ALTO WG
Internet-Draft
Intended status: Standards Track
Expires: January 13, 2011

R. Alimi, Ed.
Yale University
R. Penno, Ed.
Juniper Networks
Y. Yang, Ed.
Yale University

July 12, 2010

ALTO Protocol
draft-ietf-alto-protocol-05.txt

Abstract

Networking applications today already have access to a great amount of Inter-Provider network topology information. For example, views of the Internet routing table are easily available at looking glass servers and entirely practical to be downloaded by clients. What is missing is knowledge of the underlying network topology from the ISP or Content Provider (henceforth referred as Provider) point of view. In other words, what a Provider prefers in terms of traffic optimization -- and a way to distribute it.

The ALTO Service provides information such as preferences of network resources with the goal of modifying network resource consumption patterns while maintaining or improving application performance. This document describes a protocol implementing the ALTO Service. While such service would primarily be provided by the network (i.e., the ISP), content providers and third parties could also operate this service. Applications that could use this service are those that have a choice in connection endpoints. Examples of such applications are peer-to-peer (P2P) and content delivery networks.

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 [1].

Status of this Memo

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

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

This Internet-Draft will expire on January 13, 2011.

Copyright Notice

Table of Contents

1. Introduction

1.1. Background and Problem Statement

Today, network information available to applications is mostly from the view of endhosts. There is no clear mechanism to convey information about the network's preferences to applications. By leveraging better network-provided information, applications have the potential to become more network-efficient (e.g., reduce network resource consumption) and achieve better application performance (e.g., accelerated download rate). The ALTO Service intends to provide a simple way to convey network information to applications.

The goal of this document is to specify a simple and unified protocol that meets the ALTO requirements [8] while providing a migration path for Internet Service Providers (ISP), Content Providers, and clients that have deployed protocols with similar intentions (see below). This document is a work in progress and will be updated with further developments.

1.2. Design History and Merged Proposals

The protocol specified here consists of contributions from

- P4P [9], [10];
- ALTO Info-Export [11];
- Query/Response [12], [13];
- ATTP [ATTP];
- Proxidor [14].

See Appendix A for a list of people that have contributed significantly to this effort and the projects and proposals listed above.

1.3. Solution Benefits

The ALTO Service offers many benefits to both end-users (consumers of the service) and Internet Service Providers (providers of the service).

1.3.1. Service Providers

The ALTO Service enables ISPs to influence the peer selection process in distributed applications in order to increase locality of traffic,

improve user-experience, amongst others. It also helps ISPs to efficiently engineer traffic that traverses more expensive links such as transit and backup links, thus allowing a better provisioning of the networking infrastructure.

1.3.2. Applications

Applications that use the ALTO Service can benefit in multiple ways. For example, they may no longer need to infer topology information, and some applications can reduce reliance on measuring path performance metrics themselves. They can take advantage of the ISP's knowledge to avoid bottlenecks and boost performance.

An example type of application is a Peer-to-Peer overlay where peer selection can be improved by including ALTO information in the selection process.

2. Architecture

Two key design objectives of the ALTO Protocol are simplicity and extensibility. At the same time, it introduces additional techniques to address potential scalability and privacy issues. After an introduction to the terminology, the ALTO architecture and the ALTO Protocol's place in the overall architecture are defined.

2.1. Terminology

We use the following terms defined in [15]: Application, Overlay Network, Peer, Resource, Resource Identifier, Resource Provider, Resource Consumer, Resource Directory, Transport Address, Host Location Attribute, ALTO Service, ALTO Server, ALTO Client, ALTO Query, ALTO Reply, ALTO Transaction, Local Traffic, Peering Traffic, Transit Traffic.

We also use the following additional terms: Endpoint Address, ASN, and Network Location.

2.1.1. Endpoint Address

An endpoint address represents the communication address of an endpoint. An endpoint address can be network-attachment based (IP address) or network-attachment agnostic. Common forms of endpoint addresses include IP address, MAC address, overlay ID, and phone number.

2.1.2. ASN

An Autonomous System Number.

2.1.3. Network Location

Network Location is a generic term denoting a single endpoint or group of endpoints.

2.1.4. ALTO Information

ALTO Information is a generic term referring to the network information sent by an ALTO Server.

2.1.5. ALTO Information Base

Internal representation of the ALTO Information maintained by the ALTO Server. Note that the structure of this internal representation is not defined by this document.

2.2. ALTO Service and Protocol Scope

An ALTO Server conveys the network information from the perspective of a network region; the ALTO Server presents its "my-Internet View" [16] of the network region. A network region in this context can be an Autonomous System, an ISP, or perhaps a smaller region or set of ISPs; the details depend on the deployment scenario and discovery mechanism.

To better understand the ALTO Service and the role of the ALTO Protocol, we show in Figure 1 the overall system architecture. In this architecture, an ALTO Server prepares ALTO Information; an ALTO Client uses ALTO Service Discovery to identify an appropriate ALTO Server; and the ALTO Client requests available ALTO Information from the ALTO Server using the ALTO Protocol.

The ALTO Information provided by the ALTO Server can be updated dynamically based on network conditions, or can be seen as a policy which is updated at a larger time-scale.

More specifically, the ALTO Information provided by an ALTO Server may be influenced (at the operator's discretion) by other systems. Examples include (but are not limited to) static network configuration databases, dynamic network information, routing protocols, provisioning policies, and interfaces to outside parties. These components are shown in the figure for completeness but outside the scope of this specification.

Note that it may also be possible for ALTO Servers to exchange network information with other ALTO Servers (either within the same administrative domain or another administrative domain with the consent of both parties) in order to adjust exported ALTO information. Such a protocol is also outside the scope of this specification.

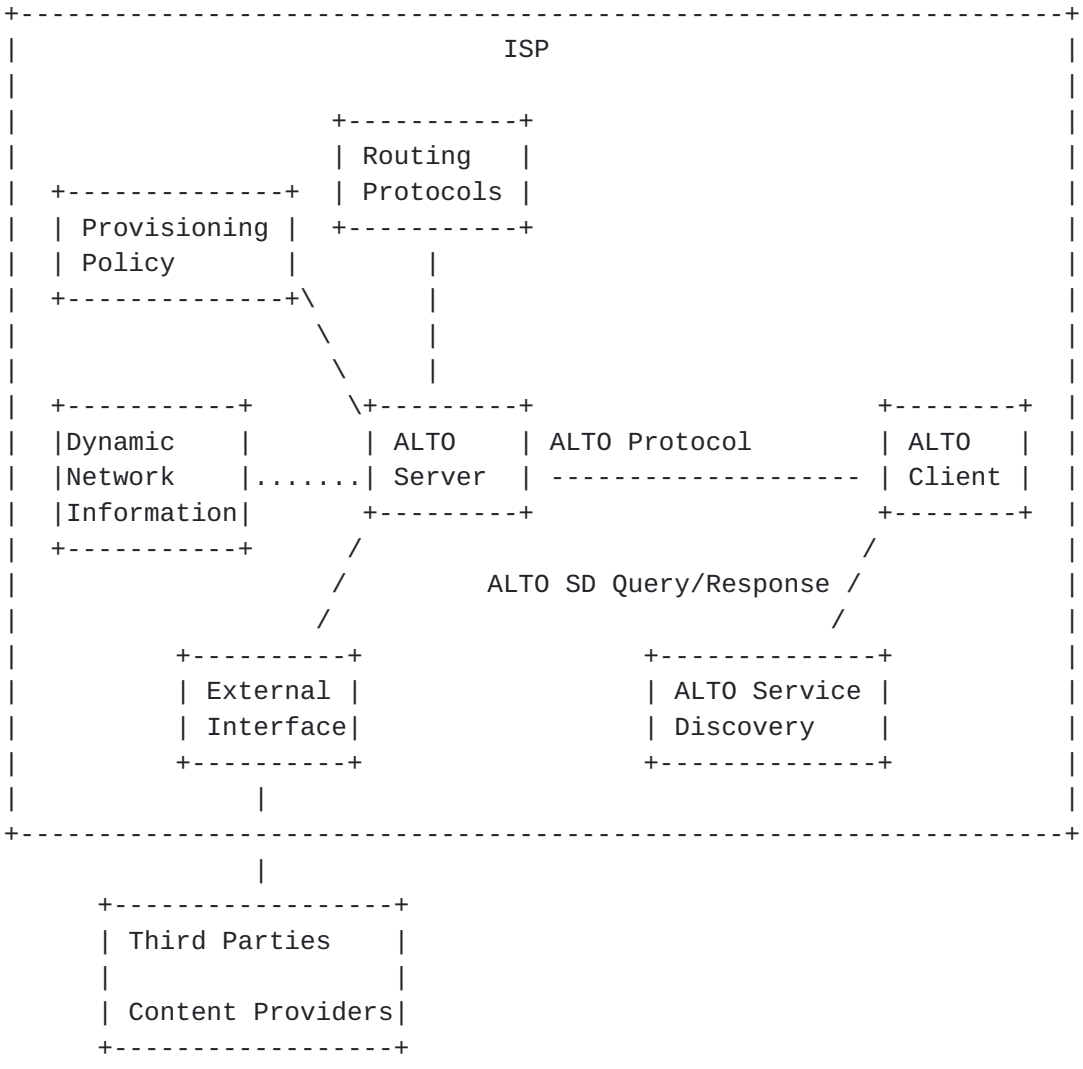

Figure 1: Basic ALTO Architecture

3. Protocol Structure

The ALTO Protocol uses a simple extensible framework to convey network information. In the general framework, the ALTO protocol will convey properties on both Network Locations and the paths between Network Locations.

In this document, we focus on a particular Endpoint property to denote the location of an endpoint, and provider-defined costs for paths between pairs of Network Locations.

The ALTO Protocol is built on a common transport protocol, messaging structure and encoding, and transaction model. The protocol is subdivided into services of related functionality. ALTO-Core provides the Server Information Service and the Map Service to provide ALTO Information. Other ALTO Information services can provide additional functionality. There are three such services defined in this document: the Map Filtering Service, Endpoint Property Service, and Endpoint Cost Service. Additional services may be defined in in companion documents. Note that functionality offered in different services are not totally non-overlapping (e.g., the Map Service and Map Filtering Service).

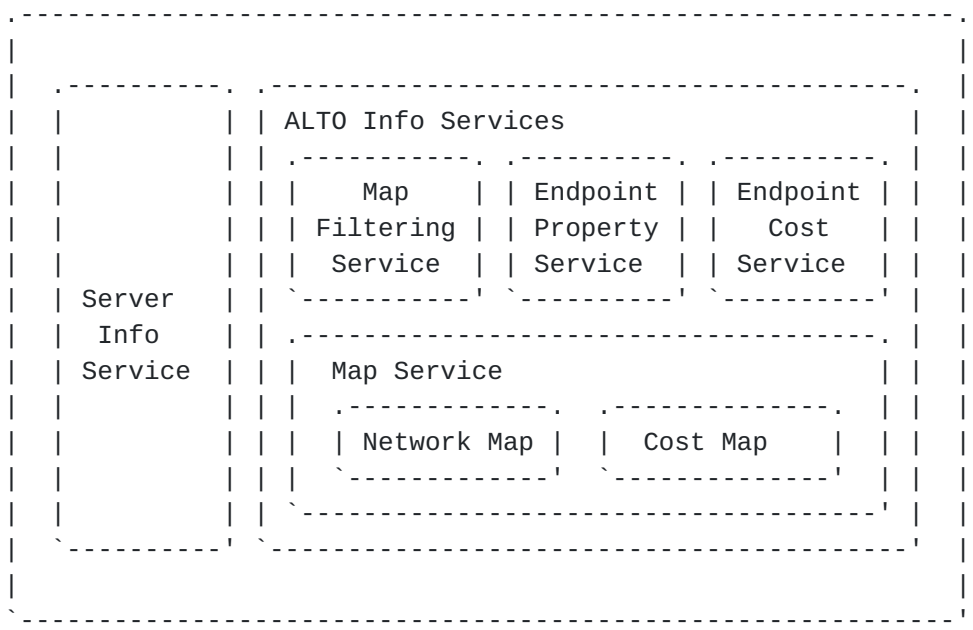

Figure 2: ALTO Protocol Structure

3.1. Server Information Service

The Server Capability Service lists the details on the information that can be provided by an ALTO Server and perhaps other ALTO Servers maintained by the network provider. The configuration includes, for example, details about the operations and cost metrics supported by the ALTO Server and other related ALTO Servers that may be usable by an ALTO Client. The capability document can be downloaded by ALTO Clients. The capability information could also be provisioned to devices, but care must be taken to update it appropriately.

3.2. ALTO Information Services

Multiple, distinct services are defined to allow ALTO Clients to query ALTO Information from an ALTO Server. The ALTO Server internally maintains an ALTO Information Base that encodes the network provider's preferences. The ALTO Information Base encodes the Network Locations defined by the ALTO Server (and their corresponding properties), as well as the provider-defined costs between pairs of Network Locations.

3.2.1. Map Service

The Map Service provides batch information to ALTO Clients in the form of a Network Map and Cost Map. The Network Map (See Section 4) provides the full set of Network Location groupings defined by the ALTO Server and the endpoints contained with each grouping. The Cost Map (see Section 5) provides costs between the defined groupings.

These two maps can be thought of (and implemented as) as simple files with appropriate encoding provided by the ALTO Server.

3.2.2. Map Filtering Service

Resource constrained ALTO Clients may benefit from query results being filtered at the ALTO Server. This avoids an ALTO Client spending network bandwidth or CPU collecting results and performing client-side filtering. The Map Filtering Service allows ALTO Clients to query for the ALTO Server Network Map and Cost Map based on additional parameters.

3.2.3. Endpoint Property Service

This service allows ALTO Clients to look up properties for individual endpoints. An example endpoint property is its Network Location (its grouping defined by the ALTO Server) or connectivity type (e.g., ADSL, Cable, or FioS).

3.2.4. Endpoint Cost Service

Some ALTO Clients may also benefit from querying for costs and rankings based on endpoints. The Endpoint Cost Service allows an ALTO Server to return either numerical costs or ordinal costs (rankings) directly amongst Endpoints.

4. Network Map

In reality, many endpoints are very close to one another in terms of

network connectivity, for example, endpoints on the same site of an enterprise. By treating a group of endpoints together as a single entity in ALTO, we can achieve much greater scalability without losing critical information.

The Network Location endpoint property allows an ALTO Server to group endpoints together to indicate their proximity. The resulting set of groupings is called the ALTO Network Map.

The definition of proximity varies depending on the granularity of the ALTO information configured by the provider. In one deployment, endpoints on the same subnet may be considered close; while in another deployment, endpoints connected to the same PoP may be considered close.

As used in this document, the Network Map refers to the syntax and semantics of the information distributed by the ALTO Server. This document does not discuss the internal representation of this data structure within the ALTO Server.

4.1. PID

Each group of Endpoints is identified by a provider-defined Network Location identifier called a PID. There can be many different ways of grouping the endpoints and assigning PIDs.

A PID is an identifier that provides an indirect and network-agnostic way to specify a network aggregation. For example, a PID may be defined by the ALTO service provider to denote a subnet, a set of subnets, a metropolitan area, a PoP, an autonomous system, or a set of autonomous systems. Aggregation of endpoints into PIDs can indicate proximity and can improve scalability. In particular, network preferences (costs) may be specified between PIDs, allowing cost information to be more compact and updated at a smaller time scale than the network aggregations themselves.

Using PIDs, the Network Map may also be used to communicate simple preferences with only minimal information from the Cost Map. For example, an ISP may prefer that endpoints associated with the same PoP (Point-of-Presence) in a P2P application communicate locally instead of communicating with endpoints in other PoPs. The ISP may aggregate endhosts within a PoP into a single PID in the Network Map. The Cost Map may be encoded to indicate that peering within the same PID is preferred; for example, cost(PID_i, PID_i) == c* and cost(PID_i, PID_j) > c* for i != j. Section 5 provides further details about Cost Map structure.

4.2. Endpoint Addresses

Communicating endpoints may have many types of addresses, such as IP addresses, MAC addresses, or overlay IDs. The current specification only considers IP addresses.

4.2.1. IP Addresses

The endpoints aggregated into a PID are denoted by a list of IP prefixes. When either an ALTO Client or ALTO Server needs to determine which PID in a Network Map contains a particular IP address, longest-prefix matching MUST be used.

A Network Map MUST define a PID for each possible address in the IP address space. A RECOMMENDED way to satisfy this property is to define a PID containing the 0.0.0.0/0 prefix for IPv4 or ::/0 (for IPv6).

4.3. Example Network Map

Figure 3 illustrates an example Network Map. PIDs are used to identify network-agnostic aggregations.

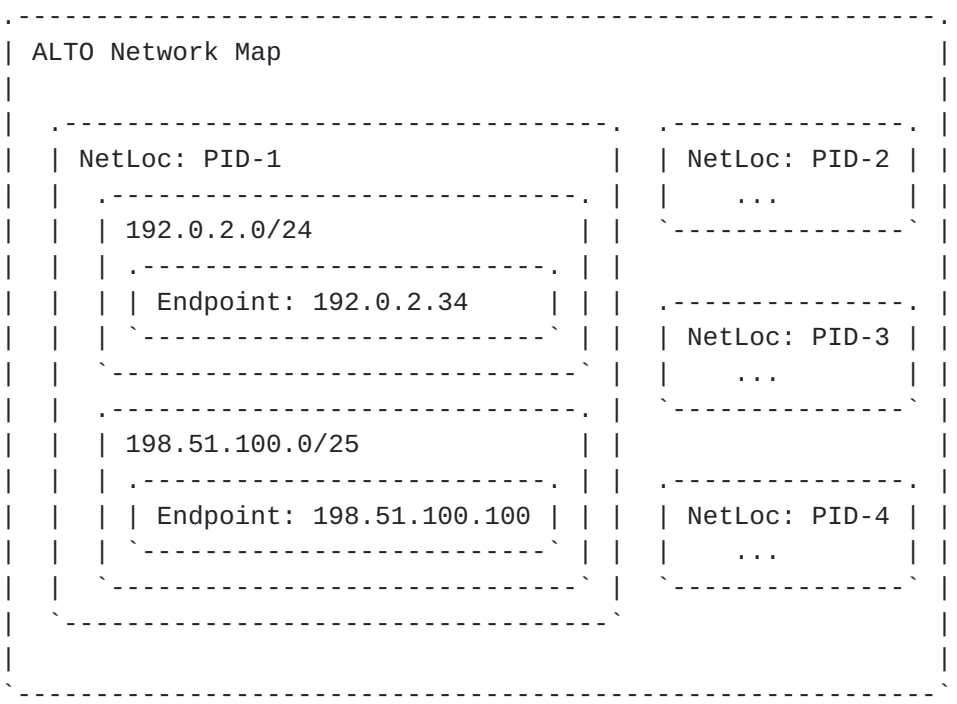

Figure 3: Example Network Map

5. Cost Map

An ALTO Server indicates preferences amongst network locations in the form of Path Costs. Path Costs are generic costs and can be internally computed by a network provider according to its own needs.

An ALTO Cost Map defines Path Costs pairwise amongst sets of source and destination Network Locations.

One advantage of separating ALTO information into a Network Map and a Cost Map is that the two components can be updated at different time scales. For example, Network Maps may be stable for a longer time while Cost Maps may be updated to reflect dynamic network conditions.

As used in this document, the Cost Map refers to the syntax and semantics of the information distributed by the ALTO Server. This document does not discuss the internal representation of this data structure within the ALTO Server.

5.1. Cost Attributes

Path Costs have attributes:

- Type: identifies what the costs represent;
- Mode: identifies how the costs should be interpreted (numerical or ordinal).

Certain queries for Cost Maps allow the ALTO Client to indicate the desired Type and Mode.

5.1.1. Cost Type

The Type attribute indicates what the cost represents. For example, an ALTO Server could define costs representing air-miles, hop-counts, or generic routing costs.

Cost types are indicated in protocol messages as alphanumeric strings. An ALTO Server MUST at least define the routing cost type denoted by the string 'routingcost'.

Note that an ISP may internally compute routing cost using any method it chooses (including air-miles or hop-count).

5.1.2. Cost Mode

The Mode attribute indicates how costs should be interpreted. For example, an ALTO Server could return costs that should be interpreted

as numerical values or ordinal rankings.

It is important to communicate such information to ALTO Clients, as certain operations may not be valid on certain costs returned by an ALTO Server. For example, it is possible for an ALTO Server to return a set of IP addresses with costs indicating a ranking of the IP addresses. Arithmetic operations, such as summation, that would make sense for numerical values, do not make sense for ordinal rankings. ALTO Clients may handle such costs differently.

Cost Modes are indicated in protocol messages as alphanumeric strings. An ALTO Server MUST at least define the modes 'numerical' and 'ordinal'.

If an ALTO Client requests a Cost Mode that is not supported, the ALTO Server MUST reply with costs having Mode either 'numerical' or 'ordinal'. Thus, an ALTO Server must implement at least one of 'numerical' or 'ordinal' Costs, but it may choose which to support. ALTO Clients may choose how to handle such situations. Two particular possibilities are to use the returned costs as-is (e.g., treat numerical costs as ordinal rankings) or ignore the ALTO information altogether.

5.2. Cost Map Structure

A query for a Cost Map either explicitly or implicitly includes a list of Source Network Locations and a list of Destination Network Locations. (Recall that a Network Location can be an endpoint address or a PID.)

Specifically, assume that a query has a list of multiple Source Network Locations, say [Src_1, Src_2, ..., Src_m], and a list of multiple Destination Network Locations, say [Dst_1, Dst_2, ..., Dst_n].

The ALTO Server will return the Path Cost for each communicating pair (i.e., Src_1 -> Dst_1, ..., Src_1 -> Dst_n, ..., Src_m -> Dst_1, ..., Src_m -> Dst_n). We refer to this structure as a Cost Map.

If the Cost Mode is 'ordinal', the Path Cost of each communicating pair is relative to the m*n entries.

5.3. Network Map and Cost Map Dependency

If a Cost Map contains PIDs in the list of Source Network Locations or the list of Destination Network Locations, the Path Costs are generated based on a particular Network Map (which defines the PIDs). Version Tags are introduced to ensure that ALTO Clients are able to

use consistent information even though the information is provided in two maps.

A Version Tag is an opaque string associated with a Network Map maintained by the ALTO Server. When the Network Map changes, the Version Tag SHOULD also be changed. (Thus, the Version Tag is defined similarly to HTTP's ETag.) Possibilities for generating a Version Tag included the last-modified timestamp for the Network Map, or a hash of its contents.

A Network Map distributed by the ALTO Server includes its Version Tag. A Cost Map referring to PIDs also includes the Version Tag of the Network Map on which it is based.

6. Protocol Overview

6.1. Design Approach

The ALTO Protocol design uses a REST-like interface with the goal of leveraging current HTTP [2] [3] implementations and infrastructure, as well as familiarity with existing REST-like services in popular use. ALTO messages are denoted with HTTP Content-Type "application/alto" and use JSON [4] to encode message bodies.

This document currently specifies both services and and the message encoding in a descriptive fashion. Care is taken to make descriptions precise and unambiguous, but it still lacks benefits of automatic tooling that exists for certain encoding formats.

Standards such as WSDL 2.0 and WADL are capable of describing available interfaces. JSON Schema [17] allows message encodings to be specified precisely and messages may be verified against the schema. It is not yet clear whether such an approach should be taken in this document.

Other benefits enabled by these design choices include easier understanding and debugging, flexible ALTO Server implementation strategies, and more importantly, simple caching and redistribution of ALTO information to increase scalability.

6.1.1. Use of Existing Infrastructure

HTTP is a natural choice for integration with existing applications and infrastructure. In particular, the ALTO Protocol design leverages:

- the huge installed base of infrastructure, including HTTP caches,
- mature software implementations,
- the fact that many P2P clients already have an embedded HTTP client, and
- authentication and encryption mechanisms in HTTP and SSL/TLS.

6.1.2. ALTO Information Reuse and Redistribution

ALTO information may be useful to a large number of applications and users. For example, an identical Network Map may be used by all ALTO Clients querying a particular ALTO Server. At the same time, distributing ALTO information must be efficient and not become a bottleneck. Therefore, the ALTO Protocol specified in this document integrates with existing HTTP caching infrastructure to allow reuse of ALTO information by ALTO Clients and reduce load on ALTO servers.

ALTO information may also be cached or redistributed using application-dependent mechanisms, such as P2P DHTs or P2P file-sharing. This document does not define particular mechanisms for such redistribution, but it does define the primitives (e.g., digital signatures) needed to support such a mechanism. See [18] for further discussion.

Note that if caching or redistribution is used, the Response message may be returned from another (possibly third-party) entity. Reuse and Redistribution is further discussed in Section 11.4. Protocol support for redistribution is specified in Section 7.8.

7. Protocol Messaging

This section specifies client and server processing, as well as messages in the ALTO Protocol. Details common to ALTO Server processing of all messages is first discussed, followed by details of the individual messages.

7.1. Notation

This document uses C-style struct notation to define the required and optional members of JSON objects. Unless explicitly noted, each member of a struct is REQUIRED.

The types 'JSONString', 'JSONNumber', 'JSONBool' indicate the JSON string, number, and boolean types respectively.

This document only includes object members used by this specification. It is possible that protocol extensions include additional members to JSON objects defined in this document; such additional members will be silently ignored by ALTO Servers and Clients only implementing the base protocol defined in this document.

7.2. Message Format

Request and Response follow the standard format for HTTP Request and Response messages [2] [3].

The following subsections provide an overview of how ALTO Requests and Responses are encoded in HTTP, and discusses rationale for certain design decisions.

7.2.1. Protocol Versioning Approach

The ALTO Protocol uses a simple and clean approach to versioning that permits evolution between versions even if ALTO information is being served as static, pre-generated files.

In particular, it is assumed that a single host responding to ALTO Requests implements a single protocol version. Note that virtual hosting can be used if multiple protocol versions need to be supported by a single physical server.

A common query (Server List, detailed in Section 7.7.1.1) to be present in all ALTO protocol versions allows an ALTO Client to discover additional ALTO Servers and the ALTO Protocol version number of each.

This approach keeps the ALTO Server implementation free from parsing and directing each request based on version number. Although ALTO Requests are free from protocol version numbers, the protocol version number is echoed in each ALTO Response to keep responses self-contained to, for example, ease reading persisted or redistributed ALTO responses.

This document specifies ALTO Protocol version 1.

7.2.2. Request Message

An ALTO Request is a standard HTTP Request generated by an ALTO Client, with certain components defined by the ALTO Protocol.

The basic syntax of an ALTO Request is:

```
<Method> /<Resource> HTTP/1.1
```

```
Host: <Host>
```

For example:

```
GET /info/capability HTTP/1.1
Host: alto.example.com:6671
```

7.2.2.1. Standard HTTP Headers

The Host header MUST follow the standard rules for the HTTP 1.1 Host Header.

The Content-Length header MUST follow the standard rules defined in HTTP 1.1.

The Content-Type HTTP Header MUST have value "application/alto" if the Body is non-empty.

7.2.2.2. Method and Resource

Next, both the HTTP Method and URI-Path (denoted as Resource) indicate the operation requested by the ALTO Client. In this example, the ALTO Client is requesting basic capability information from the ALTO Server.

7.2.2.3. Input Parameters

Certain operations defined by the ALTO Protocol (e.g., in the Map Filtering Service) allow the ALTO Client to supply additional input parameters. Such input parameters are encoded in a URI-Query-String where possible and appropriate. However, due to practical limitations (e.g. underlying HTTP implementations may have limitations on the total length of a URI and the Query-String is better-suited for simple unstructured parameters and lists), some operations in the ALTO Protocol use input parameters encoded in the HTTP Request Body.

7.2.3. Response Message

A Response message is a standard HTTP Response generated by an ALTO Server with certain components defined by the ALTO Protocol.

The basic syntax of an ALTO Response is:

```
HTTP/1.1 <StatusCode> <StatusMsg>
Content-Length: <ContentLength>
Content-Type: <ContentType>
```

```
<ALTOResponse>
```

where the HTTP Response Body is an ALTOResponse JSON Object (defined in Section 7.2.3.3). For example:

```
HTTP/1.1 200 OK
Content-Length: 1000
Content-Type: application/alto

{
    "meta" : {
        "version": 1,
        "status" : {
            "code" : 1,
            "reason" : "Success"
        },
        ...
    },
    "type" : "capability",
    "data" : {
        ...
    }
}
```

7.2.3.1. Standard HTTP Headers

The Content-Length header MUST follow the standard rules defined in HTTP 1.1.

The Content-Type HTTP Header MUST have value "application/alto" if the Body is non-empty.

7.2.3.2. Status Code and Message

Two sets of status codes are used in the ALTO Protocol. First, an ALTO Status Code provides detailed information about the success or failure of a particular operation. Second, an HTTP Status Code indicates to HTTP processing elements (e.g., intermediaries and clients) how the response should be treated.

7.2.3.3. HTTP Body

The Response body MUST encode a single top-level JSON object of type ALTOResponse:

```
    struct {
        RspMetaData   meta;
        JSONString    type;
        [RspDataType] data;
    } ALTOResponse;
```

The ALTOResponse object has distinct sections for:

- meta information encoded in an extensible way,
- the type of ALTO Information to follow, and
- the requested ALTO Information.

7.2.3.3.1. Meta Information

Meta information is encoded as a JSON object with type RspMetaData:

```
    struct {
        JSONNumber     code;
        JSONString     reason;          [OPTIONAL]
    } RspStatus;

    struct {
        JSONNumber     version;
        RspStatus      status;
        RspRedistInfo  redistribution;  [OPTIONAL]
    } RspMetaData;
```

with members:

- version: the ALTO Protocol version
- status: the ALTO Status Code indicating a more detailed reason for the success or failure of a request than HTTP Status Codes permit. See Section 7.4 for a list of ALTO Status Codes defined in this document. The corresponding 'reason' is a free-form string providing a human-readable indication of the particular status code.
- redistribution: additional meta information used by ALTO information redistribution (see Section 7.8)

7.2.3.3.2. ALTO Information

If the Response is successful (see Section 7.4), then the "type" and "data" members of the ALTOResponse object are REQUIRED. "type" encodes a Response-specific string which indicates to the ALTO Client

the type of data encoded in the message. The "data" member encodes the actual Response-specific data. The structure of this member is detailed later in this section for each particular ALTO Response.

7.2.3.4. Signature

An ALTO Server MAY additionally supply a signature asserting that it generated a particular response. In order to allow the signature to be computed over the entire response message, the signature itself is specified in an HTTP Header or Trailer (see Section 7.8.5).

7.3. General Processing

The protocol is structured in such a way that, independent of the query type, there are a set of general processing steps. The ALTO Client selects a specific ALTO Server with which to communicate, establishes a TCP connection, and constructs and sends ALTO Request messages which MUST conform to Section 7.7. In response to Request messages, an ALTO Server constructs and sends ALTO Response messages which also MUST conform to Section 7.7.

7.4. ALTO Status Codes

This document defines ALTO Status Codes to support the operations defined in this document. Additional status codes may be defined in companion or extension documents.

An ALTO Server MUST return the 'Success' code in Table 1 if and only if the Request message is successfully processed and the requested ALTO information is returned by the ALTO Server.

The HTTP Status Codes corresponding to each ALTO Status Code are defined to provide correct behavior with HTTP intermediaries and clients. When an ALTO Server returns a particular ALTO Status Code, it MUST indicate one of the corresponding HTTP Status Codes in Table 1.

If multiple errors are present in a single ALTO Request (e.g., a request uses a JSONString when a JSONInteger is expected and a required field is missing), then the ALTO Server MUST return exactly one of the detected errors. However, the reported error is implementation defined, since specifying a particular order for message processing encroaches needlessly on implementation technique.

ALTO Status Code	HTTP Status Code(s)	Description
SUCCESS	2xx	Success
E_JSON_SYNTAX	400	JSON parsing error in request
E_JSON_FIELD_MISSING	400	Required field missing
E_JSON_VALUE_TYPE	400	JSON Value of unexpected type
E_INVALID_OPERATION	501	Invalid operation requested
E_INVALID_COST_TYPE	501	Invalid cost type

Table 1: Defined ALTO Status Codes

Status codes described in Table 1 are a work in progress. The set of status codes will be modified or expanded as implementation experience is gained; feedback is welcomed.

In addition, feedback from implementers of ALTO Clients is welcomed to identify if there is a need to communicate multiple status codes in a single response.

7.5. Client Behavior

7.5.1. Successful Response

This specification does not indicate any required actions taken by ALTO Clients upon receiving a successful response from an ALTO Server. Although ALTO Clients are suggested to interpret the received ALTO Information and adapt application behavior, ALTO Clients may also choose to ignore the received information.

7.5.2. Error Conditions

If an ALTO Client does not receive a successful response from the ALTO Server, it can either choose another server or fall back to a default behavior (e.g., perform peer selection without the use of ALTO information). An ALTO Client may also retry the request at a later time.

7.6. HTTP Usage

7.6.1. Authentication and Encryption

An ALTO Server MAY support SSL/TLS to implement server and/or client authentication, as well as encryption.

An ALTO Server MAY support HTTP Digest authentication.

7.6.2. Cookies

Cookies MUST NOT be used.

7.6.3. Caching Parameters

If the Response generated by the ALTO Server is cachable, the ALTO Server MAY include 'Cache-Control' and 'Expires' HTTP headers.

If a Response generated by the ALTO Server is not cachable, the ALTO Server MUST specify the "Cache-Control: no-cache" HTTP Header.

7.7. ALTO Requests

This section documents the individual operations supported in the ALTO Protocol. See Section 7.2.2 and Section 7.2.3 for specifications of HTTP Request/Response components common to all operations in the ALTO Protocol.

Table 2 provides an summary of the HTTP Method and URI-Paths used for ALTO Requests:

Service	Operation	HTTP Method and URI-Path
Server Info	List Servers	GET /info/servers
Server Info	Capability	GET /info/capability
Map	Network Map	GET /map/core/pid/net
Map	Cost Map	GET /map/core/pid/cost
Map Filtering	Network Map	POST /map/filter/pid/net
Map Filtering	Cost Map	POST /map/filter/pid/cost
Endpoint Prop.	Lookup	GET /endpoint/prop/<name>
		POST /endpoint/prop/lookup
Endpoint Cost	Lookup	POST /endpoint/cost/lookup

Table 2: Overview of ALTO Requests

7.7.1. Server Information Service

The Server Information Service provides information about available ALTO Servers and their capabilities (e.g., supported services).

An ALTO Server MUST support the Server Information Service and MUST implement all operations defined in this section.

7.7.1.1. Server List

The Server List request allows an ALTO Client to discover other ALTO Servers provided by the ALTO Service Provider. Upon discovering an additional ALTO Server, the ALTO Client may then query the server capabilities (see Section 7.7.1.2) to test if it supports desired functionality.

The Server List request is intended to help an ALTO Client find an ALTO Server supporting the desired ALTO Protocol version and capabilities. It is not intended to serve as a substitute for the ALTO Server Discovery which helps an ALTO Client locate an initial ALTO Server.

This operation MUST be supported by the ALTO Server.

7.7.1.1.1. Request Syntax

```
GET /info/servers HTTP/1.1
Host: <Host>
```

7.7.1.1.2. Response Syntax

```
HTTP/1.1 200 <StatusMsg>
Content-Length: <BodyLength>
Content-Type: application/alto

<ALTOResponse>
```

where the ALTOResponse object has "type" member equal to the string "server-list" and "data" member of type RspServerList:

```
struct {
    JSONString   uri;
    JSONNumber   version;
} ServerItem;

struct {
    ServerItem   servers<0..*>;
} RspServerList;
```

RspServerList has members:

- servers: Array of available ALTO Servers, detailing the URI of the ALTO Server and the ALTO Protocol version that it implements. The array must at least contain an entry corresponding to the ALTO Server at the URI from which it is retrieving the server list.

7.7.1.1.3. Example

```
GET /info/servers HTTP/1.1
Host: alto.example.com:6671


HTTP/1.1 200 OK
Content-Length: [TODO]
Content-Type: application/alto

{
    "meta" : {
        "version" : 1,
        "status" : {
            "code" : 1
        }
    },
    "type" : "server-list",
    "data" : {
        "servers" : [
            {
                "uri": "http://alto.example.com:6671",
                "version" : 1
            }
        ]
    }
}
```

7.7.1.2. Server Capability

The Server Capability request allows an ALTO Client to determine the functionality supported by the queried ALTO Server.

This operation MUST be supported by the ALTO Server.

7.7.1.2.1. Request Syntax

```
GET /info/capability HTTP/1.1
Host: <Host>
```

7.7.1.2.2. Response Syntax

```
HTTP/1.1 200 <StatusMsg>
Content-Length: <BodyLength>
Content-Type: application/alto

<ALTOResponse>
```

where the ALTOResponse object has "type" member equal to the string "capability" and "data" member of type RspCapability:

```
enum {
    map,
    map_filtering,
    endpoint_property,
    endpoint_cost
} ServiceType;          [Note: encoded as JSONString's]

struct {
    ServiceType  services<0..*>;
    JSONString   cost_types<0..*>;      [OPTIONAL]
    JSONBool     cost_constraints;      [OPTIONAL]
    JSONString   service_id;            [OPTIONAL+]
    JSONString   certificate;           [OPTIONAL+]
} RspCapability;
```

See Section 7.8.1 for additional notes concerning the 'service_id' and 'certificate' fields for ALTO Servers enabling response redistribution.

RspCapability has members:

- services: Lists the services supported by the ALTO Server. The service names defined in this document are are "map", "map_filtering", "endpoint_property", and "endpoint_cost".

- cost_types: Array of cost type information for additional supported ALTO Cost types, detailing the name for each supported additional type. [[Comment.1: Need to discuss IANA implications or alternate approaches. Note that current definition assumes the unit for a cost type is fixed.]]

- cost_constraints: Indicates if the ALTO Server supports cost constraints. The value 'false' is implied if this member is not present.

- service_id: UUID [5] indicating an set of ALTO Servers (possibly just a single ALTO Server) serving equivalent ALTO Information

(see Section 7.8).

- certificate: PEM-encoded X.509 certificate used by the ALTO Server to sign distributed information (see Section 7.8).

7.7.1.2.3. Example

```
GET /info/capability HTTP/1.1
Host: alto.example.com:6671


HTTP/1.1 200 OK
Content-Length: [TODO]
Content-Type: application/alto

{
    "meta" : {
        "version" : 1,
        "status" : {
            "code" : 1
        }
    },
    "type" : "capability",
    "data" : {
        "services" : [ "map", "map-filtering" ],
        "cost_types": [
            "routingcost",
            "hopcount"
        ],
        "cost_constraints": false
    }
}
```

7.7.2. Map Service

The Map Service provides batch information to ALTO Clients in the form of two maps: a Network Map and Cost Map.

An ALTO Server MUST support the Map Service and MUST implement all operations defined in this section.

7.7.2.1. Network Map

The full Network Map lists for each PID, the network locations (endpoints) within the PID.

7.7.2.1.1. Request Syntax

```
GET /map/core/pid/net HTTP/1.1
Host: <Host>
```

7.7.2.1.2. Response Syntax

```
HTTP/1.1 200 <StatusMsg>
Content-Length: <BodyLength>
Content-Type: application/alto

<ALTOResponse>
```

where the ALTOResponse object has "type" member equal to the string "network_map" and "data" member of type RspNetworkMap:

```
struct {
    CIDRString [pidname]<0..*>;
    ...
} NetworkMapData;

struct {
    JSONString     map_vtag;
    NetworkMapData map;
} RspNetworkMap;
```

RspNetworkMap has members:

- map_vtag: The Version Tag of the Network Map (Section 5.3)

- map: The network map data itself.

NetworkMapData is a JSON object with each member representing a single PID and its associated set of IP Prefixes (encoded as a string in CIDR notation). A member's name is a JSONString denoting the PID's name.

7.7.2.1.3. Example

```
GET /map/core/pid/net HTTP/1.1
Host: alto.example.com:6671
```

```
HTTP/1.1 200 OK
Content-Length: [TODO]
Content-Type: application/alto

{
    "meta" : {
        "version" : 1,
        "status" : {
            "code" : 1
        }
    },
    "type" : "network_map",
    "data" : {
        "map_vtag" : "1266506139",
        "map" : {
            "PID1" : [
                "192.0.2.0/24",
                "198.51.100.0/25"
            ],
            "PID2" : [
                "198.51.100.128/25"
            ],
            "PID3" : [
                "0.0.0.0/0"
            ]
        }
    }
}
```

7.7.2.2. Cost Map

The Map Service Cost Map query is a batch operation in which the ALTO Server returns the Path Cost for each pair of source/destination PID defined by the ALTO Server.

The ALTO Server provides costs using the default Cost Type ('routingcost') and default Cost Mode ('numerical').

7.7.2.2.1. Request Syntax

```
GET /map/core/pid/cost HTTP/1.1
Host: <Host>
```

7.7.2.2.2. Response Syntax

```
HTTP/1.1 200 <StatusMsg>
Content-Length: <BodyLength>
Content-Type: application/alto

<ALTOResponse>
```

where the ALTOResponse object has "type" member equal to the string "cost_map" and "data" member of type RspCostMap:

```
struct DstCosts {
    JSONNumber [dstname];
    ...
};

struct {
    DstCosts [srcname]<0..*>;
    ...
} CostMapData;

struct {
    JSONString  map_vtag;
    JSONString  cost_type;
    JSONString  cost_mode;
    CostMapData map;
} RspCostMap;
```

RspCostMap has members:

- map_vtag: The Version Tag of the Network Map used to generate the Cost Map (Section 5.3).
- cost_type: Cost Type used in the map (Section 5.1.1)
- cost_mode: Cost Mode used in the map (Section 5.1.2)
- map: The cost map data itself.

CostMapData is a JSON object with each member representing a single Source PID. For each Source PID, a DstCosts structure denotes the associated cost to a set of destination PIDs (Section 5.2). DstCosts has a single member for each destination PID in the map.

7.7.2.2.3. Example

```
GET /map/core/pid/cost HTTP/1.1
Host: alto.example.com:6671
```

```
HTTP/1.1 200 OK
Content-Length: [TODO]
Content-Type: application/alto

{
    "meta" : {
        "version" : 1,
        "status" : {
            "code" : 1
        }
    },
    "type" : "cost_map",
    "data" : {
        "map_vtag"  : "1266506139",
        "cost_type" : "routingcost",
        "cost_mode" : "numerical",
        "map" : {
            "PID1": { "PID1": 1,  "PID2": 5,  "PID3": 10 },
            "PID2": { "PID1": 5,  "PID2": 1,  "PID3": 15 },
            "PID3": { "PID1": 20, "PID2": 15, "PID3": 1  }
        }
    }
}
```

7.7.3. Map Filtering Service

The Map Filtering Service allows ALTO Clients to specify filtering criteria to return a subset of the full maps available in the Map Service.

An ALTO Server MAY support the Map Filtering Service. If an ALTO Server supports the Map Filtering Service, all operations defined in this section MUST be implemented.

7.7.3.1. Network Map

ALTO Clients can query for a subset of the full network map (see Section 7.7.2.1).

7.7.3.1.1. Request Syntax

```
POST /map/filter/pid/net HTTP/1.1
Host: <Host>
Content-Length: <BodyLength>

<ReqNetworkMap>
```

where:

```
struct {
    JSONString pids<0..*>;
} ReqNetworkMap;
```

The Body of the request encodes an array of PIDs to be included in the resulting Network Map. If the list of PIDs is empty, the ALTO Server MUST interpret the list as if it contained a list of all currently-defined PIDs.

7.7.3.1.2. Response Syntax

The Response syntax is identical to that of the Map Service's Network Map Response (Section 7.7.2.1.2).

The ALTO Server MUST only include PIDs in the Response that were specified (implicitly or explicitly) in the Request. If the Request contains a PID name that is not currently defined by the ALTO Server, the ALTO Server MUST behave as if the PID did not appear in the request.

7.7.3.1.3. Example

```
POST /map/filter/pid/net HTTP/1.1
Host: alto.example.com:6671
Content-Length: <BodyLength>

{
    pids: [ "PID1", "PID2" ]
}


HTTP/1.1 200 OK
Content-Length: [TODO]
Content-Type: application/alto

{
    "meta" : {
        "version" : 1,
        "status" : {
            "code" : 1
        }
    },
    "type" : "network_map",
    "data" : {
        "map_vtag" : "1266506139",
        "map" : {
            "PID1" : [
                "192.0.2.0/24",
                "198.51.100.0/24",
            ],
            "PID2" : [
                "198.51.100.128/24",
            ]
        }
    }
}
```

7.7.3.2. Cost Map

ALTO Clients can query for the Cost Map (see Section 7.7.2.2) based on additional parameters.

7.7.3.2.1. Request Syntax

```
POST /map/filter/pid/cost?<URI-Query-String> HTTP/1.1
Host: <Host>

<ReqCostMap>
```

where:

```
struct {
    JSONString srcs<0..*>;
    JSONString dsts<0..*>;
} ReqCostMap;
```

The Query String may contain the following parameters:

- type: The requested Cost Type (Section 5.1.1). If not specified, the default value is "routingcost". This parameter MUST NOT be specified multiple times.

- mode: The requested Cost mode (Section 5.1.2). If not specified, the default value is "numerical". This parameter MUST NOT be specified multiple times.

- constraint: Defines a constraint on which elements of the Cost Map are returned. This parameter MUST NOT be used if the Server Capability Response (Section 7.7.1.2) indicates that constraint support is not available. A constraint contains two entities separated by whitespace (before URL encoding): (1) an operator either 'gt' for greater than , 'lt' for less than or 'eq' for equal to with 10 percent on either side, (2) a target numerical cost. The numerical cost is a number that MUST be defined in the units specified in the Server Capability Response. If multiple 'constraint' parameters are specified, the ALTO Server assumes they are related to each other with a logical AND. If no 'constraint' parameters are specified, then the ALTO Server returns the full Cost Map.

The Request body MAY specify a list of Source PIDs, and a list of Destination PIDs. If a list is empty, it is interpreted by the ALTO Server as the full set of currently-defined PIDs. The ALTO Server returns costs between each pair of source/destination PID. If the Request body is empty, both lists are interpreted to be empty.

7.7.3.2.2. Response Syntax

The Response syntax is identical to that of the Map Service's Cost Map Response (Section 7.7.2.2.2).

The Response MUST NOT contain any source/destination pair that was not indicated (implicitly or explicitly) in the Request. If the Request contains a PID name that is not currently defined by the ALTO Server, the ALTO Server MUST behave as if the PID did not appear in the request.

7.7.3.2.3. Example

```
POST /map/filter/pid/cost?type=hopcount HTTP/1.1
Host: alto.example.com:6671

{
    "srcs" : [ "PID1" ],
    "dsts" : [ "PID1", "PID2", "PID3" ]
}


HTTP/1.1 200 OK
Content-Length: [TODO]
Content-Type: application/alto

{
    "meta" : {
        "version" : 1,
        "status" : {
            "code" : 1
        }
    },
    "type" : "cost_map",
    "data" : {
        "map_vtag"  : "1266506139",
        "cost_type" : "hopcount",
        "cost_mode" : "numerical",
        "map" : {
            "PID1": { "PID1": 0,  "PID2": 1,  "PID3": 2 }
        }
    }
}
```

7.7.4. Endpoint Property Service

The Endpoint Property Lookup query allows an ALTO Client to lookup properties of Endpoints known to the ALTO Server. If the ALTO Server provides the Endpoint Property Service, the ALTO Server MUST define at least the 'pid' property for Endpoints.

An ALTO Server MAY support the Endpoint Property Service. If an ALTO Server supports the Endpoint Property Service, all operations defined in this section MUST be implemented.

7.7.4.1. Endpoint Property Lookup

7.7.4.1.1. Request Syntax

```
POST /endpoint/prop/lookup?<URI-Query-String> HTTP/1.1
Host: <Host>
Content-Length: <BodyLength>

<ReqEndpointProp>
```

where:

```
struct {
    JSONString endpoints<0..*>;
} ReqEndpointProp;
```

The Query String may contain the following parameters:

o prop: The requested property type. This parameter MUST be specified at least once, and MAY be specified multiple times (e.g., to query for multiple different properties at once).

The body encodes a list of endpoints (IP addresses) as strings.

An alternate syntax is supported for the case when properties are requested for a single endpoint:

```
GET /endpoint/prop/<Endpoint>?<URI-Query-String> HTTP/1.1
Host: <Host>
```

where the Query String is the same as in the first form.

7.7.4.1.2. Response Syntax

```
HTTP/1.1 200 <StatusMsg>
Content-Length: <BodyLength>
Content-Type: application/alto

<ALTOResponse>
```

where the ALTOResponse object has "type" member equal to the string "endpoint_property" and "data" member of type RspEndpointProperty:

```
struct {
    JSONString [propertyname];
    ...
} EndpointProps;

struct {
    EndpointProps  [endpointname]<0..*>;
    ...
} RspEndpointProperty;
```

RspEndpointProperty has one member for each endpoint indicated in the Request. The requested properties for each endpoint are encoded in a corresponding EndpointProps object, which encodes one name/value pair for each requested property. Note that property values are JSON Strings. If the ALTO Server does not define a requested property for a particular endpoint, then it MUST omit it from the Response for only that endpoint.

7.7.4.1.3. Example

```
POST /endpoint/prop/lookup?prop=pid HTTP/1.1
Host: alto.example.com:6671
Content-Length: [TODO]

{
    "endpoints" : [ "192.0.2.34", "203.0.113.129" ]
}


HTTP/1.1 200 OK
Content-Length: [TODO]
Content-Type: application/alto

{
    "meta" : {
        "version" : 1,
        "status" : {
            "code" : 1
        }
    },
    "type" : "endpoint_property",
    "data": {
        "192.0.2.34"    : { "pid": "PID1" },
        "203.0.113.129" : { "pid": "PID3" }
    }
}
```

7.7.5. Endpoint Cost Service

The Endpoint Cost Service allows ALTO Clients to directly supply endpoints to an ALTO Server. The ALTO Server replies with costs (numerical or ordinal) amongst the endpoints.

In particular, this service allows lists of Endpoint addresses to be ranked (ordered) by an ALTO Server.

An ALTO Server MAY support the Endpoint Cost Service. If an ALTO Server supports the Endpoint Cost Service, all operations defined in this section MUST be implemented.

7.7.5.1. Endpoint Cost Lookup

7.7.5.1.1. Request Syntax

```
POST /endpoint/cost/lookup?<URI-Query-String> HTTP/1.1
Host: <Host>
Content-Length: <BodyLength>

<ReqCostMap>
```

The request body includes a list of source and destination endpoints that should be assigned a cost by the ALTO Server. The allowed Query String parameters are defined identically to Section 7.7.3.2.

The request body MUST specify a list of source Endpoints, and a list of destination Endpoints, using an structure identical to Section 7.7.3.2 with the exception that identifiers are endpoints instead of PIDs. If the list of source Endpoints is empty (or it is not included), the ALTO Server MUST treat it as if it contained the Endpoint address of the requesting client. The list of destination Endpoints MUST NOT be empty. The ALTO Server returns costs between each pair of source/destination Endpoint.

7.7.5.1.2. Response Syntax

```
HTTP/1.1 200 <StatusMsg>
Content-Length: <BodyLength>
Content-Type: application/alto

<ALTOResponse>
```

where ALTOResponse is encoded identically to Section 7.7.2.2.2 with the following exceptions:

- ALTO Response's "type" member must be equal to "endpoint_cost_map",
- The "map_vtag" member of RspCostMap MUST be omitted, and
- Identifiers refer to endpoints instead of PIDs.

7.7.5.1.3. Example

```
POST /endpoint/cost/lookup?mode=ordinal HTTP/1.1
Host: alto.example.com:6671
Content-Length: [TODO]

{
  "src": [ "192.0.2.2" ],
  "dst": [ "192.0.2.89", "198.51.100.34", "203.0.113.45" ]
}


HTTP/1.1 200 OK
Content-Length: [TODO]
Content-Type: application/alto

{
    "meta" : {
        "version" : 1,
        "status" : {
            "code" : 1
        }
    },
    "type" : "endpoint_cost_map",
    "data" : {
        "cost-type" : "routingcost",
        "cost-mode" : "ordinal",
        "map" : {
            "192.0.2.2": {
                "192.0.2.89"    : 1,
                "198.51.100.34" : 2,
                "203.0.113.45"  : 3
            }
        }
    }
}
```

7.8. Redistributable Responses

An ALTO Server MAY indicate that a response is suitable for redistribution by including the "redistribution" member in the RspMetaData JSON object of an ALTO Response message. This additional member has type RspRedistInfo:

```
struct {
    JSONString service_id;
    JSONString request_uri;
    JSONValue  request_body;
    JSONString expires;
} RspRedistInfo;
```

If an ALTO Server indicates the that the response is redistributable, the Response message MUST satisfy all requirements in this section.

The ALTO Server generating the response indicates its own unique identifier (Service ID) and any input parameters used to generate the response. This allows ALTO Clients to which the information is distributed to understand the context of the query and interpret the results. This information is encoded in the RspRedistInfo JSON Object.

7.8.1. Server Capability Requirements

The 'service_id' and 'certificate' fields of the Server Capability response are REQUIRED if an ALTO Server generates redistributable responses.

7.8.2. Service ID

For scalability and fault tolerance, multiple ALTO Servers may be deployed to serve equivalent ALTO Information. In such a scenario, ALTO Responses from any such redundant server should be seen as equivalent for the purposes of redistribution. For example, if two ALTO Servers A and B are deployed by the service provider to distribute equivalent ALTO Information, then clients contacting Server A should be able to redistribute ALTO Responses to Server B.

One technique for doing this would be to rely on the server's DNS name. However, such an approach would mandate that all ALTO Servers resolved by a particular DNS name would need to provide equivalent ALTO information, which may be unneccessarily restrictive. Another technique would be to rely on the server's IP adddress. However, this suffers similar problems as the DNS name in deployment scenarios using IP Anycast.

To avoid such restrictions, the ALTO Protocol allows an ALTO Service Provider to explicitly denote ALTO Servers that provide equivalent ALTO Information by giving them identical Service IDs.

If an ALTO Server generates redistributable responses, the Server Capability response's 'service_id' field MUST be set to the ALTO Server's Service ID.

To help prevent ALTO Servers from mistakenly claiming to distribute equivalent ALTO Information, ALTO Server Implementations SHOULD by default generate a new UUID at installation time. The default, generated UUID may be overridden by the service provider.

7.8.3. Server and Request Parameters

This section defines the members of the RspRedistInfo JSON object.

The 'service_id' member is REQUIRED and MUST have a value equal to the ALTO Server's Service ID.

The 'request_uri' member is REQUIRED and MUST specify the HTTP Request-URI that was passed in the HTTP Request.

If the HTTP Request body was non-empty, the 'request_body' member MUST specify full JSON value passed in the HTTP Request (note that whitespace may differ, as long as the JSON Value is identical). If the HTTP Request was empty, then the 'request_body' MUST NOT be included.

Note that information about ALTO Client performing the Request and any HTTP Headers passed in the request are not included. If any such information or headers influence the response generated by the ALTO Server, the response SHOULD NOT be indicated as redistributable.

7.8.4. Expiration Time

ALTO Responses marked as redistributable SHOULD indicate a time after which the information is considered stale and should be refreshed from the ALTO Server (or possibly another ALTO Client).

The 'expires' element is RECOMMENDED and, if present, MUST specify a time in UTC formatted according to [6].

If an expiration time is present, the ALTO Server SHOULD ensure that it is reasonably consistent with the expiration time that would be computed by HTTP header fields. If the expiration time in the 'expires' element is earlier, some ALTO Clients may refresh data from the ALTO Server earlier than expected. If the expiration time

included in the response body is later, some ALTO Clients may refresh the data later than expected.

7.8.5. Signature

ALTO Responses marked as redistributable MUST include a signature used to assert that the ALTO Server Provider generated the ALTO Information.

Verification of the signature requires the ALTO Client to retrieve the ALTO Server's public key. There are multiple possibilities to retrieve it:

- SSL/TLS connection with the ALTO Server: The public key algorithm and public key may be retrieved from the ALTO Server's X.509 Certificate used on an HTTPS connection between the ALTO Server and ALTO Client.

- Included in ALTO Server's Server Capability Response: If the ALTO Client requests from the ALTO Server over a non SSL/TLS connection, an X.509 certificate (including the public key and public key algorithm) can be included in the Server Capability Response.

To reduce requirements on the underlying transport (i.e., requiring SSL/TLS), the ALTO Protocol uses the latter option. This specification makes the following requirements of the X.509 certificates:

- The certificate's corresponding private key MUST be used to sign redistributable responses.

- The certificate for each ALTO Server with an identical Service ID MUST be identical.

ALTO Clients SHOULD verify that the certificate satisfies any local policies (e.g., Issuer, expiration date, etc).

The ALTO Server may include the Hash Algorithm, Signature Algorithm, and Signature in either HTTP Headers or Trailers. Headers may be useful if Responses are pre-generated, while Trailers may be useful if Responses are dynamically generated (e.g., to avoid buffering large responses in memory while the hash value is computed).

The following HTTP Headers (the ALTO Server MAY specify them as HTTP Trailers) MUST be used to encode the Signature parameters for redistributable ALTO Responses:

```
ALTO-HashAlgorithm: <HashAlgorithm>
ALTO-SignatureAlgorithm: <SignatureAlgorithm>
ALTO-SignatureDigest: <Signature>
```

where <HashAlgorithm> and <SignatureAlgorithm> are an integer values from the IANA TLS HashAlgorithm and SignatureAlgorithm registries, and <Signature> is the corresponding PEM-encoded signature.

ALTO Clients SHOULD verify the signature on any ALTO information received via redistribution before adjusting application behavior based on it.

If an ALTO Client consumes ALTO Information from multiple ALTO Servers, it can locally maintain a map of the corresponding certificate for each ALTO Server. Upon receiving redistributed information, it may lookup the appropriate certificate to use for signature verification based on the Service ID contained in the redistributed ALTO Response. Note that verifying the signature also protects against hijacking of a Service ID as long as the initial certificate retrieval (via the Server Capability Response) is secure from hijacking.

ALTO Clients SHOULD pass the ALTO Server Certificate, Signature, and Signature Algorithm along with the body of the ALTO Response. The mechanism for redistributing such information is not specified by the ALTO Protocol, but one possibility is to add additional messages or fields to the application's native protocol.

8. Use Cases

The sections below depict typical use cases.

8.1. ALTO Client Embedded in P2P Tracker

Many P2P currently-deployed P2P systems use a Tracker to manage swarms and perform peer selection. P2P trackers may currently use a variety of information to perform peer selection to meet application-specific goals. By acting as an ALTO Client, an P2P tracker can use ALTO information as an additional information source to enable more network-efficient traffic patterns and improve application performance.

A particular requirement of many P2P trackers is that they must handle a large number of P2P clients. A P2P tracker can obtain and locally store ALTO information (the Network Map and Cost Map) from the ISPs containing the P2P clients, and benefit from the same aggregation of network locations done by ALTO Servers.

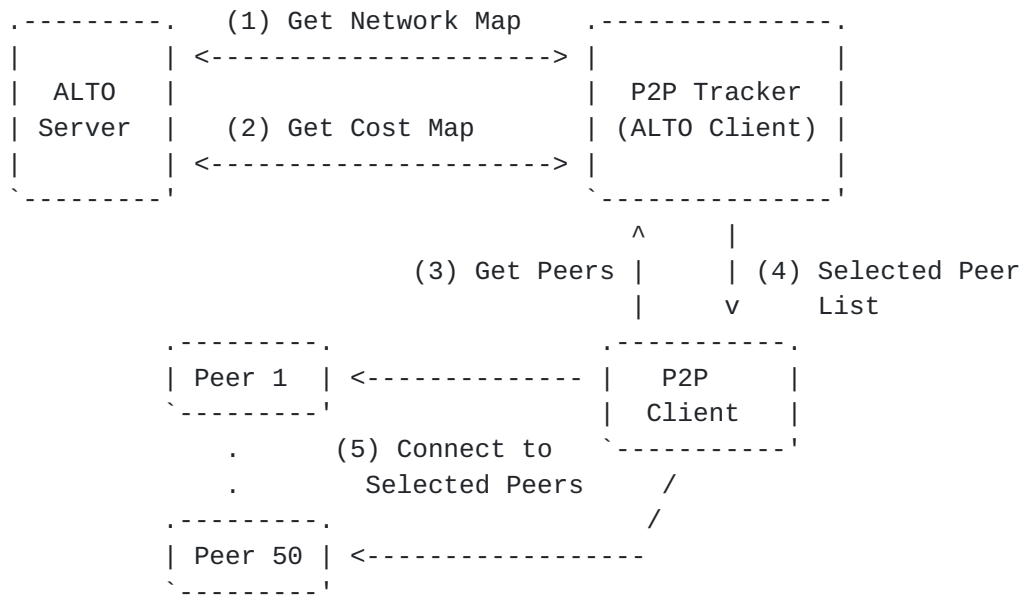

Figure 4: ALTO Client Embedded in P2P Tracker

Figure 4 shows an example use case where a P2P tracker is an ALTO Client and applies ALTO information when selecting peers for its P2P clients. The example proceeds as follows:

1. The P2P Tracker requests the Network Map covering all PIDs from the ALTO Server using the Network Map query. The Network Map includes the IP prefixes contained in each PID, allowing the P2P tracker to locally map P2P clients into a PIDs.

2. The P2P Tracker requests the Cost Map amongst all PIDs from the ALTO Server.

3. A P2P Client joins the swarm, and requests a peer list from the P2P Tracker.

4. The P2P Tracker returns a peer list to the P2P client. The returned peer list is computed based on the Network Map and Cost Map returned by the ALTO Server, and possibly other information sources. Note that it is possible that a tracker may use only the Network Map to implement hierarchical peer selection by preferring peers within the same PID and ISP.

5. The P2P Client connects to the selected peers.

Note that the P2P tracker may provide peer lists to P2P clients distributed across multiple ISPs. In such a case, the P2P tracker may communicate with multiple ALTO Servers.

8.2. ALTO Client Embedded in P2P Client: Numerical Costs

P2P clients may also utilize ALTO information themselves when selecting from available peers. It is important to note that not all P2P systems use a P2P tracker for peer discovery and selection. Furthermore, even when a P2P tracker is used, the P2P clients may rely on other sources, such as peer exchange and DHTs, to discover peers.

When an P2P Client uses ALTO information, it typically queries only the ALTO Server servicing its own ISP. The my-Internet view provided by its ISP's ALTO Server can include preferences to all potential peers.

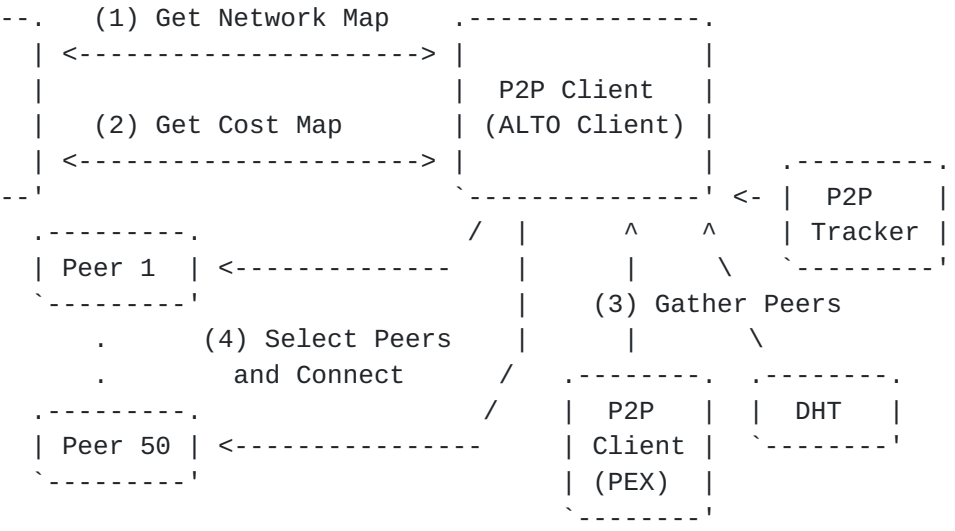

Figure 5: ALTO Client Embedded in P2P Client

Figure 5 shows an example use case where a P2P Client locally applies ALTO information to select peers. The use case proceeds as follows:

1. The P2P Client requests the Network Map covering all PIDs from the ALTO Server servicing its own ISP.

2. The P2P Client requests the Cost Map amongst all PIDs from the ALTO Server. The Cost Map by default specifies numerical costs.

3. The P2P Client discovers peers from sources such as Peer Exchange (PEX) from other P2P Clients, Distributed Hash Tables (DHT), and P2P Trackers.

4. The P2P Client uses ALTO information as part of the algorithm for selecting new peers, and connects to the selected peers.

8.3. ALTO Client Embedded in P2P Client: Ranking

It is also possible for a P2P Client to offload the selection and ranking process to an ALTO Server. In this use case, the ALTO Client gathers a list of known peers in the swarm, and asks the ALTO Server to rank them.

As in the use case using numerical costs, the P2P Client typically only queries the ALTO Server servicing its own ISP.

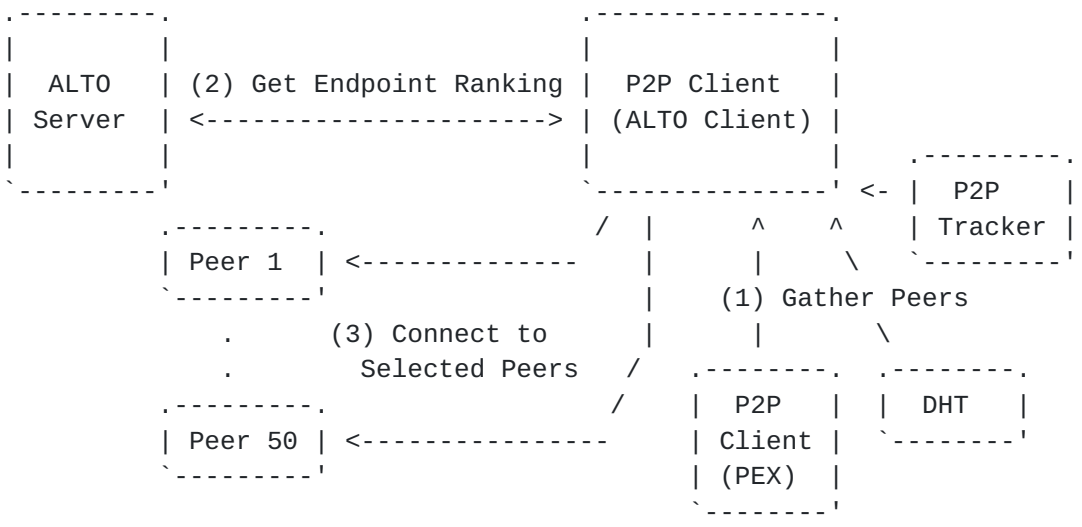

Figure 6: ALTO Client Embedded in P2P Client: Ranking

Figure 6 shows an example of this scenario. The use case proceeds as follows:

1. The P2P Client discovers peers from sources such as Peer Exchange (PEX) from other P2P Clients, Distributed Hash Tables (DHT), and P2P Trackers.

2. The P2P Client queries the ALTO Server's Ranking Service, including discovered peers as the set of Destination Endpoints, and indicates the 'ordinal' Cost Mode. The response indicates the ranking of the candidate peers.

3. The P2P Client connects to the peers in the order specified in the ranking.

9. Discussions

9.1. Discovery

The particular mechanism by which an ALTO Client discovers its ALTO Server is an important component to the ALTO architecture and numerous techniques have been discussed [19]. However, the discovery mechanism is out of scope for this document. This document assumes that an ALTO Client can discover an appropriate ALTO Server. Note that the Server List query in the Server Information Service is intended to aid an ALTO Client in selecting an available ALTO Server with the capabilities necessary for its application.

Some ISPs have proposed the possibility of delegation, in which an ISP provides information for customer networks which do not wish to run ALTO Servers themselves. A consideration for delegation is that customer networks may wish to explicitly configure such delegation.

9.2. Hosts with Multiple Endpoint Addresses

In practical deployments, especially during the transition from IPv4 to IPv6, a particular host may be reachable using multiple addresses. Furthermore, the particular network path followed when sending packets to the host may differ based on the address that is used. Network providers may perfer one path over another (e.g., one path my have a NAT64 middlebox). An additional consideration may be how to handle private address spaces (e.g., behind carrier-grade NATs).

Note that to support such behavior, Endpoints must be associated with a particular address type (e.g., IPv4 or IPv6). One simple possibility may be to prefix each endpoint address with its type (e.g., "ipv4:198.51.100.128/25"). However, we may want to discuss if a more efficient/compact encoding is possible in some cases (e.g., all addresses in the same PID are IPv6).

There are limitations as to what information ALTO can provide in this regard. In particular, a particular ALTO Service provider may not be able to determine if connectivity with a particular endhost will succeed over IPv4 or IPv6, as this may depend upon information unknown to the ISP such as particular application implementations.

Exploration of these issues is being considered in a separate Internet Draft [20]. Once a suitable solution emerges, it will be included in this document.

9.3. Network Address Translation Considerations

At this day and age of NAT v4<->v4, v4<->v6 [21], and possibly v6<->v6[22], a protocol should strive to be NAT friendly and minimize carrying IP addresses in the payload, or provide a mode of operation

where the source IP address provide the information necessary to the server.

The protocol specified in this document provides a mode of operation where the source network location is computed by the ALTO Server (via the Endpoint Property Lookup interface) from the source IP address found in the ALTO Client query packets. This is similar to how some P2P Trackers (e.g., BitTorrent Trackers - see "Tracker HTTP/HTTPS Protocol" in [23]) operate.

The ALTO client SHOULD use the Session Traversal Utilities for NAT (STUN) [7] to determine a public IP address to use as a source Endpoint address. If using this method, the host MUST use the "Binding Request" message and the resulting "XOR-MAPPED-ADDRESS" parameter that is returned in the response. Using STUN requires cooperation from a publicly accessible STUN server. Thus, the ALTO client also requires configuration information that identifies the STUN server, or a domain name that can be used for STUN server discovery. To be selected for this purpose, the STUN server needs to provide the public reflexive transport address of the host.

9.4. Mapping IPs to ASNs

It may be desired for the ALTO Protocol to provide ALTO information including ASNs. Thus, ALTO Clients may need to identify the ASN for a Resource Provider to determine the cost to that Resource Provider.

Applications can already map IPs to ASNs using information from a BGP Looking Glass. To do so, they must download a file of about 1.5MB when compressed (as of October 2008, with all information not needed for IP to ASN mapping removed) and periodically (perhaps monthly) refresh it.

Alternatively, the Network Map query in the Map Filtering Service defined in this document could be extended to map ASNs into a set of IP prefixes. The mappings provided by the ISP would be both smaller and more authoritative.

For simplicity of implementation, it's highly desirable that clients only have to implement exactly one mechanism of mapping IPs to ASNs.

9.5. Endpoint and Path Properties

An ALTO Server could make available many properties about Endpoints beyond their network location or grouping. For example, connection type, geographical location, and others may be useful to applications. The current draft focuses on network location and grouping, but the protocol may be extended to handle other Endpoint

properties.

9.6. P2P Peer Selection

This section discusses possible approaches to peer selection using ALTO information (Network Location Identifiers and associated Costs) from an ALTO Server. Specifically, the application must select which peers to use based on this and other sources of information. With this in mind, the usage of ALTO Costs is intentionally flexible, because:

Different applications may use the information differently. For example, an application that connects to just one address may have a different algorithm for selecting it than an application that connects to many.

Though initial experiments have been conducted [24], more investigation is needed to identify other methods.

In addition, the application might account for robustness, perhaps using randomized exploration to determine if it performs better without ALTO information.

9.6.1. Client-based Peer Selection

One possibility is for peer selection using ALTO costs to be done entirely by a P2P client. The following are some techniques have been proposed and/or used:

- Prefer network locations with lower ordinal rankings (i.e., higher priority) [14] [11].

- Optimistically unchoking low-cost peers with higher probability [11].

9.6.2. Server-based Peer Selection

Another possibility is for ALTO costs to be used by an Application Tracker (e.g., BitTorrent Tracker) when returning peer lists. The following are techniques that have been proposed and/or used:

- Using bandwidth matching (e.g., at an Application Tracker) and choosing solution (within bound of optimal) with minimal network cost [24].

9.6.3. Protocol Extension: 'Location-Only' Peer Selection

This section discusses a promising peer selection algorithm that was recently used in experiments with a P2P live streaming application. However, to support it, a small protocol extension would be required, but the protocol extension is general and may be applicable in other contexts as well. Feedback from the WG is greatly encouraged.

Experiments in the context of live streaming have shown significant benefits of a simple "location-only" algorithm that primarily makes use of the Network Map. A benefit of this algorithm is that it can provide a simple integration path for applications wishing to utilize ALTO.

In particular, the algorithm proceeds as follows to select an ordered list of peers for a particular incoming (or existing peer):

1. Insert into the result list a number (up to a threshold) of peers from the same PID as the incoming peer.

2. Insert into the result list a number (up to a threshold) of peers from the same ISP as the incoming peer.

3. Insert into the result list a number (up to a threshold) of peers from different ISPs than the incoming peer.

In the experiments, this algorithm was implemented at a tracker and executed for peer selection when peers initially join and when requesting new peers.

This algorithm makes two assumptions about the preferences communicated by the Network Map:

- The ISP prefers peers within the same PID to peer with each other (see Section 4); and

- The ALTO Server can indicate which PIDs describe network locations within the same ISP.

The second assumption is currently not satisfied by the ALTO protocol, but could be accomplished by including a PID attribute. For example, a boolean attribute name "Intra-Region" with value 'true' could be added to PIDs within the ALTO Server's Network Region.

10. IANA Considerations

This document request the registration of a new media type: "application/alto"

11. Security Considerations

11.1. Privacy Considerations for ISPs

ISPs must be cognizant of the network topology and provisioning information provided through ALTO Interfaces. ISPs should evaluate how much information is revealed and the associated risks. On the one hand, providing overly fine-grained information may make it easier for attackers to infer network topology. In particular, attackers may try to infer details regarding ISPs' operational policies, inter-ISP business relationships, etc. by intentionally posting a multitude of selective queries to an ALTO server (and carefully analyzing the responses). Such sophisticated attacks may reveal more information than an ISP hosting an ALTO server intends to disclose. On the other hand, revealing overly coarse-grained information may not provide benefits to network efficiency or performance improvements to ALTO Clients.

11.2. ALTO Clients

Applications using the information must be cognizant of the possibility that the information is malformed or incorrect. Even if an ALTO Server has been properly authenticated by the ALTO Client, the information provided may be malicious because the ALTO Server and its credentials have been compromised (e.g., through malware). Other considerations (e.g., relating to application performance) can be found in Section 6 of [15].

ALTO Clients should also be cognizant of revealing Network Location Identifiers (IP addresses or fine-grained PIDs) to the ALTO Server, as doing so may allow the ALTO Server to infer communication patterns. One possibility is for the ALTO Client to only rely on Network Map for PIDs and Cost Map amongst PIDs to avoid passing IP addresses of their peers to the ALTO Server.

In addition, ALTO clients should be cautious not to unintentionally or indirectly disclose the resource identifier (of which they try to improve the retrieval through ALTO-guidance), e.g., the name/ identifier of a certain video stream in P2P live streaming, to the ALTO server. Note that the ALTO Protocol specified in this document does not explicitly reveal any resource identifier to the ALTO Server. However, for instance, depending on the popularity or other

specifics (such as language) of the resource, an ALTO server could potentially deduce information about the desired resource from information such as the Network Locations the client sends as part of its request to the server.

11.3. Authentication, Integrity Protection, and Encryption

SSL/TLS can provide encryption of transmitted messages as well as authentication of the ALTO Client and Server. HTTP Basic or Digest authentication can provide authentication of the client (combined with SSL/TLS, it can additionally provide encryption and authentication of the server).

An ALTO Server may optionally use authentication (and potentially encryption) to protect ALTO information it provides. This can be achieved by digitally signing a hash of the ALTO information itself and attaching the signature to the ALTO information. There may be special use cases where encryption of ALTO information is desirable. In most cases, however, information sent out by an ALTO Server is most likely to be regarded as non-confidential information.

ISPs should be cognizant that encryption only protects ALTO information until it is decrypted by the intended ALTO Client. Digital Rights Management (DRM) techniques and legal agreements protecting ALTO information are outside of the scope of this document.

11.4. ALTO Information Redistribution

It is possible for applications to redistribute ALTO information to improve scalability. Even with such a distribution scheme, ALTO Clients obtaining ALTO information must be able to validate the received ALTO information to ensure that it was actually generated by the correct ALTO Server. Further, to prevent the ALTO Server from being a target of attack, the verification scheme must not require ALTO Clients to contact the ALTO Server to validate every set of information. Note that in any case, contacting the originating ALTO server for information validation would undermine the intended effect of redistribution and is therefore not desirable.

Note that the redistribution scheme must additionally handle details such as ensuring ALTO Clients retrieve ALTO information from the correct ALTO Server. See [18] for further discussion. Details of a particular redistribution scheme are outside the scope of this document.

To fulfill these requirements, ALTO Information meant to be redistributable contains a digital signature which includes a hash of

the ALTO information signed by the ALTO Server with its private key. The corresponding public key should either be part of the ALTO information itself, or it could be included in the server capability response. The public key SHOULD include the hostname of the ALTO Server and it SHOULD be signed by a trusted authority (i.e., in a certificate). This enables an ALTO client retrieving redistributed ALTO information to verify the correctness of the ALTO Server's signature, given that it trusts the authority which signed the ALTO Server's certificate. Note that in some cases this requires that the retrieving ALTO Client must be able to derive a transitive certificate chain (including a Root-CA) to the trusted authority which signed the ALTO Server's certificate. This requirement may not be possible to fulfill between every ALTO Client / ALTO Server combination on the Internet due to the lack of a world-wide public key infrastructure.

11.5. Denial of Service

ISPs should be cognizant of the workload at the ALTO Server generated by certain ALTO Queries, such as certain queries to the Map Filtering Service and Ranking Service. In particular, queries which can be generated with low effort but result in expensive workloads at the ALTO Server could be exploited for Denial-of-Service attacks. For instance, a simple ALTO query with n Source Network Locations and m Destination Network Locations can be generated fairly easily but results in the computation of n*m Path Costs between pairs by the ALTO Server (see Section 5.2). One way to limit Denial-of-Service attacks is to employ access control to the ALTO server. Another possible mechanism for an ALTO Server to protect itself against a multitude of computationally expensive bogus requests is to demand that each ALTO Client to solve a computational puzzle first before allocating resources for answering a request (see, e.g., [25]). The current specification does not use such computational puzzles, and discussion regarding tradeoffs of such an approach would be needed before including such a technique in the ALTO Protocol.

ISPs should also leverage the fact that the the Map Service allows ALTO Servers to pre-generate maps that can be useful to many ALTO Clients.

11.6. ALTO Server Access Control

In order to limit access to an ALTO server (e.g., for an ISP to only allow its users to access its ALTO server, or to prevent Denial-of-Service attacks by arbitrary hosts from the Internet), an ALTO server may employ access control policies. Depending on the use-case and scenario, an ALTO server may restrict access to its services more strictly or rather openly (see [26] for a more detailed discussion on

this issue).

12. References

12.1. Normative References

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

[2] Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

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

[4] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[5] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[6] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[7] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for (NAT) (STUN)", draft-ietf-behave-rfc3489bis-18 (work in progress), July 2008.

12.2. Informative References

[8] Kiesel, S., Popkin, L., Previdi, S., Woundy, R., and Y. Yang, "Application-Layer Traffic Optimization (ALTO) Requirements", draft-kiesel-alto-reqs-01 (work in progress), November 2008.

[9] Alimi, R., Pasko, D., Popkin, L., Wang, Y., and Y. Yang, "P4P: Provider Portal for P2P Applications", draft-p4p-framework-00 (work in progress), November 2008.

[10] Wang, Y., Alimi, R., Pasko, D., Popkin, L., and Y. Yang, "P4P Protocol Specification", draft-wang-alto-p4p-specification-00 (work in progress), March 2009.

[11] Shalunov, S., Penno, R., and R. Woundy, "ALTO Information Export Service", draft-shalunov-alto-infoexport-00 (work in progress), October 2008.

[12] Das, S. and V. Narayanan, "A Client to Service Query Response Protocol for ALTO", draft-saumitra-alto-queryresponse-00 (work in progress), March 2009.

[13] Das, S., Narayanan, V., and L. Dondeti, "ALTO: A Multi Dimensional Peer Selection Problem", draft-saumitra-alto-multi-ps-00 (work in progress), October 2008.

[14] Akonjang, O., Feldmann, A., Previdi, S., Davie, B., and D. Saucez, "The PROXIDOR Service", draft-akonjang-alto-proxidor-00 (work in progress), March 2009.

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

[16] Yang, Y., Popkin, L., Penno, R., and S. Shalunov, "An Architecture of ALTO for P2P Applications", draft-yang-alto-architecture-00 (work in progress), March 2009.

[17] Zyp, K., "A JSON Media Type for Describing the Structure and Meaning of JSON Documents", draft-zyp-json-schema-02 (work in progress), March 2010.

[18] Yingjie, G., Alimi, R., and R. Even, "ALTO Information Redistribution", draft-gu-alto-redistribution-03 (work in progress), July 2010.

[19] Song, H., Even, R., Pascual, V., and Y. Zhang, "Application-Layer Traffic Optimization (ALTO): Discover ALTO Servers", draft-song-alto-server-discovery-00 (work in progress), March 2009.

[20] Penno, R. and J. Medved, "ALTO and IPv4/IPv6 Co-existence and Transition", draft-penno-alto-ipv4v6-00 (work in progress), June 2010.

[21] Baker, F., Li, X., and C. Bao, "Framework for IPv4/IPv6 Translation", draft-baker-behave-v4v6-framework-02 (work in progress), February 2009.

[22] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Address Translation (NAT66)", draft-mrw-behave-nat66-02 (work in progress), March 2009.

[23] "Bittorrent Protocol Specification v1.0", http://wiki.theory.org/BitTorrentSpecification, 2009.

[24] H. Xie, YR. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz., "P4P: Provider Portal for (P2P) Applications", In SIGCOMM 2008.

[25] Jennings, C., "Computational Puzzles for SPAM Reduction in SIP", draft-jennings-sip-hashcash-06 (work in progress), July 2007.

[26] Stiemerling, M. and S. Kiesel, "ALTO Deployment Considerations", draft-stiemerling-alto-deployments-04 (work in progress), July 2010.

Appendix A. Acknowledgments

Thank you to Jan Seedorf for contributions to the Security Considerations section. We would like to thank Yingjie Gu and Roni Even for helpful input and design concerning ALTO Information redistribution.

We would like to thank the following people whose input and involvement was indispensable in achieving this merged proposal:

Obi Akonjang (DT Labs/TU Berlin),

Saumitra M. Das (Qualcomm Inc.),

Syon Ding (China Telecom),

Doug Pasko (Verizon),

Laird Popkin (Pando Networks),

Satish Raghunath (Juniper Networks),

Albert Tian (Ericsson/Redback),

Yu-Shun Wang (Microsoft),

David Zhang (PPLive),

Yunfei Zhang (China Mobile).

We would also like to thank the following additional people who were involved in the projects that contributed to this merged document: Alex Gerber (AT&T), Chris Griffiths (Comcast), Ramit Hora (Pando Networks), Arvind Krishnamurthy (University of Washington), Marty Lafferty (DCIA), Erran Li (Bell Labs), Jin Li (Microsoft), Y. Grace

Liu (IBM Watson), Jason Livingood (Comcast), Michael Merritt (AT&T), Ingmar Poese (DT Labs/TU Berlin), James Royalty (Pando Networks), Damien Saucez (UCL) Thomas Scholl (AT&T), Emilio Sepulveda (Telefonica), Avi Silberschatz (Yale University), Hassan Sipra (Bell Canada), Georgios Smaragdakis (DT Labs/TU Berlin), Haibin Song (Huawei), Oliver Spatscheck (AT&T), See-Mong Tang (Microsoft), Jia Wang (AT&T), Hao Wang (Yale University), Ye Wang (Yale University), Haiyong Xie (Yale University).

Appendix B. Authors

[[Comment.2: RFC Editor: Please move information in this section to the Authors' Addresses section at publication time.]]

Stefano Previdi
Cisco

Email: sprevidi@cisco.com

Stanislav Shalunov
BitTorrent

Email: shalunov@bittorrent.com

Richard Woundy
Comcast

Richard_Woundy@cable.comcast.com

Authors' Addresses

Richard Alimi (editor)
Yale University

Email: richard.alimi@yale.edu

Reinaldo Penno (editor)
Juniper Networks
1194 N Mathilda Avenue
Sunnyvale, CA
USA

Email: rpenno@juniper.net

Y. Richard Yang (editor)
Yale University

Email: yry@cs.yale.edu