Abstract

Many ALTO use cases involve multiple ALTO information resources like different network maps, cost maps and property maps to achieve their own specific goals. To make the ALTO client query them one by one is not only inefficient but also error-prone. The inconsistent responses can be performed because of the unstable communication environment, and finally conduct the unexpected traffic optimization. Further more, some ALTO information resources may have correlation, which means one’s input parameters may depends on another one’s response. To address those issues, some advanced query schema is required. This document proposes an ALTO extension to support the multiple ALTO resources query in the single request using the HTTP multipart message and the existing JSON query languages.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 26, 2019.
1. Introduction

Application-Layer Traffic Optimization (ALTO) protocol \cite{RFC7285} and its extensions already define several types of information resources, like Network Map, Cost Map and Property Map, to expose useful network informations to applications. However, many applications do not only use a single information resource to perform their traffic optimization. Retrieving multiple ALTO information resources is very common in many ALTO use cases.

Using the current ALTO framework defined in \cite{RFC7285}, the ALTO client can only query multiple ALTO information resources one by one. It is not only inefficient but also error-prone. Because of the network delay between different requests and the frequent change of ALTO information resources, the responses received by the ALTO client may be inconsistent.

Further more, some ALTO information resources have known dependencies, which means the ALTO client may need one’s response to decide another one’s query input parameters.

To be summarized, we need the multipart query service for three reasons:

- Clients may want to query multiple ALTO information resources in a single request to reduce the time consumption.

- Clients may want to query multiple ALTO resources consistently, which means the server should guarantee the responses of all resources are generated at the same time.

- Some use cases need to query multiple ALTO resources with a joint relationship.
This document defines a new ALTO service for: (1) querying multiple ALTO resources in a single request/response, and (2) supporting general-purpose JSON query languages to resolve the relational query.

2. Terminologies

Besides the terms defined in [RFC2045], [RFC2046], [RFC2387], and [RFC7285], this document also uses the following additional terms:

2.1. Resource Query Entry

A Resource Query Entry indicates the ResourceQuery object (see Section 6.4) for an individual resource in the accept input parameters of the Multipart Query resource.

2.2. Resource Response Entry

A Resource Response Entry indicates the entry of an individual part of the multipart response message, including the MIME headers and the body content.

2.3. Resource Response Entry Body

A Resource Response Entry Body indicates the body content of a Resource Response Entry.

3. Use Cases

The following use cases can benefit from the multipart query service.

3.1. Simple Batch Query

The simplest use case is to query a batch of ALTO resources in a single request.

Although the ALTO client can perform ALTO requests for multiple times, it is not only inefficient but also inconsistent.

For example, the ALTO server provides a network map resource A and a dependent cost map resource B. Both resources may change frequently. Assume the ALTO client queries the network map first, and it gets the revision A1. When the client queries the cost map, the network map may be already changed from A1 to A2, and the client receives cost map B2 which depends on A2 not A1. So the responded cost map B2 is not consistent with the previous network map A1.

This case requires the ALTO server to provide a way for the ALTO client to query multiple ALTO resources in a single transaction.
3.2. Properties Constrained Query

Beyond the simple batch query, there are also some other use cases requiring a new service for relational query. For example, some clients may need to query an endpoint property map first, and find endpoints with some properties fitting some conditions. And then they query the endpoint cost of these endpoints.

In this case, the endpoint cost query depends on the result of the property map query. Although the ALTO client can cache the whole property map in its local storage, it is still not efficient and may conduct the consistency issue if the property map changes frequently. So it requires a new service to provide multiple dependent resources efficiently and consistently.

A general multipart query service benefits the ALTO client in two aspects:

- It allows the ALTO client to specify the boolean test to reduce the transmission of the useless data from the ALTO server.
- It compounds multiple ALTO information resources in a single response to reduce the communication times. Thus, the transmission latency can be reduced.

3.3. Path Vector Query

Another use case requiring the multiple resource query is the relational query between the on-demand generated resources. A straightforward example is the path vector query demonstrated in [I-D.ietf-alto-path-vector].

[I-D.ietf-alto-path-vector] introduces an extension of ALTO to provide path vector information by cost map and unified property map [I-D.ietf-alto-unified-props-new]. The client using path vector extension will usually query cost map and a dynamically generated property map sequentially. It is even hard to cache the full data of resources, because both the cost map and the property map are on-demand generated by the query input here. Thus, the only way to reduce the time consumption is to compound the two resources.

4. Requirements

From the use cases described in Section 3, there are three additional requirements for ALTO protocol:
MPQ-Req1: The ALTO protocol SHOULD allow the client to query multiple ALTO resources in a single request, and return the result in a single response.

It is the basic requirement to provide the query for the compound resources. Even simple cases can benefit if this requirement is realized.

MPQ-Req2: The ALTO protocol SHOULD provide general filter schema for any ALTO resources.

Current filter schema in ALTO protocol only supports the simple boolean test of numerical comparison. And the boolean filtered query is only supported by the cost map and the endpoint cost resource. It is not enough for the general cases. Even simple property map may require more general filter schema.

MPQ-Req3: The ALTO protocol SHOULD support relational query for multiple joint resources.

Some ALTO resources are relational and cannot be used individually. The path vector query is such an example. In these use cases, the support of relational query for multiple joint resources is very helpful.

5. Design Space of Multipart Resource in ALTO

This document discusses the solution of how to apply "multipart/*" (see [RFC2045] and [RFC2046]) response to the ALTO protocol.

There are three cases applying Multipart response to ALTO:

Multpart Request and Multipart Response: In this case, an ALTO client can start a single request using Multipart encoding to query a batch of resources.

Single Request and Fixed-Layout Multipart Response: In this case, an ALTO server receives a non-Multipart request, e.g., the filtered costmap request or the endpoint cost request, but returns a Multipart response. The ALTO server MUST export the layout of the Multipart response in the IRD. A special example can be found in [I-D.ietf-alto-path-vector].

Single Request and Flexible-Layout Multipart Response: This case extends the previous case to allows the Multipart response with flexible layout. The ALTO server receives the unified query request and generate the layout of the response based on the request. The ALTO client can even use general-purpose query...
language like XQuery [W3CXMLQUERY] and JSONiq [JSONIQ] for general query process and relational joint query.

The application about Multipart request to the single object response is out of the scope of this document.

6. Multipart Query Resource

6.1. Media Type

"multipart/related" [RFC2387].

6.2. HTTP Method

An ALTO Multipart Query resource is requested using the HTTP POST method.

6.3. Capabilities

The capabilities are defined by an object of type MultipartQueryCapabilities:

```json
object {
    JSONString query-langs<0..*>;
} MultipartQueryCapabilities;
```

where "query-langs" is an array of JSONString to indicate which query languages are supported by this resource.

6.4. Accept Input Parameters

The input parameters for a Multipart Query request are supplied in the entity body of the POST request. This document specifies the input parameters with a data format indicated by the media type "application/alto-multipartquery+alto", which is a JSON object of type ReqMultipartQuery:

```json
object {
    ResourceQuery resources<1..*>;
    [JSONObject query-lang];
} ReqMultipartQuery;
```

```json
object {
    JSONObject resource-id;
    [JsonValue input];
} ResourceQuery;
```

with fields:
resources: List of ResourceQuery objects for which resources are to be queried and how to query them. Each ResourceQuery object MUST include the "resource-id" field to indicate which resource is to be queried. If the queried resource requires the POST method, the "input" field MUST be specified. The value of the "input" field MUST be either a JSONObject or a JSONString. When its value is a JSONObject, its format MUST be as the accept input parameters of its resource. When its value is a JSONString, it MUST be a program written in the query language specified by the "query-lang" field.

query-lang: Optional. The value of the "query-lang" field MUST be one of values in the "query-langs" capability. If this field is not specified in the request, the ALTO client SHOULD NOT use any query language in the "input" field.

6.5. Uses

An array with the resource ID(s) of resource(s) which this multipart query resource can compound. The used resource can be any available ALTO resources except for the multipart query resource. If the "uses" field is not specified, all the available ALTO resources can be queried except for the multipart query resource.

6.6. Response

The response of multipart query resource is a multipart message. Each part of this multipart message is the response of a queried resource in the request.

7. Protocol Errors

At the top level, the request of ALTO Multipart Query resource may conduct two types of errors: Partial Error and Entire Error.

7.1. Partial Error

The Partial Error only occurs when the value of the "resource-id" field or the "input" field is invalid.

When the Partial Error occurs, the ALTO server MUST still return the response in the media type "multipart/related". For the resource query entry with an error, the ALTO server MUST specify the "Content-Type" of its resource response entry as "application/alto-error+json", and include the ALTO error message in its resource response entry body. For the resource query entry without any error, the ALTO server MUST perform its query request normally.
The value of the "resource-id" field is invalid when this resource id is not defined by the Information Resource Directory. In this case, the ALTO server MUST return the E_INVALID_FIELD_VALUE error.

The validation of each "input" field of the multipart query input parameters depends on the queried resource:

- If the "input" field of the multipart query input parameters is neither a JSONObject nor a JSONString, the ALTO server SHOULD return the E_INVALID_FIELD_TYPE error, unless a future protocol extension supports the non-JSONObject input parameters.

- If the "input" field of the multipart query input parameters is a JSONObject, the ALTO server MUST validate the value using its queried resource and return the corresponding error if it has.

- If the "input" field of the multipart query input parameters is a JSONString:
  - If the "query-lang" is not specified, the ALTO server MUST return the E_INVALID_FIELD_TYPE error.
  - If the "query-lang" is specified, the ALTO server MUST execute this JSONString as a program written in the "query-lang". If the execution failed, the ALTO server MUST return the E_INVALID_FIELD_VALUE error. If the execution succeed but the result fails to pass the validation of the queried resource, the ALTO server MUST return the E_INVALID_FIELD_VALUE error and attach the error message returned by the queried resource into the "message" field of the ALTO error message.

The syntax error is an Entire Error.

7.2. Entire Error

Any other invalid request will conduct the Entire Error.

When the Entire Error occurs, the ALTO server MUST return the error response in the media type "application/alto-error+json" instead of "multipart/related". The process of the Entire Error is as defined in Section 8.5 of [RFC7285].

8. Incremental Update Integration

This document defines a compatible incremental update process for Multipart Query resource with [I-D.ietf-alto-incr-update-sse].
An ALTO server’s IRD can export an Update Stream service defined in
[I-D.ietf-alto-incr-update-sse] including the Resource ID of a
Multipart Query resource in the "uses" field. When an ALTO client
subscribe the incremental update for this Multipart Query resource,
the ALTO server sends the whole Multipart response message back at
the first data update message. Then the ALTO server subscribe all
nodes in this multipart resource tree automatically. Once data
updated later, the ALTO server publishes the update for each node
individually.

9. Examples

9.1. IRD Example

Assume the root IRD is like the following:

```json

{
  "meta": {
    "path-vector": {
      "cost-mode": "array",
      "cost-metric": "ane-path"
    },
    "num-routingcost": {
      "cost-mode": "numerical",
      "cost-metric": "routingcost"
    },
    "num-hopcount": {
      "cost-mode": "numerical",
      "cost-metric": "hopcount"
    }
  },
  "resources": {
    "my-default-networkmap": {
      "uri": "http://alto.example.com/networkmap",
      "media-type": "application/alto-networkmap+json"
    },
    "my-default-costmap": {
      "uri": "http://alto.example.com/costmap",
      "media-type": "application/alto-costmap+json",
      "capabilities": {
        "cost-type-names": [ "num-routingcost" ]
      },
      "uses": [ "my-default-networkmap" ]
    },
    "my-filtered-costmap": {
      "uri": "http://alto.example.com/costmap/filtered",
      "media-type": "application/alto-costmap+json",
      "accepts": "application/alto-costmapfilter+json"
    }
  }
}
```
"capabilities": {
  "cost-type-names": [ "num-hopcount" ]
},
"uses": [ "my-default-networkmap" ]
},
"endpoint-path-vector": {
  "uri": "http://alto.example.com/endpointcost",
  "media-type": "application/alto-endpointcost+json",
  "accepts": "application/alto-endpointcostparams+json",
  "capabilities": {
    "cost-constraints": true,
    "cost-type-names": [ "path-vector" ],
  },
  "property-map": "propmap-availbw"
},
"propmap-availbw-delay": {
  "uri": "http://alto.example.com/propmap/availbw",
  "media-type": "application/alto-propmap+json",
  "accepts": "application/alto-propmapparams+json",
  "capabilities": {
    "domain-types": [ "ane" ],
    "prop-types": [ "availbw" ]
  }
},
"propmap-location": {
  "uri": "http://alto.example.com/propmap/location",
  "media-type": "application/alto-propmap+json",
  "accepts": "application/alto-propmapparams+json",
  "capabilities": {
    "domain-types": [ "pid" ],
    "prop-types": [ "country", "state" ]
  }
},
"multipart-query": {
  "uri": "http://alto.example.com/multipart",
  "media-type": "multipart/related",
  "accepts": "application/alto-multipartquery+json",
  "capabilities": {
    "query-langs": [ "xquery", "jsoniq" ]
  }
}
9.2. Example 1: Simple Batch Query

POST /multipart HTTP/1.1
Host: alto.example.com
Accept: multipart/related, application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-multipartquery+json

{
    "resources": [
        {
            "resource-id": "my-default-networkmap"
        },
        {
            "resource-id": "my-default-costmap"
        }
    ]
}

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=simple-batch-query

--simple-batch-query
Content-Type: application/alto-networkmap+json

{
    "meta": {
        "vtag": {
            "resource-id": "my-default-networkmap",
            "tag": "75ed013b3cb58f896e839582504f622838ce670f"
        }
    },
    "network-map": {
        "PID1": {
            "ipv4": [
                "192.0.2.0/24",
                "198.51.100.0/25"
            ]
        },
        "PID2": {
            "ipv4": [
                "198.51.100.128/25"
            ]
        },
        "PID3": {
            "ipv4": [
                "0.0.0.0/0"
            ]
        }
    }
}
"ipv6": [ "::/0" ]
}

--simple-batch-query
Content-Type: application/alto-costmap+json

{
  "meta": {
    "dependent-vtags": [
    {
      "resource-id": "my-default-networkmap",
      "tag": "75ed013b3cb58f896e839582504f622838ce670f"
    }
    ],
    "cost-type": {
      "cost-mode": "numerical",
      "cost-metric": "routingcost"
    }
  },
  "cost-map": {
    "PID1": { "PID1": 1, "PID2": 5, "PID3": 10 },
    "PID2": { "PID1": 5, "PID2": 1, "PID3": 15 },
    "PID3": { "PID1": 20, "PID2": 15 }
  }
}

9.3. Example 2: Properties Constrained Query

NOTE: In this example, we use the "\"" block to express the raw string with unescaped characters like "\n" and "\". It is not valid HTTP body, but only used to better present. When the request is sent to the ALTO server, the "\" block should be escaped.
POST /multipart HTTP/1.1
Host: alto.example.com
Accept: multipart/related, application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-multipartquery+json

{
    "query-lang": "jsoniq",
    "resources": [
        {
            "resource-id": "propmap-location"
        },
        {
            "resource-id": "my-default-costmap",
            "input": `
                let $propmap := collection("propmap-location")
                .("property-map")
                return {
                    "cost-type": {
                        "cost-mode": "numerical",
                        "cost-metric": "hopcount"
                    },
                    "pids": {
                        "srcs": [
                            for $pid in keys($propmap)
                            where $propmap.$pid.country eq "US"
                            return substring-after($pid, "PID:")
                        ],
                        "dsts": [
                            for $pid in keys($propmap)
                            where $propmap.$pid.country eq "CA"
                            return substring-after($pid, "PID:")
                        ]
                    }
                }`,
        }
    ]
}

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=prop-const-query

--prop-const-query
Content-Type: application/alto-propmap+json

{
"property-map": {
    "pid:PID1": {
        "country": "US",
        "state": "CA"
    },
    "pid:PID2": {
        "country": "US",
        "state": "CT"
    },
    "pid:PID3": {
        "country": "CA",
        "state": "QC"
    },
    "pid:PID4": {
        "country": "CA",
        "state": "NT"
    },
    "pid:PID5": {
        "country": "FR"
    }
},

--prop-const-query
Content-Type: application/alto-costmap+json

{
    "meta": {
        "cost-type": {
            "cost-mode": "numerical",
            "cost-metric": "hopcount"
        }
    },
    "cost-map": {
        "PID1": {
            "PID3": 5,
            "PID4": 7
        },
        "PID2": {
            "PID3": 8,
            "PID4": 4
        }
    }
}
9.4. Example 3: Path Vector Query

POST /multipart HTTP/1.1
Host: alto.example.com
Accept: multipart/related, application/alto-error+json
Content-Length: [TBD]
Content-Type: application/alto-multipartquery+json

{
  "query-lang": "jsoniq",
  "resources": [
    {
      "resource-id": "endpoint-path-vector",
      "input": {
        "cost-type": {
          "cost-mode": "array",
          "cost-metric": "ane-path"
        },
        "endpoints": {
          "srcs": [ "ipv4:192.0.2.2" ],
          "dsts": [ "ipv4:192.0.2.89",
                    "ipv4:203.0.113.45" ]
        }
      }
    },
    {
      "resource-id": "propmap-availbw",
      "input": 
        let $propmap := collection("endpoint-path-vector")
            .("endpoint-cost-map")
        return {
          "entities": [ 
            distinct-values(flatten(
              for $src in keys($propmap)
              let $dsts := $propmap.$src
              return flatten(
                for $dst in keys($dsts)
                return $dsts.$dst
              ))
            ]
          },
          "properties": [ "availbw" ]
        }
    }
  ]
}
HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=path-vector-query

--path-vector-query
Content-Type: application/alto-endpointcost+json

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

--path-vector-query
Content-Type: application/alto-propmap+json

{
    "property-map": {
        "ane:L001": { "availbw": 50 },
        "ane:L003": { "availbw": 48 },
        "ane:L004": { "availbw": 55 },
        "ane:L005": { "availbw": 60 },
        "ane:L007": { "availbw": 35 }
    }
}

10. Compatibility

10.1. Compatibility with Legacy ALTO Clients/Servers

The multipart query service is a new ALTO service using the new media type. So the legacy ALTO client cannot identify this service from the IRD of the ALTO server supporting it. And the legacy ALTO server also cannot interpret the request of a multipart query service sent by the ALTO client.
10.2. Compatibility with Existing Protocol Extensions

The multipart query service can use any ALTO resources exchanging JSON data in request/response mechanism. So all the known ALTO extensions like ALTO Calendar [I-D.ietf-alto-cost-calendar], Multi-Cost [RFC8189] and the Path Vector [I-D.ietf-alto-path-vector] extension, which does not change the request/response mechanism, are compatible with the multipart query service.

11. Misc Considerations

11.1. Support Incremental Update

Because the response body entry of the multipart query resource is not a single JSON object, it may not be compatible with the current incremental update representation used in [I-D.ietf-alto-incr-update-sse].

11.2. Anonymous Resources

Some use cases may need the server generates "anonymous" ALTO resources for the on-demand information. The "anonymous" ALTO resources usually cannot appear alone but need to bind with some "non-anonymous" ALTO resources.

12. Security Considerations

Allow the ALTO clients to upload the query language script may not be safe. The code injection and many potential attacks can be conducted. The security issue should be discussed and considered.

To avoid the attacks like the code injection, this document recommends the following approaches:

Database Isolation: Some clients may attempt to access the secure database inside the server. Isolate the data into the different databases can reduce the risk of the information leak.

Application Container Isolation: Attackers may inject harmful code into the input query programs to attempt to access the system control. To avoid this, each query process is recommended to be isolated using the application container.

Resource Limit: Even attackers cannot get the permission to crack the data or the system, they can still inject some heavy-load programs to consume the server resources. Thus, limiting the memory usage and execution time of each query process is highly recommended.
13. IANA Considerations

13.1. application/alto-* Media Types

This document registers an additional ALTO media type, listed in Table 1.

<table>
<thead>
<tr>
<th>Type</th>
<th>Subtype</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>alto-multipartquery+json</td>
<td>Section 6.4</td>
</tr>
</tbody>
</table>

Table 1: Additional ALTO Media Type.

Type name: application

Subtype name: This document registers multiple subtypes, as listed in Table 1.

Required parameters: n/a

Optional parameters: n/a

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

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

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

Published specification: This document is the specification for these media types; see Table 1 for the section documenting each media type.

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

Additional information:

Magic number(s): n/a

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

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

Intended usage: COMMON

Restrictions on usage: n/a

Author: See Authors’ Addresses section.

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

14. Acknowledgements

15. References

15.1. Normative References


15.2. Informative References

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

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

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

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

[JSONIQ]

[RFC8189]

[RFC8259]

[W3CXQUERY]

Appendix A. Figures

TODO: Put additional figures here if we have.
Author’s Address

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

Email: jingxuan.n.zhang@gmail.com