ALTO WG
Internet-Draft
Intended status: Standards Track
Expires: January 5, 2018

K. Gao
Tsinghua University
X. Wang
Tongji University
Q. Xiang
Tongji/Yale University
C. Gu
Tongji University
Y. Yang
Yale University
G. Chen
Huawei

July 4, 2017

A Recommendation for Compressing ALTO Path Vectors
draft-gao-alto-routing-state-abstraction-06.txt

Abstract

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

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

In this document, an algorithm is described which can effectively reduce the redundancy in the network view while still providing the same information as in the original path vectors. The algorithm is fully compatible with the path vector extension and has several by-products which can be leveraged by other extensions to achieve better performances.

Requirements Language

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

A straw man approach, especially in the context of Software Defined Networking (SDN) where network providers have a global view, can compute the path vectors by retrieving the paths for all requested flows and returning the links on those paths as abstract network elements. However, this approach has several drawbacks:

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

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

This document describes an algorithm that identifies redundant abstract network elements and reduces them as much as possible. The algorithm, namely the "equivalent transformation" algorithm, can be integrated with any implementation of the path vector extension as a post-processing step. As the name suggests, this algorithm essentially conducts "equivalent" transformations on the original path vectors, removes redundant information and obtains a more compact view.

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

Furthermore, the redundancy check algorithm can generate filters for incremental updates of path vector queries. It can also accept customized parameters from ALTO clients to achieve even better compression results.

This document is organized as follows. Section 4 gives a concrete example to demonstrate the importance of compressing path vectors. Section 5 gives the equivalent transformation algorithm and Section 6 introduces a reference implementation of redundancy check algorithm. Section 7 discusses how to generate filters for ALTO incremental update services and Section 8 introduces an optional extension which allows ALTO clients to share certain information and further improves the performance of equivalent transformation. Finally, Section 9 and Section 10 discuss security and IANA considerations.

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

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

1. the equivalent transformation algorithm which compresses original the path vectors to provide a more compact network view,

2. a reference implementation of the redundancy check algorithm which provides near-optimal results, and

3. the client constraint map extension which allows ALTO clients to optionally provide additional client-side information to help further reduce the communication overhead.

The document also discusses how the algorithms and extension introduced here can cooperate with existing working group drafts, such as Incremental Updates, Multi-Cost and Cost Calendar.

The latest version (-06) also fixed some minor issues in -04 and -05.

3. Terminology

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

4. Compressing Path Vectors

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

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

Figure 1: Raw Network Topology

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

Table 1: Description of the Links

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

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

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

Figure 2: Path Vectors Returned by Straw Man Implementation

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

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

Figure 3: Linear Constraints Represented by the Path Vectors

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

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

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

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

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

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

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

Figure 5: Path Vectors after Removing Redundant Elements

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

To reduce the communication overhead and improve the privacy protection of the path vector extension, an algorithm is described in this document to efficiently compute the compact path vectors.

5. Equivalent Transformation Algorithm

This section describes the path vector compression algorithm, namely the "equivalent transformation" algorithm.

5.1. Parameters and Variables

The equivalent transformation algorithm accepts 3 parameters: the original path vectors P, the corresponding abstract network element property map M, and a redundancy check algorithm R(P, M).

Original path vectors P: The original path vectors P MUST have the format of a cost map or an endpoint cost map, where each cost value is a JSONArray of abstract network element names, as defined in [I-D.ietf-alto-path-vector].

Abstract network element property map M: The abstract network element property map M MUST contain all the abstract network elements whose names are included in the original path vectors P. It MUST contain at least one valid ALTO cost type which is supported by the corresponding path vector service, but MUST NOT contain ordinal values. Unless it is specifically defined in another extension, the cost values MUST follow the associative addition rule, e.g. cost(ane1) + cost(ane2) = cost(ane1 o ane2) = cost(ane2) + cost(ane1) where ne1 o ne2 represents a virtual "ane-path" consisting of ane1 and ane2. One exception is bandwidth, where the "addition" is actually a "minimum" function.

Redundancy check algorithm R(P, M): The redundancy check algorithm R(P, M) MUST accept the two parameters P and M as specified above. It MUST return a list of abstract network element names, representing those whose corresponding constraints are redundant.

In addition to the parameters mentioned above, the algorithm also maintains the following variables.

Temporary path vectors P0: P0 store the temporary values after each step of equivalent transformation.

Temporary abstract network element property map M0: M0 stores the temporary value of an abstract network element property map after each step of equivalent transformation.

Reverse abstract network element map RM: RM is a map whose key is an abstract network element name with the value being a set of endpoint/PID pairs.

Redundant abstract network element set S: S contains the result of the redundancy check algorithm R(P, M).

5.2. Algorithm Description

The equivalent transformation consists of the following steps:

1. When the algorithm starts execution, it sets P0 = P and M0 = M

2. For each abstract network element name "n", find the set of endpoint/PID pairs {(a, b)} where "n" appears in the path vector of a -> b in P0. Put the resultant set of endpoint/PID pairs in RM with "n" as the key.

3. Group RM by the value sets, e.g. put all (n, v) which have the same set of endpoint/PID pairs in the same group. It is guaranteed that each abstract network element name only appears once in one group. Now use the groups to construct a partition of all abstract network element names, where each partition contains all the abstract network element names from a single group. Each partition is associated with a unique ID, which follows the format of an abstract network element name as defined in [I-D.ietf-alto-path-vector].

4. For each endpoint/PID pair in P0, replace the abstract network element names with their group IDs. Construct an empty abstract network element property map M1. For each group, create an abstract network element property entry "e" where each abstract network element property is the "sum" of the abstract network

element property values in M0 of all abstract network elements in the group. Put "e" in M1 with the group ID as the key and also the abstract network element name. Replace M0 with M1.

5. Pass P0 and M0 to redundancy check algorithm R(P,M), and store the result in S.

6. If only bandwidth is contained in M0, go to 7. Otherwise, go to 8.

7. For each endpoint/PID pair, remove the abstract network element names in S from the path vectors. Remove the entries in M0 whose keys are in S. Go to 10.

8. Construct an empty abstract network element property map M1. For each abstract network element property entry in M0, if the abstract network element name is not in S, put the entry in M1. For each abstract network element name "n" in S, find the corresponding set of endpoint/PID pairs in RM. For each pair, replace "n" in the corresponding path vector to a new unique abstract network element name and put an entry in M1 whose key is the new abstract network element name while the value being the value of "n" in M0. Replace M0 with M1.

9. Repeat steps 2-4 and go to 10.

10. Create a virtual abstract network element "n" with a unique abstract network element name and sufficiently large bandwidth value. For each endpoint/PID pair in P, if the path vector is an empty set (this only happens when only bandwidth is requested), put the name of "n" in the path vector and add "n" to M0.

11. Return P0 as the path vector response and M0 as the corresponding abstract network element property map.

The term "sum" in step 4 is in quotes because the exact meaning depends on the property types. As stated earlier, the values MUST NOT be ordinal and MUST follow the associative addition rule unless specifically defined in a later extension. This document defines one exception -- bandwidth -- whose addition operator is the "minimum" function, which satisfies the associative addition rule.

6. Recommended Redundancy Check Algorithm

In this section, an algorithm is described as a reference implementation of the redundancy check algorithm R(P, M).

6.1. Parameters and Variables

The algorithm takes two parameters: the path vectors P and the corresponding abstract network element property map M.

Path vectors P: As specified in Section 5.1.

Abstract network element property map M: As specified in Section 5.1.

In addition to the parameters, this algorithm also maintains the following variable.

Set of linear constraints C: The set of linear constraints which can be derived from P and M.

6.2. Algorithm Description

The algorithm consists of the following steps:

1. If the abstract network element properties in M do not include bandwidth, return the set of all abstract network elements.

2. Construct the set of linear constraints, C. For each endpoint/PID pair, define a variable "x_i" with a unique ID "i". Construct RM as specified in step 3 of Section 5.2. For each abstract network element "n", find the corresponding set of endpoint/PID pairs "p_n" in RM. Construct a linear constraint "c_n: A_n X <= b_n". The left hand side is the sum of all the variables "{x_i}" whose coefficient "a_i" is 1 if the associated pair is in "p_n" and otherwise 0. The right hand side is the bandwidth of "n". Put "c_n" in C.

3. For each "c_i: A_i X <= b_i" in C, construct a new linear programming problem:

   max A_i X,

where A_j X <= b_j, j is not equal to i.

4. Solve this linear programming problem, let the maximum value be "z". If "z <= b_i", this constraint is redundant. Otherwise the constraint is NOT redundant.

5. Repeat steps 3-4 until the redundancy of all abstract network elements in the original path vectors has been identified. Return the set of abstract network elements whose corresponding linear constraints are redundant.

7. Reducing Unnecessary Incremental Updates

This section describes how an ALTO server implementation can use the results in the redundancy check algorithm described in Section 6 to filter unnecessary incremental updates.

Consider the example in Section 4 where the bandwidth of link1 (s1<=>s5) has increased from 100 Mbps to 150 Mbps. Straw man approaches may push incremental updates to ALTO clients without considering how the value changes. On the other hand, this link is not included in the path vectors after equivalent transformation, and one can conclude from the redundancy check algorithm Section 6.2 that it can only be non-redundant if "b_i < z <= 100 Mbps". Since the new "b_i" is 150 Mbps, this condition does not hold, i.e., link1 is still redundant in the updated path vector. In that case, ALTO server MAY NOT push the incremental updates.

However, this filter only works for redundant abstract network elements, e.g. "z <= b_i". In all other cases, the path vectors have to be recomputed to guarantee equivalence.

8. Extension for Customized Input Parameters

This section introduces an optional extension which can be leveraged by ALTO clients to help reduce the communication overhead of path vector services. In order to do so, a revised version of Section 6 must be introduced.

8.1. Parameters and Variables

The algorithm takes three parameters: the path vectors P, the corresponding abstract network element property map M, and client constraint map CM.

Path vectors P: Same as in Section 6.1.

Abstract network element property map M: Same as in Section 6.1.

Client constraint map CM: Client constraint map CM has the same format as a cost map, where the first key represents the source endpoint address/PID name, the second key represents the destination endpoint address/PID name, and the value is a non-negative float number representing the upper bound bandwidth value for the given endpoint/PID pair.

The algorithm also maintains the following variables:

Set of linear constraints C: The same as C in Section 6.1.

Set of client constraints CC: The set of linear constraints which can be derived from CM.

8.2. Algorithm Description

The algorithm consists of the following steps:

1. The same as step 1 in Section 6.2.

2. The same as step 2 in Section 6.2.

3. For each endpoint/PID pair in CM, assume the value is a non-negative number "u". If the pair is also in the original path vector, construct a linear constraint "cc: x_i <= u" and add it to CC where "x_i" is the variable associated with the same pair in step 2.

4. For each "c_i: A_i X <= b_i" in C, construct a linear programming problem:

```
      max A_i X
where A_j X <= b_j in C, j is not equal to i
      x_j   <= u_j in CC
```

5. The same as step 4 in Section 6.2.

6. The same as step 5 in Section 6.2.

Clearly, the feasible region for each linear programming problem in step 4 is smaller or equal to the one in Section 6.2. Thus, each abstract network element has a higher chance of being redundant.

8.3. ALTO Extension for Client Constraint Map

This section describes the extensions needed to enable client constraint map.

Any ALTO resource/service that supports client constraint map MUST also support the path vector extension and accept the cost type whose cost mode is "array" and cost metric "ane-path". This extension does not change the specifications on "media types", "HTTP methods", "uses", and "response".

8.3.1. Capabilities

The client constraint map extension requires a new capability field "client-constraint-map" in the IRD.

```
object {
    JSONString cost-type-names<1..*>;
    [JSONBool client-constraint-map;]
    ... capabilities defined by other extensions
} ClientConstraintMapCapabilities;
```

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

client-constraint-map: If present and the value is "true", it means the resource/service accepts client constraint map in the parameters. Otherwise, the client MUST assume the server does not support this extension and MUST NOT include client constraint map in input parameters.

8.3.2. Accept Input Parameters

For filtered cost map with client constraint map extension, it MUST accept the following parameters:

```
object {
    [CostType cost-type;]
    [PIDFilter pids;]
    [ClientConstraintPIDMap client-constraint-map;]
    ... input parameters defined by other extensions
} ReqFilteredCostMap;

object {
    PIDName -> ClientConstraintPIDGroup;
} ClientConstriantPIDMap;

object {
    PIDName -> JSONNumber;
} ClientConstraintPIDGroup;
```

cost-type: As defined in Section 11.3.2.3 in [RFC7285]. It MUST have the cost mode "array" and cost metric "ane-path" if the field "client-constraint-map" is present. Otherwise, the ALTO server MUST return an error with error code "E_INVALID_FIELD_VALUE".

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

client-constraint-map: The client constraint map MUST have the format as a JSON object "ClientConstraintPIDMap". All the PID names in the client constraint map MUST also be included in "pids", otherwise the ALTO server MUST return an error with error code "E_INVALID_FIELD_VALUE". If the value for a given PID pair is 0, ALTO server MUST not included this pair in the path vector.

Similarly, the input parameters for endpoint cost services with client constraint map MUST have the following format:

```
object {
    [CostType cost-type;]
    EndpointFilter endpoints;
    [ClientConstraintEndpointMap client-constraint-map;]
    ... input parameters defined by other extensions
} ReqEndpointCostMap;

object {
    TypedEndpointAddr -> ClientConstraintEndpointGroup;
} ClientConstriantEndpointMap;

object {
    TypedEndpointAddr -> JSONNumber;
} ClientConstraintEndpointGroup;
```

cost-type: Same as above.

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

client-constraint-map: The client constraint map contained in the parameter MUST have the format of a JSON object "ClientConstriantEndpointMap". All the endpoint addresses MUST also appear in the "endpoints", otherwise the ALTO server MUST return an error with an error code "E_INVALID_FIELD_VALUE". If the value for a given endpoint pair is 0, ALTO server MUST NOT included this pair in the path vector.

8.4. Examples

This section contains a series of examples for the client constraint map extension.

8.4.1. Information Resource Directory Example

```
   {
     "meta": {
       "cost-types": {
         "pv-ane": {
           "cost-mode": "array",
           "cost-metric": "ane-path"
         }
       }
     },
     "resource": {
       "default-network-map": {
         "uri": "http://alto.example.com/networkmap",
         "media-type": "application/alto-networkmap+json"
       },
       "filtered-multi-cost-map": {
         "uri": "http://alto.example.com/costmap/filtered",
         "media-type": "application/alto-costmap+json",
         "accepts": "application/alto-costmapfilter+json",
         "uses": ["default-network-map"],
         "capabilities": {
           "cost-type-names": ["pv-ane"],
           "property-map": "default-prop-map",
           "client-constraint-map": true
         }
       },
       "default-endpoint-cost-map": {
         "uri": "http://alto.example.com/endpointcost/lookup",
         "media-type": "application/alto-endpointcostmap+json",
         "accepts": "application/alto-endpointcostparams+json",
         "capabilities": {
           "cost-type-names": ["pv-ane"],
           "client-constraint-map": true
         }
       },
       "default-prop-map": {
         "uri": "http://alto.example.com/default-prop-map",
         "media-type": "application/alto-propmap+json",
         "accepts": "application/alto-propmapparams+json",
         "capabilities": {
           "domain-types": ["ane"],
           "prop-types": [ "availbw" ]
         }
       }
     }
   }
```

8.4.2. Filtered Cost Map Example

Assume we use the example in Section 4 and PID1-PID4 are mapped to eh1-eh4 respectively.

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

{
  "cost-type": {
    "cost-mode": "array",
    "cost-metric": "ane-path"
  },
  "pids": {
    "srcs": ["PID1", "PID3"],
    "dsts": ["PID2", "PID4"]
  },
  "client-constraint-map": {
    "PID1": { "PID2": 40, "PID4": 0 },
    "PID3": { "PID2": 50, "PID4": 50 }
  }
}

HTTP/1.1 200 OK
Content-Length: [TBD]
Content-Type: multipart/related; boundary=31415926
```

```
--31415926
Content-Type: application/alto-costmap+json

{
  "meta": {
    "dependent-vtags": [
      {
        "resource-id": "default-network-map",
        "tag": "75ed013b3cb58f896e839582504f622838ce670f"
      }
    ],
    "vtag": {
      "resource-id": "cost-map-pv",
      "tag": "27612897acf278ffu3287c284dd28841da78213",
      "query-id": "query-cost-map-pv-276128"
    },
    "cost-type": {
      "cost-mode": "array",
      "cost-metric": "ane-path"
    }
  },
  "cost-map": {
    "PID1": {
      "PID2": [ "ane:1" ]
    },
    "PID3": {
      "PID2": [ "ane:1" ],
      "PID4": [ "ane:1" ]
    }
  }
}

--31415926
Content-Type: application/alto-propmap+json

{
  "property-map": {
    "ane:1": { "availbw": 100 }
  }
}

--31415926--
```

Since the bandwidth of all links is 100 Mbps, it can be easily concluded that only link5 can potentially become a bottleneck with the given client constraints. Thus, only one abstract network element is returned.

8.4.3. Endpoint Cost Service Example

Assume we use the example in Section 4 and eh1-eh4 are associated with IP addresses 192.0.2.1-192.0.2.4 respectively.

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

{
  "cost-type": {
    "cost-mode": "array",
    "cost-metric": "ane-path"
  },
  "endpoints": {
    "srcs": ["ipv4:192.0.2.1", "ipv4:192.0.2.3"],
    "dsts": ["ipv4:192.0.2.2", "ipv4:192.0.2.4"]
  },
  "client-constraint-map": {
    "ipv4:192.0.2.1": { "ipv4:192.0.2.2": 40, "ipv4:192.0.2.4": 0 },
    "ipv4:192.0.2.3": { "ipv4:192.0.2.2": 50, "ipv4:192.0.2.4": 50 }
  }
}

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

```
{
  "meta": {
    "vtag": {
      "resource-id": "default-prop-map",
      "tag": "a911354dfd1ef6555bfe7af07d3af0bfebe7c8a9",
      "query-id": "query-ecs-a91135"
    },
    "cost-type": {
      "cost-mode": "array",
      "cost-metric": "ane-path"
    }
  },
  "endpoint-cost-map": {
    "ipv4:192.0.2.1": {
      "ipv4:192.0.2.2": [ "ane:1" ]
    },
    "ipv4:192.0.2.3": {
      "ipv4:192.0.2.2": [ "ane:1" ],
      "ipv4:192.0.2.4": [ "ane:1" ]
    }
  }
}
```

Since the bandwidth for all links is 100 Mbps, it can be easily concluded that only link5 can potentially become a bottleneck with the given client constraints. Thus, only one abstract network element is returned.

In this example, the abstract network element property map is not attached so the client SHOULD send another request to fetch the details about the abstract network elements.

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

{
  "query-id": "query-ecs-a91135",
  "entities": [ "ane:1" ],
  "properties": [ "availbw" ]
}
```

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

{
  "property-map": {
    "ane:1": { "availbw": 100 }
  }
}
```

8.5. Compatibility

The client constraint map is fully compatible with the path vector extension. For other extensions such as multi-cost [I-D.ietf-alto-multi-cost] and cost calendar [I-D.ietf-alto-cost-calendar], the input parameters MUST still follow the definition of "client-constraint-map" but can adjust the requirements for other fields.

Since the client constraint map extension is fully compatible with the path vector extension, it does not alter the compatibility with other extensions such as multi-cost and cost calendar.

9. Security Considerations

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

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

The client constraint map extension defined in Section 8.3 can potentially leak client-side information to ALTO servers. ALTO client implementations MUST take information security into consideration when using this extension, for example, only activating this extension when the ALTO server is considered a trusted party.

ALTO clients can also obfuscate the information contained in a request, for example, providing larger values than actual upper bounds. Such obfuscation will not affect the correctness of the response, but can potentially affect the reduction effect of client constraint map.

10. IANA Considerations

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

11. Acknowledgments

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

12. References

12.1. Normative References

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

12.2. Informative References

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

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

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

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

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

Authors’ Addresses

Kai Gao
Tsinghua University
30 Shuangqinglu Street
Beijing 100084
China

Email: gaok12@mails.tsinghua.edu.cn

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

Email: xinwang2014@hotmail.com

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

Email: qiao.xiang@cs.yale.edu

Chen Gu
Tongji University
4800 CaoAn Road
Shanghai 210000
China

Email: gc19931011jy@gmail.com

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

Email: yry@cs.yale.edu

G. Robert Chen
Huawei
Nanjing
China

Email: chenguohai@huawei.com