

Workgroup: ALTO  
Internet-Draft:  
draft-ietf-alto-new-transport-22  
Published: 3 January 2024  
Intended Status: Standards Track  
Expires: 6 July 2024  
Authors: K. Gao R. Schott  
Sichuan University Deutsche Telekom  
Y. R. Yang L. Delwiche L. Keller  
Yale University Yale University Yale University  
**The ALTO Transport Information Publication Service**

## Abstract

The ALTO Protocol (RFC 7285) leverages HTTP/1.1 and is designed for the simple, sequential request-reply use case, in which an ALTO client requests a sequence of information resources and the server responds with the complete content of each resource one at a time.

ALTO incremental updates using Server-Sent Events (SSE) (RFC 8895) defines a multiplexing protocol on top of HTTP/1.x, so that an ALTO server can incrementally push resource updates to clients whenever monitored network information resources change, allowing the clients to monitor multiple resources at the same time. However, HTTP/2 and later versions already support concurrent, non-blocking transport of multiple streams in the same HTTP connection.

To take advantage of newer HTTP features, this document introduces the ALTO Transport Information Publication Service (TIPS). TIPS uses an incremental RESTful design to give an ALTO client the new capability to explicitly, concurrently (non-blocking) request (pull) specific incremental updates using native HTTP/2 or HTTP/3, while still functioning for HTTP/1.1.

## Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Application-Layer Traffic Optimization Working Group mailing list ([alto@ietf.org](mailto:alto@ietf.org)), which is archived at <https://mailarchive.ietf.org/arch/browse/alto/>.

Source for this draft and an issue tracker can be found at <https://github.com/ietf-wg-alto/draft-ietf-alto-new-transport>.

## 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 6 July 2024.

## Copyright Notice

Copyright (c) 2024 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. [Introduction](#)
  - 1.1. [Requirements Language](#)
  - 1.2. [Notations](#)
2. [TIPS Overview](#)
  - 2.1. [Transport Requirements](#)
  - 2.2. [TIPS Terminology](#)
3. [TIPS Updates Graph](#)
  - 3.1. [Basic Data Model of Updates Graph](#)
  - 3.2. [Updates Graph Modification Invariants](#)
4. [TIPS Workflow and Resource Location Schema](#)
  - 4.1. [Workflow](#)
  - 4.2. [Resource Location Schema](#)
5. [TIPS Information Resource Directory \(IRD\) Announcement](#)
  - 5.1. [Media Type](#)
  - 5.2. [Capabilities](#)
  - 5.3. [Uses](#)
  - 5.4. [An Example](#)
6. [TIPS Management](#)
  - 6.1. [Open Request](#)
  - 6.2. [Open Response](#)

- [6.3. Open Example](#)
  - [6.3.1. Basic Example](#)
  - [6.3.2. Example using Digest Authentication](#)
  - [6.3.3. Example using ALTO/SSE](#)
- [7. TIPS Data Transfers - Client Pull](#)
  - [7.1. Request](#)
  - [7.2. Response](#)
  - [7.3. Example](#)
  - [7.4. New Next Edge Recommendation](#)
    - [7.4.1. Request](#)
    - [7.4.2. Response](#)
    - [7.4.3. Example](#)
- [8. Operation and Processing Considerations](#)
  - [8.1. Considerations for Load Balancing](#)
  - [8.2. Considerations for Cross-Resource Dependency Scheduling](#)
  - [8.3. Considerations for Managing Shared TIPS Views](#)
  - [8.4. Considerations for Offering Shortcut Incremental Updates](#)
- [9. Security Considerations](#)
  - [9.1. TIPS: Denial-of-Service Attacks](#)
  - [9.2. ALTO Client: Update Overloading or Instability](#)
- [10. IANA Considerations](#)
  - [10.1. application/alto-tips+json Media Type](#)
  - [10.2. application/alto-tipsparams+json Media Type](#)
- [11. References](#)
  - [11.1. Normative References](#)
  - [11.2. Informative References](#)
- [Appendix A. A High-Level Deployment Model](#)
- [Appendix B. Conformance to "Building Protocols with HTTP" Best Current Practices](#)
- [Appendix C. Push-mode TIPS using HTTP Server Push](#)
- [Appendix D. Persistent HTTP Connections](#)
- [Acknowledgments](#)
- [Authors' Addresses](#)

## **1. Introduction**

Application-Layer Traffic Optimization (ALTO) provides means for network applications to obtain network status information. So far, the ALTO information can be transported in two ways:

1. The ALTO base protocol [[RFC7285](#)], which is designed for the simple use case in which an ALTO client requests a network information resource, and the server sends the complete content of the requested information (if any) resource to the client.
2. ALTO incremental updates using Server-Sent Events (ALTO/SSE) [[RFC8895](#)], which is designed for an ALTO client to indicate to the server that it wants to receive updates for a set of resources and the server can then concurrently and

incrementally push updates to that client whenever monitored resources change.

Both protocols are designed for HTTP/1.1 [[RFC9112](#)], but HTTP/2 [[RFC9113](#)] and HTTP/3 [[RFC9114](#)] can support HTTP/1.1 workflows. However, HTTP/2 and HTTP/3 provide features that can improve certain properties of ALTO and ALTO/SSE.

\*First, consider the ALTO base protocol, which is designed to transfer only complete information resources. A client can run the base protocol on top of HTTP/2 or HTTP/3 to request multiple information resources in concurrent streams, but each request must be for a complete information resource: there is no capability for the server to transmit incremental updates. Hence, there can be a large overhead when the client already has an information resource and then there are small changes to the resource.

\*Next, consider ALTO/SSE [[RFC8895](#)]. Although ALTO/SSE can transfer incremental updates, it introduces a customized multiplexing protocol on top of HTTP, assuming a total-order message channel from the server to the client. The multiplexing design does not provide naming (i.e., a resource identifier) to individual incremental updates. Such a design cannot use concurrent data streams available in HTTP/2 and HTTP/3, because both cases require a resource identifier. Additionally, ALTO/SSE is a push-only protocol, which denies the client flexibility in choosing how and when it receives updates.

To mitigate these concerns, this document introduces a new ALTO service called the Transport Information Publication Service (TIPS). TIPS uses an incremental RESTful design to provide an ALTO client with a new capability to explicitly, concurrently issue non-blocking requests for specific incremental updates using native HTTP/2 or HTTP/3, while still functioning for HTTP/1.1.

While ALTO/SSE [[RFC8895](#)] and TIPS both can transport incremental updates of ALTO information resources to clients, they have different design goals. The TIPS extension enables more scalable and robust distribution of incremental updates, but is missing the session management and built-in server push capabilities of ALTO/SSE. From the performance perspective, TIPS is optimizing throughput by leveraging concurrent and out-of-order transport of data, while ALTO/SSE is optimizing latency as new events can be immediately transferred to the clients without waiting for another round of communication when there are multiple updates. Thus, we do not see TIPS as a replacement but as a complement of ALTO/SSE. One example of combining these two extensions is as shown in [Section 6.3.3](#).

Note that future extensions may leverage server push, a feature of HTTP/2 [[RFC9113](#)] and HTTP/3 [[RFC9114](#)], as an alternative of SSE. We discuss why this alternative design is not ready in [Appendix C](#).

Specifically, this document specifies:

- \*Extensions to the ALTO Protocol for dynamic subscription and efficient uniform update delivery of an incrementally changing network information resource.

- \*A new resource type that indicates the TIPS updates graph model for a resource.

- \*URI patterns to fetch the snapshots or incremental updates.

Some operational complexities that must be taken into consideration when implementing this extension are discussed in [Section 8](#), including load balancing [Section 8.1](#), fetching and processing incremental updates of dependent resources [Section 8.2](#)

[Appendix B](#) discusses to what extent the TIPS design adheres to the Best Current Practices for building protocols with HTTP [[RFC9205](#)].

## 1.1. Requirements Language

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

## 1.2. Notations

This document uses the same syntax and notations as introduced in [Section 8.2](#) of [[RFC7285](#)] to specify the extensions to existing ALTO resources and services.

## 2. TIPS Overview

### 2.1. Transport Requirements

The ALTO Protocol and its extensions support two transport mechanisms: First, a client can directly request an ALTO resource and obtain a complete snapshot of that ALTO resource, as specified in the base protocol [[RFC7285](#)]; Second, a client can subscribe to incremental changes of one or multiple ALTO resources using the incremental update extension [[RFC8895](#)], and a server pushes the updates to the client through Server Sent Events (SSE).

However, the current transport mechanisms are not optimized for storing, transmitting, and processing (incremental) updates of ALTO information resources. Specifically, the new transport mechanism must satisfy the following requirements:

**Incremental updates:** Incremental updates only maintain and transfer the "diff" upon changes. Thus, it is more efficient than storing and transferring the full updates, especially when the change of an ALTO resource is minor. The base protocol does not support incremental updates and the current incremental update mechanism in [[RFC8895](#)] has limitations (as discussed below).

**Concurrent, non-blocking update transmission:** When a client needs to receive and apply multiple incremental updates, it is desired to transmit the updates concurrently to fully utilize the bandwidth and to reduce head-of-line blocking. The ALTO incremental update extension [[RFC8895](#)], unfortunately, does not satisfy this requirement -- even though the updates can be multiplexed by the server to avoid head-of-line blocking between multiple resources, the updates are delivered sequentially and can suffer from head-of-line blocking inside the connection, for example, when there is a packet loss.

**Long-polling updates:** Long-polling updates can reduce the time to send the request, making it possible to achieve sub-RTT transmission of ALTO incremental updates. In [[RFC8895](#)], this requirement is fulfilled using server-sent event (SSE) and is still desired in the ALTO new transport.

**Backward compatibility:** While some of the previous requirements are offered by HTTP/2 [[RFC9113](#)] and HTTP/3 [[RFC9114](#)], it is desired that the ALTO new transport mechanism can work with HTTP/1.1 as many development tools and current ALTO implementations are based on HTTP/1.1.

The ALTO new transport specified in this document satisfies all the design requirements:

\*This document reuses the data format introduced in [[RFC8895](#)] that enables incremental updates using JSON patches or merge patches.

\*This document introduce a unified data model to describe the changes (snapshots and incremental updates) of an ALTO resource, referred to as a TIPS view. In the data model, snapshots and incremental updates are indexed as individual HTTP resources following a unified naming convention, independent of the HTTP version. Thus, these updates can be concurrently requested and be transferred in a non-blocking manner either by using multiple

connections or leveraging multiplexed data transfer offered by HTTP/2 or HTTP/3.

\*The unified naming convention is based on a monotonically increasing sequence number, making it possible for a client to construct the URL of a future update and send a long-polling request.

\*The unified naming convention is independent of the HTTP versions and can operate atop HTTP/1.1, HTTP/2 or HTTP/3.

This document assumes the deployment model discussed in [Appendix A](#).

## 2.2. TIPS Terminology

In addition to the terms defined in [[RFC7285](#)], this document uses the following terms:

**Transport Information Publication Service (TIPS):** Is a new type of ALTO service, as specified in this document, to enable a uniform transport mechanism for updates of an incrementally changing ALTO network information resource.

**Network information resource:** Is a piece of retrievable information about network state, per [[RFC7285](#)].

**TIPS view (tv):** Is defined in this document to be the container of incremental transport information about the network information resource. The TIPS view has one basic component, updates graph (ug), but may include other transport information.

**Updates graph (ug):** Is a directed, acyclic graph whose nodes represent the set of versions of an information resource, and edges the set of update items to compute these versions. An ALTO map service (e.g., Cost Map, Network Map) may need only a single updates graph. A dynamic network information service (e.g., Filtered Cost Map) may create an updates graph (within a new TIPS view) for each unique request. Encoding of a updates graph is specified in [Section 6.1](#).

**Version:** Represents a historical content of an information resource. For an information resource, each version is associated with and uniquely identified by a monotonically and consecutively increased sequence number. This document uses the term "version s" to refer to the version associated with sequence number "s". Version is encoded as a JSONNumber, as specified in [Section 6.1](#).

**Start sequence number (start-seq):** Is the smallest non-zero sequence number in an updates graph.

**End sequence number (end-seq):**

Is the largest sequence number in an updates graph.

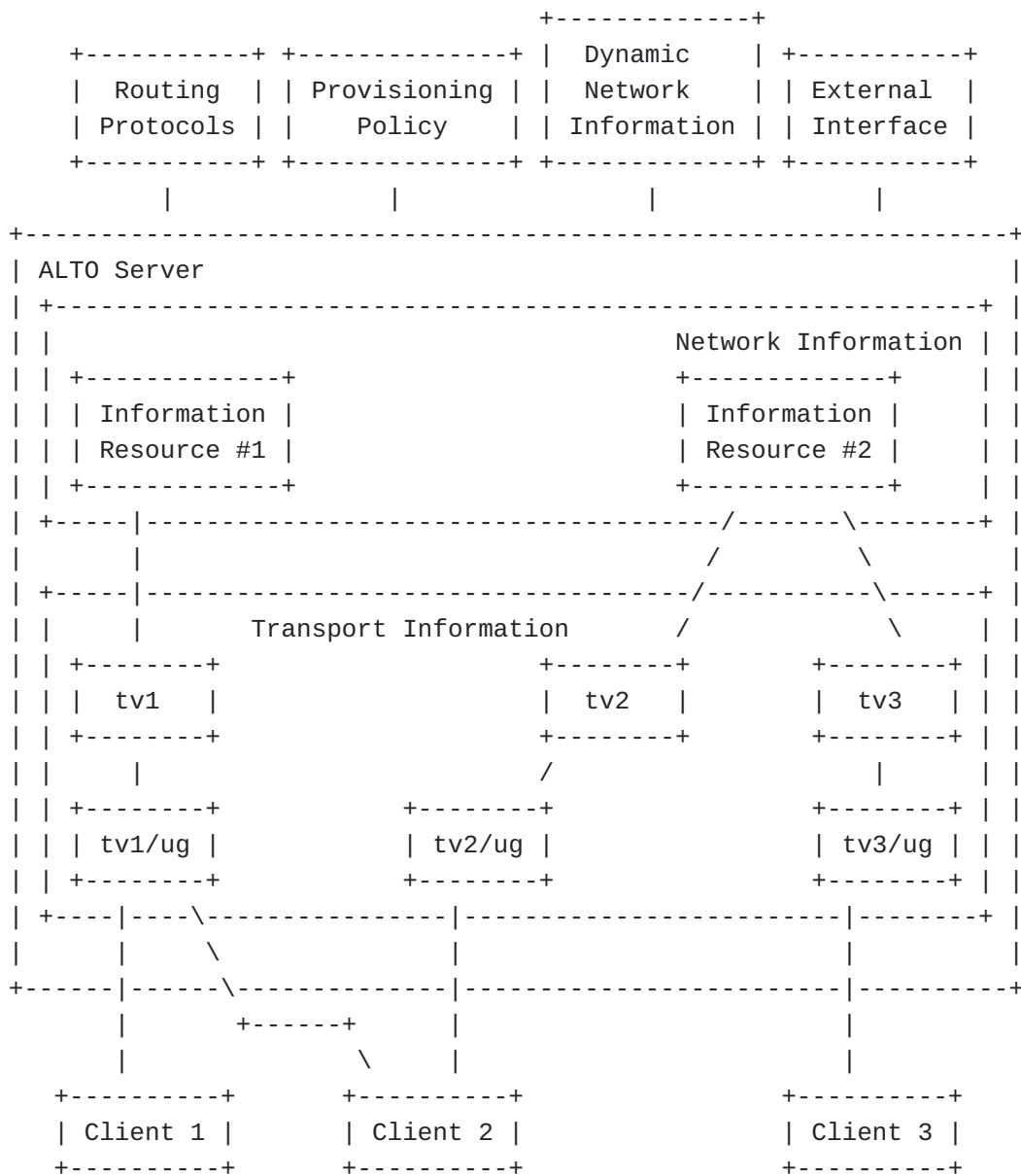
**Snapshot:** Is a full replacement of a resource and is contained within an updates graph.

**Incremental update:** Is a partial replacement of a resource contained within an updates graph, codified in this document as a JSON Merge Patch or JSON Patch. An incremental update is mandatory if the source version ( $i$ ) and target version ( $j$ ) are consecutive, i.e.,  $i + 1 = j$ , and optional or a shortcut otherwise. Mandatory incremental updates are always in an updates graph, while optional/shortcut incremental updates may or may not be included in an updates graph.

**Update item:** Refers to the content on an edge of the updates graph, which can be either a snapshot or an incremental update. An update item can be considered as a pair ( $op, data$ ) where  $op$  denotes whether the item is an incremental update or a snapshot, and  $data$  is the content of the item.

**ID# $i$ -# $j$ :** Denotes the update item on a specific edge in the updates graph to transition from version  $i$  to version  $j$ , where  $i$  and  $j$  are the sequence numbers of the source node and the target node of the edge, respectively.





tvi = TIPS view i  
tvi/ug = incremental updates graph associated with tvi

Figure 1: Overview of ALTO TIPS

[Figure 1](#) shows an example illustrating an overview of the ALTO TIPS service. The server provides the TIPS service of two information resources (#1 and #2) where #1 is an ALTO map service, and #2 is a filterable service. There are 3 ALTO clients (Client 1, Client 2, and Client 3) that are connected to the ALTO server.

Each client uses the TIPS view to retrieve updates. Specifically, a TIPS view (tv1) is created for the map service #1, and is shared by multiple clients. For the filtering service #2, two different TIPS

views (tv2 and tv3) are created upon different client requests with different filter sets.

### 3. TIPS Updates Graph

In order to provide incremental updates for a resource, an ALTO server creates an updates graph, which is a directed, acyclic graph that contains a sequence of incremental updates and snapshots (collectively called update items) of a network information resource.

#### 3.1. Basic Data Model of Updates Graph

For each resource (e.g., a cost map, a network map), the incremental updates and snapshots can be represented using the following directed acyclic graph model, where the server tracks the change of the resource maps with version IDs that are assigned sequentially (i.e., incremented by 1 each time):

- \*Each node in the graph is a version of the resource, which is identified by a sequence number (defined as a JSONNumber). Version 0 is reserved as the initial state (empty/null).

- \*A tag identifies the content of a node. A tag has the same format as the "tag" field in [Section 10.3](#) of [\[RFC7285\]](#) and is valid only within the scope of resource.

- \*Each edge is an update item. In particular, the edge from *i* to *j* is the update item to transit from version *i* to version *j*.

- \*Version is path-independent (different paths arrive at the same version/node has the same content)

A concrete example is shown in [Figure 2](#). There are 7 nodes in the graph, representing 7 different versions of the resource. Edges in the figure represent the updates from the source version to the target version. Thick lines represent mandatory incremental updates (e.g., ID103-104), dotted lines represent optional incremental updates (e.g., ID103-105), and thin lines represent snapshots (e.g., ID0-103). Note that node content is path independent: the content of node *v* can be obtained by applying the updates from any path that ends at *v*. For example, assume the latest version is 105 and a client already has version 103. The base version of the client is 103 as it serves as a base upon which incremental updates can be applied. The target version 105 can either be directly fetched as a snapshot, computed incrementally by applying the incremental updates between 103 and 104, then 104 and 105, or if the optional update from 103 to 105 exists, computed incrementally by taking the "shortcut" path from 103 to 105.

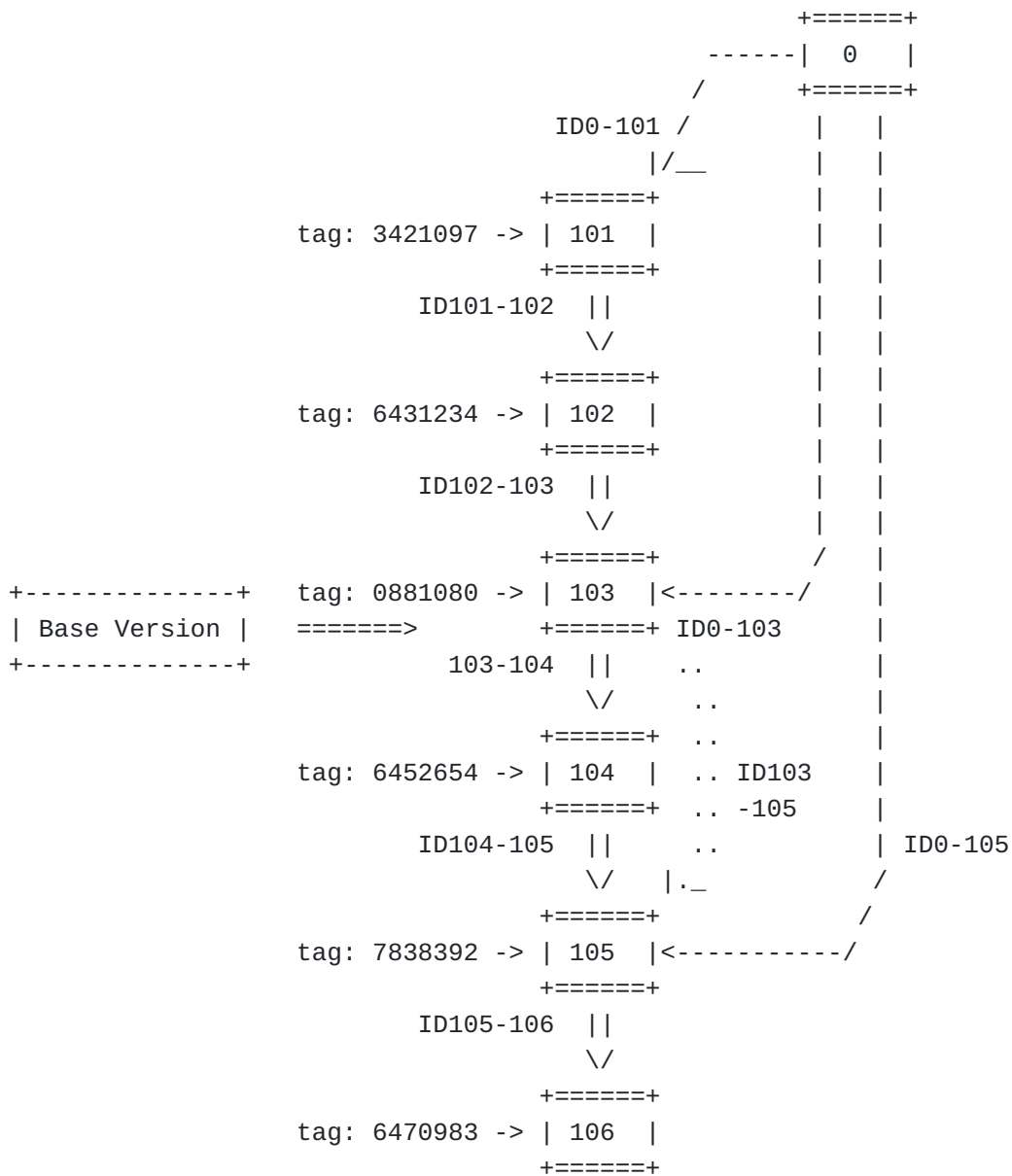


Figure 2: TIPS Model Example

### 3.2. Updates Graph Modification Invariants

A server may change its updates graph (to compact, to add nodes, etc.), but it must ensure that any resource state that it makes available is reachable by clients, either directly via a snapshot (that is, relative to  $\emptyset$ ) or indirectly by requesting an earlier snapshot and a contiguous set of incremental updates. Additionally, to allow clients to proactively construct URIs for future update items, the ID of each added node in the updates graph must increment

contiguously by 1. More specifically, the updates graph **MUST** satisfy the following invariants:

\*Continuity: At any time, let  $n_s$  denote the smallest non-zero version (i.e., start-seq) in the update graph and  $n_e$  denote the latest version (i.e., end-seq). Then any version in between  $n_s$  and  $n_e$  **MUST** also exist. This implies that the incremental update from  $n_i$  to  $n_i + 1$  exists for any  $n_s \leq n_i \leq n_e$ , and all versions in the update graph (except 0) is an integer interval  $[n_s, n_e]$ .

\*Feasibility: Let  $n_s$  denote the start-seq in the update graph. The server **MUST** provide a snapshot of  $n_s$  and, in other words, there is always a direct link to  $n_s$  in the update graph.

\*"Right shift" only: Assume a server provides versions in  $[n_1, n_2]$  at time  $t$  and versions in  $[n_1', n_2']$  at time  $t'$ . If  $t' > t$ , then  $n_1' \geq n_1$  and  $n_2' \geq n_2$ .

For example, consider the case that a server compacts a resource's updates graph to conserve space, using the example model in [Section 3.1](#). Assume at time 0, the server provides the versions {101, 102, 103, 104, 105, 106}. At time 1, both {103, 104, 105, 106} and {105, 106} are valid sets. However, {102, 103, 104, 105, 106} and {104, 105, 106} are not valid sets as there is no snapshot to version 102 or 104 in the update graph. Thus, there is a risk that the right content of version 102 (in the first example) or 104 (in the second example) cannot be obtained by a client that does not have the previous version 101 or 103, respectively.

## 4. TIPS Workflow and Resource Location Schema

### 4.1. Workflow

At a high level, an ALTO client first uses the TIPS service (denoted as TIPS-F and F is for frontend) to indicate the information resource(s) that the client wants to monitor. For each requested resource, the server returns a JSON object that contains a URI, which points to the root of a TIPS view (denoted as TIPS-V), and a summary of the current view, which contains the information to correctly interact with the current view. With the URI to the root of a TIPS view, clients can construct URIs (see [Section 4.2](#)) to fetch incremental updates.

An example workflow is shown in [Figure 3](#). After the TIPS-F service receives the request from the client to monitor the updates of an ALTO resource, it creates a TIPS view service and returns the corresponding information to the client. The URI points to that specific TIPS-V instance and the summary contains the start-seq and end-seq of the update graph, and a server-recommended edge to consume first, e.g., from  $i$  to  $j$ .

An ALTO client can then continuously pull each additional update with the information. For example, the client in [Figure 3](#) first fetches the update from  $i$  to  $j$ , and then from  $j$  to  $j+1$ . Note that the update item at `<tips-view-uri>/ug/<j>/<j+1>` may not yet exist, so the server holds the request until the update becomes available (long polling).

A server **MAY** close a TIPS view at any time, e.g., under high system load or due to client inactivity. In the event that a TIPS view is closed, an edge request will receive error code 404 in response, and the client will have to request a new TIPS view URI.

If resources allow, a server **SHOULD** avoid closing TIPS views that have active polling edge requests or have recently served responses until clients have had a reasonable interval to request the next update, unless guided by specific control policies.

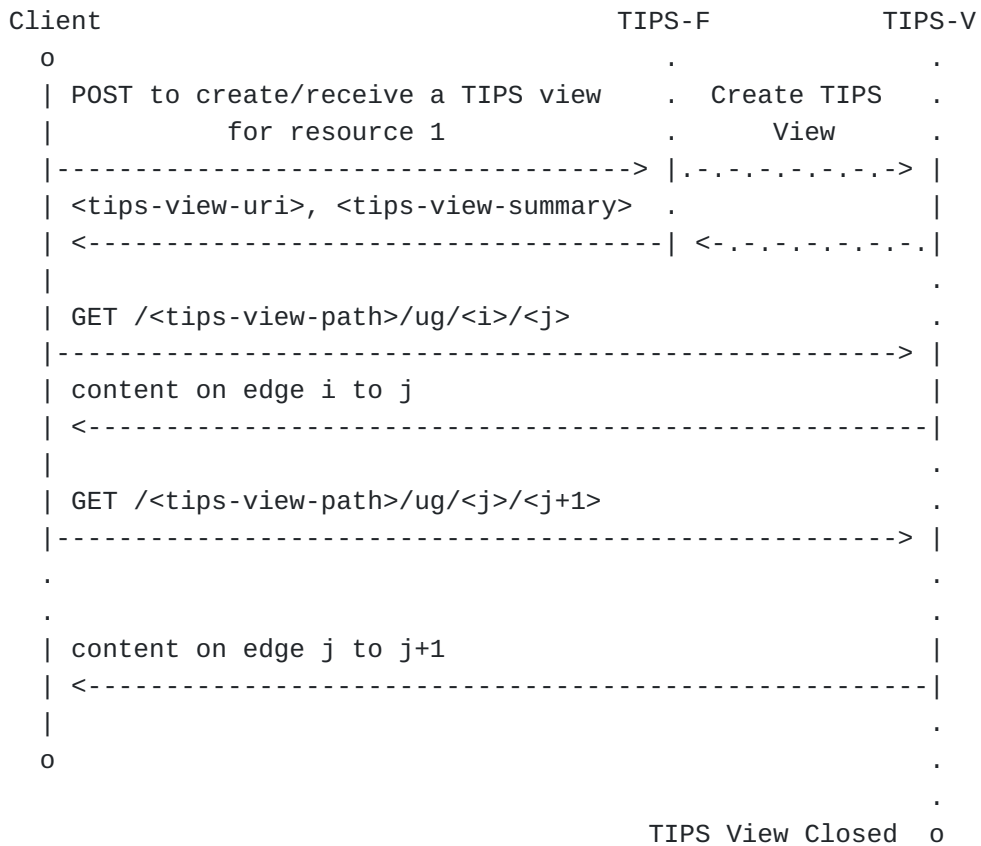


Figure 3: ALTO TIPS Workflow Supporting Client Pull

#### 4.2. Resource Location Schema

The resource location schema defines how a client constructs URI to fetch incremental updates.

To access each update in an updates graph, consider the model represented as a "virtual" file system (adjacency list), contained within the root of a TIPS view URI (see [Section 6.2](#) for the definition of tips-view-uri). For example, assuming that the update graph of a TIPS view is as shown in [Figure 2](#), the location schema of this TIPS view will have the format as in [Figure 4](#).

```
<tips-view-path> // root path to a TIPS view
|_ ug // updates graph
|  |_ 0
|  |  |_ 101 // full 101 snapshot
|  |  |_ 103
|  |  \_ 105
|  |  |_ 101
|  |  \_ 102 // 101 -> 102 incremental update
|  |  |_ 102
|  |  \_ 103
|  |  |_ 103
|  |  |_ 104
|  |  \_ 105 // optional shortcut 103 -> 105 incr. update
|  |  |_ 104
|  |  \_ 105
|  \_ 105
|     \_ 106
\_ ...
```

Figure 4: Location Schema Example

TIPS uses this directory schema to generate template URIs which allow clients to construct the location of incremental updates after receiving the tips-view-uri from the server. The generic template for the location of the update item on the edge from node 'i' to node 'j' in the updates graph is:

```
<tips-view-uri>/ug/<i>/<j>
```

Due to the sequential nature of the update item IDs, a client can long poll a future update that does not yet exist (e.g., the incremental update from 106 to 107) by constructing the URI for the next edge that will be added, starting from the sequence number of the current last node (denoted as end-seq) in the graph to the next sequential node (with the sequence number of end-seq + 1):

```
<tips-view-uri>/ug/<end-seq>/<end-seq + 1>
```

Incremental updates of a TIPS view are read-only. Thus, they are fetched using the HTTP GET method.

## 5. TIPS Information Resource Directory (IRD) Announcement

To announce a TIPS information resource in the information resource directory (IRD), an ALTO server **MUST** specify the "media-type", "capabilities" and "uses" as follows.

### 5.1. Media Type

The media type of the Transport Information Publication Service resource is "application/alto-tips+json".

### 5.2. Capabilities

The capabilities field of TIPS is modeled on that defined in Section 6.3 of [[RFC8895](#)].

Specifically, the capabilities are defined as an object of type TIPSCapabilities:

```
object {
  IncrementalUpdateMediaTypes incremental-change-media-types;
} TIPSCapabilities;

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

Figure 5: TIPSCapabilities

with field:

**incremental-change-media-types:** If a TIPS can provide updates with incremental changes for a resource, the "incremental-change-media-types" field has an entry for that resource-id, and the value is the supported media types of incremental changes, separated by commas. For the implementation of this specification, this **MUST** be "application/merge-patch+json", "application/json-patch+json", or "application/merge-patch+json,application/json-patch+json", unless defined by a future extension.

When choosing the media types to encode incremental updates for a resource, the server **MUST** consider the limitations of the encoding. For example, when a JSON merge patch specifies that the value of a field is null, its semantics are that the field is removed from the target and hence the field is no longer defined (i.e., undefined). 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 **MUST**

choose JSON patch over JSON merge patch if JSON patch is indicated as a capability of the TIPS. If the server does not support JSON patch to handle such a case, the server then needs to send a full replacement.

### 5.3. Uses

The "uses" attribute **MUST** be an array with the resource-ids of every network information resource for which this TIPS can provide service.

This set **MAY** be any subset of the ALTO server's network information resources and **MAY** include resources defined in linked IRDs. However, it is **RECOMMENDED** that the ALTO server selects a set that is closed under the resource dependency relationship. That is, if a TIPS' "uses" set includes resource R1 and resource R1 depends on ("uses") resource R0, then the TIPS' "uses" set should include R0 as well as R1. For example, if a TIPS provides a TIPS view for a cost map, it should also provide a TIPS view for the network map upon which that cost map depends.

If the set is not closed, at least one resource R1 in the "uses" field of a TIPS depends on another resource R0 which is not in the "uses" field of the same TIPS. Thus, a client cannot receive incremental updates for R0 from the same TIPS service. If the client observes in an update of R1 that the version tag for R0 has changed, it must request the full content of R0, which is likely to be less efficient than receiving the incremental updates of R0.

### 5.4. An Example

Extending the IRD example in Section 8.1 of [\[RFC8895\]](#), [Figure 6](#) is the IRD of an ALTO server supporting ALTO base protocol, ALTO/SSE, and ALTO TIPS.



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

```
"update-my-costs-tips": {
  "uri": "https://alto.example.com/updates-new/costs",
  "media-type": "application/alto-tips+json",
  "accepts": "application/alto-tipsparams+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",
      "my-simple-filtered-cost-map": "application/merge-patch+json"
    }
  }
},
"tips-sse": {
  "uri": "https://alto.example.com/updates/tips",
  "media-type": "text/event-stream",
  "accepts": "application/alto-updatestreamparams+json",
  "uses": [ "update-my-costs-tips" ],
  "capabilities": {
    "incremental-change-media-types": {
      "update-my-costs-tips": "application/merge-patch+json"
    }
  }
}
```

Figure 6: Example of an ALTO Server Supporting ALTO Base Protocol, ALTO/SSE, and ALTO TIPS

Note that it is straightforward for an ALTO server to run HTTP/2 and support concurrent retrieval of multiple resources such as "my-network-map" and "my-routingcost-map" using multiple HTTP/2 streams.

The resource "update-my-costs-tips" provides an ALTO TIPS service, and this is indicated by the media-type "application/alto-tips+json".

## 6. TIPS Management

Upon request, a server sends a TIPS view to a client. This TIPS view may be created at the time of the request or may already exist (either because another client has already created a TIPS view for the same requested network resource or because the server perpetually maintains a TIPS view for an often-requested resource).

### 6.1. Open Request

An ALTO client requests that the server provide a TIPS view for a given resource by sending an HTTP POST body with the media type "application/alto-tipsparams+json". That body contains a JSON object of type TIPSReq, where:

```
object {
  ResourceID  resource-id;
  [JSONString tag;]
  [Object     input;]
} TIPSReq;
```

Figure 7: TIPSReq

with the following fields:

**resource-id:** The resource-id of an ALTO resource and **MUST** be in the TIPS' "uses" list ([Section 5](#)). If a client does not support all incremental methods from the set announced in the server's capabilities, the client **MUST NOT** use the TIPS service.

**tag:** If the resource-id is a GET-mode resource with a version tag (or "vtag"), as defined in Section 10.3 of [[RFC7285](#)], and the ALTO client has previously retrieved a version of that resource from ALTO, the ALTO client **MAY** set the "tag" field to the tag part of the client's version of that resource. The server **MAY** use the tag when calculating a recommended starting edge for the client to consume. Note that the client **MUST** support all

incremental methods from the set announced in the server's capabilities for this resource.

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

## 6.2. Open Response

The response to a valid request **MUST** be a JSON object of type AddTIPSResponse, denoted as media type "application/alto-tips+json":

```
object {
  URI          tips-view-uri;
  TIPSTableViewSummary tips-view-summary;
} AddTIPSResponse;

object {
  UpdatesGraphSummary updates-graph-summary;
} TIPSTableViewSummary;

object {
  JSONNumber start-seq;
  JSONNumber end-seq;
  StartEdgeRec start-edge-rec;
} UpdatesGraphSummary;

object {
  JSONNumber seq-i;
  JSONNumber seq-j;
} StartEdgeRec;
```

Figure 8: AddTIPSResponse

with the following fields:

**tips-view-uri:** URI to the requested TIPS view. The value of this field **MUST** have the following format:

```
scheme "://" tips-view-host "/" tips-view-path
```

```
tips-view-host = host [ ":" port]
```

```
tips-view-path = path
```

where scheme **MUST** be "http" or "https" unless specified by a future extension, and host, port and path are as specified in Sections 3.2.2, 3.2.3, and 3.3 in [RFC3986]. An ALTO server **SHOULD** use the "https" scheme unless the contents of the TIPS view are intended to be publicly accessible and does not raise

security concerns. The field **MUST** contain only ASCII characters. In case the original URL contains international characters (e.g., in the domain name), the ALTO server implementation **MUST** properly encode the URL into the ASCII format (e.g., using the "urlencode" function).

A server **MUST NOT** use the same URI for different TIPS views, either for different resources or different request bodies to the same resource. URI generation is implementation specific, for example, one may compute a Universally Unique Identifier (UUID, [[RFC4122](#)]) or a hash value based on the request, and append it to a base URL. For performance considerations, it is **NOT RECOMMENDED** to use properties that are not included in the request body to determine the URI of a TIPS view, such as cookies or the client's IP address, which may result in duplicated TIPS views in cases such as mobile clients. However, this is not mandatory as a server may intentionally use client information to compute the TIPS view URI to provide service isolation between clients.

**tips-view-summary:** Contains an updates-graph-summary.

The updates-graph-summary field contains the starting sequence number (start-seq) of the updates graph and the last sequence number (end-seq) that is currently available, along with a recommended edge to consume (start-edge-rec). If the client does NOT provide a version tag, the server **MUST** recommend the edge of the latest snapshot available. If the client does provide a version tag, the server **MUST** either recommend the first incremental update edge starting from the client's tagged version or the edge of the latest snapshot. Which edge is selected depends on the implementation. For example, a server **MAY** calculate the cumulative size of the incremental updates available from that version onward and compare it to the size of the complete resource snapshot. If the snapshot is bigger, the server recommends the first incremental update edge starting from the client's tagged version. Otherwise, the server recommends the latest snapshot edge.

If the request has any errors, the TIPS service **MUST** return an HTTP "400 Bad Request" to the ALTO client; the body of the response follows the generic ALTO error response format specified in Section 8.5.2 of [[RFC7285](#)]. Hence, an example ALTO error response has the format shown in [Figure 9](#).

HTTP/1.1 400 Bad Request  
Content-Length: 131  
Content-Type: application/alto-error+json

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

Figure 9: ALTO Error Example

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

\*If the TIPS request does not have a "resource-id" field, the error code of the error message **MUST** be E\_MISSING\_FIELD and the "field" field, if present, **MUST** be "resource-id". The TIPS service **MUST NOT** create any TIPS view.

\*If the "resource-id" field is invalid or is not associated with the TIPS, the error code of the error message **MUST** be E\_INVALID\_FIELD\_VALUE. If present, the "field" field **MUST** be the full path of the "resource-id" field, and the "value" field **MUST** be the invalid resource-id.

\*If the resource is a POST-mode service that requires input, the client **MUST** set the "input" field to a JSON object with the parameters that that resource expects. If the "input" field is missing or invalid, TIPS **MUST** return the same error response that resource would return for missing or invalid input (see [\[RFC7285\]](#)).

Furthermore, it is **RECOMMENDED** that the server uses the following HTTP codes to indicate other errors, with the media type "application/alto-error+json".

\*429 (Too Many Requests): when the number of TIPS views open requests exceeds the server threshold. The server **MAY** indicate when to re-try the request in the "Re-Try After" headers.

It is **RECOMMENDED** that the server provide the ALTO/SSE support for the TIPS resource. Thus, the client can be notified of the version updates of all the TIPS views that it monitors and make better cross-resource transport decisions (see [Section 8.2](#) for related considerations).

### 6.3. Open Example

#### 6.3.1. Basic Example

For simplicity, assume that the ALTO server is using the Basic authentication. If a client with username "client1" and password "helloalto" wants to create a TIPS view of an ALTO Cost Map resource with resource ID "my-routingcost-map", it can send the request depicted in [Figure 10](#).

```
POST /tips HTTP/1.1
Host: alto.example.com
Accept: application/alto-tips+json, application/alto-error+json
Authorization: Basic Y2xpZW50MTpoZWxsb2FsdG8K
Content-Type: application/alto-tipsparams+json
Content-Length: 41

{
  "resource-id": "my-routingcost-map"
}
```

Figure 10: Request Example of Opening a TIPS View

If the operation is successful, the ALTO server returns the message shown in [Figure 11](#).

```
HTTP/1.1 200 OK
Content-Type: application/alto-tips+json
Content-Length: 255

{
  "tips-view-uri": "https://alto.example.com/tips/2718281828",
  "tips-view-summary": {
    "updates-graph-summary": {
      "start-seq": 101,
      "end-seq": 106,
      "start-edge-rec" : {
        "seq-i": 0,
        "seq-j": 105
      }
    }
  }
}
```

Figure 11: Response Example of Opening a TIPS View

### 6.3.2. Example using Digest Authentication

Below is another example of the same query using Digest authentication, a mandatory authentication method of ALTO servers as defined in [Section 8.3.5](#) of [\[RFC7285\]](#). The content of the response is the same as in [Figure 11](#) and thus omitted for simplicity.



```
POST /tips HTTP/1.1
Host: alto.example.com
Accept: application/alto-tips+json, application/alto-error+json
Authorization: Basic Y2xpZW50MTpoZWxsb2FsdG8K
Content-Type: application/alto-tipsparams+json
Content-Length: 41
```

```
{
  "resource-id": "my-routingcost-map"
}
```

```
HTTP/1.1 401 UNAUTHORIZED
WWW-Authenticate: Digest
  realm="alto.example.com",
  qop="auth",
  algorithm="MD5",
  nonce="173b5aba4242409ee2ac3a4fd797f9d7",
  opaque="a237ff9ab865379a69d9993162ef55e4"
```

```
POST /tips HTTP/1.1
Host: alto.example.com
Accept: application/alto-tips+json, application/alto-error+json
Authorization: Digest
  username="client1",
  realm="alto.example.com",
  uri="/tips",
  qop=auth,
  algorithm=MD5,
  nonce="173b5aba4242409ee2ac3a4fd797f9d7",
  nc=00000001,
  cnonce="ZTg3MTI3NDZmMDQ0NzI1MDQ3MWE3ZTFjZmM5MTNiM2I=",
  response="8e937ae696c1512e4f990fa21c7f9347",
  opaque="a237ff9ab865379a69d9993162ef55e4"
Content-Type: application/alto-tipsparams+json
Content-Length: 41
```

```
{
  "resource-id": "my-routingcost-map"
}
```

```
HTTP/1.1 200 OK
Content-Type: application/alto-tips+json
Content-Length: 258
```

```
{....}
```

Figure 12: Open Example with Digest Authentication

### 6.3.3. Example using ALTO/SSE

This section gives an example of receiving incremental updates of the TIPS view summary using ALTO/SSE [RFC8895]. Consider the tips-sse resource, as announced by the IRD in [Figure 6](#), which provides ALTO/SSE for the update-my-cost-tips resource, a client may send the following request to receive updates of the TIPS view (authentication is omitted for simplicity).

```
POST /updates/tips HTTP/1.1
Host: alto.example.com
Accept: text/event-stream,application/alto-error+json
Content-Type: application/alto-updatestreamparams+json
Content-Length: 76

{
  "add": {
    "tips-123": { "resource-id": "update-my-cost-tips" }
  }
}
```

Figure 13: Example of Monitoring TIPS view with ALTO/SSE

Then, the client will be able to receive the TIPS view summary as follows.

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

event: application/alto-tips+json,tips-123
data: {
data:   "tips-view-uri": "https://alto.example.com/tips/2718281828",
data:   "tips-view-summary": {
data:     "updates-graph-summary": {
data:       "start-seq": 101,
data:       "end-seq": 106,
data:       "start-edge-rec" : {
data:         "seq-i": 0,
data:         "seq-j": 105
data:       }
data:     }
data:   }
data: }
```

When there is an update to the TIPS view, for example, the end-seq is increased by 1, the client will be able to receive the incremental update of the TIPS view summary as follows.

```
event: application/merge-patch+json,tips-123
data: {
  data: "tips-view-summary": {
    data: "updates-graph-summary": {
      data: "end-seq": 107
    }
  }
}
```

## 7. TIPS Data Transfers - Client Pull

TIPS allows an ALTO client to retrieve the content of an update item from the updates graph, with an update item defined as the content (incremental update or snapshot) on an edge in the updates graph.

### 7.1. Request

The client sends an HTTP GET request, where the media type of an update item resource **MUST** be the same as the "media-type" field of the update item on the specified edge in the updates graph.

The GET request **MUST** have the following format:

```
GET /<tips-view-path>/ug/<i>/<j>
HOST: <tips-view-host>
```

For example, consider the updates graph in [Figure 4](#). If the client wants to query the content of the first update item (0 -> 101) whose media type is "application/alto-costmap+json", it sends a request to "/tips/2718281828/ug/0/101" and sets the "Accept" header to "application/alto-costmap+json,application/alto-error+json". See [Section 7.3](#) for a concrete example.

### 7.2. Response

If the request is valid (ug/<i>/<j> exists), the response is encoded as a JSON object whose data format is indicated by the media type.

A client **MAY** conduct proactive fetching of future updates, by long polling updates that have not been provided in the directory yet. For such updates, the client **MUST** indicate all media types that may appear. It is **RECOMMENDED** that the server allows for at least the long polling of <end-seq> -> <end-seq + 1>.

Hence, the server processing logic **MUST** be:

\*If ug/<i>/<j> exists: return content using encoding.

\*Else if long polling ug/<i>/<j> is acceptable: put request in a backlog queue, then either a response is triggered when the

content is ready or the request is interrupted, e.g., by a network error.

\*Else: return error.

It is **RECOMMENDED** that the server uses the following HTTP codes to indicate errors, with the media type "application/alto-error+json", regarding update item requests.

\*404 (Not Found): if the requested update does not exist, or the requested TIPS view does not exist or is closed by the server.

\*410 (Gone): if an update has a seq that is smaller than the start-seq.

\*415 (Unsupported Media Type): if the media type(s) accepted by the client does not include the media type of the update chosen by the server.

\*425 (Too Early): if the seq exceeds the server long-polling window

\*429 (Too Many Requests): when the number of pending (long-poll) requests exceeds the server threshold. The server **MAY** indicate when to re-try the request in the "Re-Try After" headers.

### 7.3. Example

Assume the client wants to get the contents of the update item on edge 0 to 101. The format of the request is shown in [Figure 14](#).

```
GET /tips/2718281828/ug/0/101 HTTP/1.1
Host: alto.example.com
Accept: application/alto-costmap+json, \
        application/alto-error+json
```

Figure 14: GET Example

The response is shown in [Figure 15](#).

```
HTTP/1.1 200 OK
Content-Type: application/alto-costmap+json
Content-Length: 50

{ ... full replacement of my-routingcost-map ... }
```

Figure 15: Response to a GET Request

## 7.4. New Next Edge Recommendation

While intended TIPS usage is for the client to receive a recommended starting edge in the TIPS summary, consume that edge, then construct all future URIs by incrementing the sequence count by 1, there may be cases in which the client needs to request a new next edge to consume. For example, if a client has an open TIPS view yet has not polled in a while, the client may request the next logical incremental URI but the server has compacted the updates graph so it no longer exists. Thus, the client **MAY** request a new next edge to consume based on its current version of the resource.

### 7.4.1. Request

An ALTO client requests that the server provide a next edge recommendation for a given TIPS view by sending an HTTP POST request with the media type "application/alto-tipsparams+json". The URL of the request **MUST** have the format of

```
<tips-view-path>/ug
```

and the HOST field **MUST** be the <tips-view-host>.

The POST body has the same format as the TIPSReq [Figure 7](#). The resource-id **MUST** be the same as the resource ID used to create the TIPS view, and the optional input field **MUST NOT** be present.

### 7.4.2. Response

The response to a valid request **MUST** be a JSON merge patch to the object of type AddTIPSResponse (defined in [Section 6.2](#)), denoted as media type "application/merge-patch+json". The "update-graph-summary" field **MUST** be present in the response and hence its parent field "tips-view-summary" **MUST** be present as well.

If the tag field is present in the request, the server **MUST** check if any version within the range [start-seq, end-seq] has the same tag value. If the version exists, e.g., denoted as tag-seq, the server **MUST** compute the paths from both tag-seq and 0 to the end-seq, and choose the one with the minimal cost. The cost **MAY** be implementation specific, e.g., number of messages, accumulated data size, etc. The first edge of the selected path **MUST** be returned as the recommended next edge.

If the tag field is NOT present, it **MUST** be interpreted as the tag-seq is 0.

It is **RECOMMENDED** that the server uses the following HTTP codes to indicate errors, with the media type "application/alto-error+json", regarding new next edge requests.

\*404 (Not Found): if the requested TIPS view does not exist or is closed by the server.

### 7.4.3. Example

We give an example of the new next edge recommendation service. Assume that a client already creates a TIPS view as in [Section 6.3](#), whose updates graph is as shown in [Figure 2](#). Now assume that the client already has tag 0881080 whose corresponding sequence number is 103, and sends the following new next edge recommendation request (authentication is omitted for simplicity):

```
POST /tips/2718281828/ug HTTP/1.1
HOST alto.example.com
Accept: application/merge-patch+json, application/alto-error+json
Content-Type: application/alto-tipsparams+json
Content-Length: 62
```

```
{
  "resource-id": "my-routingcost-map",
  "tag": "0881080"
}
```

According to [Figure 2](#), there are 3 potential paths: 103 -> 104 -> 105 -> 106, 103 -> 105 -> 106, and 0 -> 105 -> 106. Assume that the server chooses shortest update path by the accumulated data size and the best path is 103 -> 105 -> 106. Thus, the server responds with the following message:

```
HTTP/1.1 200 OK
Content-Type: application/merge-patch+json
Content-Length: 193
```

```
{
  "tips-view-summary": {
    "updates-graph-summary": {
      "start-seq": 101,
      "end-seq": 106,
      "start-edge-rec": {
        "seq-i": 103,
        "seq-j": 105
      }
    }
  }
}
```

## 8. Operation and Processing Considerations

TIPS has some common operational considerations as ALTO/SSE [[RFC8895](#)], including:

- \*server choosing update messages ([Section 9.1](#) of [[RFC8895](#)]);
- \*client processing update messages ([Section 9.2](#) of [[RFC8895](#)]);
- \*updates of filtered map services ([Section 9.3](#) of [[RFC8895](#)]);
- \*updates of ordinal mode costs ([Section 9.4](#) of [[RFC8895](#)]).

There are also some operation considerations specific to TIPS, which we discuss below.

### 8.1. Considerations for Load Balancing

There are two levels of load balancing in TIPS. The first level is to balance the load of TIPS views for different clients, and the second is to balance the load of incremental updates.

Load balancing of TIPS views can be achieved either at the application layer or at the infrastructure layer. For example, an ALTO server **MAY** set <tips-view-host> to different subdomains to distribute TIPS views, or simply use the same host of the TIPS service and rely on load balancers to distribute the load.

TIPS allows a client to make concurrent pulls of incremental updates for the same TIPS view potentially through different HTTP connections. As a consequence, it introduces additional complexities when the ALTO server is being load balanced. For example, a request may be directed to a wrong backend server and get incorrectly processed if the following two conditions both hold:

- \*the backend servers are stateful, i.e., the TIPS view is created and stored only on a single server;
- \*the ALTO server is using layer-4 load balancing, i.e., the requests are distributed based on the TCP 5-tuple.

Thus, additional considerations are required to enable correct load balancing for TIPS, including:

- \*Use a stateless architecture: One solution is to follow the stateless computing pattern: states about the TIPS view are not maintained by the backend servers but are stored in a distributed database. Thus, concurrent requests to the same TIPS view can be processed on arbitrary stateless backend servers, which all fetches data from the same database.

\*Configure the load balancers properly: In case when the backend servers are stateful, the load balancers must be properly configured to guarantee that requests of the same TIPS view always arrive at the same server. For example, an operator or a provider of an ALTO server **MAY** configure layer-7 load balancers that distribute requests based on the tips-view-path component in the URI.

## 8.2. Considerations for Cross-Resource Dependency Scheduling

Dependent ALTO resources result in cross-resource dependencies in TIPS. Consider the following pair of resources, where my-cost-map (C) is dependent on my-network-map (N). The updates graph for each resource is shown, along with links in between the respective updates graphs to show dependency:

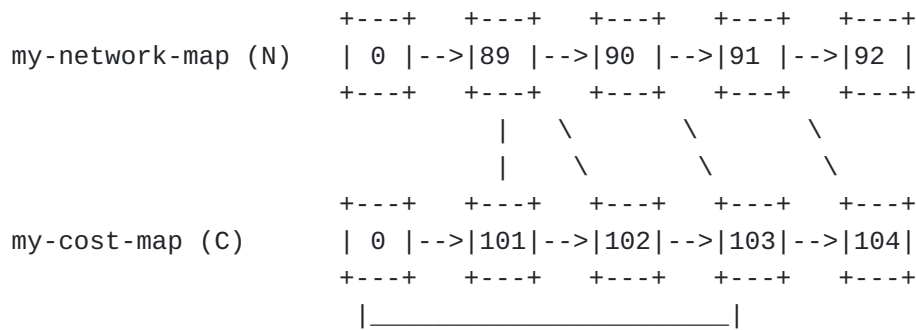


Figure 16: Example Dependency Model

In [Figure 16](#), the cost-map versions 101 and 102 (denoted as C101 and C102) are dependent on the network-map version 89 (denoted as N89). The cost-map version 103 (C103) is dependent on the network-map version 90 (N90), and so on.

Thus, the client must decide the order in which to receive and apply the updates. The order may affect how fast the client can build a consistent view and how long the client needs to buffer the update.

\*Example 1: The client requests N89, N90, N91, C101, C102 in that order. The client either gets no consistent view of the resources or has to buffer N90 and N91.

\*Example 2: The client requests C101, C102, C103, N89. The client either gets no consistent view or has to buffer C103.

To get consistent ALTO information, a client must process the updates following the guidelines specified in [Section 9.2](#) of [\[RFC8895\]](#). If resource permits (i.e., sufficient updates can be buffered), an ALTO client can safely use long polling to fetch all the updates. This allows a client to build consistent views quickly



as the updates are already stored in the buffer. Otherwise, it is **RECOMMENDED** to request

### 8.3. Considerations for Managing Shared TIPS Views

From a client's point of view, it sees only one copy of the TIPS view for any resource. However, on the server side, there are different implementation options, especially for common resources (e.g., network map or cost map) that may be frequently queried by many clients. Some potential options are listed below:

\*An ALTO server creates one TIPS view of the common resource for each client.

\*An ALTO server maintains one copy of the TIPS view for each common resource and all clients requesting the same resources use the same copy. There are two ways to manage the storage for the shared copy:

- the ALTO server maintains the set of clients that have sent a polling request to the TIPS view, and only removes the view from the storage when the set becomes empty and no client immediately issues a new edge request;

- the TIPS view is never removed from the storage.

Developers may choose different implementation options depending on criteria such as request frequency, available resources of the ALTO server, the ability to scale, and programming complexity.

### 8.4. Considerations for Offering Shortcut Incremental Updates

Besides the mandatory stepwise incremental updates (from  $i$  to  $i+1$ ), an ALTO server **MAY** optionally offer shortcut incremental updates, or simple shortcuts, between two non-consecutive versions  $i$  and  $i+k$  ( $k > 1$ ). Such shortcuts offer alternative paths in the update graph and can potentially speed up the transmission and processing of incremental updates, leading to faster synchronization of ALTO information, especially when the client has limited bandwidth and computation. However, implementors of an ALTO server must be aware that:

1. Optional shortcuts may increase the size of the update graph, in the worst case being the square of the number of updates (i.e., when a shortcut is offered for each version to all future versions).
2. Optional shortcuts require additional storage on the ALTO server.

3. Optional shortcuts may reduce concurrency when the updates do not overlap, e.g., when the updates apply to different parts of an ALTO resource. In such a case, the total size of the original updates is close to the size of the shortcut, but the original updates can be transmitted concurrently while the shortcut is transmitted in a single connection.

## 9. Security Considerations

The security considerations (Section 15 of [\[RFC7285\]](#)) of the base protocol fully apply to this extension. For example, the same authenticity and integrity considerations (Section 15.1 of [\[RFC7285\]](#)) still fully apply; the same considerations for the privacy of ALTO users (Section 15.4 of [\[RFC7285\]](#)) also still fully apply. Additionally, operators of the ALTO servers **MUST** follow the guidelines in [\[RFC9325\]](#) to avoid new TLS vulnerabilities discovered after [\[RFC7285\]](#) was published.

The additional services (addition of update read service and update push service) provided by this extension extend the attack surface described in Section 15.1.1 of [\[RFC7285\]](#). The following sub-sections discuss the additional risks and their remedies.

### 9.1. TIPS: Denial-of-Service Attacks

Allowing TIPS views enables new classes of Denial-of-Service attacks. In particular, for the TIPS server, one or multiple malicious ALTO clients might create an excessive number of TIPS views, to exhaust the server resource and/or to block normal users from accessing the service.

To avoid such attacks, the server **MAY** choose to limit the number of active views and reject new requests when that threshold is reached. TIPS allows predictive fetching and the server **MAY** also choose to limit the number of pending requests. If a new request exceeds the threshold, the server **MAY** log the event and return the HTTP status "429 Too many requests".

It is important to note that the preceding approaches are not the only possibilities. For example, it may be possible for TIPS to use somewhat more clever logic involving TIPS view eviction policies, IP reputation, rate-limiting, and compartmentalization of the overall threshold into smaller thresholds that apply to subsets of potential clients. If service availability is a concern, ALTO clients **MAY** establish service level agreements with the ALTO server.

### 9.2. ALTO Client: Update Overloading or Instability

The availability of continuous updates can also cause overload for an ALTO client, in particular, an ALTO client with limited

processing capabilities. The current design does not include any flow control mechanisms for the client to reduce the update rates from the server. For example, TCP, HTTP/2 and QUIC provide stream and connection flow control data limits, which might help prevent the client from being overloaded. Under overloading, the client **MAY** choose to remove the information resources with high update rates.

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

## 10. IANA Considerations

IANA is requested to register the following media types from the registry available at [[IANA-Media-Type](#)]:

\*application/alto-tips+json: as described in [Section 6.2](#);

\*application/alto-tipsparams+json: as described in [Section 6.1](#);

Note to the RFC Editor: Please replace This-Document with the RFC number to be assigned to this document.

### 10.1. application/alto-tips+json Media Type

**Type name:** application

**Subtype name:** alto-tips+json

**Required parameters:** N/A

**Optional parameters:** N/A

**Encoding considerations:** Encoding considerations are identical to those specified for the "application/json" media type. See [[RFC8259](#)].

**Security considerations:** See the Security Considerations section of This-Document.

**Interoperability considerations:** N/A.

**Published specification:** [Section 6.2](#) of This-Document.

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

**Fragment identifier considerations:** N/A

**Additional information:**

**Deprecated alias names for this type:** N/A

**Magic number(s):** N/A

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

**Macintosh file type code(s):** N/A

**Person and 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).

**Provisional registration?:** No

**10.2. application/alto-tipsparams+json Media Type**

**Type name:** application

**Subtype name:** alto-tipsparams+json

**Required parameters:** N/A

**Optional parameters:** N/A

**Encoding considerations:** Encoding considerations are identical to those specified for the "application/json" media type. See [\[RFC8259\]](#).

**Security considerations:** See the Security Considerations section of This-Document.

**Interoperability considerations:** N/A.

**Published specification:** [Section 6.1](#) of This-Document.

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

**Fragment identifier considerations:**

N/A

**Additional information:**

**Deprecated alias names for this type:** N/A

**Magic number(s):** N/A

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

**Macintosh file type code(s):** N/A

**Person and 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).

**Provisional registration?:** No

## 11. References

### 11.1. 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/rfc/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/rfc/rfc3986>>.
- [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/rfc/rfc7285>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/rfc/rfc8259>>.
- [RFC8895] Roome, W. and Y. Yang, "Application-Layer Traffic Optimization (ALTO) Incremental Updates Using Server-Sent Events (SSE)", RFC 8895, DOI 10.17487/RFC8895, November 2020, <<https://www.rfc-editor.org/rfc/rfc8895>>.
- [RFC9112] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <<https://www.rfc-editor.org/rfc/rfc9112>>.
- [RFC9113] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <<https://www.rfc-editor.org/rfc/rfc9113>>.
- [RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <<https://www.rfc-editor.org/rfc/rfc9114>>.
- [RFC9325] Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 9325, DOI 10.17487/RFC9325, November 2022, <<https://www.rfc-editor.org/rfc/rfc9325>>.

## 11.2. Informative References

- [IANA-Media-Type] "Media Types", June 2023, <<https://www.iana.org/assignments/media-types/media-types.xhtml>>.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <<https://www.rfc-editor.org/rfc/rfc4122>>.
- [RFC9205] Nottingham, M., "Building Protocols with HTTP", BCP 56, RFC 9205, DOI 10.17487/RFC9205, June 2022, <<https://www.rfc-editor.org/rfc/rfc9205>>.

## Appendix A. A High-Level Deployment Model

Conceptually, the TIPS system consists of three types of resources:

- \*(R1) TIPS frontend to create TIPS views.
- \*(R2) TIPS view directory, which provides metadata (e.g., references) about the network resource data.
- \*(R3) The actual network resource data, encoded as complete ALTO network resources (e.g., cost map, network map) or incremental updates.

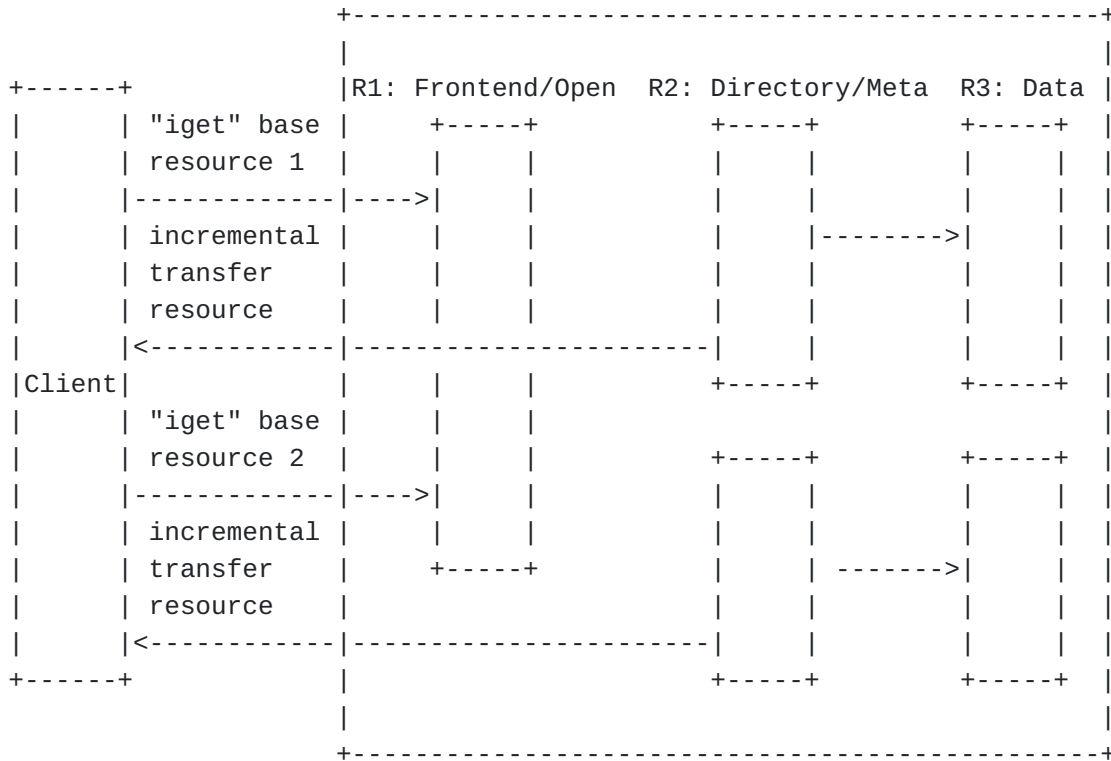


Figure 17: Sample TIPS Deployment Model

Design Point: Component Resource Location

- \*Design 1 (Single): all the three resource types at the same, single server (accessed via relative reference)
- \*Design 2 (Flexible): all three resource types can be at their own server (accessed via absolute reference)
- \*Design 3 (Dir + Data): R2 and R3 must remain together, though R1 might not be on the same server

This document supports Design 1 and Design 3. For Design 1, the TIPS service simply needs to always use the same host for the TIPS views. For Design 3, the TIPS service can set tips-view-host to a different server. Note that the deployment flexibility is at the logical level, as these services can be distinguished by different paths and potentially be routed to different physical servers by layer-7 load balancing. See [Section 8.1](#) for a discussion on load balancing considerations. Future documents may extend the protocol to support Design 2.

## **Appendix B. Conformance to "Building Protocols with HTTP" Best Current Practices**

This specification adheres fully to [\[RFC9205\]](#) as further elaborated below:

\*TIPS does not "redefine, refine, or overlay the semantics of generic protocol elements such as methods, status codes, or existing header fields" and instead focuses on "protocol elements that are specific to [the TIPS] application -- namely, [its] HTTP resources" ([Section 3.1](#) of [\[RFC9205\]](#)).

\*There are no statically defined URI components ([Section 3.2](#) of [\[RFC9205\]](#)).

\*No minimum version of HTTP is specified by TIPS which is recommended ([Section 4.1](#) of [\[RFC9205\]](#)).

\*The TIPS design follows the advice that "When specifying examples of protocol interactions, applications should document both the request and response messages with complete header sections, preferably in HTTP/1.1 format" ([Section 4.1](#) of [\[RFC9205\]](#)).

\*TIPS uses URI templates which is recommended ([Section 4.2](#) of [\[RFC9205\]](#)).

\*TIPS follows the pattern that "a client will begin interacting with a given application server by requesting an initial document that contains information about that particular deployment, potentially including links to other relevant resources. Doing so ensures that the deployment is as flexible as possible (potentially spanning multiple servers), allows evolution, and also allows the application to tailor the "discovery document" to the client" ([Section 4.4.1](#) of [\[RFC9205\]](#)).

\*TIPS uses existing HTTP schemes ([Section 4.4.2](#) of [\[RFC9205\]](#)).

\*TIPS defines its errors "to use the most applicable status code" ([Section 4.6](#) of [\[RFC9205\]](#)).



\*TIPS does not "make assumptions about the relationship between separate requests on a single transport connection; doing so breaks many of the assumptions of HTTP as a stateless protocol and will cause problems in interoperability, security, operability, and evolution" ([Section 4.11](#) of [[RFC9205](#)]). The only relationship between requests is that a client must first discover where a TIPS view of a resource will be served, which is consistent with the URI discovery in [Section 4.4.1](#) of [[RFC9205](#)].

### **Appendix C. Push-mode TIPS using HTTP Server Push**

TIPS allows ALTO clients to subscribe to incremental updates of an ALTO resource, and the specification in this document is based on the current best practice of building such a service using native HTTP. Earlier versions of this document had investigated the possibility of enabling push-mode TIPS, i.e., by taking advantage of the server push feature in HTTP/2 and HTTP/3.

In the ideal case, push-mode TIPS can potentially improve performance (e.g., latency) in more dynamic environments and use cases, with wait-free message delivery. Using native server push also results in minimal changes to the current protocol. While not adopted due to the lack of server push support and increased protocol complexity, push-mode TIPS remains a potential direction of protocol improvement.

### **Appendix D. Persistent HTTP Connections**

Previous versions of this document use persistent HTTP connections to detect the liveness of clients. This design, however, does not conform well with the best current practice of HTTP. For example, if an ALTO client is accessing a TIPS view over an HTTP proxy, the connection is not established directly between the ALTO client and the ALTO server, but between the ALTO client and the proxy and between the proxy and the ALTO server. Thus, using persistent connections may not correctly detect the right liveness state.

### **Acknowledgments**

The authors of this document would like to thank Mark Nottingham and Spencer Dawkins for providing invaluable reviews of earlier versions of this document, Adrian Farrel, Qin Wu, and Jordi Ros Giralt for their continuous feedback, Russ White, Donald Eastlake, Martin Thomson, Bernard Adoba, Spencer Dawkins, Linda Dunbar and Sheng Jiang for the directorate reviews, Martin Duke for the Area Director review, Francesca Palombini, Wesley Eddy, Roman Danyliw, Murray Kucherawy and Zaheduzzaman Sarker for the telechat and IESG reviews, and Mohamed Boucadair for shepherding the document.

## Authors' Addresses

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

Email: [kaigao@scu.edu.cn](mailto:kaigao@scu.edu.cn)

Roland Schott  
Deutsche Telekom  
Ida-Rhodes-Straße 2  
64295 Darmstadt  
Germany

Email: [Roland.Schott@telekom.de](mailto:Roland.Schott@telekom.de)

Yang Richard Yang  
Yale University  
51 Prospect Street  
New Haven, CT  
United States of America

Email: [yry@cs.yale.edu](mailto:yry@cs.yale.edu)

Lauren Delwiche  
Yale University  
51 Prospect Street  
New Haven, 3408  
United States of America

Email: [lauren.delwiche@yale.edu](mailto:lauren.delwiche@yale.edu)

Lachlan Keller  
Yale University  
51 Prospect Street  
New Haven, 3408  
United States of America

Email: [lachlan.keller@yale.edu](mailto:lachlan.keller@yale.edu)