Applications of Green Information Centric Networking), a research project supported jointly by the European Commission under its 7th Framework Program (contract no. 608518) and the National Institute of Information and Communications Technology (NICT) in Japan (contract no. 167). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the GreenICN project, the European Commission, or NICT.

11. References

11.1. Normative References

[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>.

[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>.

[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>.

[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>.

[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>.

[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.

11.2. Informative References

[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-07 (work in progress), July 2017.

[I-D.jenkins-alto-cdn-use-cases] Niven-Jenkins, B., Watson, G., Bitar, N., Medved, J., and S. Previdi, "Use Cases for ALTO within CDNs", draft-jenkins-alto-cdn-use-cases-03 (work in progress), June 2012.

[I-D.ietf-alto-unified-props-new] Roome, W. and Y. Yang, "Extensible Property Maps for the ALTO Protocol", draft-ietf-alto-unified-props-new-00 (work in progress), July 2017.

Authors' Addresses

Jan Seedorf
HFT Stuttgart - Univ. of Applied Sciences
Schellingstrasse 24
Stuttgart 70174
Germany

Phone: +49-0711-8926-2801
Email: jan.seedorf@hft-stuttgart.de

Y.R. Yang
Tongji/Yale University
51 Prospect Street
New Haven, CT 06511
United States of America

Email: yry@cs.yale.edu
URI: http://www.cs.yale.edu/~yry/

Kevin J. Ma
Ericsson
43 Nagog Park
Acton, MA 01720
United States of America

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

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

Email: jon.peterson@neustar.biz

X.S. Lin
Tongji University
4800 Cao'an Hwy
Shanghai 201804
China

Email: x.shawn.lin@gmail.com

ALTO WG
Internet-Draft
Intended status: Standards Track
Expires: January 2, 2019

W. Roome
Nokia Bell Labs
Y. Yang
Tongji/Yale University
S. Chen
Tongji University

July 1, 2018

ALTO Incremental Updates Using Server-Sent Events (SSE)
draft-ietf-alto-incr-update-sse-12

Abstract

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

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

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

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

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

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

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

This Internet-Draft will expire on January 2, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 3
2. Major Changes Since Version -01 5
3. Terms 5
4. Background 6
  4.1. Server-Sent Events (SSEs) 6
  4.2. JSON Merge Patch 7
    4.2.1. JSON Merge Patch Encoding 7
    4.2.2. Merge Patch ALTO Messages 9
  4.3. JSON Patch 12
    4.3.1. JSON Patch Encoding 12
    4.3.2. JSON Patch ALTO Messages 12
5. Overview of Approach 14
6. Update Messages: Data Update and Control Update Messages . . 16
  6.1. ALTO Update Message Format 16
  6.2. ALTO Data Update Message 17
  6.3. ALTO Control Update Message 17
7. Update Stream Service 18
  7.1. Media Type 18
  7.2. HTTP Method 18

7.3. Accept Input Parameters 18
7.4. Capabilities 20
7.5. Uses 21
7.6. Response 21
7.6.1. Keep-Alive Messages 22
7.6.2. Event Sequence Requirements 22
7.6.3. Cross-Stream Consistency Requirements 22
8. Update Stream Control Service 23
8.1. URI 23
8.2. Media Type 24
8.3. HTTP Method 24
8.4. Accept Input Parameters 24
8.5. Capabilities & Uses 25
8.6. Response 25
9. Examples 26
9.1. Example: IRD Announcing Update Stream Services 26
9.2. Example: Simple Network and Cost Map Updates 28
9.3. Example: Advanced Network and Cost Map Updates 31
9.4. Example: Endpoint Property Updates 34
10. Client Actions When Receiving Update Messages 38
11. Design Decisions and Discussions 39
11.1. HTTP/2 Server-Push 39
11.2. Not Allowing Stream Restart 40
11.3. Data Update Choices 41
11.3.1. Full Replacement or Incremental Change 41
11.3.2. JSON Merge Patch or JSON Patch 41
11.4. Requirements on Future ALTO Services to Use this Design 42
12. Miscellaneous Considerations 42
12.1. Considerations for Updates to Filtered Cost Maps 42
12.2. Considerations for Incremental Updates to Ordinal Mode Costs 43
12.3. Considerations Related to SSE Line Lengths 43
13. Security Considerations 44
13.1. Denial-of-Service Attacks 44
13.2. Spoofed Control Requests 44
13.3. Privacy 44
14. IANA Considerations 44
15. Appendix A 47
16. References 47
Appendix A. Acknowledgments 48
Authors' Addresses 48

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

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

While primarily intended to provide updates to GET-mode network and cost maps, the mechanism defined in this document can also provide updates to POST-mode ALTO services, such as the endpoint property and endpoint cost services. We intend that the mechanism can also support new ALTO services to be defined by future extensions, but a future service need to satsify requirements specified in Section 11.4.

The rest of this document is organized as follows. Section 4 gives background on the basic techniques used in this design: (1) Server-Sent Events to allow server push; (2) JSON merge patch and JSON patch to allow incremental update. With the background, Section 5 gives a non-normative overview of the design. Section 6 defines individual messages in an update stream, and Section 7 defines the overall update stream service. Section 8 defines the update stream control service. Section 9 gives several examples. Section 10 describes how a client should handle incoming updates. Section 11 and Section 12 discusses the design decisions behind this update mechanism and other considerations. The next two sections review the security and IANA considerations.

2. Major Changes Since Version -01

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

- Incremental encoding: Added JSON patch as an alternative incremental delta encoding.

- Update concurrent requests of the same resouce: The client now assigns a unique client-id to each resource in an update stream. The server puts the client-id in each update event for that resource (before, the server used the server's resource-id). This allows a client to use one update stream to get updates to two different requests with the same server resource-id; before, that required two separate update streams.

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

3. Terms

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

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

Update Message: An Update Message is either a Data Update Message or a Control Update Message.

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

Full Replacement: A full replacement for a resource sends the content of the resource in its original ALTO encoding.

Incremental Change: An incremental change specifies only the difference between the new content and the previous version. An incremental change can be specified using either JSON merge patch or JSON patch in this document.

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

Update Stream Control Service: An Update Stream Control Service provides an HTTP URI so that the client of an Update Stream can use it to request addition or removal of resources receiving update messages. We refer to a server receiving the addition or removal requests as the ALTO Update Stream Control server, or just stream control server for short.

Stream Control: A shorthand for Updatea Strem Control Service.

4. Background

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

4.1. Server-Sent Events (SSEs)

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

Server-Sent Events enable a server to send new data to a client by "server-push". The client establishes an HTTP ([RFC7230], [RFC7231]) connection to the server, and keeps the connection open. The server continually sends messages. Each message has one or more lines, where a line is terminated by a carriage-return immediately followed by a new-line, a carriage-return not immediately followed by a new-line, or a new-line not immediately preceded by a carriage-return. A message is terminated by a blank line (two line terminators in a row).

Each line in a message is of the form "field-name: string value". Lines with a blank field-name (that is, lines which start with a colon) are ignored, as are lines which do not have a colon. The

protocol defines three field names: event, id, and data. If a message has more than one "data" line, the value of the data field is the concatenation of the values on those lines. There can be only one "event" or "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: good bye
```

Figure 1: A Sample SSE stream.

4.2. JSON Merge Patch

4.2.1. JSON Merge Patch Encoding

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

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

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

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

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

```
define MergePatch(Target, Patch) {
  if Patch is an Object {
    if Target is not an Object {
      Target = {} # Ignore the contents and
                  # set it to an empty Object
    }
    for each Name/Value pair in Patch {
      if Value is null {
        if Name exists in Target {
          remove the Name/Value pair from Target
        }
      } else {
        Target[Name] = MergePatch(Target[Name], Value)
      }
    }
    return Target
  } else {
    return Patch
  }
}
```

Note that null as the value of a name/value pair will delete the element with "name" in the original JSON value.

4.2.2. Merge Patch ALTO Messages

Both as examples of JSON merge patch and a demonstration of the feasibility to apply JSON merge patch to ALTO, we look at the application of JSON merge patch to two key ALTO messages.

4.2.2.1. Merge Patch Network Map Messages

Section 11.2.1.6 of [RFC7285] defines the format of a 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 merge patch update message, which (1) adds an ipv4 prefix "193.51.100.0/25" and an ipv6 prefix "2001:db8:8000::/33" to "PID1", (2) deletes "PID2", and (3) assigns a new "tag" to the network map:

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

Applying the 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",
                 "193.51.100.0/25" ],
      "ipv6" : [ "2001:db8:8000::/33" ]
    },
    "PID3" : {
      "ipv4" : [ "0.0.0.0/0" ],
      "ipv6" : [ "::/0" ]
    }
  }
}
```

4.2.2.2. Merge Patch Cost Map Messages

Section 11.2.3.6 of [RFC7285] defines the format of a cost map message. Assume is 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 merge patch message updates the example cost map so that PID1->PID2 is 9 instead of 5, PID3->PID1 is no longer available, and PID3->PID3 is now defined as 1:

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

Hence applying the 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"
    }
  },
  "cost-map" : {
    "PID1": { "PID1": 1,  "PID2": 9,  "PID3": 10 },
    "PID2": { "PID1": 5,  "PID2": 1,  "PID3": 15 },
    "PID3": {             "PID2": 15, "PID3": 1  }
  }
}
```

4.3. JSON Patch

4.3.1. JSON Patch Encoding

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

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

4.3.2. JSON Patch ALTO Messages

Both as examples of JSON patch and demonstration of difference between JSON patch and JSON merge patch, we look at the application of JSON patch to the same updates shown in Section 4.2.2.

4.3.2.1. JSON Patch Network Map Messages

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

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

4.3.2.2. JSON patch Cost Map Messages

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

```
[
  {
    "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
  }
]
```

5. Overview of Approach

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

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

An ALTO server may provide any number of update stream services, where each update stream service may provide updates for a given subset of the 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 server selects the resource set for each stream. 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.

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

The key objective of an update stream is to update the client on data value changes to ALTO resources. We refer to messages sending such updates as data update messages. Although an update stream may update one or more ALTO resources, each data update message updates only one resource, and is sent as a Server-Sent Event (SSE), as defined by [SSE]. An data update message is encoded either as a full replacement or 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 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.

A update stream can run for a long time, and hence there can be status changes at the server side during the life time of an update stream; for example, the server may encounter an error or need to shutdown for maintenance. To support robust, flexible protocol design, this design allows the server to send server state updates to the client as well, showing as control update messages from the server to the client.

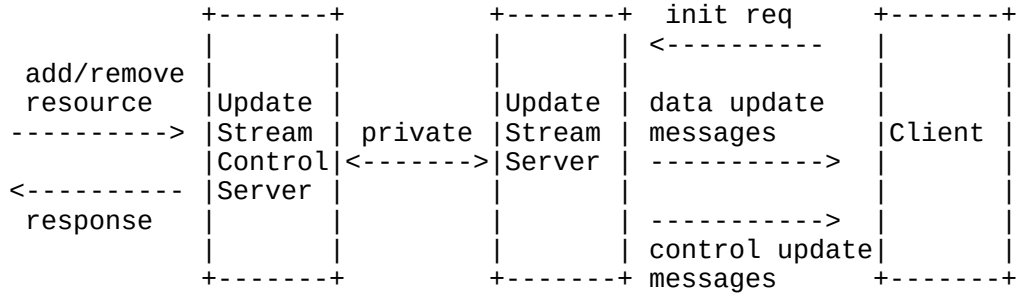

Figure 2: ALTO SSE Overview

In addition to state changes triggered from the server side, in a flexible design, a client may initiate changes as well, in particular, by adding or removing ALTO resources receiving updates. A client initiates such changes using the Update Stream Control Service. For an Update Stream Service supporting Update Stream Control, the server responds by sending an event (a control update message) with the URI for update stream control. The client can then use the URI to ask the 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.

6. Update Messages: Data Update and Control Update Messages

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

6.1. ALTO Update Message Format

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

JSON specifications use the type ClientId for a client-id.

The two sub-fields of the event field are encoded as comma-separated strings:

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

Note that media type names may not contain a comma (character code 0x2c).

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

6.2. ALTO Data Update Message

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

Figure 3 shows some examples of ALTO data update messages:

```
event: application/alto-networkmap+json,1
data: { ... full network map message ... }

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

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

Figure 3: Examples of ALTO data update messages.

6.3. ALTO Control Update Message

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

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

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

The "started" field is a list of client-ids of resources. It notifies the client that the server starts data update messages for each resource listed.

The "stopped" field is a list of client-ids of resources. It notifies the client that the server will no longer send data update messages for the listed resources. There can be multiple reasons for a server to stop sending data update messages for a resource, including a request from the client using stream control (Section 7.6.2) or an internal server event.

The "description" field is a non-normative text providing an explanation for the control event. When a server stops sending data update messages for a resource, it is RECOMMENDED that the server provides a short reason text, providing details.

7. Update Stream Service

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

7.1. Media Type

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

7.2. HTTP Method

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

7.3. Accept Input Parameters

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

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

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

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

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

An update stream request MUST have an "add" field specifying one or more resources. If it does not, the server MUST return an E_INVALID_FIELD_VALUE error response (see Section 8.5.2 of [RFC7285]), and MUST close the stream without sending any events.

The "resource-id" field is the resource-id of an ALTO resource, and MUST be in the update stream's "uses" list (see Section 7.5). If any resource-id is invalid, or is not associated with this update stream, the server MUST return an E_INVALID_FIELD_VALUE error response (see Section 8.5.2 of [RFC7285]), and MUST close the stream without sending any events.

If the resource-id is a GET-mode resource with a version tag (or "vtag"), as defined in Sections 6.3 and 10.3 of [RFC7285], and if the client has previously retrieved a version of that resource from the server, the 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 server MUST send a full replacement before sending any patch updates, as described in Section 7.6.2. If that version is still current, the ALTO server MAY omit the initial full replacement.

If the "incremental-changes" field for a resource-id is "true", the server MAY send incremental changes for this resource-id (assuming the server supports incremental changes for that resource; see Section 7.4). If the "incremental-changes" field is "false", the ALTO server MUST NOT send incremental changes for that resource.

The default for "incremental-changes" is "true", so to suppress incremental changes, the client MUST explicitly set "incremental-changes" to "false". Note that the client cannot suppress full replacements.

When the client sets "incremental-changes" to "false", whenever a change occurs, the server MUST send a full replacement instead of an incremental change. The 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 a client which does.

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 resource expects. If the "input" field is missing or invalid, the ALTO server MUST return the same error response that that resource would return for missing or invalid input (see [RFC7285]). In this case, the server MUST close the update stream without sending any events. If the inputs for several POST-mode resources are missing or invalid, the server MUST pick one error response and return it.

The "remove" field is used in update stream control requests (see Section 8), and is not allowed in the initial update stream request. If the "remove" field exists, the server MUST return an E_INVALID_FIELD_VALUE error response (see Section 8.5.2 of [RFC7285]), and MUST close the stream without sending any events.

7.4. Capabilities

The capabilities are defined by 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 media-type of the incremental change. Normally this will be "application/merge-patch+json", "application/json-patch+json", or "application/merge-patch+json,application/json-patch+json", because,

as described in Section 6, they are the only incremental change types defined by this document. However future extensions may define other types of incremental changes.

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

The "support-stream-control" field specifies whether the given update stream supports stream control.

7.5. Uses

The "uses" attribute MUST be an array with the resource-ids of every resource for which this stream can provide updates. Each resource specified in the "uses" MUST support full replacement: server can always send full replacement, and the 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 select a set that is closed under the resource dependency relationship. That is, if an update stream's "uses" set includes resource R1, and resource R1 depends on ("uses") resource R0, then the update stream's "uses" set SHOULD include R0 as well as R1. For example, an update stream for a cost map SHOULD also provide updates for the network map upon which that cost map depends.

7.6. Response

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

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

There are additional requirements on the server's response, as described below.

7.6.1. 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 ALTO server SHOULD send a comment line (or an event) every 15 seconds to prevent clients and proxy servers from dropping the HTTP connection.

7.6.2. Event Sequence Requirements

- The first event MUST be a control update message with the URI of the stream control service (Section 8) for this update stream (Section 6.3).

- As soon as possible after the client initiates the connection, the ALTO server MUST send a full replacement for each resource-id requested by the client. The only exception is for a GET-mode resource with a version tag. In this case the server MAY omit the initial full replacement for that resource, if the "tag" field the client provided for that resource-id matches the tag of the server’s current version.

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

- If this stream provides updates for resource-ids R0 and R1, and if R1 depends on R0, then the ALTO server SHOULD send an update for R1 as soon as possible after sending the update for R0. For example, when a network map changes, the ALTO server SHOULD send data update messages for the dependent cost maps as soon as possible after the data update messages for the network map.

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

7.6.3. Cross-Stream Consistency Requirements

If several clients create multiple update streams for updates to the same resource, the server MUST send the same updates to all of them. However, the 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 clients create update streams for the same cost map, and suppose the ALTO 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 server MAY send a single patch event (with all three cost points) to one client, while sending three separate patch events (with one cost point per event) to the other client.

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

8. Update Stream Control Service

An update stream control service allows a 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 a client to gracefully shutdown an update stream.

When a server creates a new update stream, and if the server supports stream control for the update stream, the server creates an update stream control service for that update stream. A client uses the update stream control service to remove resources from the update stream instance, or to request updates for additional resources. A client cannot obtain the update stream control service through the IRD. Instead, the first event that the server sends to the client has the URI for the associated update stream control service (see Section 6.3.

Each stream control request is an individual HTTP request. If the client and the server support multiple HTTP requests from the client to the server ([RFC7230]), the client MAY send multiple stream control requests to the server using the same HTTP connection.

8.1. URI

The URI for a stream control service, by itself, MUST uniquely specify the update stream instance which it controls. The 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, a server MUST NOT re-use a control service URI once the associated update stream has been closed.

The client MUST evaluate a non-absolute control URI (for example, a URI without a host, or with a relative path) in the context of the

URI used to create the update stream. The controller's host MAY be different from the update stream's host.

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

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

8.2. Media Type

An ALTO stream control response does not have a specific media type. If a request is successful, the server returns an HTTP "204 No Content" response. If a request is unsuccessful, the server returns an ALTO error response (Section 8.5.2 of [RFC7285])

8.3. HTTP Method

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

8.4. Accept Input Parameters

A stream control service accepts the same input media type and input parameters as the update stream service (Section 7.3). The only difference is that a stream control service also accepts the "remove" field.

If specified, the "remove" field is an array of client-ids the client previously added to this update stream. An empty "remove" array is equivalent to a list of all currently active resources; the server responds by removing all resources and closing the stream.

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

If a request has any error, the server MUST NOT add or remove any resources from the associated update stream. In particular,

- Each "add" request must satisfy the requirements in Section 7.3. If not, the server MUST return the error response defined in Section 7.3.

- As described in Section 7.6.2, for each "add" request, the ALTO server MUST send a full replacement for that resource before sending any incremental changes. The only exception is for a GET-mode resource with a version tag. In this case the server MAY omit the full replacement for that resource if the "tag" field the client provided matches the server's current version.

- The server MUST return an E_INVALID_FIELD_VALUE error if a client-id in the "remove" field was not added in a prior request. Thus it is illegal to "add" and "remove" the same client-id in the same request. However, it is legal to remove a client-id twice.

- The server MUST return an E_INVALID_FIELD_VALUE error if a client-id in the "add" field has been used before in this stream.

- The server MUST return an E_INVALID_FIELD_VALUE error if the request has a non-empty "add" field and a "remove" field with an empty list of client-ids (to replace all active resources with a new set, the client MUST explicitly enumerate the client-ids to be removed).

- If the associated update stream has been closed, the server MUST return either an ALTO E_INVALID_FIELD_VALUE error, or else an HTTP error, such as "404 Not Found".

8.5. Capabilities & Uses

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

8.6. Response

If a request is successfully accepted by the server, the server returns an HTTP "204 No Content" response with no data. If there are any errors, the server MUST return the appropriate ALTO error code, and MUST NOT add or remove any resources from the update stream.

It should be noted that an HTTP "204 No Content" response does not mean that the request is successfully processed by the server. Transitions of states (i.e., started, stopped) of resources in the update stream are notified by the server using control update messages.

The server MUST process the "add" field before the "remove" field. If the request removes all active resources without adding any

additional resources, the server MUST close the update stream. Thus an update stream cannot have zero resources.

9. Examples

9.1. Example: IRD Announcing Update Stream Services

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

Note that in the "update-my-costs" update stream shown in the example IRD, the ALTO 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 12.1, the update stream provides updates for the second, but not the first.

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

```
      "accepts": "application/alto-costmapfilter+json",
      "uses": ["my-networkmap"],
      "capabilities": {
        "cost-type-names": ["num-routingcost", "num-hopcount"],
        "cost-constraints": true
      }
    },
    "my-simple-filtered-cost-map": {
      "uri": "http://alto.example.com/costmap/filtered/simple",
      "media-type": "application/alto-costmap+json",
      "accepts": "application/alto-costmapfilter+json",
      "uses": ["my-networkmap"],
      "capabilities": {
        "cost-type-names": ["num-routingcost", "num-hopcount"],
        "cost-constraints": false
      }
    },
    "my-props": {
      "uri": "http://alto.example.com/properties",
      "media-type": "application/alto-endpointprops+json",
      "accepts": "application/alto-endpointpropparams+json",
      "capabilities": {
        "prop-types": ["priv:ietf-bandwidth"]
      }
    },
    "update-my-costs": {
      "uri": "http://alto.example.com/updates/costs",
      "media-type": "text/event-stream",
      "accepts": "application/alto-updatestreamparams+json",
      "uses": [
         "my-network-map",
         "my-routingcost-map",
         "my-hopcount-map",
         "my-simple-filtered-cost-map"
      ],
      "capabilities": {
        "incremental-change-media-types": {
          "my-network-map": "application/json-patch+json",
          "my-routingcost-map": "application/merge-patch+json",
          "my-hopcount-map": "application/merge-patch+json"
        },
        "support-stream-control": true
      }
    },
    "update-my-props": {
      "uri": "http://alto.example.com/updates/properties",
      "media-type": "text/event-stream",
      "uses": [ "my-props" ],
```

```
      "accepts": "application/alto-updatestreamparams+json",
      "capabilities": {
        "incremental-change-media-types": {
          "my-props": "application/merge-patch+json"
        },
        "support-stream-control": true
      }
    }
```

9.2. Example: Simple Network and Cost Map Updates

Given the update streams announced in the preceding example IRD, below we show an example of a client's request and the server's immediate response, using the update stream resource "update-my-costs". In the example, the client requests updates for the network map and "routingcost" cost map, but not for the "hopcount" cost map. The client uses the server's resource-ids as the client-ids. Because the client does not provide a "tag" for the network map, the server must send a full update for the network map as well as for the cost map. The client does not set "incremental-changes" to "false", so it defaults to "true". Thus server will send patch updates for the cost map and the network map.

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

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

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

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

  event: application/alto-networkmap+json,my-network-map
```

```
data: {
data:   "meta" : {
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 ALTO 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: a ipv4 prefix "193.51.100.0/25" is added to PID1. It triggers changes to the cost map. The ALTO server chooses to send an incremental change for the network map, and send a full replacement instead of an incremental change for the cost map:

```
event: application/json-patch+json,my-network-map
data: {
data:   {
data:     "op": "replace",
data:     "path": "/meta/vtag/tag",
data:     "value" :"a10ce8b059740b0b2e3f8eb1d4785acd42231bfe"
data:   },
data:   {
data:     "op": "add",
data:     "path": "/network-map/PID1/ipv4/2",
data:     "value": "193.51.100.0/25"
data:   }
data: }

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

9.3. Example: Advanced Network and Cost Map Updates

This example is similar to the previous one, except that the 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 server is not required to send the full network map data update message at the beginning of the stream. In this example, the client uses the client-ids "net", "routing" and "hops" for those resources. The ALTO server sends the stream control URI and the full cost maps, followed by updates for the network map and cost maps as they become available:

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

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

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

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

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

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

   (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 client wishes to stop receiving updates for the "hopcount" cost map, the 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: ###

{
  "remove": [ "hops" ]
}


HTTP/1.1 204 No Content
Content-Length: 0

    (stream closed without sending data content)
```

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

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

If the client no longer needs any updates, and wishes to shut the update stream down gracefully, the client can send a "remove" request with an empty array:

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

{
  "remove": [ ]
}


HTTP/1.1 204 No Content
Content-Length: 0

    (stream closed without sending data content)
```

The ALTO server sends a final control update message on the original request stream to inform the 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)
```

9.4. Example: Endpoint Property Updates

As another example, here is how a client can request updates for the property "priv:ietf-bandwidth" for one set of endpoints, and "priv:ietf-load" for another. The ALTO server immediately sends full replacements with the property values for all endpoints. After that, the 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: ###

{ "add": {
    "props-1": {
      "resource-id": "my-props",
      "input": {
        "properties" : [ "priv:ietf-bandwidth" ],
        "endpoints" : [
          "ipv4:198.51.100.1",
          "ipv4:198.51.100.2",
          "ipv4:198.51.100.3"
        ]
      }
    },
    "props-2": {
      "resource-id": "my-props",
      "input": {
        "properties" : [ "priv:ietf-load" ],
        "endpoints" : [
          "ipv6:2001:db8:100::1",
          "ipv6:2001:db8:100::2",
          "ipv6:2001:db8:100::3",
        ]
      }
    },
  }
}
```

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

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

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

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

   (pause)

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

   (pause)

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

If the client needs the "bandwidth" property for additional endpoints, the client can send an "add" request on the stream control URI:

```
POST /updates/streams/1414213562373" HTTP/1.1
Host: alto.example.com
Accept: text/plain,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: ###

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


HTTP/1.1 204 No Content
Content-Length: 0

    (stream closed without sending data content)
```

The ALTO 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: }
```

10. Client Actions When Receiving Update Messages

In general, when a client receives a full replacement for a resource, the client should replace the current version with the new version. When a client receives an incremental change for a resource, the 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 a client can do that. We will illustrate these techniques by referring to network and cost map messages, although these techniques apply to any dependent resources.

Note that when a network map changes, the ALTO server MUST send the network map update message before sending the updates for the dependent cost maps (see Section 7.6.2).

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

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

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

Although not as efficient as possible, this recovery method is simple and reliable.

11. Design Decisions and Discussions

11.1. HTTP/2 Server-Push

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

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

the client, "Add this resource to your cache, because a resource you have requested needs it."

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

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

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

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

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

11.2. Not Allowing Stream Restart

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

However, that mechanism adds additional complexity. The server must save SSE messages in a buffer, in case clients reconnect. But that mechanism will never be perfect: if the client waits too long to reconnect, or if the client sends an invalid id, then the server will have to resend the complete maps anyway.

Furthermore, this is unlikely to be a problem in practice. Clients who want continuous updates for large resources, such as full Network and cost maps, are likely to be things like P2P trackers. These

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 ALTO server will have to retransmit only a small amount of redundant data.

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

The Update Stream service does allow the client to specify the tag of the last received version of any tagged resource, and if that is still current, the server need not retransmit the full resource. Hence 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.

11.3. Data Update Choices

11.3.1. Full Replacement or Incremental Change

At this point we do not have sufficient experience with ALTO deployments to know how frequently the resources will change, or how extensive those changes will be. For stable resources with minor changes, the server may choose to send incremental changes; for resources that frequently change, the server may choose to send a full replacement after a while. Whether to send full replacement or incremental change depends on the server.

11.3.2. JSON Merge Patch or JSON Patch

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

The case is not as clear for incremental changes to network maps. First consider small changes such as moving a prefix from one PID to another. JSON patch could encode that as a simple insertion and deletion, while 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 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, 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 server may decide on its own whether to use JSON merge patch or JSON patch according to the changes in network maps.

Other JSON-based incremental change formats may be introduced in the future.

11.4. Requirements on Future ALTO Services to Use this Design

Although this design is quite flexible, it has underlying requirements. In particular, the key requirements are that (1) each update message is for a single resource; (2) incremental changes can be applied only to a resource that is a single JSON object, as both 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 extention to this design is made.

If an update stream provides updates to a filtered cost map that allows constraint tests, the requirements for such services are stated in Section 12.1.

12. Miscellaneous Considerations

12.1. Considerations for Updates to Filtered Cost Maps

If an update stream provides updates to a Filtered cost map which allows constraint tests, then a client MAY request updates to a Filtered cost map request with a constraint test. In this case, when a cost changes, the server MUST send an update if the new value satisfies the test. If the new value does not, whether the server sends an update depends on whether the previous value satisfied the

test. If it did not, the server SHOULD NOT send an update to the client. But if the previous value did, then the server MUST send an update with a "null" value, to inform the client that this cost no longer satisfies the criteria.

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

12.2. Considerations for Incremental 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 a server to offer incremental updates for ordinal mode cost maps, server implementors should be aware that incremental updates for ordinal costs are more complicated than for numerical costs, and clients should be aware that small changes may result in large updates.

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

12.3. Considerations Related to SSE Line Lengths

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

First, SSE clients might not be able to handle a multi-megabyte data "line". Hence it is RECOMMENDED that an ALTO server limit data lines to at most 2,000 characters.

Second, some SSE client packages read all the data for an event into memory, and then present it to the client as a single character array. However, a client computer may not have enough memory to hold the entire JSON text for a large cost map. Hence an ALTO client SHOULD consider using an SSE library which presents the event data in manageable chunks, so the client can parse the cost map incrementally and store the underlying data in a more compact format.

13. Security Considerations

13.1. Denial-of-Service Attacks

Allowing persistent update stream connections enables a new class of Denial-of-Service attacks. A client might create an unreasonable number of update stream connections, or add an unreasonable number of client-ids to one update stream. To avoid those attacks, an ALTO server MAY choose to limit the number of active streams, and reject new requests when that threshold is reached. A server MAY also choose to limit the number of active client-ids on any given stream, or limit the total number of client-ids used over the lifetime of a stream, and reject any stream control request which would exceed those limits. In these cases, the server SHOULD return the HTTP status "503 Service Unavailable".

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

Alternatively an ALTO 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.

13.2. Spoofed Control Requests

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

13.3. Privacy

This extension does not introduce any privacy issues not already present in the ALTO protocol.

14. IANA Considerations

This document defines two new media-types, "application/alto-updatestreamparams+json", as described in Section 7.3, and "application/alto-updatestreamcontrol+json", as described in Section 6.3. All other media-types used in this document have

already been registered, either for ALTO, JSON merge patch, or JSON patch.

Type name: application

Subtype name: alto-updatestreamparams+json

Required parameters: n/a

Optional parameters: n/a

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

Security considerations: Security considerations relating to the generation and consumption of ALTO Protocol messages are discussed in Section 13 of this document and Section 15 of [RFC7285].

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

Published specification: Section 7.3 of this document.

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

Additional information:

Magic number(s): n/a

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

Macintosh file type code(s): n/a

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

Intended usage: COMMON

Restrictions on usage: n/a

Author: See Authors' Addresses section.

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

Type name: application

Subtype name: alto-updatestreamcontrol+json

Required parameters: n/a

Optional parameters: n/a

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

Security considerations: Security considerations relating to the generation and consumption of ALTO Protocol messages are discussed in Section 13 of this document and Section 15 of [RFC7285].

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

Published specification: Section 6.3 of this document.

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

Additional information:

Magic number(s): n/a

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

Macintosh file type code(s): n/a

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

Intended usage: COMMON

Restrictions on usage: n/a

Author: See Authors' Addresses section.

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

15. Appendix A

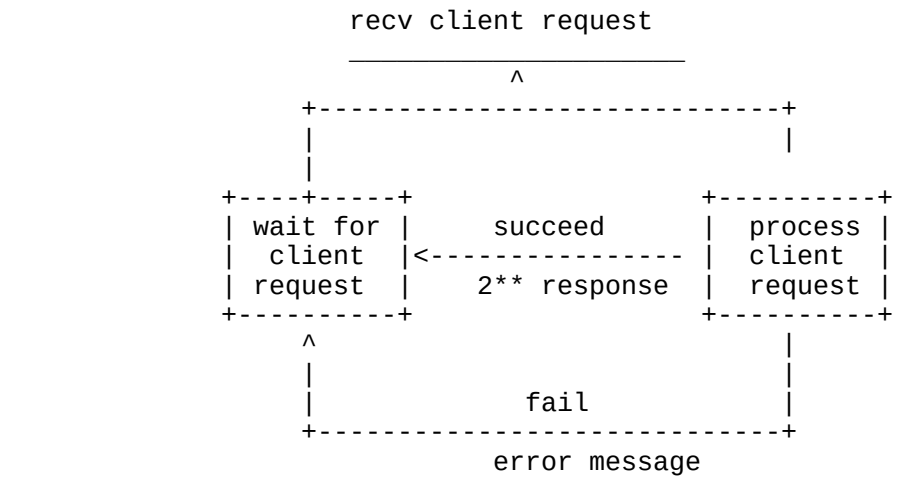

Figure 4: Finite State Machine of Control Server

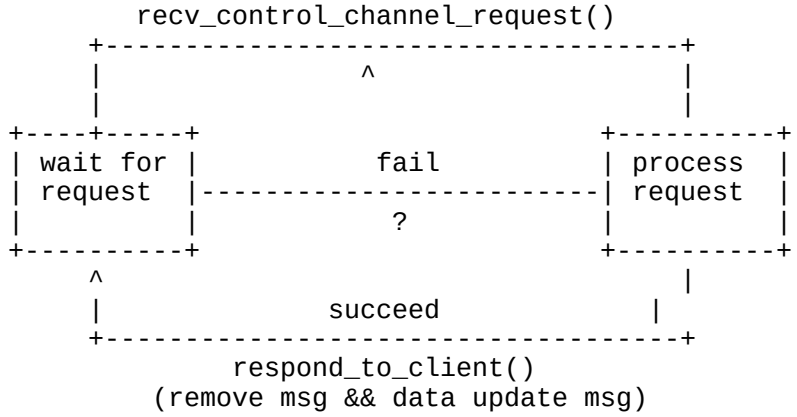

Figure 5: Finite State Machine of SSE Channel

16. References

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

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

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

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[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.

[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.

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

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

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

Appendix A. Acknowledgments

Thank you to Xiao Shi (Yale University) for his contributions to an earlier version of this document.

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
Tongji/Yale University
51 Prospect St
New Haven CT
USA

Email: yang.r.yang@gmail.com

Shiwei Dawn Chen
Tongji University
4800 Caoan Road
Shanghai 201804
China

Email: dawn_chen_f@hotmail.com

ALTO WG
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

G. Bernstein
Grotto Networking
S. Chen
Tongji University
K. Gao
Tsinghua University
Y. Lee
Huawei
W. Roome
M. Scharf
Nokia
Y. Yang
Yale University
J. Zhang
Tongji University
July 2, 2018

ALTO Extension: Path Vector Cost Type
draft-ietf-alto-path-vector-04

Abstract

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

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute

working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 3
2. Terminology 5
3. Use Case: Capacity Region for Multi-Flow Scheduling 5
4. Overview of Path Vector Extensions 7
4.1. New Cost Type to Encode Path Vectors 7
4.2. New ALTO Entity Domain to Provide ANE Properties 8
4.3. Extended Cost Map/Endpoint Cost Service for Compound Resources 8
5. Cost Type 8
5.1. Cost Mode: array 9
5.2. Cost Metric: ane-path 9
5.3. Path Vector Cost Type Semantics 9
6. ANE Domain 10
6.1. Domain Name 10
6.2. Domain-Specific Entity Addresses 10
6.3. Hierarchy and Inheritance 10
7. Protocol Extensions for Path Vector Compound Query 10
7.1. Filtered Cost Map Extensions 11
7.1.1. Capabilities 11
7.1.2. Accept Input Parameters 12
7.1.3. Response 12

7.2. Endpoint Cost Service Extensions 12
7.2.1. Capabilities 13
7.2.2. Accept Input Parameters 13
7.2.3. Response 13
8. Examples 13
8.1. Workflow 13
8.2. Information Resource Directory Example 14
8.3. Example # 1 16
8.4. Example # 2 18
8.5. Example #3 20
9. Compatibility 22
9.1. Compatibility with Legacy ALTO Clients/Servers 22
9.2. Compatibility with Multi-Cost Extension 23
9.3. Compatibility with Incremental Update 23
10. General Discussions 23
10.1. Provide Calendar for Property Map 23
10.2. Constraint Tests for General Cost Types 24
10.3. General Compound Resources Query 24
11. Security Considerations 24
11.1. Privacy Concerns 24
11.2. Resource Consumption on ALTO Servers 25
12. IANA Considerations 25
12.1. ALTO Cost Mode Registry 25
12.2. ALTO Cost Metric Registry 25
12.3. ALTO Domain Registry 25
12.4. ALTO Network Element Property Type Registry 26
13. Acknowledgments 26
14. References 26
14.1. Normative References 26
14.2. Informative References 26
Authors' Addresses 27

1. Introduction

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

Although the "single-node" network view abstraction works well in many settings, it lacks the ability to support emerging use cases, such as applications requiring large bandwidth or latency sensitivity [I-D.bernstein-alto-topo], and inter-datacenter data transfers [I-D.lee-alto-app-net-info-exchange]. For these use cases, applications require a more powerful network view abstraction beyond the "single-node" abstraction to support application capabilities, in particular, the ability multi-flow scheduling.

To support capabilities like multi-flow scheduling, this document uses a "path vector" abstraction to represent more detailed network graph information like capacity regions. The path vector abstraction uses path vectors with abstract network elements to provide network graph view for applications. A path vector consists of a sequence of Abstract Network Elements (ANEs) that end-to-end traffic goes through. ANEs can be links, switches, middleboxes, their aggregations, etc.; they have properties like "bandwidth", "delay", etc. These information may help the application avoid network congestion and achieve better application performance.

Providing path vector abstraction using ALTO introduces the following additional requirements (ARs):

AR-1: The ALTO protocol SHOULD include the support for encoding array-like cost values rather than scalar cost values in cost maps or endpoint cost maps.

The ALTO server providing path vector abstraction SHOULD convey sequences of ANEs between sources and destinations the ALTO client requests. Theses information cannot be encoded by the scalar types (numerical or ordinal) which the base ALTO protocol supports. A new cost type is required to encode path vectors as costs.

AR-2: The ALTO protocol SHOULD include the support for encoding properties of ANEs.

Only the sequences of ANEs are not enough for most use cases mentioned previously. The properties of ANEs like "bandwidth" and "delay" are required by applications to build the capacity region or realize the latency sensitivity.

AR-3: The ALTO server SHOULD allow the ALTO client to query path vectors and the properties of abstract network elements consistently.

Path vectors and the properties of abstract network elements are correlated information, but can be separated into different ALTO information resources. A mechanism to query both of them consistently is necessary.

This document proposes the path vector extension which satisfies these additional requirements to the ALTO protocol. Specifically, the ALTO protocol encodes the array of ANEs over an end-to-end path using a new cost type, and conveys the properties of ANEs using unified property map [I-D.ietf-alto-unified-props-new]. We also provide an optional solution to query separated path vectors and

properties of ANEs in a consistent way. But querying general separated resources consistently is not the scope in this document.

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

2. Terminology

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

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

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

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

Assume that an application has control over a set of flows, which may go through shared links or switches and share a bottleneck. The application hopes to schedule the traffic among multiple flows to get better performance. The capacity region information for those flows will benefit the scheduling. However, existing cost maps 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.

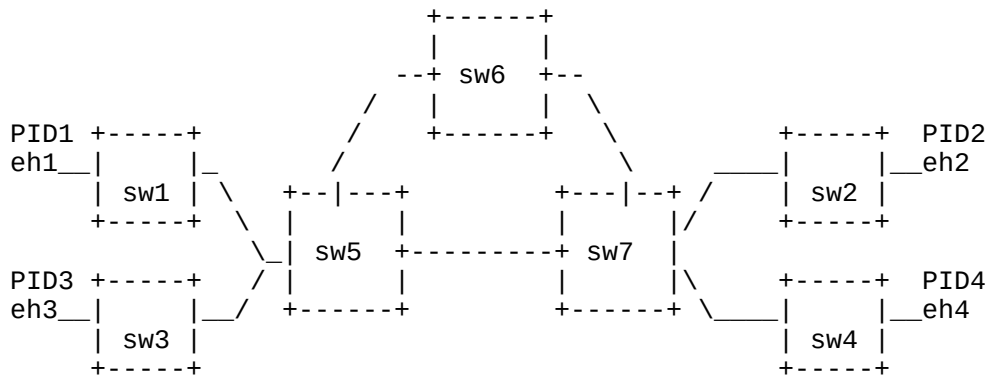

Figure 1: Raw Network Topology.

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

Figure 2: Base Single-Node Topology Abstraction.

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

- Case 1: If eh1 -> eh2 uses the path eh1 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh2 and 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:

- The network needs to expose more detailed routing information to show the shared bottlenecks.

- The network needs to provide the necessary abstraction to hide the real topology information while providing enough information to applications.

The path vector extension defined in this document propose a solution to provide these details.

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

4. Overview of Path Vector Extensions

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

The path vector extension is composed of three building blocks: (1) a new cost type to encode path vectors; (2) a new ALTO entity domain for unified property extension [I-D.ietf-alto-unified-props-new] to encode properties of ANEs; and (3) an extension to the cost map and endpoint cost resource to provide path vectors and properties of ANEs in a single response.

4.1. New Cost Type to Encode Path Vectors

Existing cost types defined in [RFC7285] allow only scalar cost values. However, the "path vector" abstraction requires to convey vector format information. To achieve this requirement, this document defines a new cost mode to enable the cost value to carry an array of elements, and a new cost metric to take names of ANEs as elements in the array. We call such an array of ANEs a path vector. In this way, the cost map and endpoint cost service can convey the path vector to represent the routing information. Detailed information and specifications are given in Section 5.1 and Section 5.2.

4.2. New ALTO Entity Domain to Provide ANE Properties

The path vector can only represent the route between the source and the destination. Although the application can find the shared ANEs among different paths, it is not enough for most use cases, which requires the bandwidth or delay information of the ANEs. So this document adopts the property map defined in [I-D.ietf-alto-unified-props-new] to provide the general properties of ANEs. The document registers a new entity domain called "ane" to represent the ANE. The address of the ANE entity is just the ANE name used by the path vector. By requesting the property map of entities in the "ane" domain, the client can retrieve the properties of ANEs in path vectors.

4.3. Extended Cost Map/Endpoint Cost Service for Compound Resources

Providing the path vector information and the ANE properties by separated resources have several known benefits: (1) can be better compatible with the base ALTO protocol; (2) can make different property map resources reuse the same cost map or endpoint cost resource. However, it conducts two issues:

- Efficiency: The separated resources will require the ALTO client to invoke multiple requests/responses to collect all needed information. It increases the communication overhead.

- Consistency: The path vectors and properties of ANEs are correlated. So querying them one by one may conduct consistency issue. Once the path vector changes during the client requests the ANE properties, the ANE properties may be inconsistent with the previous path vector.

To solve these issues, this document introduces an extension to cost map and endpoint cost service, which allows the ALTO server to attach a property map in the data entry of a cost map or an endpoint cost service response.

These issues may exist in all general cases for querying separated ALTO information resources. But solving this general problem is not in the scope of this document.

5. Cost Type

This document extends the cost types defined in Section 6.1 of [RFC7285] by introducing a new cost mode "array" and a new cost metric "ane-path". In the rest content, this document uses "path-vector" to indicate the combination cost type of the cost mode "array" and the cost metric "ane-path".

5.1. Cost Mode: array

This document extends the CostMode defined in Section 10.5 of [RFC7285] with a new cost mode: "array". This cost mode indicates that every cost value in a cost map represents an array rather than a simple value. The values are arrays of JSONValue. The specific type of each element in the array depends on the cost metric.

5.2. Cost Metric: ane-path

This document specifies a new cost metric: "ane-path". This cost metric indicates that the cost value is a list of ANEs which the path from a source to a destination goes across. The values are arrays of ANE names which are defined in Section 6.2.

The cost metric "ane-path" SHOULD NOT be used when the cost mode is not "array" unless it is explicitly specified by a future extension. If an ALTO client send queries with the cost metric "ane-path" and a non "array" cost mode, the ALTO server SHOULD return an error with the error code "E_INVALID_FIELD_VALUE"; If an ALTO server declares the support of a cost type with the cost metric "ane-path" and a non "array" cost mode, the ALTO client SHOULD assume such a cost type is invalid and ignore it.

5.3. Path Vector Cost Type Semantics

The new cost type follows the convention of the cost types in the base ALTO protocol. Table 1 lists some of the current defined cost types and their semantics.

Cost Mode	Cost Metric	Semantics
numerical	routingcost	a number representing the routing cost
numerical	hopcount	a number representing the hop count
ordinal	routingcost	a ranking representing the routing cost
ordinal	hopcount	a ranking representing the hop count
array	ane-path	a list representing the ane path

Table 1: Cost Types and Their Semantics

The "routingcost" and "hopcount" can encoded in "numerical" or "ordinal", however, the cost metric "ane-path" can only be applied to the cost mode "array" defined in this document to convey path vector information. The cost metric "ane-path" can not be used in

"numerical" or "ordinal" unless it is defined in future extensions. If the ALTO server declares that it support cost type with cost metric being "ane-path" and cost mode not being "array", the ALTO client SHOULD ignore them.

6. ANE Domain

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

6.1. Domain Name

ane

6.2. Domain-Specific Entity Addresses

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

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

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

6.3. Hierarchy and Inheritance

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

7. Protocol Extensions for Path Vector Compound Query

To make the ALTO client query the path vectors and properties of ANEs efficiently and consistently, this document extends the Filtered Cost Map and Endpoint Cost Service.

7.1. Filtered Cost Map Extensions

This document extends Filtered Cost Map, as defined in Section 11.3.2 of [RFC7285], by adding new input parameters and capabilities, and by augmenting the property map into the data entry of the response.

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) remain the same.

7.1.1. Capabilities

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

- dependent-property-map
- allow-compound-response

The capability "dependent-property-map" indicates which property map this resource depends on, and the capability "allow-compound-response" indicates whether the ALTO server supports the resource to compound the property map with its own response data. With these two additional members, the FilteredCostMapCapabilities object in Section 11.3.2.4 of [RFC7285] is extended as follows:

```
object {
  [ResourceID dependent-property-map;]
  [JSONBool   allow-compound-response;]
} PVFCMCapabilities : FilteredCostMapCapabilities;
```

dependent-property-map: This field MUST be specified when the "cost-type-names" includes a cost type name indicating a "ane-path" metric. Its value MUST be a resource id indicating a property map including "ane" domain. If not, the ALTO client SHOULD consider this resource is invalid.

allow-compound-response: If present, the true value means the ALTO client can request the resource to augment its dependent property map into the response automatically; the false value means the ALTO client cannot request the compound response. If omitted, the default value is false;

To be noticed that the capability "cost-constraints" is unexpected for the "array" cost mode. The syntax and semantics of constraint tests on the "array" cost mode depends on the implementation and can be defined in the future documents. But it is not in the scope of this document.

7.1.2. Accept Input Parameters

The ReqFilteredCostMap object in Section 11.3.2.3 of [RFC7285] is extended as follows:

```
object {
  [PropertyName compound-properties<1..*>;]
} ReqPVFilteredCostMap : ReqFilteredCostMap;
```

compound-properties: If the capability "allow-compound-response" is false, the ALTO client MUST NOT specify this field, and the ALTO server MUST reject the request and return "E_INVALID_FILED_VALUE" error when it receives a request including this field. If this field is specified and accepted, the ALTO server MUST augment the dependent property map with the properties in this field into the response automatically.

7.1.3. Response

If the ALTO client specifies the "cost-type" input parameter with "ane-path" metric, the "dependent-vtags" field in the "meta" field of the response MUST include the version tag of its dependent property map following its dependent network map.

If the ALTO client specifies the "compound-properties" input parameter which is accepted by the ALTO server, the response MUST include a "property-map" field following the "cost-map" field, and its value MUST be a PropertyMapData object. This PropertyMapData object MUST be equivalent to the result when query the dependent property map resource using the following request: the "entities" field includes all the ANE names appearing in the cost values of the "cost-map" field, the "properties" field has the same value as the "compound-properties" field does. The properties shown in the "compound-properties" input parameter but are not supported by the dependent property map SHOULD be omitted from the response.

7.2. Endpoint Cost Service Extensions

This document extends the Endpoint Cost Service, as defined in Section 11.5.1 of [RFC7285], by adding new input parameters and capabilities and by augmenting the property map into the data entry of the response.

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.

7.2.1. Capabilities

The extensions to the Endpoint Cost Service capabilities are identical to the extensions to the Filtered Cost Map (see Section 7.1.1).

7.2.2. Accept Input Parameters

The ReqEndpointCostMap object in Section 11.5.1.3 of [RFC7285] is extended as follows:

```
object {
  [PropertyName compound-properties<1..*>;]
} ReqPVEndpointCostMap : ReqEndpointCostMap;
```

The "compound-properties" has the same interpretation as defined in Section 7.1.2

7.2.3. Response

If the ALTO client specifies the "cost-type" input parameter with "ane-path" metric, the response MUST include the "meta" field with the "dependent-vtags" in it, and the "dependent-vtags" field MUST include the version tag of its dependent property map.

If the ALTO client specifies the "compound-properties" input parameter which is accepted by the ALTO server, the response MUST include a "property-map" field following the "endpoint-cost-map" field, and its value MUST be a PropertyMapData object. This PropertyMapData object MUST be equivalent to the result when query the dependent property map resource using the following request: the "entities" field includes all the ANE names appearing in the cost values of the "endpoint-cost-map" field, the "properties" field has the same value as the "compound-properties" field does. The properties shown in the "compound-properties" input parameter but are not supported by the dependent property map SHOULD be omitted from the response.

8. Examples

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

8.1. Workflow

This section gives a typical workflow of how an ALTO client query path vectors using the extension.

1. Send a GET request for the whole Information Resource Directory.

2. Look for the resource of the (Filtered) Cost Map/Endpoint Cost Service which supports the "ane-path" cost metric and get the resource ID of the dependent property map.

3. Check whether the capabilities of the property map includes the desired "prop-types".

4. Check whether the (Filtered) Cost Map/Endpoint Cost Service allows the compound response.

    1. If allowed, the ALTO client can send a request including the desired ANE properties to the ALTO server and receive a compound response with the cost map/endpoint cost map and the property map.

    2. If not allowed, the ALTO client sends a query for the cost map/endpoint cost map first. After receiving the response, the ALTO client interprets all the ANE names appearing in the response and sends another query for the property map on those ANE names.

8.2. Information Resource Directory Example

Here is an example of an Information Resource Directory. In this example, filtered cost map "cost-map-pv" doesn't support the multi-cost extension but support the path-vector extension, "endpoint-multicost-map" supports both multi-cost extension and path-vector extension. Filtered Property Map "propmap-availbw-delay" supports properties "availbw" and "delay".

```
{
  "meta": {
    "cost-types": {
      "path-vector": {
        "cost-mode": "array",
        "cost-metric": "ane-path"
      },
      "num-routingcost": {
        "cost-mode": "numerical",
        "cost-metric": "routingcost"
      },
      "num-hopcount": {
        "cost-mode": "numerical",
        "cost-metric": "hopcount"
      }
    }
```

```
      },
      "resources": {
        "my-default-networkmap": {
          "uri" : "http://alto.example.com/networkmap",
          "media-type" : "application/alto-networkmap+json"
        },
        "my-default-cost-map": {
          "uri": "http://alto.example.com/costmap/pv",
          "media-type": "application/alto-costmap+json",
          "accepts": "application/alto-costmapfilter+json",
          "capabilities": {
            "cost-type-names": [ "num-hopcount",
                                 "num-routingcost" ]
          },
          "uses": [ "my-default-networkmap" ]
        },
        "cost-map-pv": {
          "uri": "http://alto.example.com/costmap/pv",
          "media-type": "application/alto-costmap+json",
          "accepts": "application/alto-costmapfilter+json",
          "capabilities": {
            "cost-type-names": [ "path-vector" ],
            "dependent-property-map": "propmap-availbw-delay"
          },
          "uses": [ "my-default-networkmap" ]
        },
        "endpoint-cost-pv": {
          "uri": "http://alto.exmaple.com/endpointcost/pv",
          "media-type": "application/alto-endpointcost+json",
          "accepts": "application/alto-endpointcostparams+json",
          "capabilities": {
            "cost-type-names": [ "path-vector" ],
            "dependent-property-map": "propmap-availbw-delay",
            "allow-compound-response": true
          }
        },
        "invalid-cost-map" : {
          "uri": "http://alto.example.com/costmap/invalid",
          "media-type": "application/alto-costmap+json",
          "accepts": "application/alto-costmapfilter+json",
          "capabilities": {
            "cost-type-names": [ "path-vector" ],
            "allow-compound-response": true
          },
          "uses": [ "my-default-networkmap" ]
        },
        "propmap-availbw-delay": {
          "uri": "http://alto.exmaple.com/propmap/ane-prop",
```

```
            "media-type": "application/alto-propmap+json",
            "accepts": "application/alto-propmapparams+json",
            "capabilities": {
              "domain-types": [ "ane" ],
              "prop-types": [ "availbw", "delay" ]
            },
            "uses": [ "cost-map-pv", "endpoint-cost-pv" ]
          }
        }
      }
```

8.3. Example # 1

Query filtered cost map to get the path vectors.

```
POST /costmap/pv 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": "array",
    "cost-metric": "ane-path"
  },
  "pids": {
    "srcs": [ "PID1" ],
    "dsts": [ "PID2", "PID3" ]
  }
}
```

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

{
  "meta": {
    "dependent-vtags": [
      {
        "resource-id": "my-default-networkmap",
        "tag": "75ed013b3cb58f896e839582504f622838ce670f"
      }
    ],
    "cost-type": {
      "cost-mode": "array",
      "cost-metric": "ane-path"
    }
  },
  "cost-map": {
    "PID1": {
      "PID2": [ "ane:L001", "ane:L003" ],
      "PID3": [ "ane:L001", "ane:L004" ]
    }
  }
}
```

Then query the properties of ANEs in path vectors.

```
POST /propmap/ane-prop HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json,
        application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-propmapparams+json

{
  "entities": [ "ane:L001", "ane:L003", "ane:L004" ],
  "properties": [ "delay" ]
}
```

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

{
  "meta": {
    "dependent-vtags": [
      {
        "resource-id": "cost-map-pv",
        "tag": "a7d57e120ab63124e3c9a82f7a54bc120fc96216"
      }
    ]
  },
  "property-map": {
    "ane:L001": { "delay": 46},
    "ane:L003": { "delay": 50},
    "ane:L004": { "delay": 70}
  }
}
```

8.4. Example # 2

```
POST /endpointcost/pv 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

{
  "multi-cost-types": [
    {
      "cost-mode": "array",
      "cost-metric": "ane-path"
    },
    {
      "cost-mode": "numerical",
      "cost-metric": "routingcost"
    }
  ],
  "endpoints": {
    "srcs": [ "ipv4:192.0.2.2" ],
    "dsts": [ "ipv4:192.0.2.89",
              "ipv4:203.0.113.45",
              "ipv6:2001:db8::10" ]
  }
}
```

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

{
  "meta": {
    "cost-type": [
      {"cost-mode": "array", "cost-metric": "ane-path"}
    ]
  },
  "endpoint-cost-map": {
    "ipv4:192.0.2.2": {
      "ipv4:192.0.2.89":   [ "ane:L001", "ane:L003",
                             "ane:L004" ],
      "ipv4:203.0.113.45": [ "ane:L001", "ane:L004",
                             "ane:L005" ],
      "ipv6:2001:db8::10": [ "ane:L001", "ane:L005",
                             "ane:L007" ]
    }
  }
}

POST /endpointcost/pv 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

{
  "entities": [ "ane:L001", "ane:L003", "ane:L004",
                "ane:L005", "ane:L007" ],
  "properties": [ "availbw" ]
}
```

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

{
  "meta": {
    "dependent-vtags": [
      {
        "resource-id": "endpoint-cost-pv",
        "tag": "12c0889c3c0892bb67df561ed16d93f5d1fa75cf"
      }
    ]
  },
  "property-map": {
    "ane:L001": { "availbw": 50 },
    "ane:L003": { "availbw": 48 },
    "ane:L004": { "availbw": 55 },
    "ane:L005": { "availbw": 60 },
    "ane:L007": { "availbw": 35 }
  }
}
```

8.5. Example #3

```
POST /endpointcost/pv 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

{
  "multi-cost-types": [
    {
      "cost-mode": "array",
      "cost-metric": "ane-path"
    },
    {
      "cost-mode": "numerical",
      "cost-metric": "routingcost"
    }
  ],
  "endpoints": {
    "srcs": [ "ipv4:192.0.2.2" ],
    "dsts": [ "ipv4:192.0.2.89",
              "ipv4:203.0.113.45",
              "ipv6:2001:db8::10" ]
  },
  "properties": [ "delay", "availbw" ]
}
```

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

{
  "meta": {
    "dependent-vtags": [
      {
        "resource-id": "propmap-availbw-delay",
        "tag": "bb6bb72eafe8f9bdc4f335c7ed3b10822a391cef"
      }
    ],
    "cost-type": [
      {"cost-mode": "array", "cost-metric": "ane-path"}
    ]
  },
  "endpoint-cost-map": {
    "ipv4:192.0.2.2": {
      "ipv4:192.0.2.89":   [ "ane:L001", "ane:L003",
                             "ane:L004" ],
      "ipv4:203.0.113.45": [ "ane:L001", "ane:L004",
                             "ane:L005" ],
      "ipv6:2001:db8::10": [ "ane:L001", "ane:L005",
                             "ane:L007" ]
    }
  },
  "property-map": {
    "ane:L001": { "availbw": 50, "delay": 46 },
    "ane:L003": { "availbw": 48, "delay": 50 },
    "ane:L004": { "availbw": 55, "delay": 70 },
    "ane:L005": { "availbw": 60, "delay": 100 },
    "ane:L007": { "availbw": 35, "delay": 100 }
  }
}
```

9. Compatibility

9.1. Compatibility with Legacy ALTO Clients/Servers

The path vector extension on Filtered Cost Map and Endpoint Cost Service is backward compatible with the base ALTO protocol:

- If the ALTO server provides extended capabilities "dependent-property-map" and "allow-compound-response" for Filtered Cost Map or Endpoint Cost Service, but the client only supports the base ALTO protocol, then the client will ignore those capabilities without conducting any incompatibility.

- If the client sends a request with the input parameter "properties", but the server only supports the base ALTO protocol, the server will ignore this field.

9.2. Compatibility with Multi-Cost Extension

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

As Section 7.1.1 mentions, the syntax and semantics of whether "constraints" or "or-constraints" field for the "array" cost mode is not specified in this document. So if an ALTO server provides a resource with the "array" cost mode and the capability "cost-constraints" or "testable-cost-types-names", the ALTO client MAY ignore the capability "cost-constraints" or "testable-cost-types-names" unless the implementation or future docuements specify the behavior.

9.3. Compatibility with Incremental Update

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

- When using the compound response, updates on both cost map and property map SHOULD be notified.

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

10. General Discussions

10.1. Provide Calendar for Property Map

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

For example, as the properties of ANEs (e.g., available bandwidth and link delay) are usually the real-time network states, they change

frequently in the real network. It is very helpful to get the historical value of these properties. Applications may predicate the network status using these information to better optimize their performance.

So the coming requirement may be a general calendar service for the ALTO information resources.

10.2. Constraint Tests for General Cost Types

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

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

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

10.3. General Compound Resources Query

As the last paragraph of Section 4.3 mentions, querying multiple ALTO information resources continuously is 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

11.1. Privacy Concerns

We can identify multiple potential security issues. A main security issue is network privacy, as the path vector information may reveal more network internal structures than the more abstract single-node abstraction. The network should consider protection mechanisms to reduce information exposure, in particular, in settings where the network and the application do not belong to the same trust domain. On the other hand, in a setting of the same trust domain, a key

benefit of the path-vector abstraction is reduced information transfer from the network to the application.

Beyond the privacy issues, the computation of the path vector is unlikely to be cacheable, in that the results will depend on the particular requests (e.g., where the flows are distributed). Hence, this service may become an entry point for denial of service attacks on the availability of an ALTO server. Hence, authenticity and authorization of this ALTO service may need to be better protected.

11.2. Resource Consumption on ALTO Servers

The dependent Property Map of path vectors is dynamically enriched when the (Filtered) Cost Map/Endpoint Cost Service is queried of the path-vector information. The properties of the abstract network elements can consume a large amount of resources when cached. So, a time-to-live is needed to remove outdated entries in the Abstract Network Element Property Map.

12. IANA Considerations

12.1. ALTO Cost Mode Registry

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

12.2. ALTO Cost Metric Registry

A new cost metric needs to be registered in the "ALTO Cost Metric Registry", listed in Table 2.

Identifier	Intended Semantics
ane-path	See Section 5.2

Table 2: ALTO Cost Metrics

12.3. ALTO Domain Registry

As proposed in Section 9.2 of [I-D.ietf-alto-unified-props-new], "ALTO Domain Registry" is requested. Besides, a new domain is to be registered, listed in Table 3.

Identifier	Entity Address Encoding	Hierarchy & Inheritance
ane	See Section 6.2	None

Table 3: ALTO Domain

12.4. ALTO Network Element Property Type Registry

The "ALTO Abstract Network Element Property Type Registry" is required by the ALTO Domain "ane", listed in Table 4.

Identifier	Intended Semantics
availbw	The available bandwidth
delay	The transmission delay

Table 4: ALTO Abstract Network Element Property Types

13. Acknowledgments

The authors would like to thank discussions with Randriamasy Sabine, Andreas Voellmy, Erran Li, Haibin Son, Haizhou Du, Jiayuan Hu, Qiao Xiang, Tianyuan Liu, Xiao Shi, Xin Wang, and Yan Luo.

14. References

14.1. 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>.

14.2. Informative References

[I-D.amante-i2rs-topology-use-cases] Medved, J., Previdi, S., Lopez, V., and S. Amante, "Topology API Use Cases", draft-amante-i2rs-topology-use-cases-01 (work in progress), October 2013.

[I-D.bernstein-alto-topo] Bernstein, G., Yang, Y., and Y. Lee, "ALTO Topology Service: Uses Cases, Requirements, and Framework", draft-bernstein-alto-topo-00 (work in progress), October 2013.

[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-08 (work in progress), January 2018.

[I-D.ietf-alto-unified-props-new]
Roome, W., Chen, S., xinwang2014@hotmail.com, x., Yang, Y., and J. Zhang, "Extensible Property Maps for the ALTO Protocol", draft-ietf-alto-unified-props-new-01 (work in progress), December 2017.

[I-D.lee-alto-app-net-info-exchange]
Lee, Y., Dhody, D., Wu, Q., Bernstein, G., and T. Choi, "ALTO Extensions to Support Application and Network Resource Information Exchange for High Bandwidth Applications in TE networks", draft-lee-alto-app-net-info-exchange-04 (work in progress), October 2013.

[I-D.yang-alto-topology]
Bernstein, G., Lee, Y., Roome, W., Scharf, M., and Y. Yang, "ALTO Topology Extensions: Node-Link Graphs", draft-yang-alto-topology-06 (work in progress), March 2015.

[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>.

Authors' Addresses

Greg Bernstein
Grotto Networking
Fremont, CA
USA

Email: gregb@grotto-networking.com

Shiwei Dawn Chen
Tongji University
4800 Caoan Road
Shanghai 201804
China

Email: dawn_chen_f@hotmail.com

Kai Gao
Tsinghua University
Beijing Beijing
China

Email: gaok12@mails.tsinghua.edu.cn

Young Lee
Huawei
TX
USA

Email: leeyoung@huawei.com

Wendy Roome
Nokia/Bell Labs (Retired)
124 Burlington Rd
Murray Hill, NJ 07974
USA

Phone: +1-908-464-6975
Email: wendy@wdroome.com

Michael Scharf
Nokia
Germany

Email: michael.scharf@nokia.com

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

Email: yry@cs.yale.edu

Jingxuan Jensen Zhang
Tongji University
4800 Caoan Road
Shanghai 201804
China

Email: jingxuan.n.zhang@gmail.com

ALTO WG
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2018

W. Roome
Nokia Bell Labs
S. Chen
Tongji University
S. Randriamasy
Nokia Bell Labs
Y. Yang
Yale University
J. Zhang
Tongji University

June 29, 2018

Unified Properties for the ALTO Protocol
draft-ietf-alto-unified-props-new-04

Abstract

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

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

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

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

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

This Internet-Draft will expire on December 31, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 3
2. Definitions and Concepts 4
2.1. Entity 4
2.2. Entity Domain 5
2.3. Domain Name 5
2.4. Entity Address 5
2.5. Property Name 6
2.6. Hierarchy and Inheritance 6
2.7. Relationship with Other ALTO Resources 6
3. Entity Domains 7
3.1. Internet Address Domains 7
3.1.1. IPv4 Domain 7
3.1.2. IPv6 Domain 8
3.1.3. Hierarchy and Inheritance of ipv4/ipv6 Domains . . . 8
3.1.4. Relationship to Network Maps 9
3.2. PID Domain 10
3.2.1. Domain Name 10
3.2.2. Domain-Specific Entity Addresses 10
3.2.3. Hierarchy and Inheritance 10
3.2.4. Relationship To Internet Addresses Domains 10
3.3. Internet Address Properties vs. PID Properties 10
4. Property Map Resource 11
4.1. Media Type 11
4.2. HTTP Method 11
4.3. Accept Input Parameters 11
4.4. Capabilities 11
4.5. Uses 12
4.6. Response 12
5. Filtered Property Map Resource 13
5.1. Media Type 13
5.2. HTTP Method 13

5.3. Accept Input Parameters 13
5.4. Capabilities 14
5.5. Uses 14
5.6. Response 14
6. Impact on Legacy ALTO Servers and ALTO Clients 14
6.1. Impact on Endpoint Property Service 15
6.2. Impact on Resource-Specific Properties 15
6.3. Impact on the pid Property 15
6.4. Impact on Other Properties 16
7. Examples 16
7.1. Network Map 16
7.2. Property Definitions 16
7.3. Information Resource Directory (IRD) 16
7.4. Property Map Example 18
7.5. Filtered Property Map Example #1 19
7.6. Filtered Property Map Example #2 20
7.7. Filtered Property Map Example #3 21
7.8. Filtered Property Map Example #4 22
8. Security Considerations 24
9. IANA Considerations 24
9.1. application/alto-* Media Types 24
9.2. ALTO Entity Domain Registry 25
9.2.1. Consistency Procedure between ALTO Address Type Registry and ALTO Entity Domain Registry 26
9.2.2. ALTO Entity Domain Registration Process 27
9.3. ALTO Endpoint Property Type Registry 28
10. References 28
10.1. Normative References 28
10.2. Informative References 29
Authors' Addresses 29

1. Introduction

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

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

Second, the EPS is only defined as a POST-mode service. Clients must request the properties for an explicit set of addresses. By

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

This document proposes a new approach to retrieve ALTO properties. Specifically, it defines two new resource types, namely Property Maps (see Section 4) and Filtered Property Maps (see Section 5). The former are GET-mode resources which return the property values for all entities in a domain, and are analogous to the ALTO's Network Maps and Cost Maps. The latter are POST-mode resources which return the values for a set of properties and entities requested by the client, and are analogous to the ALTO's Filtered Network Maps and Filtered Cost Maps.

Additionally, this document introduces ALTO Entity Domains, where entities extend the concept of endpoints to objects that may be endpoints as defined in [RFC7285] but also, for example, PIDs, Abstract Network Elements as defined in [I-D.ietf-alto-path-vector] or cells. As a consequence, ALTO Entity Domains are a super-set of ALTO Address Types and their relation is specified in Section 9.2.1.

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

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

2. Definitions and Concepts

2.1. Entity

The entity is an extended concept of the endpoint defined in Section 2.1 of [RFC7285]. An entity is an object with a (possibly

empty) set of properties. Every entity is in a domain, such as the IPv4 and IPv6 domains, and has a unique address.

2.2. Entity Domain

An entity domain is a family of entities. Two examples are the Internet address and PID domain (see Section 3.1 and Section 3.2) that this document will define.

2.3. Domain Name

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

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

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

2.4. Entity Address

Each entity has a unique address of the format:

```
domain-name : domain-specific-entity-address
```

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

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

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

Note that an entity address MAY have different textual representations, for a given entity domain. For example, the strings

"ipv6:2001:db8::1" and "ipv6:2001:db8:0:0:0:0:0:1" refer to the same entity.

2.5. Property Name

The space of property names associated with entities defined by this document is the same as, and is shared with, the endpoint property names defined by [RFC7285]. Thus entity property names are as defined in Section 10.8.2 of that document, and must be registered with the "ALTO Endpoint Property Type Registry" defined in Section 9.3 of that document. The type PropertyName denotes a JSON string with a property name in this format.

This document defines uniform property names specified in a single property name space rather than being scoped by a specific entity domain, although some properties may only be applicable for particular entity domains. This design decision is to enforce a design so that similar properties are named similarly. The interpretation of the value of a property, however, may depend on the entity domain. For example, suppose the "geo-location" property is defined as the coordinates of a point, encoded as (say) "latitude longitude [altitude]." When applied to an entity that represents a specific host computer, such as an Internet address, the property defines the host’s location. When applied to an entity that represents a set of computers, such as a CIDR, the property would be the location of the center of that set. If it is necessary to represent the bounding box of a set of hosts, another property, such as "geo-region", should be defined.

2.6. Hierarchy and Inheritance

Entities in a given domain MAY form hierarchy based on entity address. Each entity domain MUST define its own hierarchy and inheritance rules when registered. The hierarchy and inheritance rule makes it possible for an entity to inherit a property value from another entity in the same domain. If and only if the property of an entity is undefined, the hierarchy and inheritance rules are applied.

2.7. Relationship with Other ALTO Resources

[RFC7285] recognizes that some properties MAY be specific to another ALTO resource, such as a network map. Accordingly [RFC7285] defines the concept of "resource-specific endpoint properties" (see Section 10.8.1), and indicates that dependency by prefixing the property name with the ID of the resource on which it depends. That document defines one resource-specific property, namely the "pid" property, whose value is the name of the PID containing that endpoint in the associated network map.

This document takes a different approach. Instead of defining the dependency by qualifying the property name, this document attaches the dependency to the entity domains. Thus all properties of a specific entity domain depend on the same resource, the properties of another entity domain may depend on another resource. For example, entities in the PID domain depend on a network map.

The "uses" field in an IRD entry defines the dependencies of a property map resource, and the "dependent-vtags" field in a property map response defines the dependencies of that map. These fields are defined in Sections 9.1.5 and 11.1 of [RFC7285], respectively.

The "uses" field in an IRD entry MUST NOT include two dependent resources with the same media type. This is similar to how [RFC7285] handles dependencies between cost maps and network maps. Recall that cost maps present the costs between PIDs, and PID names depend on a network map. If an ALTO server provides the "routingcost" metric for the network maps "net1" and "net2", then the server defines two separate cost maps, one for "net1" and the other for "net2".

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

3. Entity Domains

This document defines the following entity domains. For the definition of each entity domain, it includes the following template: domain name, domain-specific addresses, and hierarchy and inheritance semantics.

3.1. Internet Address Domains

The document defines two entity domains (IPv4 and IPv6) for Internet addresses. Both entity domains include individual addresses and blocks of addresses.

3.1.1. IPv4 Domain

3.1.1.1. Domain Name

ipv4

3.1.1.2. Domain-Specific Entity Addresses

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

3.1.2. IPv6 Domain

3.1.2.1. Domain Name

ipv6

3.1.2.2. Domain-Specific Entity Addresses

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

3.1.3. Hierarchy and Inheritance of ipv4/ipv6 Domains

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

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

As an example, suppose that a server defines the 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 A 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 this property has no value.

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

3.1.4. Relationship to Network Maps

An Internet address domain MAY be associated with an ALTO network map resource. Logically, there is a map of Internet address entities to property values for each network map defined by the ALTO server, plus an additional property map for Internet address entities which are

not associated with a network map. So, if there are n network maps, the server can provide n+1 maps of Internet address entities to property values. These maps are separate from each other. The prefixes in the property map do not have to correspond to the prefixes defining the network map's PIDs. For example, the property map for a network map MAY assign properties to "ipv4:192.0.2.0/24" even if that prefix is not associated with any PID in the network map.

3.2. PID Domain

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

3.2.1. Domain Name

pid

3.2.2. Domain-Specific Entity Addresses

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

3.2.3. Hierarchy and Inheritance

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

3.2.4. Relationship To Internet Addresses Domains

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

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

3.3. Internet Address Properties vs. PID Properties

Because the Internet address and PID domains are completely separate, the question may arise as to which entity domain is the best for a property. In general, the Internet address domain is RECOMMENDED for

properties that are closely related to the Internet address, or are associated with, and inherited through, blocks of addresses.

The PID domain is RECOMMENDED for properties that arise from the definition of the PID, rather than from the Internet address prefixes in that PID.

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

4. Property Map Resource

A Property Map returns the properties defined for all entities in one or more domains.

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

4.1. Media Type

The media type of an ALTO Property Map resource is "application/alto-propmap+json".

4.2. HTTP Method

An ALTO Property Map resource is requested using the HTTP GET method.

4.3. Accept Input Parameters

None.

4.4. Capabilities

The capabilities are defined by an object of type PropertyMapCapabilities:

```
object {
  DomainName entity-domain-types<1..*>;
  PropertyName prop-types<1..*>;
} PropertyMapCapabilities;
```

where "entity-domain-types" is an array with the domains of the entities in this property map, and "prop-types" is an array with the names of the properties returned for entities in those domains.

4.5. Uses

An array with the resource ID(s) of resource(s) with which the entity domains in this map are associated. In most cases, this array will have at most one ID, for example, for a network map resource. However, the "uses" field MUST NOT contain two resources of the same resource type. For example, if a property map depends on network map resource, the "uses" field MUST include exactly one network map resource.

4.6. Response

If the entity domains in this property map depend on other resources, the "dependent-vtags" field in the "meta" field of the response MUST be an array that includes the version tags of those resources. 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 {
  EntityAddr -> EntityProps;
} PropertyMapData;

object {
  PropertyName -> JSONValue;
} EntityProps;
```

The ResponseEntityBase type is defined in Section 8.4 of [RFC7285].

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

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

5. Filtered Property Map Resource

A Filtered Property Map returns the values of a set of properties for a set of entities selected by the client.

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

5.1. Media Type

The media type of an ALTO Property Map resource is "application/alto-propmap+json".

5.2. HTTP Method

An ALTO Filtered Property Map resource is requested using the HTTP POST method.

5.3. Accept Input Parameters

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

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

with fields:

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

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

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

5.4. Capabilities

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

5.5. Uses

An array with the resource ID(s) of resource(s) with which the entity domains in this map are associated. In most cases, this array will have at most one ID, and it will be for a network map resource.

5.6. Response

The response is the same as for the property map (see Section 4.6), except that it only includes the entities and properties requested by the client.

Also, the Filtered Property Map response MUST include all inherited property values for the specified entities (unlike the Full Property Map, the Filtered Property Map response does not include enough information for the client to calculate the inherited values).

If an entity in "entities" in the request is invalid, the ALTO server MUST return an "E_INVALID_FIELD_VALUE" error defined in Section 8.5.2 of [RFC7285]. An entity can be invalid if the domain of the entity is not defined in the IRD for this service or the entity address is an invalid address of the entity domain. On the other hand, a valid entity address is not an error, even if the server does not define a value for a requested property. In this case, the server MUST omit that property from the response for only that entity. If a request contains a property in "properties" and the property is not specified in the IRD for the service, the ALTO server MUST return an "E_INVALID_FIELD_VALUE" error defined in Section 8.5.2 of [RFC7285]. The "value" of the error message SHOULD indicate the wrong property.

If the ALTO server does not define a requested property’s value for a particular entity, then it MUST omit that property from the response for only that endpoint.

If the ALTO server does not support a requested entity’s domain, then it MUST return an E_INVALID_FIELD_VALUE error defined in Section 8.5.2 of [RFC7285].

6. Impact on Legacy ALTO Servers and ALTO Clients

6.1. Impact on Endpoint Property Service

The Property Maps defined in this document provide the same functionality as the Endpoint Property Service (EPS) defined in Section 11.4 of [RFC7285]. Accordingly, it is RECOMMENDED that the EPS be deprecated in favor of Property Maps. However, ALTO servers MAY provide an EPS for the benefit of legacy clients.

6.2. Impact on Resource-Specific Properties

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

6.3. Impact on the pid Property

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

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

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

Second, a client MAY request the "pid" property for a block of addresses. An ALTO server determines the value of "pid" for an address block C as follows. Let CS be the set of all address blocks in the network map. If C is in CS, then the value of "pid" is the name of the PID associated with C. Otherwise, find the longest block C’ in CS such that C’ prefix-matches C, but is shorter than C. If

there is such a block C', the value of "pid" is the name of the PID associated with C'. If not, then "pid" has no value for block C.

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

6.4. Impact on Other Properties

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

7. Examples

7.1. Network Map

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

```
defaultpid:  ipv4:0.0.0.0/0  ipv6:::0/0
pid1:        ipv4:192.0.2.0/25
pid2:        ipv4:192.0.2.0/28  ipv4:192.0.2.16/28
```

Figure 3: Example Network Map

7.2. Property Definitions

The examples in this section use four additional properties, "ISP", "ASN", "country" and "state", with the following values:

	ISP	ASN	country	state
ipv4:192.0.2.0/24:	BitsRus	-	us	-
ipv4:192.0.2.0/28:	-	12345	-	NJ
ipv4:192.0.2.16/28:	-	12345	-	CT
ipv4:192.0.2.0:	-	-	-	PA

Figure 4: Example Property Values

7.3. Information Resource Directory (IRD)

The following IRD defines the relevant resources of the ALTO server. It provides two Property Map resources, one for the "ISP" and "ASN" properties, and another for the "country" and "state" properties. The server could have provided a Property Map resource for all four properties, but did not, presumably because the organization that

runs the ALTO server believes any given client is not interested in all four properties.

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

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

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

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

```
          }
        },
        "pid-property-map" : {
           "uri" : "http://alto.example.com/propmap/lookup/pid",
           "media-type" : "application/alto-propmap+json",
           "accepts" : "application/alto-propmapparams+json",
           "uses" : [ "default-network-map" ]
           "capabilities" : {
             "entity-domain-types" : [ "ipv4", "ipv6" ],
             "prop-types" : [ "pid" ]
           }
        },
        "location-property-map": {
          "uri": "http://alto.exmaple.com/propmap/location",
          "media-type": "application/alto-propmap+json",
          "accepts": "application/alto-propmapparams+json",
          "uses" : [ "default-network-map" ],
          "capabilities": {
            "domain-types": [ "pid" ],
            "prop-types": [ "country", "state" ]
          }
        },
        "legacy-pid-property" : {
           "uri" : "http://alto.example.com/legacy/eps-pid",
           "media-type" : "application/alto-endpointprop+json",
           "accepts" : "application/alto-endpointpropparams+json",
           "capabilities" : {
             "prop-types" : [ "default-network-map.pid" ]
           }
        }
      }
```

Figure 5: Example IRD

7.4. Property Map Example

The following example uses the properties and IRD defined above to retrieve a property map for entities with the "ISP" and "ASN" properties. Note that the response does not include the entity "ipv4:192.0.2.0", because it does not have a value for either of those properties. Also note that the entities "ipv4:192.0.2.0/28" and "ipv4:192.0.2.16/28" are refinements of "ipv4:192.0.2.0/24", and hence inherit its value for "ISP" property. But because that value is inherited, it is not explicitly listed in the property map.

```
GET /propmap/full/inet-ia HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json


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

{
  "property-map": {
    "ipv4:192.0.2.0/24":   {"ISP": "BitsRus"},
    "ipv4:192.0.2.0/28":   {"ASN": "12345"},
    "ipv4:192.0.2.16/28":  {"ASN": "12345"}
  }
}
```

7.5. Filtered Property Map Example #1

The following example uses the Filtered Property Map resource to request the "ISP", "ASN" and "state" properties for several IPv4 addresses. Note that the value of "state" for "ipv4:192.0.2.0" is the only explicitly defined property; the other values are all derived by the inheritance rules for Internet address entities.

```
POST /propmap/lookup/inet-iacs HTTP/1.1
Host: alto.example.com
Accept: application/alto-propmap+json,application/alto-error+json
Content-Length: ###
Content-Type: application/alto-propmapparams+json

{
  "entities" : [ "ipv4:192.0.2.0",
                 "ipv4:192.0.2.1",
                 "ipv4:192.0.2.17" ],
  "properties" : [ "ISP", "ASN", "state" ]
}


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

{
  "property-map": {
    "ipv4:192.0.2.0":
           {"ISP": "BitsRus", "ASN": "12345", "state": "PA"},
    "ipv4:192.0.2.1":
           {"ISP": "BitsRus", "ASN": "12345", "state": "NJ"},
    "ipv4:192.0.2.17":
           {"ISP": "BitsRus", "ASN": "12345", "state": "CT"}
  }
}
```

7.6. Filtered Property Map Example #2

The following example uses the Filtered Property Map resource to request the "ASN", "country" and "state" properties for several IPv4 prefixes. Note that none of the returned property values is explicitly defined; all values are derived by the inheritance rules for Internet address entities.

Also note the "ASN" property has the value "12345" for both the blocks "ipv4:192.0.2.0/28" and "ipv4:192.0.2.16/28", so every address in the block "ipv4:192.0.2.0/27" has that property value. However the block "ipv4:192.0.2.0/27" itself does not have a value for "ASN": address blocks cannot inherit properties from blocks with longer prefixes, even if every such block has the same value.

```
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",
                 "ipv4:192.0.2.0/27",
                 "ipv4:192.0.2.0/28" ],
  "properties" : [ "ASN", "country", "state" ]
}


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

{
  "property-map": {
    "ipv4:192.0.2.0/26":  {"country": "us"},
    "ipv4:192.0.2.0/27":  {"country": "us"},
    "ipv4:192.0.2.0/28":  {"ASN": "12345",
                           "country": "us",
                           "state": "NJ"}
 }
}
```

7.7. Filtered Property Map Example #3

The following example uses the Filtered Property Map resource to request the "pid" property for several IPv4 addresses and prefixes.

Note that the value of "pid" for the prefix "ipv4:192.0.2.0/26" is "pid1", even though all addresses in that block are in "pid2", because "ipv4:192.0.2.0/25" is the longest prefix in the network map which prefix-matches "ipv4:192.0.2.0/26", and that prefix is in "pid1".

```
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" : [
                "ipv4:192.0.2.0",
                "ipv4:192.0.2.16",
                "ipv4:192.0.2.64",
                "ipv4:192.0.2.128",
                "ipv4:192.0.2.0/26",
                "ipv4:192.0.2.0/30" ],
  "properties" : [ "pid" ]
}


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": {
    "ipv4:192.0.2.0":     {"pid": "pid2"},
    "ipv4:192.0.2.16":    {"pid": "pid2"},
    "ipv4:192.0.2.64":    {"pid": "pid1"},
    "ipv4:192.0.2.128":   {"pid": "defaultpid"},
    "ipv4:192.0.2.0/26":  {"pid": "pid1"},
    "ipv4:192.0.2.0/30":  {"pid": "pid2"}
  }
}
```

7.8. Filtered Property Map Example #4

The following example uses the Filtered Property Map resource to request the "country" and "state" property for several PIDs defined in "default-network-map".

```
POST /propmap/lookup/location 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" : ["pid:pid3",
                "pid:pid4",
                "pid:pid5",
                "pid:pid6",
                "pid:pid7"],
  "properties" : [ "country", "state" ]
}

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

{
  "meta" : {
    "dependent-vtags" : [
       {"resource-id": "default-network-map",
        "tag": "7915dc0290c2705481c491a2b4ffbec482b3cf62"}
    ]
  },
  "property-map": {
    "pid:pid3": {
      "country": "us",
      "state": "CA"
    },
    "pid:pid4": {
      "country": "us",
      "state": "CT"
    },
    "pid:pid5": {
      "country": "ca",
      "state": "QC"
    },
    "pid:pid6": {
      "country": "ca",
      "state": "NT"
    },
    "pid:pid7": {
      "country": "fr"
    }
  }
}
```

8. Security Considerations

As discussed in Section 15 of [RFC7285], properties MAY have sensitive customer-specific information. If this is the case, an ALTO Server MAY limit access to those properties by providing several different Property Maps. For non-sensitive properties, the ALTO Server would provide a URI which accepts requests from any client. Sensitive properties, on the other hand, would only be available via a secure URI which would require client authentication.

Also, while technically this document does not introduce any security risks not inherent in the Endpoint Property Service defined by [RFC7285], the GET-mode property map resource defined in this document does make it easier for a client to download large numbers of property values. Accordingly, an ALTO Server SHOULD limit GET-mode Property Maps to properties which do not contain sensitive data.

9. IANA Considerations

This document defines additional application/alto-* media types, and extends the ALTO endpoint property registry.

9.1. application/alto-* Media Types

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

Type	Subtype	Specification
application	alto-propmap+json	Section 4.1
application	alto-propmapparams+json	Section 5.3

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).

9.2. ALTO Entity Domain Registry

This document requests IANA to create and maintain the "ALTO Entity Domain Registry", listed in Table 2.

Identifier	Entity Address Encoding	Hierarchy & Inheritance
ipv4	See Section 3.1.1	See Section 3.1.3
ipv6	See Section 3.1.2	See Section 3.1.3
pid	See Section 3.2	None

Table 2: ALTO Entity Domains.

This registry serves two purposes. First, it ensures uniqueness of identifiers referring to ALTO entity domains. Second, it states the requirements for allocated entity domains.

9.2.1. Consistency Procedure between ALTO Address Type Registry and ALTO Entity Domain Registry

One potential issue of introducing the "ALTO Entity Domain Registry" is its relationship with the "ALTO Address Types Registry" already defined in Section 14.4 of [RFC7285]. In particular, the entity address of an entity domain registered in the "ALTO Entity Domain Registry" MAY match an address type defined in "ALTO Address Type Registry". It is necessary to precisely define and guarantee the consistency between "ALTO Address Type Registry" and "ALTO Entity Domain Registry".

We define that the ALTO Entity Domain Registry is consistent with ALTO Address Type Registry if two conditions are satisfied:

- When an address type is already or able to be registered in the ALTO Address Type Registry [RFC7285], the same identifier MUST be used when a corresponding entity domain is registered in the ALTO Entity Domain Registry.

- If an ALTO entity domain has the same identifier as an ALTO address type, their addresses encoding MUST be compatible.

To achieve this consistency, the following items MUST be checked before registering a new ALTO entity domain in a future document:

- Whether the ALTO Address Type Registry contains an address type that can be used as an entity address for the candidate domain identifier. This has been done for the identifiers "ipv4" and "ipv6" in Table 2.

- Whether the candidate entity address of the entity domain is able to be an endpoint address, as defined in Sections 2.1 and 2.2 of [RFC7285].

When a new ALTO entity domain is registered, the consistency with the ALTO Address Type Registry MUST be ensured by the following procedure:

- test: Do corresponding entity addresses match a known "network" address type?
  - if yes: (e.g., cell, MAC or socket addresses)
    - test: Is such an address type present in the ALTO Address Type Registry?
      - if yes: Set the new ALTO entity domain identifier to be the found ALTO address type identifier.
      - if no: Define a new ALTO entity domain identifier and use it to register a new address type in the ALTO Address Type Registry following Section 14.4 of [RFC7285].
    - Use the new ALTO entity domain identifier to register a new ALTO entity domain in the ALTO Entity Domain Registry following Section 9.2.2 of this document.
  - if no (e.g., pid name, ane name or country code): Proceed with the ALTO Entity Domain registration as described in Section 9.2.2.

9.2.2. ALTO Entity Domain Registration Process

New ALTO entity domains are assigned after IETF Review [RFC5226] to ensure that proper documentation regarding the new ALTO entity domains and their security considerations has been provided. RFCs defining new entity domains SHOULD indicate how an entity in a registered domain is encoded as an EntityAddr, and, if applicable, the rules defining the entity hierarchy and property inheritance. Updates and deletions of ALTO entity domains follow the same procedure.

Registered ALTO entity domain identifiers MUST conform to the syntactical requirements specified in Section 2.3. Identifiers are to be recorded and displayed as strings.

Requests to the IANA to add a new value to the registry MUST include the following information:

- Identifier: The name of the desired ALTO entity domain.

- Entity Address Encoding: The procedure for encoding the address of an entity of the registered type as an EntityAddr (see Section 2.4). If corresponding entity addresses of an entity domain match a known "network" address type, the Entity Address Encoding of this domain identifier MUST include both Address Encoding and Prefix Encoding of the same identifier registered in the ALTO Address Type Registry [RFC7285]. For the purpose of defining properties, an individual entity address 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.

- Security Considerations: In some usage scenarios, entity 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 the registered type SHOULD be made aware of how (or if) the addressing scheme relates to private information and network proximity.

This specification requests registration of the identifiers "ipv4", "ipv6" and "pid", as shown in Table 2.

9.3. ALTO Endpoint Property Type Registry

The ALTO Endpoint Property Type Registry was created by [RFC7285]. If possible, the name of that registry SHOULD be changed to "ALTO Entity Property Type Registry", to indicate that it is not restricted to Endpoint Properties. If it is not feasible to change the name, the description MUST be amended to indicate that it registers properties in all entity domains, rather than just the Internet address domain.

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, <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>.

[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>.

10.2. Informative References

[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 Type", draft-ietf-alto-path-vector-03 (work in progress), March 2018.

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

Shiwei Dawn Chen
Tongji University
4800 Caoan Road
Shanghai 201804
China

Email: dawn_chen_f@hotmail.com

Sabine Randriamasy
Nokia Bell Labs
Route de Villejust
NOZAY 91460
FRANCE

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

Y. Richard Yang
Yale University
51 Prospect Street
New Haven, CT 06511
USA

Phone: +1-203-432-6400
Email: yry@cs.yale.edu

Jingxuan Jensen Zhang
Tongji University
4800 Caoan Road
Shanghai 201804
China

Email: jingxuan.n.zhang@gmail.com

ALTO WG
Internet-Draft
Intended status: Informational
Expires: January 14, 2021

D. Lachos
C. Rothenberg
Unicamp

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.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 14, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
2. Terminology 3
3. Scope 4
4. Problem Statement and Challenges 5
5. Proposed Approach 6
5.1. Inter-domain Resource (IdR) Component 7
5.2. Inter-domain Topology (IdT) Component 7
5.3. ALTO Server Functionalities 7
5.4. Filtered Cost Map Extension 8
5.4.1. Accept Input Parameters 8
5.4.2. Response 9
5.5. Examples of Message Exchange 9
5.5.1. Property Map Service 9
5.5.2. Filtered Cost Map Service 10
6. Discussion 13
6.1. Benefits 14
6.2. Open Issues 14
7. IANA Considerations 15
8. Security Considerations 15
9. Acknowledgments 15
10. References 15
10.1. Normative References 15
10.2. Informative References 16
10.3. URIs 18
Appendix A. Proof of Concept Use Case Implementation 18
Authors' Addresses 22

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.

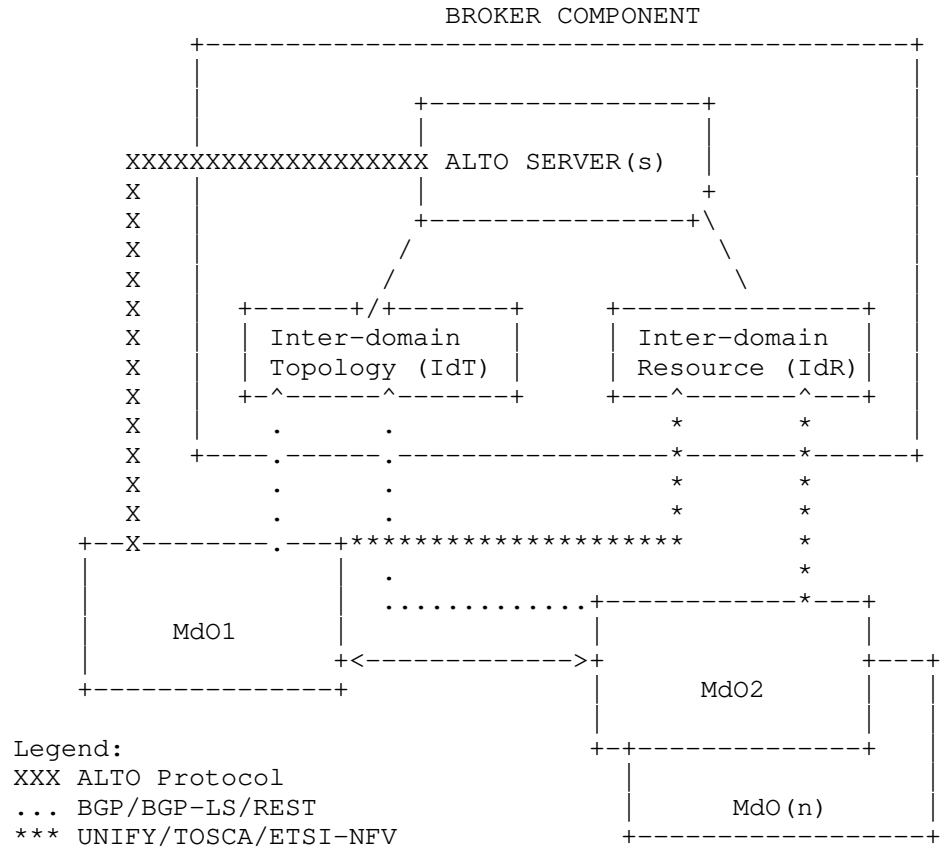

Figure 1: Broker-assisted Multi-operator Network Architecture

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 ]
          }
        ]
      }
    }
```

- 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:

- Avoid the distribution of topology and resource information in a peer-to-peer fashion (MdO-to-MdO)
- The (abstracted) information and offered resources, services are maintained in each local MdO.
- Allow domains without physical infrastructure (hence without BGP or BGP-LS) to advertise their capabilities.
- An ALTO-based privacy-preserving information model to provide computing, storage, and networking resource info.
- 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.

- 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].
- 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.

- 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.

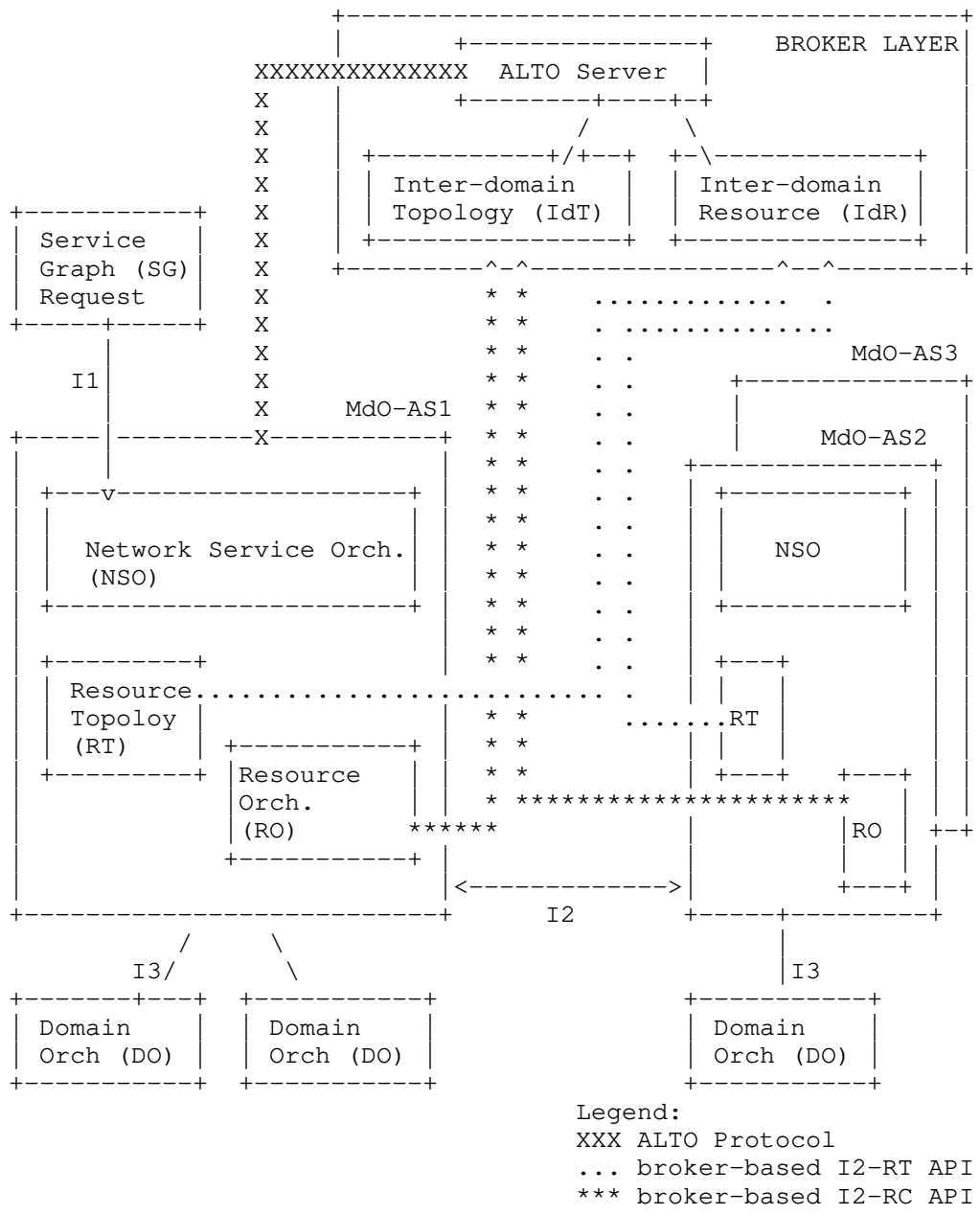

Figure 2: Broker-assisted 5GEx Info Exchange

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 WG
Internet-Draft
Intended status: Informational
Expires: 13 January 2022

Q. Xiang
Xiamen University
J. Zhang
Tongji/Yale University
F. Le
IBM
Y. Yang
Yale University
H. Newman
California Institute of Technology

12 July 2021

Resource Orchestration for Multi-Domain, Exascale, Geo-Distributed Data Analytics

draft-xiang-alto-multidomain-analytics-06.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

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

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

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

This Internet-Draft will expire on 13 January 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 3
2. Requirements Language 4
3. Changes Since Version -05 4
4. Characteristics of Multi-Domain, Geo-Distributed Data Analytics 4
4.1. Dynamic Data Analytics Workload 4
4.2. Dynamic Resource Availability 5
5. Design Requirements 6
6. Review of Resource Orchestration Designs for Data Analytics 7
6.1. Centralized resource-graph-based orchestration 7
6.2. Centralized ClassAds-based orchestration 7
6.3. Distributed opportunistic orchestration 7
6.4. Inadequacy of Existing Designs for Multi-Domain, Geo-Distributed Data Analytics 8
7. Unicorn Design 8
7.1. Choosing ALTO as the Resource Information Model 8
7.2. Architecture of Unicorn 9
7.2.1. Three-Phase Resource Discovery 11
7.2.2. Proactive Full-Mesh Resource Discovery 15

7.3. Example 15
8. ALTO Extension: Privacy-Preserving Interdomain Information Aggregation for Resource Discovery 16
8.1. Extension Specification 16
8.2. Example 18
9. Implementation and Demonstration Experience 19
10. Discussion 19
10.1. Discovering the Domain-Paths Using a New Interdomain Routing Protocol 20
10.2. Comparison of the Efficiency to Achieve Optimal Resource Orchestration using ALTO and without using ALTO 20
10.3. Future Work 1: Unified Resource Representation as an ALTO extension 20
10.4. Future Work 2: Integrating Unicorn and ALTO into Rucio 21
11. Security Considerations 21
12. IANA Considerations 21
13. References 21
13.1. Normative References 22
13.2. Informative References 22
Authors' Addresses 23

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 -05

* Add Section 10.4 to discuss how Ruzio, the latest scientific data management framework for the ATLAS experiment at LHC, can benefit from integrating with ALTO and Unicorn.

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

* allows member networks to provide accurate information on different types of resources, e.g., the computing, storage and networking resources, they own;

* keeps the private information of member networks, such as physical topologies and policies, from the data analytics applications; and

* 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:

* 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.

* 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.

* 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:

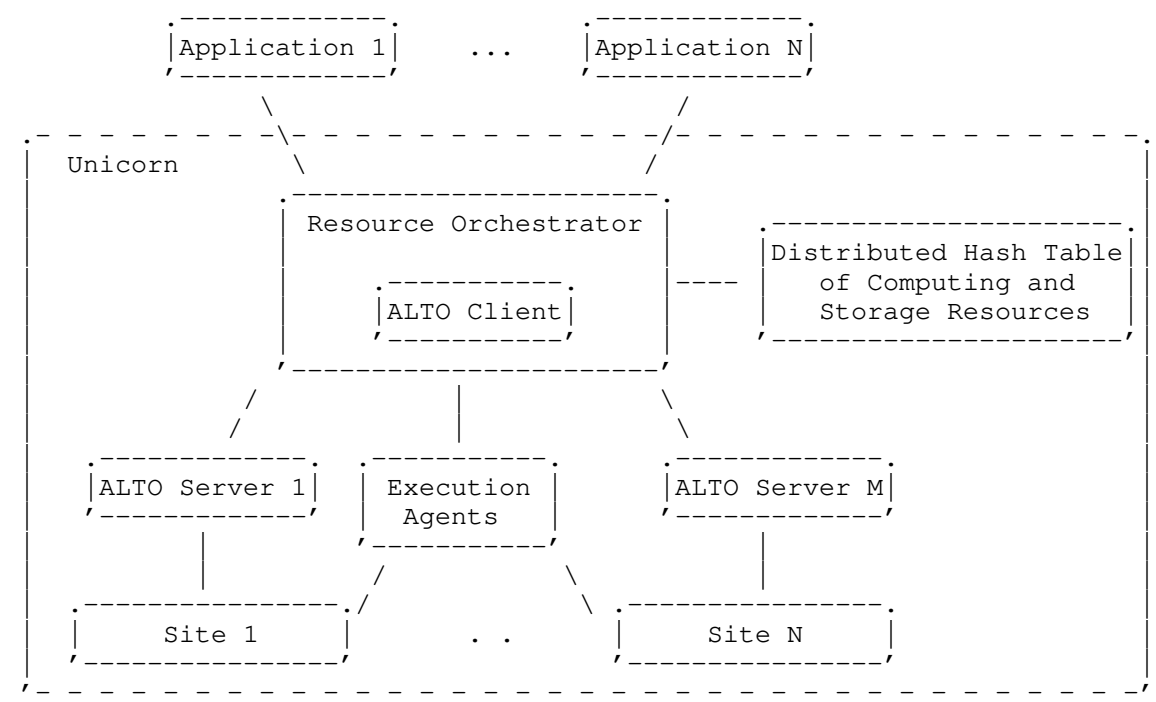

Figure 1: Architecture of Unicorn.

* 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.

* 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.

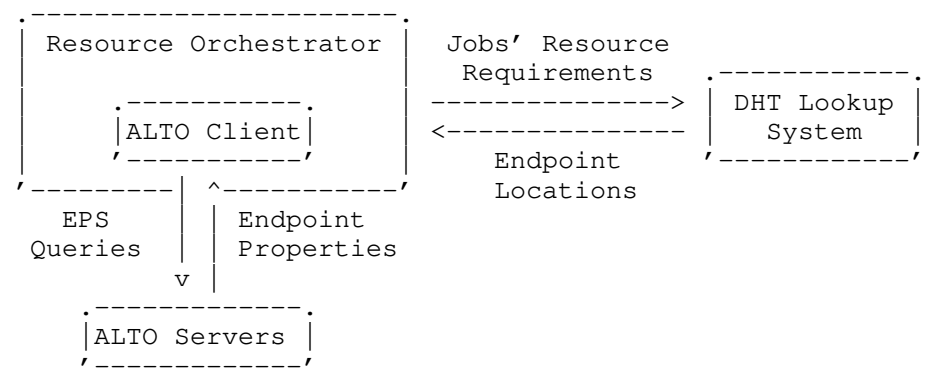

Figure 2: The Endpoint Property Discovery Phase.

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.

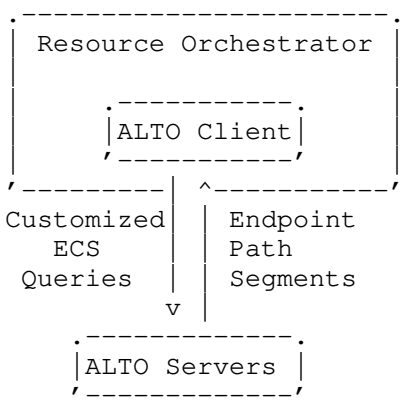

Figure 3: The Endpoint Path Discovery Phase.

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.

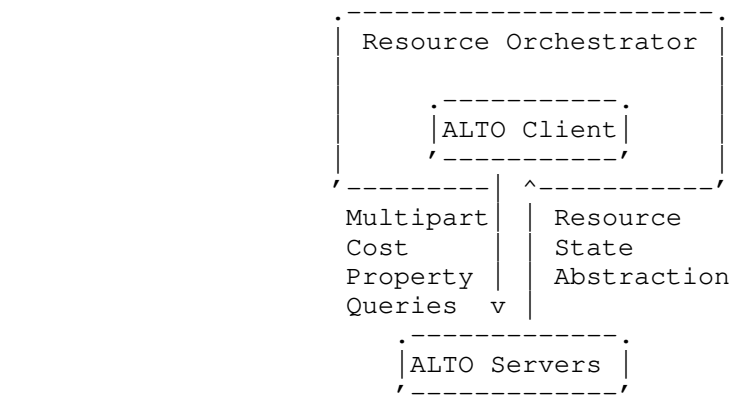

Figure 4: The Resource State Abstraction Discovery Phase.

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.

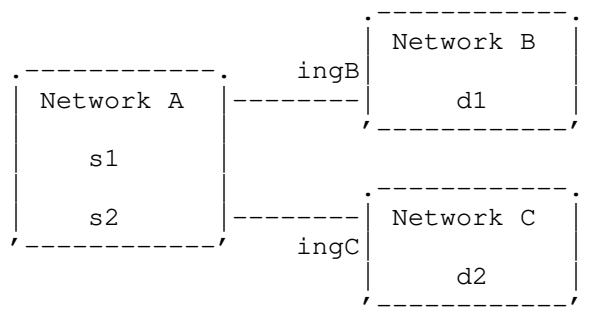

Figure 5: An Illustrating Example for Unicorn.

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:

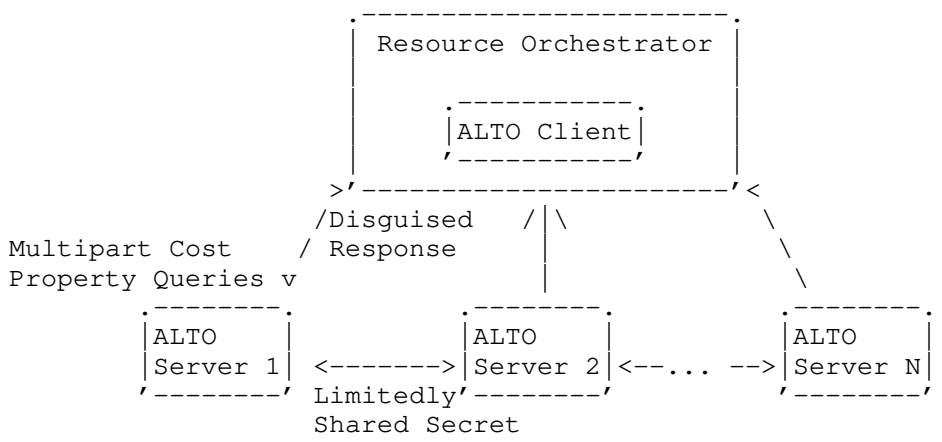

Figure 6: The Privacy-Preserving Interdomain Resource Information Aggregation.

* 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. Future Work 1: 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.

10.4. Future Work 2: Integrating Unicorn and ALTO into Rucio

Ruzio [RUCIO] is a scientific data management framework that supports scientific collaborations with the functionality to organize, manage, and access data at exascales. It is currently deployed at the ATLAS experiment of the LHC project, to manage detector data, simulation data and user data. Rucio interacts with the heterogeneous storage systems that are in use in ATLAS, and schedules the dataset transfers in the ATLAS science network using the network infrastructure provided by multiple national research and education networks, including ESnet and Internet2, as well as commercial cloud storage providers, including Google and Amazon.

However, the same as other scientific data management frameworks, e.g., PhEDEx, Rucio does not see the underlying network infrastructure. As such, it may suffer from underutilization of network resources, leading to a low data transfer efficiency. In contrast, ALTO and Unicorn can provide accurate, yet privacy-preserving network resource information. By integrating ALTO and Unicorn into the Rucio framework, Rucio will possess the capability to retrieve accurate resource information from the underlying network infrastructure, and use such information to schedule the exascale scientific data flows with better predictable performances (e.g., latency and throughput). As such, we plan to systematically investigate the integration of ALTO/Unicorn and Rucio in the new charter cycle of the ALTO working group.

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. R. 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. R. 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. R., 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. R. 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>.

[RUCIO] Barisits, M., "Rucio: Scientific Data Management", 2019, <https://doi.org/10.1007/s41781-019-0026-3>.

[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
Xiamen University
School of Informatics
Xiamen
Fujian Province,
China

Email: xiangq27@gmail.com

J. Jensen Zhang
Tongji/Yale University
51 Prospect Street
New Haven, CT
United States of America

Email: jingxuan.zhang@yale.edu

Franck Le
IBM
Thomas J. Watson Research Center
Yorktown Heights, NY,
United States of America

Email: fle@us.ibm.com

Y. Richard Yang
Yale University
51 Prospect Street
New Haven, CT
United States of America

Email: yry@cs.yale.edu

Harvey Newman
California Institute of Technology
1200 California Blvd.
Pasadena, CA
United States of America

Email: newman@hep.caltech.edu

ALTO WG
Internet-Draft
Intended status: Standards Track
Expires: January 14, 2021

Q. Xiang
Yale University
J. Zhang
Tongji/Yale University
F. Le
IBM
Y. Yang
Yale University

July 13, 2020

ALTO Extension: Unified Resource Representation
draft-xiang-alto-unified-representation-03.txt

Abstract

The ALTO protocol [RFC7285] provides network information to applications so that applications can make network informed decisions to improve the performance. However, the base ALTO protocol only provides coarse-grained end-to-end metrics, which are insufficient to satisfy the demands of applications to solve more complex network optimization problems. The ALTO Path Vector extension [DRAFT-PV] has been introduced to allow ALTO clients to query information such as capacity regions for a given set of flows. However, the current design of this extension has a limited expressiveness. The goal of this document is to introduce a unified resource representation service as an extension of ALTO (ALTO-UR), which allows the ALTO clients to query and get the capacity regions of more complex resource information, such as Shared-Risk-Link-Group (SRLG), multi-path routing, multicast and on-demand routing, for a given set of flows.

Status of This Memo

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

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

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 3
2. Requirements Language 3
3. Changes Since Version -02 3
4. Limitations of the ALTO Path Vector Extension 3
5. Overview of the Unified Representation Extension 5
5.1. Basic idea 5
5.2. New Cost Type to Encode Mathematical Programming Variables 6
5.2.1. Cost Mode: array 6
5.2.2. Cost Metric: variable-list 7
5.3. New Entity Domain to Provide Mathematical Programming Constraints 7
5.4. Multipart Response to Provide the Unified Representation 7
5.5. Designing New Interfaces for More Flexible Queries . . . 7
6. Example 8
6.1. Protocol Extension 8
6.2. Workflow 9
6.3. Information Resource Directory Example 9
6.4. Cost Map Service Example 10
7. Use Case: Programmable End-to-End Multi-Domain Route Control 12
8. Ongoing Design Update: A Language-Based Resource Discovery Framework 12
9. Security and Privacy Considerations 13
10. References 13
10.1. Normative References 13
10.2. Informative References 13
Authors' Addresses 14

1. Introduction

As discussed in [DRAFT-PV], the "one-big-switch" abstraction used in the ALTO base protocol lacks the ability to support emerging use cases, such as inter-datacenter data transfers, because this abstraction cannot reveal the resource sharing, i.e., the capacity region, for a set of flows. The ALTO Path Vector extension addresses this insufficiency by using the path vector abstraction to express the capacity region in a set of linear inequalities. However, in an internal discussion with the leading persons of several important ALTO use cases, it is revealed that the expressiveness of the ALTO Path Vector extension is limited in three aspects:

- It cannot provide compact encoding of the SRLG for a set of flows;
- It assumes that each flow in the client's query will use a single-path route, and hence cannot encode the resource sharing for flows that are forwarded along multi-path routes or multicast flows;
- It assumes that the route of each flow in the client's query is pre-computed, and hence cannot encode the resource sharing for flows that use on-demand routing, e.g., the path computation element (PCE) protocol.

To cope with these issues, this document introduces a new ALTO extension, the unified representation (ALTO-UR). This extension expands the linear inequality encoding of capacity regions used in ALTO-PV, to a generic, complete encoding, which uses mathematical programming constraints to represent the capacity regions for a set of flows.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Changes Since Version -02

- Add a discussion of an ongoing investigation of a language-based resource discovery framework utilizing the unified resource representation in Section 8.

4. Limitations of the ALTO Path Vector Extension

The limitations of the ALTO-PV extension are illustrated with the same dumbbell topology used in [DRAFT-PV]. Assume that the bandwidth of every link is 100 Mbps, and that the SRLG of each link is shown in

Figure 1. Consider an application overlay (e.g., a large data analytics system) which wants to schedule the traffic among a set of end host source-destination pairs, say eh1 -> eh2 and eh1 -> eh4.

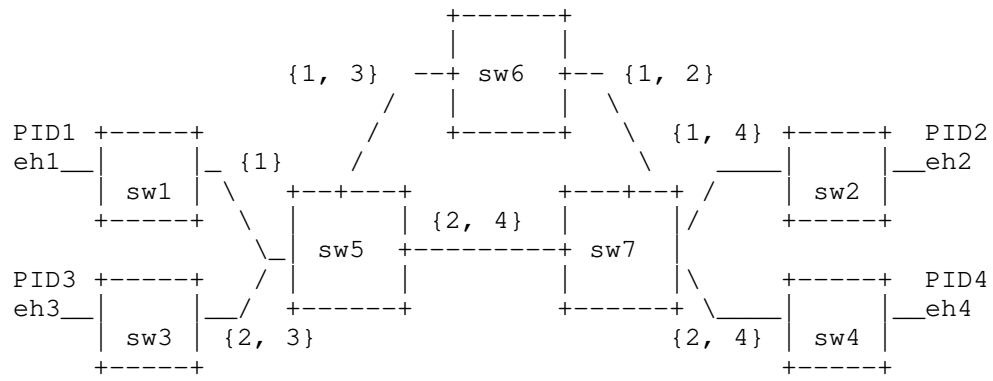

Figure 1: A Dumbbell Network Topology

- Assume the application is only interested in the SRLG of both flows, not the bandwidth. The route of eh1 -> eh2 is eh1 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh2, and the route of eh3 -> eh4 is eh3 -> sw1 -> sw5 -> sw7 -> sw2 -> eh4. The minimal yet accurate information returned to the application should be {2, 3, 4}, the SRLG of both flows, since flow 1 has a set of SRLG {1, 2, 3, 4} and flow 2 has a set of SRLG {2, 3, 4}. In contrast, in the current ALTO-PV service, the ALTO server needs to return the ane-path of each flow and the SRLG of every ane, where ane and ane-path are defined in [DRAFT-PV]. This response is redundant and causes unnecessary information exposure to the application, e.g., the information of flow has an SRLG 1 should not be returned to the application.

- Assume the application is only interested in the bandwidth of both flows. The route of eh1 -> eh2 is eh1 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh2, and the route of eh3 -> eh4 is a multi-path route, i.e., {eh3 -> sw1 -> sw5 -> sw7 -> sw2 -> eh4, eh3 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh4}, where each path would forward 50 percent of the traffic for eh3 -> eh4. The ALTO-PV service cannot reveal the traffic split of the multi-path route for eh3 -> eh4, or the bandwidth sharing for both flows on link sw5 -> sw6.

- Assume the network has a PCE sever, through which the application can reserve bandwidth for both flows. Before the application makes the reservation request, the application queries the ALTO server to get the bandwidth capacity region of both flows, which it wants to use to decide how much bandwidth to reserve for each flow. Suppose when the ALTO server receives a query, the network only has two precomputed routes for both flows: eh1 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh2, and eh3 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh4. Through the ALTO-PV service, the application receives the information that the total bandwidth it can reserve for both flows cannot exceed 100 Mbps. However, one important feature of the PCE server is that it can compute the route for reservation request on-demand, and hence it can find routes with a larger bandwidth. In this example, if the application submits a request to reserve 100 Mbps bandwidth for each flow, the PCE server can compute two on-demand routes, i.e., eh1 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh2 and eh3 -> sw1 -> sw5 -> sw7 -> sw2 -> eh4, and still return a success signal to the application. This shows that the ALTO-PV service cannot encode the capacity region for flows who use on-demand routing.

5. Overview of the Unified Representation Extension

Although different patches and extensions can be introduced to address the aforementioned insufficiencies of the ALTO-PV service, it is desirable to design a service that provides a generic solution that can encode different types of resource sharing for a set of flows. To this end, this document introduces the ALTO Unified Representation (ALTO-UR) service.

5.1. Basic idea

The basic idea of the ALTO-UR service is to use mathematical programming constraints to represent the capacity region for a set of flows. Different from linear inequalities used in the ALTO-PV service, mathematical programming constraints can represent a much wider range of resource information. To illustrate the expressiveness of mathematical programming constraints, we revisit the examples in Figure 1.

Assume the route of eh1 -> eh2 is eh1 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh2, and the route of eh3 -> eh4 is eh3 -> sw1 -> sw5 -> sw7 -> sw2 -> eh4. Denote the SRLG of flow eh1 -> eh2 as f1:SRLG, and that of flow eh3->eh4 as f2:SRLG. Then the SRLG of both flows can be represented as

f1:SRLG intersect f2:SRLG = {2, 3, 4}

Assume the route of eh1 -> eh2 is eh1 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh2, and the route of eh3 -> eh4 is a multi-path route, i.e., {eh3 -> sw1 -> sw5 -> sw7 -> sw2 -> eh4, eh3 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh4}, where each path would forward 50 percent of the traffic for eh3 -> eh4. Denote the available bandwidth of eh1 -> eh2 as f1-bw and those of eh3 -> eh4 along two paths as f2-bw-p1 and f2-bw-p2. The bandwidth sharing of two flows can be represented as:

```
f1:bw <= 100 Mbps, for (sw1, sw5), (sw7, sw2)
f2:bw:p1 + f2:bw:p2 <= 100 Mbps, for (sw3, sw5), (sw7, sw4)
f1:bw  + f2:bw:p1 <= 100 Mbps, for (sw5, sw6), (sw6, sw7)
f2:bw:p2 <= 100 Mbps, for (sw5, sw7)
f2:bw:p1 = f2:bw:p2
```

Assume the routes for both flows are computed on demand and each flow can only use a single path. For eh1 -> eh2, use f1-bw-p1 and f1-bw-p2 to represent the available bandwidth of routes eh1 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh2 and eh1 -> sw1 -> sw5 -> sw7 -> sw2 -> eh2, respectively. For eh3 -> eh4, use f2-bw-p1 and f2-bw-p2 to represent the available bandwidth of routes eh3 -> sw1 -> sw5 -> sw6 -> sw7 -> sw2 -> eh4 and eh3 -> sw1 -> sw5 -> sw7 -> sw2 -> eh4, respectively. The bandwidth capacity region of both flows can be represented as:

```
f1:bw:p1 + f1:bw:p2 <= 100 Mbps, for (sw1, sw5), (sw7, sw2)
f2:bw:p1 + f2:bw:p2 <= 100 Mbps, for (sw3, sw5), (sw7, sw4)
f1:bw:p1 + f2:bw:p1 <= 100 Mbps, for (sw5, sw6), (sw6, sw7)
f1:bw:p2 + f2:bw:p2 <= 100 Mbps, for (sw5, sw7)
f1:bw:p1 = 0 or f1:bw:p2 = 0
f2:bw:p1 = 0 or f2:bw:p2 = 0
```

The next few subsections present the approaches adopted by the ALTO unified representation extension.

5.2. New Cost Type to Encode Mathematical Programming Variables

This document introduces the unified representation cost type, with the following cost mode and cost metric.

5.2.1. Cost Mode: array

The cost mode of the notation cost type is "array", which is defined in [DRAFT-PV]. The values are arrays of JSONValue. The specific type of each element in the array depends on the cost metric.

5.2.2. Cost Metric: variable-list

This document specifies a new cost metric called "variable-list". This cost metric indicates that the cost value is a list of variables that will be used in mathematical programming constraints.

5.3. New Entity Domain to Provide Mathematical Programming Constraints

This document adopts the property map defined in [DRAFT-UP] to encode the properties of abstract network elements. A new domain "cstr" (short for constraint) is registered in the property map. Each entity in the "cstr" domain has an identifier of an CSTR. Each CSTR has one property, which represents the semantics of this constraint, e.g., a "bw-cstr" property indicates that this constraint represents the bandwidth sharing among flows. This property is provided in information resources called "Property Map Resource" and "Filtered Property Map Resource". The "Filtered Property Map" resource which supports the "cstr" domain is used to encode the properties of cstr entities, and it is called a cstr Property Map in this document.

5.4. Multipart Response to Provide the Unified Representation

To ensure the consistency between the unified representation cost map and the corresponding CSTR property map, this document adopts the design of [DRAFT-PV] to allow a response to contain both the unified representation in a filtered cost map and the associated CSTR property map.

5.5. Designing New Interfaces for More Flexible Queries

Current ALTO protocol and its extensions allow applications to query resource information by specifying IP addresses of endpoints and simple filters. However, with the emerging of new networking architecture (e.g., software defined networking and network function virtualization) and the fine-grained resource requirement of applications (e.g., link-disjoint paths and endpoint precedence), applications need a more flexible interface to specify queries of resource information.

[DRAFT-FCS] proposes a flexible flow query extension service to allow applications to specify query entities based on flexible matching conditions (e.g., TCP/IP 5-tuple) instead of IP addresses only. However, it still does not allow an application to specify more fine-grained resource requirements. For example, a multi-domain service function chaining application may want to get the endpoint cost information of a chain of endpoints in the order of firewall, load balancer and data analyzer. The endpoint cost information of a chain

of endpoints not in this order is of no interest to the application and should not be returned to the application.

As such, this document makes an initial proposal to design new interfaces for application to express fine-grained requirements of resource in the query. In addition to allowing the ALTO client to specify endpoints using flexible matching conditions (e.g., TCP/IP 5 tuple), one key idea in this proposal is to use a resource filter design

Specifically, two types of resource properties are defined. The first type is value property, which are typical quality of services metrics for a given flow. Examples of the value property include the bandwidth, delay, loss rate and so on. The second type is set property. Examples of such a property include the forwarding and processing devices used by a flow (denoted as nodes), the physical links used by a flow (denoted as links) and the shared risk link group (SRLG) of all the devices and links used by a flow.

With these two types of resource properties, the atomic resource requirement predicates in resource filters are the comparison expressions on value properties and set properties. Resource requirement predicates can be composited using conjunction, disjunction and negation. The ALTO client can send resource query with composed resource requirement predicates to ALTO server, which only returns the information of resources that satisfy such predicates to ALTO client.

The details of this new interface will be fully specified in the next version of this document.

6. Example

6.1. Protocol Extension

To allow the ALTO client to query and receive the mathematical programming constraints for a set of flows, the Filtered Cost Map and Endpoint Cost Service of the ALTO protocol need to be extended. The current design adopted in this document uses a similar approach as the ALTO-PV extension does in [DRAFT-PV]: (1) extending the FilteredCostMapCapabilities object with a new member "property-map" and (2) using a multipart service to send both the unified representation cost map and the CSTR property map together. This design is illustrated in the next few subsections.

However, for the ALTO-UR service, this is still an early stage design. As a major next step for ALTO-UR service, other design options are being investigated with the aim of enabling better

modularity and extensibility, and the protocol extension will be updated in the next version accordingly.

6.2. Workflow

A typical workflow of an ALTO client using the unified representation extension is as follows:

1. Send a GET request for the whole Information Resource Directory.

2. Look for the resource of the Cost Map Service which contains the unified representation cost type and get the resource ID of the dependent cstr property map.

3. Check whether the capabilities of the property map includes the desired "prop-types".

4. Send a unified-representation request which accepts "multipart/related" media type following "application/alto-costmap+json" or "application/endpointcost+json"

6.3. Information Resource Directory Example

An example of an Information Resource Directory is as follows. In this example, filtered cost map "cost-map-ur" supports the unified-representation extension. The property map "propmap-cstr" support two properties, "bw-cstr" and "srlg-cstr", representing bandwidth constraint and SRLG constraint, respectively.

```
{
  "meta": {
    "cost-types": {
      "ur": {
        "cost-mode": "array",
        "cost-metric": "variable-list"
        }
    }
    "resources": {
      "my-default-networkmap": {
        "uri" : "http://alto.example.com/networkmap",
        "media-type" : "application/alto-networkmap+json"
        }
      "cost-map-ur" : {
        "uri": "http://alto.example.com/costmap/ur",
        "media-type": "application/alto-costmap+json",
        "accepts": "application/alto-costmapfilter+json",
        "capabilities": {
          "cost-type-names": [ "ur"]
        },
        "property-map": "propmap-cstr",
        "uses": [ "my-default-networkmap" ]
      },
      "propmap-cstr" : {
        "uri": "http://alto.exmaple.com/propmap/cstr",
        "media-type": "application/alto-propmap+json",
        "accepts": "application/alto-propmapparams+json",
        "capabilities": {
          "domain-types": [ "cstr" ],
          "prop-types": [ "bw-cstr", "srlg-cstr" ]
        }
      }
    }
}
```

6.4. Cost Map Service Example

The following is an example of the cost map service in the ALTO-UR extension. In the returned cost map, flow 1 has two bandwidth variables, f1:bw:p1 and f2:bw:p2, and one SRLG variable, f1:srlg. And flow 2 has one bandwidth variable, f2:bw, and one SRLG variable, f2:srlg. Four mathematical programming constraints are returned in the property map. The first three are bandwidth sharing constraints, and the fourth is an SRLG constraint.

```
POST /costmap/pv HTTP/1.1
Host: alto.example.com
```

```
Accept: multipart/related, application/alto-costmap+json,
application/alto-propmap+json, application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-costmapfilter+json
{
  "cost-type": {
    "cost-mode": "array",
    "cost-metric": "variable-list"
  },
  "pids": {
    "srcs": [ "PID1" ],
    "dsts": [ "PID2", "PID3" ]
  }
}

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=42

--42
Content-Type: application/alto-costmap+json

{
  "meta": {
    "dependent-vtags": [
      {
        "resource-id": "default-network-map",
        "tag": "75ed013b3cb58f896e839582504f622838ce670f"
      }
    ],
    "cost-type": {
      "cost-mode": "array",
      "cost-metric": "variable-list"
    },
  },

  "cost-map": {
    "PID1": {
      "PID2": [ "f1:bw:p1", "bw:p2", "f1:srlg"]
      "PID3": [ "f2:bw:p1", "f2:srlg" ]
    }
  }
}

--42
Content-Type: application/alto-propmap+json

{
```

```
   "property-map": {
     "cstr:001": { "bw-cstr": "[0][0] add [0][1] leq 100"},
     "cstr:002": { "bw-cstr": "[0][0] eq [0][1]"},
     "cstr:003": { "bw-cstr": "[1][0] add [0][0] leq 100"},
     "cstr:004": { "srlg-cstr": "[0][2] intersect [1][1] eq {2, 3, 4}"},
   }
 }
```

7. Use Case: Programmable End-to-End Multi-Domain Route Control

This section describes a multi-domian use case that can benefit from a unified resource presentation extension of ALTO. Specifically, a novel multi-domain network programming framework is recently proposed to achieve programmable, end-to-end interdomain route control. Specifically, in this framework, each autonomus system (AS) deploys an ALTO server to provide a programming abstraction. Collectively, these ALTO servers provide the client a single, abstract, programmable network spanning multiple individual networks. Unlike existing SDN abstractions (e.g., OpenFlow and P4), it includes a built-in layer to extract and learn the interactions of interdomain policies of individual networks (e.g., route selection preference), providing a unified abstraction.

In particular, given a destination IP prefix p specified by a client, each autonomous system (AS) exposes its routing information base (RIB), i.e., all available routes it has to reach p, and the corresponding price for the client to use each route, by deploying an ALTO server. A client queries the available routes and the corresponding prices at different ASes. With the retrieved information, it then itereatively selects the best route based on its internal objective function and budget, and interacts with different ASes to check if the selected route is policy compliant, i.e., whether all ASes along this route will export the preceding segment to its upstream neighbor. After finding the best policy compliant route, the client then interacts with ASes of the route in a backward order along the route to setup the AS path. A blackbox based optimization algorithm has been developed to allow the client to swiftly find the best policy compliant route. A paper describing this framework has been accepted by IEEE INFOCOM 2020.

8. Ongoing Design Update: A Language-Based Resource Discovery Framework

To further investigate the feasibility, challenges and benefits of using a unified resource representation to facilitate application-network integration, the authors of this draft are investigating a language-based resource discovery framework. This framework consists of three key design points. First, this framework uses generic

mathematical programming constraints as a unified, compact representation of network information, as proposed in this draft. Second, it utilizes the equivalence between relational algebra and first order logic to provide a SQL-style language for application / ALTO client to express their intents on discovering resources in the network. Third, the framework develops a compiler to translate an application's resource discovery intent into a constraint programming problem with a set of logical constraints, whose feasible solutions correspond to qualified resources for application. In addition, an algorithm is also designed to improve the network's efficiency in finding qualified resources. Details of this framework and its early evaluation results have been accepted by ACM SIGCOMM NAI 2020 Workshop, with title ""Towards Deep Network & Application Integration: Possibilities, Challenges, and Research Directions".

9. Security and Privacy Considerations

The unified representation extension may expose more private information to applications than the ALTO base protocol does. However, as shown in the motivating example of providing SRLG information for a set of flows, this extension has the capability of exposing less private information than the ALTO-PV extension does, while having a better expressiveness on providing fine-grained resource information to applications. A systematic study on the security and privacy issues of the ALTO-UR extension is one of the major next steps.

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

10.2. Informative References

[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-NFCHAIN]
Perez, D. and C. Rothenberg, "ALTO-based Broker-assisted Multi-domain Orchestration", 2018, <https://datatracker.ietf.org/doc/html/draft-lachosrothenberg-alto-brokermdo-01>.

[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-UP]
Roome, W., Chen, S., Randriamasy, S., Yang, Y., and J. Zhang, "Unified Properties for the ALTO Protocol", 2015, <https://datatracker.ietf.org/doc/draft-ietf-alto-unified-props-new/>.

[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>.

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

ALTO WG
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

S. Chen
J. Zhang
Tongji University
July 02, 2018

Multiple ALTO Resources Query Using Multipart Message
draft-zhang-alto-multipart-00

Abstract

Many ALTO use cases involve multiple ALTO information resources like network map, cost map and property map to achieve their own goal. To make the ALTO client query them one by one is not only inefficient but also possible to introduce inconsistent issues. Further more, some ALTO information resources may have correlation, which means one's input parameters may depends on another one's response. So some advanced query schema is required. This document proposes an extension to support the multiple ALTO resources query with HTTP multipart message and the existing JSON query languages.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
1.1. Background 3
1.2. Requirements Language 3
2. Terminologies 4
3. Use Cases 4
3.1. Simple Batch Query 4
3.2. Properties Constrained Query 4
3.3. Path Vector Query 5
4. Requirements 5
5. Overview of Approach 6
6. Multipart Query Resource 6
6.1. Media Type 6
6.2. HTTP Method 6
6.3. Capabilities 6
6.4. Accept Input Parameters 6
6.5. Uses 7
6.6. Response 7
7. Protocol Errors 7
8. Examples 8
8.1. IRD Example 8
8.2. Example 1: Simple Batch Query 10
8.3. Example 2: Properties Constrained Query 11
8.4. Example 3: Path Vector Query 14
9. Compatibility 15
9.1. Compatibility with Legacy ALTO Clients/Servers 15
9.2. Compatibility with Existing Protocol Extensions 16
9.3. Compatibility with New Communication Mechanism 16
10. Misc Considerations 16
10.1. Support Incremental Update 16
10.2. Anonymous Resources 16
11. Security Considerations 16
12. IANA Considerations 16
12.1. application/alto-* Media Types 16
13. Acknowledgements 18
14. References 18
14.1. Normative References 18
14.2. Informative References 18
Appendix A. Figures 19
Authors' Addresses 19

1. Introduction

1.1. Background

Application-Layer Traffic Optimization (ALTO) protocol [RFC7285] and its extensions already define several kinds of information resources, like network map, cost map, property map, to expose useful network informations to applications. However, many applications cannot only use a single information resource to achieve their optimization goal. Retrieving multiple ALTO information resources is very common in many ALTO use cases.

Although the ALTO client can query multiple ALTO information resources one by one, it is not inefficient. And because the network delay between different requests and the frequent change of ALTO information resources, the responses may be inconsistent.

Further more, some ALTO information resources have known dependencies, which means the ALTO client may need one’s response to decide another one’s query input parameters.

To be summarized, we need the multipart query service for three reasons:

- Clients may want to query multiple ALTO information resources in a single request to reduce the time consumption.
- Clients may want to query multiple ALTO resources consistently, which means the server should guarantee the responses of all resources are generated at the same time.
- Some use cases need to query multiple ALTO resources with a joint relationship.

This document defines a new ALTO services for: (1) querying multiple ALTO resources in a single request/response, and (2) supporting general-purpose JSON query languages to resolve the relational query.

1.2. 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].

2. Terminologies

TBD.

3. Use Cases

We take the following potential use cases which may benefit from the multipart query service.

3.1. Simple Batch Query

The simplest use case is to query a batch of ALTO resources in a single request.

Although the ALTO client can perform ALTO requests for multiple times, it is not only inefficient but also inconsistent.

For example, the ALTO server provides a network map resource A and a dependent cost map resource B. Both resources may change frequently. Assume the ALTO client queries the network map first, and it gets the revision A1. When the client queries the cost map, the network map may be already changed from A1 to A2, and the client receives cost map B2 which depends on A2 not A1. So the responded cost map B2 is not consistent with the previous network map A1.

This case requires the ALTO server to provide a way for the ALTO client to query multiple ALTO resources in a single transaction.

3.2. Properties Constrained Query

Beyond the simple batch query, there are also some another use cases requiring a new service for relational query. For example, Some clients may need to query an endpoint property map first, and find endpoints with some properties fitting some conditions. And then they query the endpoint cost of these endpoints.

In this case, the endpoint cost query depends on the result of the property map query. Although the ALTO client can cache the whole property map in its local storage, it is still not efficient and may conduct the consistency issue if the property map changes frequently. So it requires a new service to provide multiple dependent resources efficiently and consistently.

A general multipart query service benefits the ALTO client in two aspects:

- It allows the ALTO client to specify the boolean test to reduce the transmission of the useless data from the ALTO server.

- It compounds multiple ALTO information resources in a single response to reduce the communication times. Thus, the transmission latency can be reduced.

3.3. Path Vector Query

Another use case requiring the multiple resource query is the relational query between the on-demand generated resources. A straightforward example is the path vector query demonstrated in [I-D.ietf-alto-path-vector].

[I-D.ietf-alto-path-vector] introduces an extension of ALTO to provide path vector information by cost map and unified property map [I-D.ietf-alto-unified-props-new]. The client using path vector extension will usually query cost map and a dynamically generated property map sequentially. It is even hard to cache the full data of resources, because both the cost map and the property map are on-demand generated by the query input here. Thus, the only way to reduce the time consumption is to compound the two resources.

4. Requirements

From the use cases described in Section 3, there are three additional requirements for ALTO protocol:

MPQ-Req1: The ALTO protocol SHOULD allow the client to query multiple ALTO resources in a single request, and return the result in a single response.

It is the basic requirement to provide the query for the compound resources. Even simple cases can benefit if this requirement is realized.

MPQ-Req2: The ALTO protocol SHOULD provide general filter schema for any ALTO resources.

Current filter schema in ALTO protocol only supports the simple boolean test of numerical comparison. And the boolean filtered query is only supported by the cost map and the endpoint cost resource. It is not enough for the general cases. Even simple property map may require more general filter schema.

MPQ-Req3: The ALTO protocol SHOULD support relational query for mulitple joint resources.

Some ALTO resources are relational and cannot be used individually. The path vector query is such an example. In these

use cases, the support of relational query for multiple joint resources is very helpful.

5. Overview of Approach

This document uses two key techniques to realize the general multiple resources query:

- Use Multipart message [RFC2046] to deliver compound resources.
- Accept JSON Query Language like XQuery [W3CXQUERY] and JSONiq [JSONIQ] for general query process and relational joint query.

6. Multipart Query Resource

6.1. Media Type

"multipart/related" [RFC2387].

6.2. HTTP Method

An ALTO Multipart Query resource is requested using the HTTP POST method.

6.3. Capabilities

The capabilities are defined by an object of type MultipartQueryCapabilities:

```
object {
  JSONString query-langs<0..*>;
} MultipartQueryCapabilities;
```

where "query-langs" is an array of JSONString to indicate which query languages are supported by this resource.

6.4. Accept Input Parameters

The input parameters for a Multipart Query request are supplied in the entity body of the POST request. This document specifies the input parameters with a data format indicated by the media type "application/alto-multipartquery+alto", which is a JSON object of type ReqMultipartQuery:

```
object {
  ResourceQuery   resources<1..*>;
  [JsonString     query-lang;]
} ReqMultipartQuery;

object {
  JsonString      resource-id;
  [JsonValue      input;]
} ResourceQuery;
```

with fields:

resources: List of ResourceQuery objects for which resources are to be queried and how to query them. Each ResourceQuery object MUST include the "resource-id" field to indicate which resource is to be queried. If the queried resource requires the POST method, the "input" field MUST be specified. The value of the "input" field MUST be either a JSONString or a JSONObject. When its value is a JSONObject, its format MUST be as the accept input parameters of its resource. When its value is a JSONString, it MUST be a program written in the query language specified by the "query-lang" field.

query-lang: Optional. The value of the "query-lang" field MUST be one of values in the "query-langs" capability. If this field is not specified in the request, the ALTO client SHOULD NOT use any query language in the "input" field.

6.5. Uses

An array with the resource ID(s) of resource(s) which this multipart query resource can compound. The used resource can be any available ALTO resources except for the multipart query resource. If the "uses" field is not specified, all the available ALTO resources can be queried except for the multipart query resource.

6.6. Response

The response of multipart query resource is a multipart message. Each part of this multipart message is the response of a queried resource in the request.

7. Protocol Errors

The validation of each "input" field of the multipart query input parameters depends on the queried resource:

- If the "input" field of the multipart query input parameters is neither a JSONObject nor a JSONString, the ALTO server SHOULD return the E_INVALID_FIELD_TYPE error, unless a future protocol extension supports the non-JSONObject input parameters.

- If the "input" field of the multipart query input parameters is a JSONObject, the ALTO server MUST validate the value using its queried resource and return the corresponding error if it has.

- If the "input" field of the multipart query input parameters is a JSONString:

  * If the "query-lang" is not specified, the ALTO server MUST return the E_INVALID_FIELD_TYPE error.

  * If the "query-lang" is specified, the ALTO server MUST execute this JSONString as a program written in the "query-lang". If the execution failed, the ALTO server MUST return the E_INVALID_FIELD_VALUE error. If the execution succeed but the result fails to pass the validation of the queried resource, the ALTO server MUST return the E_INVALID_FIELD_VALUE error and attach the error message returned by the queried resource into the "message" field of the ALTO error message.

8. Examples

8.1. IRD Example

Assume the root IRD is like the following:

```
{
  "meta": {
      "path-vector": {
        "cost-mode": "array",
        "cost-metric": "ane-path"
      },
      "num-routingcost": {
        "cost-mode": "numerical",
        "cost-metric": "routingcost"
      },
      "num-hopcount": {
        "cost-mode": "numerical",
        "cost-metric": "hopcount"
      }
  },
  "resources": {
    "my-default-networkmap": {
      "uri": "http://alto.example.com/networkmap",
```

```
      "media-type": "application/alto-networkmap+json"
    },
    "my-default-costmap": {
      "uri": "http://alto.example.com/costmap",
      "media-type": "application/alto-costmap+json",
      "capabilities": {
        "cost-type-names": [ "num-routingcost" ]
      },
      "uses": [ "my-default-networkmap" ]
    },
    "my-filtered-costmap": {
      "uri": "http://alto.example.com/costmap/filtered",
      "media-type": "application/alto-costmap+json",
      "accepts": "application/alto-costmapfilter+json",
      "capabilities": {
        "cost-type-names": [ "num-hopcount" ]
      },
      "uses": [ "my-default-networkmap" ]
    },
    "endpoint-path-vector": {
      "uri": "http://alto.exmaple.com/endpointcost",
      "media-type": "application/alto-endpointcost+json",
      "accepts": "application/alto-endpointcostparams+json",
      "capabilities": {
        "cost-constraints": true,
        "cost-type-names": [ "path-vector" ],
      },
      "property-map": "propmap-availbw"
    },
    "propmap-availbw-delay": {
      "uri": "http://alto.exmaple.com/propmap/availbw",
      "media-type": "application/alto-propmap+json",
      "accepts": "application/alto-propmapparams+json",
      "capabilities": {
        "domain-types": [ "ane" ],
        "prop-types": [ "availbw" ]
      }
    },
    "propmap-location": {
      "uri": "http://alto.exmaple.com/propmap/location",
      "media-type": "application/alto-propmap+json",
      "accepts": "application/alto-propmapparams+json",
      "capabilities": {
        "domain-types": [ "pid" ],
        "prop-types": [ "country", "state" ]
      }
    },
    "multipart-query": {
```

```
          "uri": "http://alto.example.com/multipart",
          "media-type": "multipart/related",
          "accepts": "application/alto-multipartquery+json",
          "capabilities": {
            "query-langs": [ "xquery", "jsoniq" ]
          }
        }
      }
    }
```

8.2. Example 1: Simple Batch Query

```
POST /multipart HTTP/1.1
Host: alto.example.com
Accept: multipart/related, application/alto-error+json
Content-Lenght: [TBD]
Content-Type: application/alto-multipartquery+json

{
  "resources": [
    {
      "resource-id": "my-default-networkmap"
    },
    {
      "resource-id": "my-default-costmap"
    }
  ]
}

HTTP/1.1 200 OK
Content-Lenght: [TBD]
Content-Type: multipart/related; boundary=simple-batch-query

--simple-batch-query
Content-Type: application/alto-networkmap+json

{
  "meta": {
    "vtag": {
      "resource-id": "my-default-networkmap",
      "tag": "75ed013b3cb58f896e839582504f622838ce670f"
    }
  },
  "network-map": {
    "PID1" : {
      "ipv4" : [
        "192.0.2.0/24",
        "198.51.100.0/25"
```

```
        ]
      },
      "PID2" : {
        "ipv4" : [
          "198.51.100.128/25"
        ]
      },
      "PID3" : {
        "ipv4" : [
          "0.0.0.0/0"
        ],
        "ipv6" : [
          "::/0"
        ]
      }
    }
  }

  --simple-batch-query
  Content-Type: application/alto-costmap+json

  {
    "meta": {
      "dependent-vtags": [
        {
          "resource-id": "my-default-networkmap",
          "tag": "75ed013b3cb58f896e839582504f622838ce670f"
        }
      ],
      "cost-type": {
        "cost-mode": "numerical",
        "cost-metric": "routingcost"
      }
    },
    "cost-map": {
      "PID1": { "PID1": 1,  "PID2": 5,  "PID3": 10 },
      "PID2": { "PID1": 5,  "PID2": 1,  "PID3": 15 },
      "PID3": { "PID1": 20, "PID2": 15  }
    }
  }
```

8.3. Example 2: Properties Constrained Query

NOTE: In this example, we use the "‘" block to express the raw string with unescaped characters like "\n" and "\"". It is not valid HTTP body, but only used to better present. When the request is sent to the ALTO server, the "‘" block should be escaped.

```
POST /multipart HTTP/1.1
Host: alto.example.com
Accept: multipart/related, application/alto-error+json
Content-Lenght: [TBD]
Content-Type: application/alto-multipartquery+json

{
  "query-lang": "jsoniq",
  "resources": [
    {
      "resource-id": "propmap-location"
    },
    {
      "resource-id": "my-default-costmap",
      "input": '
        let $propmap := collection("propmap-location")
                        .("property-map")
        return {
          "cost-type": {
            "cost-mode": "numerical",
            "cost-metric": "hopcount"
          },
          "pids": {
            "srcs": [
              for $pid in keys($propmap)
              where $propmap.$pid.country eq "US"
              return substring-after($pid, "PID:")
            ],
            "dsts": [
              for $pid in keys($propmap)
              where $propmap.$pid.country eq "CA"
              return substring-after($pid, "PID:")
            ]
          }
        }
      '
    }
  ]
}

HTTP/1.1 200 OK
Content-Lenght: [TBD]
Content-Type: multipart/related; boundary=prop-const-query

--prop-const-query
Content-Type: application/alto-propmap+json

{
```

```
    "property-map": {
      "pid:PID1": {
        "country": "US",
        "state": "CA"
      },
      "pid:PID2": {
        "country": "US",
        "state": "CT"
      },
      "pid:PID3": {
        "country": "CA",
        "state": "QC"
      },
      "pid:PID4": {
        "country": "CA",
        "state": "NT"
      },
      "pid:PID5": {
        "country": "FR"
      }
    }
  }

  --prop-const-query
  Content-Type: application/alto-costmap+json

  {
    "meta": {
      "cost-type": {
        "cost-mode": "numerical",
        "cost-metric": "hopcount"
      }
    },
    "cost-map": {
      "PID1": {
        "PID3": 5,
        "PID4": 7
      },
      "PID2": {
        "PID3": 8,
        "PID4": 4
      }
    }
  }
```

8.4. Example 3: Path Vector Query

```
POST /multipart HTTP/1.1
Host: alto.example.com
Accept: multipart/related, application/alto-error+json
Content-Lenght: [TBD]
Content-Type: application/alto-multipartquery+json

{
  "query-lang": "jsoniq",
  "resources": [
    {
      "resource-id": "endpoint-path-vector",
      "input": {
        "cost-type": {
          "cost-mode": "array",
          "cost-metric": "ane-path"
        },
        "endpoints": {
          "srcs": [ "ipv4:192.0.2.2" ],
          "dsts": [ "ipv4:192.0.2.89",
                    "ipv4:203.0.113.45" ]
        }
      }
    },
    {
      "resource-id": "propmap-availbw",
      "input": '
        let $propmap := collection("endpiont-path-vector")
                          .("endpoint-cost-map")
        return {
          "entities": [
            distinct-values(flatten(
              for $src in keys($propmap)
              let $dsts := $propmap.$src
              return flatten(
                for $dst in keys($dsts)
                return $dsts.$dst
              )
            ))
          ],
          "properties": [ "availbw" ]
        }
      '
    }
  ]
}
```

```
HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=path-vector-query

--path-vector-query
Content-Type: application/alto-endpointcost+json

{
  "meta": {
    "cost-type": {
      "cost-mode": "array",
      "cost-metric": "ane-path"
    }
  },
  "endpoint-cost-map": {
    "ipv4:192.0.2.2": {
      "ipv4:192.0.2.89":   [ "ane:L001", "ane:L003", "ane:L004" ],
      "ipv4:203.0.113.45": [ "ane:L001", "ane:L004", "ane:L005" ],
      "ipv6:2001:db8::10": [ "ane:L001", "ane:L005", "ane:L007" ]
    }
  }
}

--path-vector-query
Content-Type: application/alto-propmap+json

{
  "property-map": {
    "ane:L001": { "availbw": 50 },
    "ane:L003": { "availbw": 48 },
    "ane:L004": { "availbw": 55 },
    "ane:L005": { "availbw": 60 },
    "ane:L007": { "availbw": 35 }
  }
}
```

9. Compatibility

9.1. Compatibility with Legacy ALTO Clients/Servers

The multipart query service is a new ALTO service using the new media type. So the legacy ALTO client cannot identify this service from the IRD of the ALTO server supporting it. And the legacy ALTO server also cannot interpret the request of a multipart query service sent by the ALTO client.

9.2. Compatibility with Existing Protocol Extensions

The multipart query service can use any ALTO resources exchanging JSON data in request/response mechanism. So all the known ALTO extensions like ALTO Calendar [I-D.ietf-alto-cost-calendar], Multi-Cost [RFC8189] and the Path Vector [I-D.ietf-alto-path-vector] extension, which does not change the request/response mechanism, are compatible with the multipart query service.

9.3. Compatibility with New Communication Mechanism

Since the multipart query service use multipart messages as the response instead of the JSON data, the incremental update service defined in [I-D.ietf-alto-incr-update-sse] does not support it. If the update service does not notify the incremental change to the ALTO client but only notify the full replacement, it can still work. But it is very inefficient. So an extension to integrate multipart query and the incremental update smoothly is required. HTTP/2 may be a candidate solution to this problem.

10. Misc Considerations

10.1. Support Incremental Update

Because the response body entry of the multipart query resource is not a single JSON object, it may not be compatible with the existing incremental update representation.

10.2. Anonymous Resources

Some use cases may need the server generates "anonymous" ALTO resources for the on-demand information. The "anonymous" ALTO resources usually cannot appear alone but need to bind with some "non-anonymous" ALTO resources.

11. Security Considerations

Allow the ALTO clients to upload the query language script may not be safe. The script injection and many potential attacks can be conducted. The security issue should be discussed and considered.

12. IANA Considerations

12.1. application/alto-* Media Types

This document registers an additional ALTO media type, listed in Table 1.

Type	Subtype	Specification
application	alto-multipartquery+json	Section 6.4

Table 1: Additional ALTO Media Type.

Type name: application

Subtype name: This document registers multiple subtypes, as listed in Table 1.

Required parameters: n/a

Optional parameters: n/a

Encoding considerations: Encoding considerations are identical to those specified for the "application/json" media type. See [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).

13. Acknowledgements

14. References

14.1. 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>.

[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>.

14.2. 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-06 (work in progress), July 2018.

[I-D.ietf-alto-incr-update-sse] Roome, W., Yang, Y., and S. Chen, "ALTO Incremental Updates Using Server-Sent Events (SSE)", draft-ietf-alto-incr-update-sse-11 (work in progress), June 2018.

[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 Type", draft-ietf-alto-path-vector-03 (work in progress), March 2018.

[I-D.ietf-alto-unified-props-new]
Roome, W., Chen, S., Randriamasy, S., Yang, Y., and J. Zhang, "Unified Properties for the ALTO Protocol", draft-ietf-alto-unified-props-new-04 (work in progress), June 2018.

[JSONIQ] Robie, J., Fourny, G., Brantner, M., Florescu, D., Westmann, T., and M. Zaharioudakis, "JSONiq - the SQL of NoSQL 1.0", JSONiq , 2015.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[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>.

[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>.

[W3CXQUERY]
Robie, J., Chamberlin, D., Dyck, M., and J. Snelson, "XQuery 3.0: An XML query language", W3C Recommendation, W3C, 2014.

Appendix A. Figures

TODO: Put additional figures here if we have.

Authors' Addresses

Shiwei Dawn Chen
Tongji University
4800 Cao'An Hwy
Shanghai 201804
China

Email: dawn_chen_f@hotmail.com

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

Email: jingxuan.n.zhang@gmail.com