ALTO Working Group
Internet-Draft
Intended status: Informational
Expires: September 10, 2015

X. Shi
Yale University
Y. Yang
Tongji/Yale University
M. Scharf
Alcatel-Lucent Bell Labs
March 9, 2015

A YANG Data Model for Base ALTO Data
draft-shi-alto-yang-model-01

Abstract

yry: add

Table of Contents

1. Introduction

This document defines a YANG model for the base ALTO information resources defined in [RFC7285]. Such a model can provide value in multiple aspects. For example, the tree diagram of the YANG model may provide a concise visualization of the information contained in ALTO base information resources.

A particular goal of our designing the model is to provide a standard way for a management entity or even an ALTO client to provision or update ALTO information resources stored at an ALTO server. We refer

to an entity that interacts with the ALTO server through the YANG model as an ALTO/YANG client. ALTO clients and ALTO/YANG clients are different in that the former uses [RFC7285] to obtain ALTO information resources. An ALTO/YANG client, on the other hand, uses NETCONF [RFC6241] or RESTCONF [RESTCONF], which will be based on the

YANG model, to retrieve, and more importantly, update the ALTO information resources at the ALTO server. Figure 1 is a non-normative diagram illustrating the related entities. How an ALTO/YANG server stores and pushes updates to the ALTO server belongs to implementation.

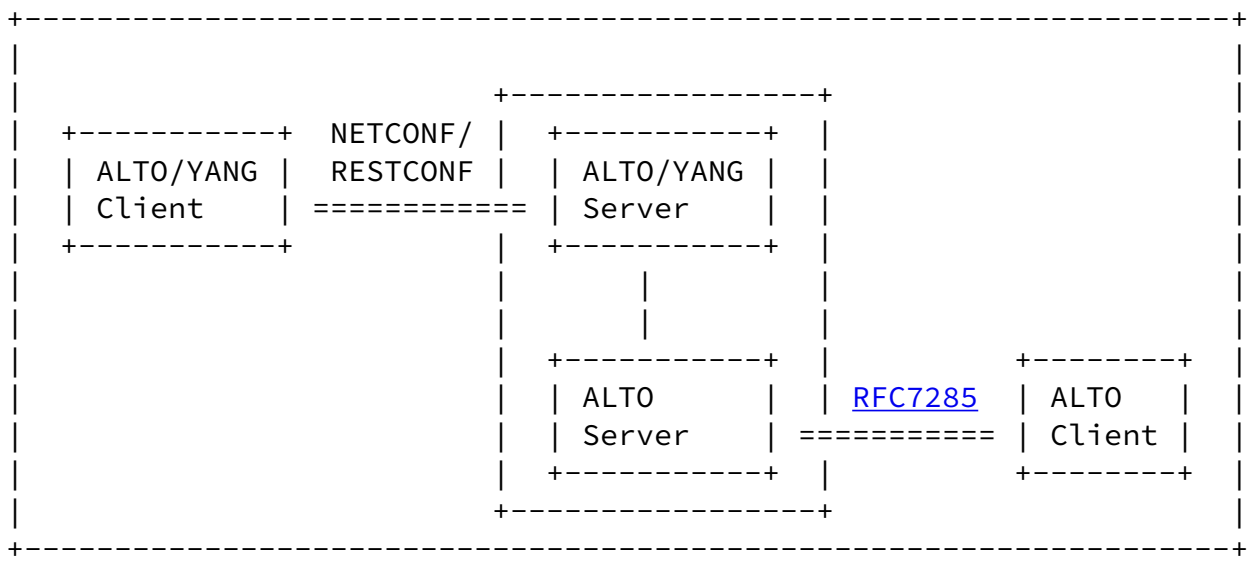

Figure 1: ALTO/YANG Components.

The ALTO YANG model defined in this document contains only data instances. There are no definitions for RPCs or notifications. Specifically, the model defines data instances for ALTO information resource directory (IRD), network maps, cost maps, and the endpoint property map.

As a result of the preceding data-instances only design, the model will not directly provide other ALTO information services such as filtered maps and endpoint cost services. This is consistent with the design goal: using the YANG model as a basis for ALTO base information provisioning. For example, an ALTO/YANG client may use the NETCONF filtering features (See Section 6 and Section 8.9 of

[RFC6241]) to approximate the ALTO filtering services. We provide examples and templates in the appendix.

The rest of the document is organized as follows. In Section 2, we give the details of the ALTO YANG model. In Section 3, we give a non-normative specification on how one may use NETCONF and RESTCONF to update ALTO YANG data. Section 4 and Section 5 give IANA and security considerations respectively.

2. ALTO YANG Model

Figure 2 shows the tree diagram of the structure of the ALTO/YANG data model. See Appendix A for the complete YANG model.

```
module: alto-service
   +--rw resources
      +--rw IRD
      |  +--rw meta
      |  |  +--rw cost-types* [cost-type-name]
      |  |  |  +--rw cost-type-name    cost-type-name
      |  |  |  +--rw cost-mode         cost-mode
      |  |  |  +--rw cost-metric       cost-metric
      |  |  |  +--rw description?      string
      |  |  +--rw default-alto-network-map    resource-id
      |  +--rw resources* [resource-id]
      |     +--rw resource-id     resource-id
      |     +--rw uri             inet:uri
      |     +--rw media-type      media-type
      |     +--rw accepts*        media-type
      |     +--rw capabilities
      |     |  +--rw cost-constraints?   boolean
      |     |  +--rw cost-type-names*    cost-type-name
      |     |  +--rw prop-types*         endpoint-property-type
      |     +--rw uses*           resource-id
      +--rw network-maps
      |  +--rw network-map* [resource-id]
      |     +--rw resource-id    alto:resource-id
      |     +--rw tag            alto:tag-string
```

```
|     +--rw map* [pid]
|        +--rw pid                          alto:pid-name
|        +--rw endpoint-address-group* [address-type]
|           +--rw address-type       endpoint-address-type
|           +--rw endpoint-prefix*   endpoint-prefix
+--rw cost-maps
|  +--rw cost-map* [resource-id]
|     +--rw resource-id    alto:resource-id
|     +--rw tag            alto:tag-string
|     +--rw meta
|     |  +--rw dependent-vtags*
|     |  |  +--rw resource-id    resource-id
|     |  |  +--rw tag            tag-string
|     |  +--rw cost-type
|     |     +--rw cost-mode      cost-mode
|     |     +--rw cost-metric    cost-metric
|     |     +--rw description?   string
|     +--rw map* [src]
|        +--rw src          alto:pid-name
```



```
|        +--rw dst-costs* [dst]
|           +--rw dst     alto:pid-name
|           +--rw cost
+--rw endpoint-property-map
   +--rw meta
   |  +--rw dependent-vtags*
   |     +--rw resource-id    resource-id
   |     +--rw tag            tag-string
   +--rw endpoint-properties* [endpoint]
      +--rw endpoint      typed-endpoint-address
      +--rw properties* [property-type]
         +--rw property-type    endpoint-property-type
         +--rw property         endpoint-property-value
```

Figure 2: Tree Diagram of ALTO YANG Model.

This model has certain limitations concerning multiple versions of resources. In particular, network maps and cost maps are modeled as lists keyed by their resource ids only. Therefore, the datastore may store only the most updated copy of the resources with their appropriate tags. The advantages of this design include: 1) the server does not need to keep all version history of all resources; 2)

the clients need not know the tag in order to request the most recent resource. The disadvantages include: 1) the server must keep all updated resources consistent as they are updated; 2) there is no way for a client to request an old version of a resource.

3. Use Case: Using NETCONF/RESTCONF to Update ALTO Information Resources

The Network Configuration (NETCONF) Protocol [RFC6241] provides mechanisms to manipulate configuration data and state data of network devices via some of its standard operations (e.g, <get-config>). The NETCONF protocol messages are encoded in YANG-modeled XML and transported via persistent connections such as SSH. Given the datastore, NETCONF will be able to handle simple retrieval and update (insertion, deletion, and replacement) of configuration or state data. Our focus is on the updates of configuration or state data.

The RESTCONF Protocol ([RESTCONF]) uses restful HTTP operations to provide CRUD operations on a NETCONF datastore containing YANG-defined data. RESTCONF is similar to NETCONF in terms of usage, hence we elaborate on NETCONF update operations in this section and provide more details in Appendix C.

3.1. Update Operations

The most relevant update operation that NETCONF provides is the <edit-config> operation, which allows a client to edit the configuration data in different ways (e.g., merge, replace, etc.). Section 7.2 of [RFC6241] contains the specification of <edit-config>. This section gives two non-normative examples on modifying network and cost maps to illustrate the usage.

3.1.1. <edit-config> Examples

3.1.1.1. Modifying Network Maps

Suppose we would like to modify a network map in the following manner: remove endpoint prefix "192.0.2.0/24" from PID1 and add it to

PID 2. The <edit-config> operation would be the following:


```
<rpc message-id=SEQ-NUM
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config
      xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
        <resources
          xmlns="urn:ietf:params:xml:ns:yang:alto-service">
          <network-maps>
            <network-map>
              <resource-id>myNetMap1</resource-id>
              <tag>ANEWTAG</tag> <TODO NEED VALID>
              <map>
                <pid>PID1</pid>
                <endpoint-address-group>
                  <endpoint-prefix xc:operation="delete">
                    192.0.2.0/24
                  </endpoint-prefix>
                </endpoint-address-group>
              </map>
              <map>
                <pid>PID2</pid>
                <endpoint-address-group>
                  <endpoint-prefix xc:operation="create">
                    192.0.2.0/24
                  </endpoint-prefix>
                </endpoint-address-group>
              </map>
            </network-map>
          </network-maps>
        </resources>
      </config>
    </edit-config>
  </rpc>
```

Note that we specified different "suboperations" for the two endpoint prefixes, one to "delete" and the other to "create". The reply from the NETCONF server would be either <rpc-reply> with <OK> or an <rpc-error>.

3.1.1.2. Modifying Cost Maps

To change the cost from PID3 to PID2 from 15 to 10 in myCostMap1, the NETCONF RPC would be the following:


```
  <rpc message-id=SEQ-NUM
```

```
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
      <target>
        <running/>
      </target>
      <config
        xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <resources
          xmlns="urn:ietf:params:xml:ns:yang:alto-service">
          <cost-maps>
            <cost-map>
              <resource-id>myCostMap1</resource-id>
              <tag>ANEWTAG</tag> <TODO NEED VALID>
              <map>
                <src>PID3</src>
                <dst-costs>
                  <dst>PID2</dst>
                  <cost>10</cost>
                </dst-costs>
              </map>
            </cost-map>
          </cost-maps>
        </resources>
      </config>
    </edit-config>
  </rpc>
```

3.1.2. Consistency Considerations

The ALTO/YANG server MUST validate update operations to maintain the consistency of the ALTO resources. Specifically, this document specifies the following consistency requirements:

- ALTO/YANG server MUST check that each update operation MUST include a new tag to indicate the update.

- TODO: Add more

3.2. Other Operations

3.2.1. <delete-config>

3.2.1.1. Examples

4. Security Considerations

This document depends on standard NETCONF/RESTCONF security mechanisms to achieve authentication and authorization of update operations.

5. IANA Considerations

This document does not have IANA considerations.

6. References

[RESTCONF] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-04 (work in progress), January 2015.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2010.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6020, October 2010.

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

[RFC7285] Almi, R., Penno, R., Yang, Y., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", RFC 7285, September 2014.

[draft-ietf-netmod-yang-json] Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-01 (work in progress), October 2014.

[draft-shi-alto-yang-json] Shi, X. and Y. Yang, "Modeling JSON Messages Using YANG", draft-shi-alto-yang-json-00 (work in progress), October 2014.

Appendix A. YANG Data Model for ALTO Protocol

A.1. ALTO/YANG: Common Data Types

```
module alto-service-types {
  yang-version 1;

  namespace "urn:ietf:params:xml:ns:yang:alto-service-types";
  // TODO: replace with IANA namespace when assigned

  prefix "alto";

  import ietf-inet-types {
    prefix inet;
  }

  organization "ALTO WG";
  contact "alto@ietf.org";

  description
    "This module defines the data types and groupings for a semantically
     equivalent data model for the ALTO services defined in RFC7285.";

  revision 2014-11-01 {
    description "Separate types module";
  }

  revision 2014-10-24 {
    description "Initial version.";
  }

  /*********************
   * TYPE DEFINITIONS *
   *********************/

  /*************************************************************
    Definitions for addresses
```

ALTO RFC7285 uses the following addresses, as shown in the examples below:

- Endpoint property service (Sec. 11.4.1.7):
  "endpoints" : ["ipv4:192.0.2.34",
                   "ipv4:203.0.113.129"]
- Endpoint cost service (Sec. 11.5.1.7):
  "endpoints" : {
  "srcs": ["ipv4:192.0.2.2"],


```
      "dsts": [
        "ipv4:192.0.2.89",
        "ipv4:198.51.100.34",
        "ipv4:203.0.113.45"
    - Network map (Sec. 11.2.1.7.):
        "ipv4": [
          "192.0.2.0/24",
          "198.51.100.0/25"
        ],
        "ipv6": [
          "2001:db8:0:1::/64",
          "2001:db8:0:2::/64"
        ]

   To handle the proceeding, we need the following definitions:
      ipv4-address (e.g., 192.0.2.0, already defined in rfc6991),
      ipv6-address (already defined in rfc6991),
      ipv4-prefix (e.g., 192.0.2.0/24, already defined in rfc6991),
      ipv6-prefix (defined in rfc6991),
      typed-ipv4-address (e.g., ipv4:192.0.2.1, to be defined below)
      typed-ipv6-address
      typed-ipv4-prefix-list (e.g., "ipv4": [
           "192.0.2.0/24",
           "198.51.100.0/25"
        ],

  **************************************************************/

  /*
     First define typed-ipv4-address and typed-ipv6-address, as used
     by endpoint services.
```

```
     The ideal case is to define it as "ipv4:"+ipv4-address, but there
     is not such a type constructor (YANG EXTENSION).  Hence, the
     current definition cuts-and-pastes (i.e., repeats verbatim) the
     definition of ipv4-address and prepend "ipv4:". The downside is
     that if someone redefines ipv4-address, there could be
     inconsistency.
   */

  typedef typed-ipv4-address {
    type string {
        pattern
          'ipv4:(([0-9]|[1-9][0-9]|1[0-9][0-9]|'
        + '2[0-4][0-9]|25[0-5])\.){3}'
        + '([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])'
        + '(%[\p{N}\p{L}]+)?';
      }
```



```
  }


  typedef typed-ipv6-address {
    type string {
      pattern 'ipv6:((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}'
            + '((([0-9a-fA-F]{0,4}:)?(:|[0-9a-fA-F]{0,4}))|'
            + '(((25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])\.){3}'
            + '(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])))'
            + '(%[\p{N}\p{L}]+)?';
      pattern 'ipv6:(([^:]+:){6}(([^:]+:[^:]+)|(.*\..*))|'
            + '((([^:]+:)*[^:]+)?::(([^:]+:)*[^:]+)?)'
            + '(%.+)?';
    }
  }

  typedef typed-endpoint-address {
    type union {
      type typed-ipv4-address;
      type typed-ipv6-address;
      // EXTENSION: ADD NEW TYPE HERE.
    }
    description
      "Ref: RFC7285 Sec. 10.4.1 Typed Endpoint Addresses" +
      "= AddressType:EndpointAddr";
```

```
  }

  /* Next, we define endpoint address group, as used in the definition
     of ALTO network maps. Specifically, an endpoint address group in
     ALTO is defined as a key-value store, with address type as key,
     and an array of prefix as the value of each key:

     EndpointAddrGroup. RFC7285 Sec. 10.4.5." +
       object-map {
       AddressType -> endpoint-prefix<0..*>;
     } EndpointAddrGroup;

     There are two challenges:

     1) To specify that AddressType is key, we must use the list type,
     which is the only type that one can specify key. However, the
     current JSON-YANG encoding generates an array, instead of a
     key-value map;

     2) Ideally, we want to enforce address type and prefix
     consistency; for example, an ipv6 prefix in an ipv4 type should
     not be allowed. However, we encounter problems. We leave this as
     an OPEN ISSUE.
```



```
  */

  typedef endpoint-address-type {
    type union {
      type enumeration {
        enum ipv4;
        enum ipv6;
        // EXTENSION: ADD NEW TYPE HERE
      }
    }
    description
      "Ref: RFC7285 Sec 2.2.";
  }

  typedef endpoint-prefix {
    type inet:ip-prefix;
    description
      "endpoint prefix, identical to ip-prefix defined in RFC6991.";
```

```
  }

  grouping endpoint-address-group {
    list endpoint-address-group {
      key address-type;
      leaf address-type {
        type endpoint-address-type;
        mandatory true;
      }
      leaf-list endpoint-prefix {
        type endpoint-prefix;
      }
    }
    description
      "EndpointAddrGroup. RFC7285 Sec. 10.4.5." +
      " object-map {
          AddressType -> endpoint-prefix<0..*>;
        } EndpointAddrGroup;";
  }

  /*****************************************************************
   * Definitions for IDs and names
   *
   * ALTO defines the following concepts that are names and IDs:
   *
   *   pid name (used in network map, cost map),
   *   resource IDs (used to identify alto network/cost maps),
   *   version tag (used to indicate uniqueness of resource),
   *   cost-type-name (used in IRD),
   *   cost-metric,
```



```
   *   cost-mode
   *
   * We group their definitions together below.
   *****************************************************************/

  typedef valid-id-string {
    type string {
      length "1..64";
      pattern "[0-9a-zA-Z_\-:@\.]+";
    }
    description
```

```
      "Type for valid ID strings.";
  }

  typedef tag-string {
    type string {
      length "1..64";
      pattern "[!-~]+";
    }
    description
      "Tag. RFC7285 Sec. 10.3. U+0021-U+007E";
  }

  typedef pid-name {
    type valid-id-string;
    description
      "Name for the PID." +
      "RFC7285, Section 10.1. Note: the '.' separator MUST NOT be" +
      "used unless specifically indicated in RFC7285 or an" +
      " extension document.";
  }

  typedef resource-id {
    type valid-id-string;
    description
      "Resource-ID.";
  }

  grouping vtag {
    leaf resource-id {
      type resource-id;
      mandatory true;
    }
    leaf tag {
      type tag-string;
      mandatory true;
    }
    description
```



```
      "Version tag. Both resource-id and tag must be equal
       byte-for-byte. RFC7285 Sec. 10.3." +
      " object {
          ResourceID resource-id;
```

```
          JSONString tag;
        } VersionTag;";
  }

  grouping dependent-vtags {
    list dependent-vtags {
      key "resource-id tag";
      uses vtag;
      min-elements 1;
    }
  }

 /************************************
    Definitions for cost type and cost types

    In ALTO, a cost type consists of two required components:

      cost-metric,
      cost-mode
      and an optional description component.

    In the IRD, one can name each cost type. Such info is collected
    in a hash map called cost types.
  ************************************/

  typedef cost-metric {
    type union {
      type enumeration {
        enum routingcost {
          description
          "Default metric. MUST support. RFC7285 Sec. 6.1.1.1.";
        }
        enum hopcount {
          description
          "Hopcount metric.";
        }
        // EXTENSION: Additional cost-metric will be defined here.
      }
      type string {
        length 1..32;
        pattern "priv:[0-9a-zA-Z_\-:\.]+";
      }
    }
    description
```

```
      "Cost metric. for type string,
      'priv:' reserved for Private Use.";
  }

  typedef cost-mode {
    type enumeration {
      enum numerical {
        description
          "Numerical cost mode.";
      }
      enum ordinal {
        description
          "Ordinal cost mode.";
      }
      // EXTENSION: Additional cost-mode will be defined here.
    }
    description
      "Cost mode. MUST support at least one of numerical and ordinal";
  }

  grouping cost-type {
    leaf cost-mode {
      type cost-mode;
      mandatory true;
      description
        "Cost mode.";
    }
    leaf cost-metric {
      type cost-metric;
      mandatory true;
      description
        "Cost metric.";
    }
    leaf description {
      type string;
      description
        "Optional description field.";
    }
    description
      "Cost type. RFC7285 Sec. 10.7." +
      " object {
          CostMetric cost-metric;
          CostMode   cost-mode;
          [JSONString description;]
        } CostType;";
  }
```

```
  typedef cost-type-name {
```



```
    type valid-id-string;
    // NOTE: not fully specified in RFC7285, default as valid id
  }

  grouping cost-types {
    list cost-types {
      key cost-type-name;
      leaf cost-type-name {
        type cost-type-name;
      }
      uses cost-type;
    }
    description
      "RFC 7285 Sec. 9.2.2." +
      "object-map {
         JSONString -> CostType;
       } IRDMetaCostTypes;";
  }


  /*************************************
  * Definitions for endpoint properties *
  *************************************/
  typedef global-endpoint-property {
    type union {
      type enumeration {
        enum pid {
          description "PID property.";
        }
        // EXTENSION: other options here
      }
      type string {
        pattern "priv:[\w\-:@]+";
      }
    }
    description
      "Global endpoint property. RFC7285 Sec. 10.8.2." +
      "'priv:' for Private Use " +
      " length 1..32; '.' is not allowed";
  }
```

```
  /*
   * Ideally we would want to extend the typedef of resource-id and
   * global endpoint properties, however, YANG 1.0 does not allow
   * that, hence we simply copied the regex for resource-id over
   * verbatim.
   */
```



```
  typedef resource-specific-endpoint-property {
    type string {
      length "3..97"; //len(resource-id) + 1 + len(global-property)
      pattern "(priv:)?[\w\-:@\.]+\.[\w\-:_]+"; // resource-id.property
    }
    description
      "Resource-specific endpoint property.";
  }

  typedef endpoint-property-type {
    type union {
      type resource-specific-endpoint-property;
      type global-endpoint-property;
    }
    description
      "Endpoint property type. RFC7285 Sec. 10.8.";
  }

  typedef endpoint-property-value {
    type string;
    description
      "Endpoint property (value).";
  }

 /***********************************
  * Definitions for response header
  ***********************************/

  typedef media-type {
    type union {
      type string {
        pattern "application/alto\-.*";
      }
```

```
      type enumeration {
        enum alto-directory+json;
        enum alto-networkmap+json;
        enum alto-networkmapfilter+json;
        enum alto-costmap+json;
        enum alto-costmapfilter+json;
        enum alto-endpointprop+json;
        enum alto-endpointpropparams+json;
        enum alto-endpointcost+json;
        enum alto-endpointcostparams+json;
        enum alto-error+json;
      }
    }
  }
```



```
  grouping alto-cost {
    anyxml cost {
      mandatory true;
      description
        "ALTO cost is a JSONValue, which could be
        an object, array, string, etc. (Ref: RFC 7159 Sec.3.)";
    }
  }

  typedef constraint {
    type string {
      pattern "(gt|ge|lt|le|eq) [0-9]+";
    }
    description
      "RFC7285 Sec. 11.3.2.3. The second part must be in the" +
      "same unit as cost-metric, IEEE 754 2008 floating point.";
  }

  /*****************************************
    Groupings for ALTO information resource
  *****************************************/

  /* meta */
  grouping IRD-meta {
    uses cost-types;
    leaf default-alto-network-map {
```

```
      type resource-id;
      mandatory true;
    }
  }

  grouping network-map-meta {
    container vtag {
      uses vtag;
    }
  }

  grouping cost-map-meta {
    uses dependent-vtags {
      refine dependent-vtags {
        max-elements 1;
      }
    }
    container cost-type {
      uses cost-type;
    }
  }
```



```
  grouping endpoint-property-meta {
    uses dependent-vtags;
  }

  /* accepts (optional) */
  grouping accepts {
    leaf-list accepts {
      type media-type;
      min-elements 1;
    }
  }

  /* capabilities (capabilities) */
  grouping IRD-capabilities {
    container capabilities {
      leaf cost-constraints {
        type boolean;
      }
      leaf-list cost-type-names {
```

```
        type cost-type-name;
      }
      leaf-list prop-types {
        type endpoint-property-type;
      }
    }
  }

  /* uses (optional) */
  grouping uses {
    leaf-list uses {
      type resource-id;
      min-elements 1;
    }
  }

  /* Information Resource Directory Grouping */
  grouping IRD {
    container meta {
      uses IRD-meta;
    }
    uses IRD-data;
  }

  grouping IRD-data {
    list resources {
      key resource-id;
      leaf resource-id {
        type resource-id;
```



```
        mandatory true;
      }
      leaf uri {
        type inet:uri;
        mandatory true;
      }
      leaf media-type {
        type media-type;
        mandatory true;
      }
      uses accepts {
        when "current()";
```

```
      }
      uses IRD-capabilities {
        when "current()";
      }
      uses uses {
        when "current()";
      }
      description
        "IRDResourceEntry. RFC7285 9.2.2." +
        " object {
            JSONString      uri;
            JSONString      media-type;
            [JSONString     accepts;]
            [Capabilities   capabilities;]
            [ResourceID     uses<0..*>;]
          } IRDResourceEntry;" +
        "IRDResourceEntries. RFC7285 9.2.2." +
        " object-map {
            ResourceID  -> IRDResourceEntry;
          } IRDResourceEntries;" +
        "InformationResourceDirectory. RFC7285 9.2.2." +
        " object {
            IRDResourceEntries resources;
          } InfoResourceDirectory : ResponseEntityBase;";
    }
  }

  /* Network Map Grouping */
  grouping network-map {
    container meta {
      uses network-map-meta;
    }
    uses network-map-data;
  }

  grouping network-map-data {
```



```
    list network-map {
      key "pid";
      leaf pid {
        type pid-name;
      }
```

```
      uses endpoint-address-group;
      description
        "RFC7285 Sec. 11.2.1.6." +
        " object-map {
            PIDName -> EndpointAddrGroup;
          } NetworkMapData;";
    }
    description
      "Network map. RFC7285 Sec. 11.2.1.6." +
      "object {
         NetworkMapData network-map;
       } InfoResourceNetworkMap : ResponseEntityBase;";
  }

  /* Cost Map Grouping */
  grouping cost-map {
    container meta {
      uses cost-map-meta;
    }
    uses cost-map-data;
  }

  grouping cost-map-data {
    list cost-map {
      leaf src {
        type pid-name;
        description
          "Source PID.";
      }
      key "src";
      list dst-costs {
        leaf dst {
          type pid-name;
          description
            "Destination PID.";
        }
        key "dst";
        uses alto-cost {
          description
            "Cost from source to destination.";
        }
        description
          "The list represents the inner part of the cost matrix." +
```

```
        "DstCosts. RFC7285 Sec. 11.2.3.6." +
        " object-map {
            PIDName -> JSONValue;
          } DstCosts;";
    }
    description
      "The list represents the outer part of the cost matrix." +
      "CostMapData. RFC7285 Sec. 11.2.3.6." +
      " object-map {
          PIDName -> DstCosts;
        } CostMapData;";
  }
  description
    "Cost map. RFC7285 Sec. 11.2.3.6." +
    " object {
        CostMapData cost-map;
      } InfoResourceCostMap : ResponseEntityBase;";
}

/* Endpoint Property Map Grouping */
grouping endpoint-property-map {
  container meta {
    uses endpoint-property-meta;
  }
  uses endpoint-property-map-data;
}

grouping endpoint-property-map-data {
  list endpoint-properties {
    key endpoint;
    leaf endpoint {
      type typed-endpoint-address;
      mandatory true;
    }
    list properties {
      key property-type;
      leaf property-type {
        type endpoint-property-type;
        mandatory true;
      }
      leaf property {
        type endpoint-property-value;
        mandatory true;
      }
      description
        "EndpointProps. RFC7285 Sec. 11.4.1.6." +
        " object {
            EndpointPropertyType -> JSONValue;
```

```
            } EndpointProps;";
      }
      description
        "EndpointPropertyMapData. Sec. 11.4.1.6." +
        " object-map {
            TypedEndpointAddr -> EndpointProps;
          } EndpointPropertyMapData;";
    }
    description
      "InfoResourceEndpointProperties. Sec. 11.4.1.6." +
      " object {
          EndpointPropertyMapData endpoint-properties;
        } InfoResourceEndpointProperties : ResponseEntityBase;";
  }
}
```

Figure 3: ALTO/YANG Common Types.

A.2. ALTO/YANG Model

```
   module alto-service {
     yang-version 1;

     namespace "urn:ietf:params:xml:ns:yang:alto-service";
     // TODO: replace with IANA namespace when assigned

     prefix "as";

     import alto-service-types {
       prefix alto;
     }

     organization "ALTO WG";
     contact "alto@ietf.org";

     description
       "This module defines a semantically equivalent data model
       for the ALTO services defined in RFC7285.";

     revision 2015-03-03 {
```

```
    description "Revise according to IETF91.";
  }

  revision 2014-11-01 {
    description "Inherit from alto-service-types.";
  }
```



```
  revision 2014-10-24 {
    description "Initial version.";
  }

  /*****************************************
    Groupings for ALTO information resource
  *****************************************/

  grouping network-map-data {
    list map {
      key "pid";
      leaf pid {
        type alto:pid-name;
      }
      uses alto:endpoint-address-group;
    }
  }

  /* Network Map Grouping */
  grouping network-map {
    leaf resource-id {
      type alto:resource-id;
      mandatory true;
    }
    leaf tag {
      type alto:tag-string;
      mandatory true;
    }
    uses network-map-data;
  }

  grouping cost-map-data {
    list map {
      leaf src {
        type alto:pid-name;
```

```
      }
      key "src";
      list dst-costs {
        leaf dst {
          type alto:pid-name;
        }
        key "dst";
        uses alto:alto-cost;
      }
    }
  }

  /* Cost Map Grouping */
```



```
  grouping cost-map {
    leaf resource-id {
      type alto:resource-id;
      mandatory true;
    }
    leaf tag {
      type alto:tag-string;
      mandatory true; //TODO: sx: mandatory tag? tag manager?
    }
    container meta {
      must "current()";
      uses alto:cost-map-meta;
    }
    uses cost-map-data;
  }

  grouping alto-resources {
    container IRD {
      uses alto:IRD;
    }

    container network-maps {
      list network-map {
        key "resource-id";
        uses network-map;
      }
    }
```

```
      container cost-maps {
        list cost-map {
          key "resource-id";
          uses cost-map;
        }
      }

      container endpoint-property-map {
        uses alto:endpoint-property-map;
      }
    }

    /*************************************************
       DATA INSTANCES of all ALTO information resources

       unfiltered network-maps, unfiltered cost-maps are all instances
       of resources. IRD is also modeled as data.

       The design uses augment as the basic approach to implement
       inheritance.
```



```
     *************************************************/

    container resources {
      uses alto-resources;
    }
  }
```

Figure 4: ALTO YANG Model.

Appendix B. Using NETCONF to Read ALTO/YANG Information

NETCONF has provided two RPCs to retrieve data: the <get> operation and the <get-config> operation. The <get> operation can be used to retrieve both configuration data and state data, whereas the <get-config> operation is used to retrieve only configuration data. Since state data is read-only, in the current design, we model ALTO resources as configuration data. The <get> operation retrieves only the configuration data in the running datastore where as the <get-config> operation can specify the target datastore. Hence we use <get-config> as an example, and <get> operation is analogous. The

<get-config> operation defines two filter types in the NETCONF base protocol [RFC6241], subtree filtering and optional XPATH filtering capabilities. Hence we show examples for both types.

B.1. Full Network Map Service

B.1.1. Approach 1: Using Subtree Filtering

To retrieve a network map with resource-id INPUT-NETWORK-MAP-RESOURCE-ID, the client specifies it in the following netconf RPC template:


```
<rpc message-id=SEQ-NUM
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="subtree">
      <resources
        xmlns="urn:ietf:params:xml:ns:yang:alto-service">
        <network-maps>
          <network-map>
            <resource-id>INPUT-NETWORK-MAP-RESOURCE-ID</resource-id>
            <tag/>
            <map/>
```

```
          </network-map>
        </network-maps>
      </resources>
    </filter>
  </get-config>
</rpc>
```

One can observe that the query mapping specifies not only the resource-id as a content matching node, but also the tag and map nodes, as two selection nodes, to indicate that these two fields should be included in the filter output. A simpler mapping, using the default processing of filtering output (the last output rule of Section 6.2.5 of [RFC6241]), is to omit <tag/> and <map/>. This will give the same output. We suggest the more complete template for more explicit results.

An example of the query, for a network map with resource-id myNetMap1, is the following:


```
<rpc message-id=SEQ-NUM
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="subtree">
```

```
        <resources
          xmlns="urn:ietf:params:xml:ns:yang:alto-service">
          <network-maps>
            <network-map>
              <resource-id>myNetMap1</resource-id>
              <tag/>
              <map/>
            </network-map>
          </network-maps>
        </resources>
      </filter>
    </get-config>
  </rpc>
```

An example reply from the server then will be:

```
<rpc-reply message-id=SEQ-NUM
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <resources
      xmlns="urn:ietf:params:xml:ns:yang:alto-service">
      <network-maps>
        <network-map>
          <resource-id>myNetMap1</resource-id>
          <tag>da65eca2tus10ce8b0740a1938e3f8eb1d4785</tag>
          <map>
            <pid>PID1</pid>
            <endpoint-address-group>
              <address-type>ipv4</address-type>
              <endpoint-prefix>192.0.2.0/24</endpoint-prefix>
              <endpoint-prefix>198.51.100.0/25</endpoint-prefix>
            </endpoint-address-group>
          </map>
          <map>
            <pid>PID2</pid>
            <endpoint-address-group>
              <address-type>ipv4</address-type>
              <endpoint-prefix>198.51.100.128/25</endpoint-prefix>
            </endpoint-address-group>
          </map>
          <map>
            <pid>PID3</pid>
            <endpoint-address-group>
              <address-type>ipv4</address-type>
              <endpoint-prefix>0.0.0.0/0</endpoint-prefix>
            </endpoint-address-group>
            <endpoint-address-group>
              <address-type>ipv6</address-type>
              <endpoint-prefix>::/0</endpoint-prefix>
            </endpoint-address-group>
          </map>
        </network-map>
      </network-maps>
    </resources>
  </data>
</rpc-reply>
```

B.1.2. Approach 2: Using XPATH Filtering

To retrieve a network map with resource-id INPUT-NETWORK-MAP-RESOURCE-ID, the client specifies it in the following netconf RPC template:

```
<rpc message-id=SEQ-NUM
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter
      xmlns:t="urn:ietf:params:xml:ns:yang:alto-service"
      type="xpath"
      select="/t:resources/t:network-maps/t:network-map
              [t:resource-id=INPUT-NETWORK-MAP-RESOURCE-ID]" />
  </get-config>
</rpc>
```

Note that [RFC6241] requires only that a NETCONF server MAY support xpath. Hence, this approach may or may not work on a given NETCONF server.

An example of the query, for a network map with resource-id myNetMap1, is the following. Note that the XPATH expression would select the appropriate <network-map> node including its subtree, but not the ancestor node <resources> or <network-maps>. According to the NETCONF modification rules defined in Section 8.9.5.1 of [RFC6241], the ancestor nodes of the XPATH result are also encoded, in particular, the tags <resources> and <network-maps>.

```
<rpc message-id=SEQ-NUM
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter
      xmlns:t="urn:ietf:params:xml:ns:yang:alto-service"
      type="xpath"
      select="/t:resources/t:network-maps/t:network-map
              [t:resource-id='myNetMap1']" />
  </get-config>
</rpc>
```

The output of this query is the same as the subtree based query.

B.2. Filtered Cost Map Service

Then we consider filtered cost map service. For consistency, we consider the same yang module as in Figure 2. For ease of reading, we duplicate the tree diagram here:


```
+--ro cost-maps
|  +--ro cost-map* [resource-id]
|     +--ro resource-id    alto:resource-id
|     +--ro tag            alto:tag-string
|     +--ro meta
|     |  +--ro dependent-vtags*
|     |  |  +--ro resource-id    resource-id
|     |  |  +--ro tag            tag-string
|     |  +--ro cost-type
|     |     +--ro cost-mode      cost-mode
|     |     +--ro cost-metric    cost-metric
|     |     +--ro description?   string
|     +--ro map* [src]
|        +--ro src          alto:pid-name
|        +--ro dst-costs* [dst]
|           +--ro dst     alto:pid-name
|           +--ro cost
```

B.2.1. Approach 1: Using Subtree Filtering

To retrieve INPUT-SRC-PID-1, INPUT-SRC-PID-2, ..., INPUT-SRC-PID-p and INPUT-DST-PID-1, INPUT-DST-PID-2, ..., INPUT-DST-PID-q of a cost map with resource-id INPUT-COST-MAP-RESOURCE-ID, the client specifies it in the following NETCONF RPC template:

```
   <rpc message-id=SEQ-NUM
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <get-config>
       <source>
         <running/>
       </source>
       <filter type="subtree">
         <resources
           xmlns="urn:ietf:params:xml:ns:yang:alto-service">
           <cost-maps>
             <cost-map>
               <resource-id>
                 INPUT-COST-MAP-RESOURCE-ID
               </resource-id>
               <tag/>
               <meta/>
               <map>
                 <src>INPUT-SRC-PID-1</src>
                 <dst-costs>
                   <dst>INPUT-DST-PID-1</dst>
                   <cost/>
                 </dst-costs>
                 ...
                 <dst-costs>
                   <dst>INPUT-DST-PID-q</dst>
                   <cost/>
                 </dst-costs>
               </map>
               ...
```

```
            <map>
              <src>INPUT-SRC-PID-p</src>
              <dst-costs>
                <dst>INPUT-DST-PID-1</dst>
                <cost/>
              </dst-costs>
              ...
              <dst-costs>
                <dst>INPUT-DST-PID-q</dst>
                <cost/>
              </dst-costs>
            </map>
          </cost-map>
        </cost-maps>
      </resources>
    </filter>
  </get-config>
</rpc>
```


One can observe that in the template, client must list all combinations of src and dst to retrieve the cost, which is very inefficient when the number of src and dst grows.

An example of the query, for src: PID1, PID3 and dst: PID2, PID3 of a cost map with resource-id myCostMap1, is the following:

```
   <rpc message-id=SEQ-NUM
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <get-config>
       <source>
         <running/>
       </source>
       <filter type="subtree">
         <resources
           xmlns="urn:ietf:params:xml:ns:yang:alto-service">
           <cost-maps>
             <cost-map>
               <resource-id>
                 myCostMap1
               </resource-id>
               <tag/>
```

```
              <meta/>
              <map>
                <src>PID1</src>
                <dst-costs>
                  <dst>PID2</dst>
                  <cost/>
                </dst-costs>
                <dst-costs>
                  <dst>PID3</dst>
                  <cost/>
                </dst-costs>
              </map>
              <map>
                <src>PID3</src>
                <dst-costs>
                  <dst>PID2</dst>
                  <cost/>
                </dst-costs>
                <dst-costs>
                  <dst>PID3</dst>
                  <cost/>
                </dst-costs>
              </map>
            </cost-map>
          </cost-maps>
        </resources>
      </filter>
    </get-config>
  </rpc>
```

An example reply from the server then will be:


```
  <rpc-reply message-id=SEQ-NUM
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <data>
      <resources
        xmlns="urn:ietf:params:xml:ns:yang:alto-service">
        <cost-maps>
          <cost-map>
            <resource-id>myCostMap1</resource-id>
```

```
            <tag>tus10ce8b0740a1938e3f8eb1d4785da65eca2</tag>
            <meta>
              <dependent-vtags>
                <resource-id>myNetMap1</resource-id>
                <tag>da65eca2tus10ce8b0740a1938e3f8eb1d4785</tag>
              </dependent-vtags>
              <cost-type>
                <cost-mode>numerical</cost-mode>
                <cost-metric>routingcost</cost-metric>
              </cost-type>
            </meta>
            <map>
              <src>PID1</src>
              <dst-costs>
                <dst>PID2</dst>
                <cost>5</cost>
              </dst-costs>
              <dst-costs>
                <dst>PID3</dst>
                <cost>10</cost>
              </dst-costs>
            </map>
            <map>
              <src>PID3</src>
              <dst-costs>
                <dst>PID2</dst>
                <cost>15</cost>
              </dst-costs>
            </map>
          </cost-map>
        </cost-maps>
      </resources>
    </data>
  </rpc-reply>
```

B.2.2. Approach 2 : Using XPath

To retrieve INPUT-SRC-PID-1, INPUT-SRC-PID-2, ..., INPUT-SRC-PID-p and INPUT-DST-PID-1, INPUT-DST-PID-2, ..., INPUT-DST-PID-q of a cost

map with resource-id INPUT-COST-MAP-RESOURCE-ID, the client specifies

it in the following netconf RPC template:

```
<rpc message-id=SEQ-NUM
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter
      xmlns:t="urn:ietf:params:xml:ns:yang:alto-service"
      type="xpath"
      select="/t:resources/t:cost-maps/t:cost-map
          [t:resource-id=INPUT-COST-MAP-RESOURCE-ID]/t:resource-id
        | /t:resources/t:cost-maps/t:cost-map
          [t:resource-id=INPUT-COST-MAP-RESOURCE-ID]/t:tag
        | /t:resources/t:cost-maps/t:cost-map
          [t:resource-id=INPUT-COST-MAP-RESOURCE-ID]/t:meta
        | /t:resources/t:cost-maps/t:cost-map
          [t:resource-id=INPUT-COST-MAP-RESOURCE-ID]/
          t:map[t:src=INPUT-SRC-PID-1 or t:src=INPUT-SRC-PID-2
            or ... or t:src=INPUT-SRC-PID-p]/t:src
        | /t:resources/t:cost-maps/t:cost-map
          [t:resource-id=INPUT-COST-MAP-RESOURCE-ID]/
          t:map[t:src=INPUT-SRC-PID-1 or t:src=INPUT-SRC-PID-2
            or ... or t:src=INPUT-SRC-PID-p]/t:dst-costs
          [t:dst=INPUT-DST-PID-1 or t:dst=INPUT-DST-PID-2
            or ... or t:dst=INPUT-DST-PID-q]" />
  </get-config>
</rpc>
```

One observe that similar to the xpath template for filtered network map, the xpath template for filtered cost map also used XPATH union in the select attribute of the filter, which is necessary because the XPATH operation will not select the resource-id and tag and src nodes if we only have the XPATH expression "/t:resources/t:cost-maps/ t:cost-map[t:resource-id=INPUT-COST-MAP-RESOURCE-ID]/t:map[t:src=INPUT-SRC-PID-1 or t:src=INPUT-SRC-PID-2 or ... or t:src=INPUT-SRC-PID-p]/t:dst-costs[t:dst=INPUT-DST-PID-1 or t:dst=INPUT-DST-PID-2 or ... or t:dst=INPUT-DST-PID-q]".

Despite that one needs to specify the src PIDs multiple times, there is no need to provide a cartesian product of src PIDs and dst PIDs, which is more efficient compared to the subtree filtering approach.

An example of the query, for srcs: PID1, PID3 and dsts: PID2, PID3 of a cost map with resource-id myCostMap1, is the following:

```
<rpc message-id=SEQ-NUM
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter
      xmlns:t="urn:ietf:params:xml:ns:yang:alto-service"
      type="xpath"
      select="/t:resources/t:cost-maps/t:cost-map
          [t:resource-id='myCostMap1']/t:resource-id
        | /t:resources/t:cost-maps/t:cost-map
          [t:resource-id='myCostMap1']/t:tag
        | /t:resources/t:cost-maps/t:cost-map
          [t:resource-id='myCostMap1']/t:meta
        | /t:resources/t:cost-maps/t:cost-map
          [t:resource-id='myCostMap1']/
          t:map[t:src='PID1' or t:src='PID3']/t:src
        | /t:resources/t:cost-maps/t:cost-map
          [t:resource-id='myCostMap1']/
          t:map[t:src='PID1' or t:src='PID3']/t:dst-costs
          [t:dst='PID2' or t:dst='PID3']" />
  </get-config>
</rpc>
```

An example reply from the server then will be:

```
   <rpc-reply message-id=SEQ-NUM
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <data>
       <resources
         xmlns="urn:ietf:params:xml:ns:yang:alto-service">
         <cost-maps>
           <cost-map>
             <resource-id>myCostMap1</resource-id>
             <tag>tus10ce8b0740a1938e3f8eb1d4785da65eca2</tag>
             <meta>
               <dependent-vtags>
                 <resource-id>myNetMap1</resource-id>
                 <tag>da65eca2tus10ce8b0740a1938e3f8eb1d4785</tag>
               </dependent-vtags>
               <cost-type>
                 <cost-mode>numerical</cost-mode>
                 <cost-metric>routingcost</cost-metric>
               </cost-type>
             </meta>
             <map>
               <src>PID1</src>
               <dst-costs>
                 <dst>PID2</dst>
                 <cost>5</cost>
               </dst-costs>
               <dst-costs>
                 <dst>PID3</dst>
                 <cost>10</cost>
               </dst-costs>
             </map>
             <map>
               <src>PID3</src>
               <dst-costs>
                 <dst>PID2</dst>
                 <cost>15</cost>
               </dst-costs>
             </map>
           </cost-map>
         </cost-maps>
```

```
      </resources>
    </data>
  </rpc-reply>
```

Appendix C. Using RESTCONF to Update ALTO/YANG Information

The RESTCONF Protocol [RESTCONF] provides a RESTful interface to data defined in YANG. RESTCONF must support XML and may support JSON

encoding. We can retrieve information with HTTP GET method and update information with PATCH method.

RESTCONF uses the following HTTP request structure for clients to encode query parameters:

```
        <OP> /<restconf>/<path>?<query>#<fragment>

          ^       ^        ^       ^         ^
          |       |        |       |         |
        method  entry  resource  query    fragment

          M       M        O        O         I

        M=mandatory, O=optional, I=ignored

        <text> replaced by client with real values
```

The particular relevant components are the path component and the query component. The goal of the path component (Section 3.5.1 of [RESTCONF]) is to identify a single node (referred to as the target resource) in the data tree. This is different from xpath, whose basic Location Path concept is built on node set; that is, the goal of xpath is to identify a set of nodes (address parts of an XML document), not a single node. The query component (Section 3.8 of [RESTCONF]) consists of a set of "name=value" pairs, with a given set of names (query parameters) to control the query behavior. A particularly relevant parameter is select, which allows a client to request a subset of the target resource contents. The exact syntax of select is specified in Section 3.8.4 of [RESTCONF]. Note that all query parameters are optional to implement by the server and optional to use by the client.

C.1. Read Queries

C.1.1. Example: Full Network Map Service

To request a network map with resource-id INPUT-NETWORK-MAP-RESOURCE-ID, the RESTCONF client sends the following HTTP request:

```
GET /restconf/data/alto-service:resources/network-maps
  /network-map=INPUT-NETWORK-MAP-RESOURCE-ID?content=all HTTP/1.1
Host: alto.example.com
Accept: application/yang.data+json,application/yang.errors+json
```

An example query for a network map whose resource-id is myNetMap1 is:


```
GET /restconf/data/alto-service:resources/network-maps
  /network-map=myNetMap1?content=all HTTP/1.1
Host: alto.example.com
Accept: application/yang.data+json,application/yang.errors+json
```

An example response from a RESTCONF server can be:

```
HTTP/1.1 200 OK
Date: Mon, 31 Oct 2011 23:59:00 GMT
Server: example-alto-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json

{
  "alto-service:network-map" : {
    "resource-id" : "myNetMap1",
    "tag" : "da65eca2tus10ce8b0740a1938e3f8eb1d4785",
    "map": [
      {
        "pid": "PID1",
        "endpoint-address-group": {
          "address-type": "ipv4",
          "endpoint-prefix": [
            "192.0.2.0/24",
```

```
              "198.51.100.0/25"
            ]
          }
        },
        {
          "pid": "PID2",
          "endpoint-address-group": {
            "address-type": "ipv4",
            "endpoint-prefix": ["198.51.100.128/25"]
          }
        },
        {
          "pid": "PID3",
          "endpoint-address-group": [
            {
              "address-type": "ipv4",
              "endpoint-prefix": ["0.0.0.0/0"]
            },
            {
              "address-type": "ipv6",
              "endpoint-prefix": ["::/0"]
            }
          ]
        }
      ]
    }
  }
```


C.1.2. Impossibility to Encode Filtered Maps and Endpoint Properties using Standard RESTCONF Query

The preceding sections provided templates to implement full network map service using standard operations in RESTCONF; full cost map services are analogous. However, it is not possible to do so for the filtered map services or the endpoint property service.

Specifically, supporting the filtered map services requires the ability to select multiple nodes from the data tree based on the data content. Recall that the path component in the uri can return only a

single node. Hence, the only component that allows one to select multiple nodes is the query component, specifically, the "fields" query parameter. However, the definition of the fields expression (Section 4.8.5 of [RESTCONF]) only includes terms defined in a schema, not any data content (i.e., select does not include any content match capabilities). Hence, it is impossible to implement filtered maps.

C.2. Update Queries

RESTCONF uses HTTP POST, PUT, PATCH, and DELETE methods to edit server resources. They are closely related to the NETCONF operations (see Section 4. of [RESTCONF]).

Authors' Addresses

Xiao Shi
Yale University
51 Prospect Street
New Haven, CT 06511
USA

Email: xiao.shi@yale.edu

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

Email: yang.r.yang@gmail.com

Michael Scharf
Alcatel-Lucent Bell Labs
Lorenzstrasse 10
Stuttgart 70435

Germany

Email: michael.scharf@alcatel-lucent.com