Using YANG to Specify ALTO Protocol and Services

draft-shi-alto-yang-model-01
draft-shi-alto-yang-json-00

Xiao Shi (xiao.shi@yale.edu)

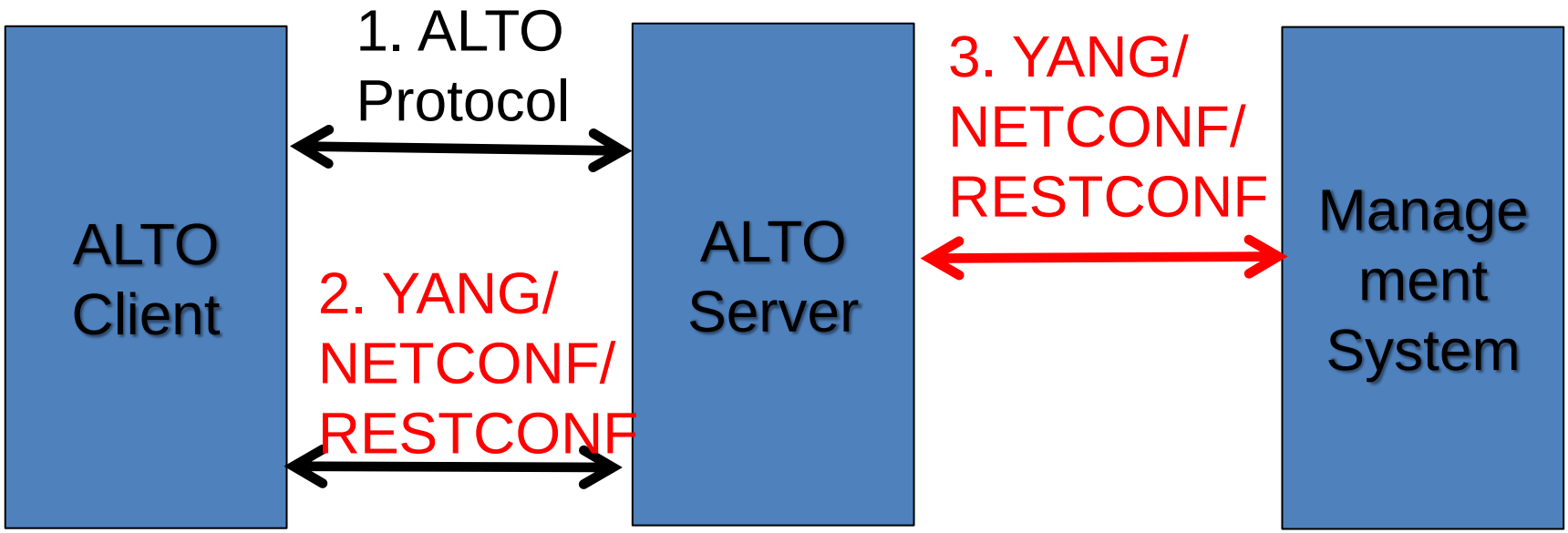
Y. Richard Yang (yry@cs.yale.edu)

M. Scharf (michael.scharf@alcatel-lucent.com)

Nov. 13, 2014 @ IETF 91

Overview: Three Use Scenarios of Applying YANG to ALTO

1. Use YANG to specify ALTO protocol messages
2. Use YANG to provide ALTO services to ALTO clients
3. Use YANG to configure (r/w info on) ALTO server



In the rest of the presentation, we will first go over background and issues. The main goal is to get WG discussion on direction.

Outline

- Overview
- Use YANG to specify ALTO protocol

Using YANG to Specify ALTO Protocol

- Ideal goal: achieve interop between native ALTO and YANG ALTO
- Since YANG by itself defines only input/output data encoding, we first focus on message body, using JSON
  - `draft-ietf-netmod-yang-json-01`

Example: ALTO Filtered Network Maps

```
POST /networkmap/filtered HTTP/1.1
Host: custom.alto.example.com
Content-Length: 33
Content-Type: application/
  alto-networkmapfilter+json
Accept: application/
  alto-networkmap+json,
  application/alto-error+json
```

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

Focus on YANG specification of these JSON objects

Q: What is the condition that YANG can specify these JSON input/output, efficiently?

```
HTTP/1.1 200 OK
Content-Length: 342
Content-Type: application/alto-networkmap+json
```

```
{
  "meta" : {
    "vtag" : {
      "resource-id": "my-default-network-map",
      "tag": "c0ce023b8678a7b9ec003246754656d1f6d"
    }
  },
  "network-map" : {
    "PID1" : {
      "ipv4" : [
        "192.0.2.0/24",
        "198.51.100.0/24"
      ]
    },
    "PID2" : {
      "ipv4": [
        "198.51.100.128/24"
      ]
    }
  }
}
```

Impossible to Encode JSON Messages

- Non object
- Non-YANG array
- YANG/JSON Key Space
  - Observation: For a YANG model
    - Only some identifiers defined the model and a few YANG predefined keywords (e.g., notification name, rpc name, `<eventTime>`) can appear as XML tags in XML encoding
    - Only XML tags can appear as JSON keys in JSON encoding
      =>
    - Assume that a YANG model provides a model for a JSON based protocol. If a key can ever appear in one JSON message, the key must appear as an identifier of the YANG model.

Implications

- Consider key-value stores, commonly used in many designs. Then each possible key will need to appear as an identifier in the model.

WG Discussion

- Since we have shown ALTO JSON encoding will not have a compact YANG model, and hence interop is not an option (we cannot use YANG to specify the msg of native ALTO protocol)

- Work items
  - Change YANG: Modify YANG encoding rule to generate ALTO msg
  - Extend ALTO: specify additional ALTO msg according to an YANG model
    - Implication: two encoding of the same ALTO resource

YANG ALTO JSON Encoding Example

```
POST /networkmap/filtered HTTP/1.1
Host: custom.alto.example.com
Content-Length: 33
Content-Type: application/
  alto-networkmapfilter+json
Accept: application/
  alto-networkmap-yang+json,
  application/alto-error+json
```

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

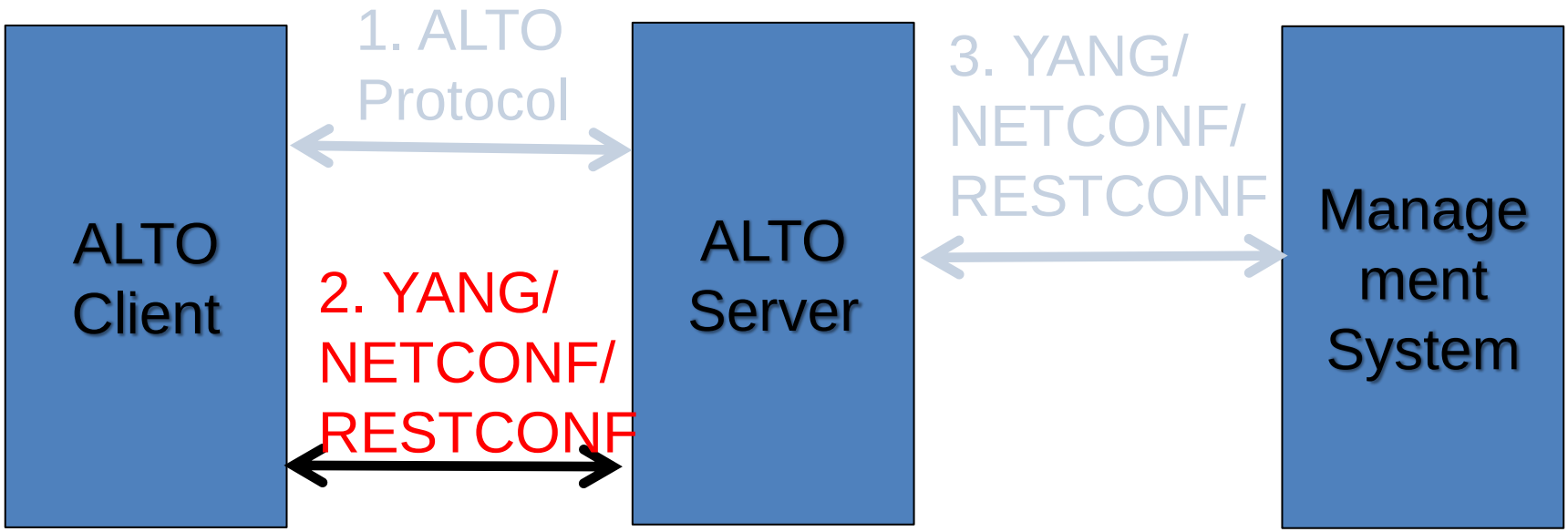
Encoded YANG object

```
HTTP/1.1 200 OK
Content-Length: 342
Content-Type: application/alto-networkmap-
yang+json
```

```
"alto-service:network-map" : {
   "resource-id" : "myNetMap1",
   "tag" : "da65eca2tus10ce8b0740a1938e",
   "map": [
      {
       "pid": "PID1",
       "endpoint-address-group": [ {
         "address-type": "ipv4",
         "endpoint-prefix": [
            "192.0.2.0/24",
            "198.51.100.0/25"
         ]
       }]
    },
…
```

Outline

- Overview
- Use YANG to specify ALTO protocol
- Use YANG to provide ALTO services



Embedding in a YANG System

- Already defined *"RPCs"*
- Additional ALTO RPCs

Input/output precisely defined using YANG grammar

- Already defined data instances
- ALTO data instances

Goal: realizing 5 base services:

- Full network/cost map services
- Filtered maps
- Endpoint properties
- IRD
- ECS

Outline

- Overview
- Use YANG to specify ALTO protocol
- Use YANG to provide ALTO services
  - NETCONF

Embedding in NETCONF

- Already defined *"RPCs"*
  - get, get-config, edit-config, copy-config, delete-config, close/kill-session, lock/unlock, commit/cancel-commit, get-schema, …
- Additional RPCs
  - endpoint-cost-service

- Already defined data instances
- ALTO data instances

Positive: It is possible to provide semantically equivalent ALTO services in NETCONF. Using both existing NETCONF RPCs and custom RPC (ECS)

Example: Cost Map Yang Model

```
+--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
```

Filtered Cost Map using Subtree

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

xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <resources
xmlns="urn:ietf:params:xml:ns:yang:alto-service-
netconf">
        <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>
</rpc>
```

Note the need to write out all cross-product. Overall: missing join/cross product as in SQL

Filtered Cost Map using XPATH

```
<rpc message-id=SEQ-NUM
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter xmlns:t="urn:ietf:params:xml:ns:yang:alto-service-netconf"
      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 ... 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 ... or t:src=INPUT-SRC-PID-p]/
          t:dst-costs[t:dst=INPUT-DST-PID-1 or ... or t:dst=INPUT-DST-PID-q]" />
  </get>
</rpc>
```

- Can avoid writing cross product
- Need union to get all needed outputs

Issues to Embedding ALTO in NETCONF

- Need to “disable” extra RPCs defined in NETCONF

- Scalability
  - ALTO IRD allows different resources to be distributed on different servers for more flexible, scalable deployment, e.g.,
    - Full network map net-map1 served only by CDN server 1
    - Filtering of net-map1 served only by server 2
    - Full cost map cost-map1 served only by server 3
  - The base YANG model assumes a single device (server)
    - New proposal of YANG mount allows a central view from multiple servers’ data stores, but uses a complete “fetch-through” design, creating a single point of bottleneck at the server who does the mounting.

- The transport is typically SSH, which can be a deployment issue for some settings.

Outline

- Overview
- Use YANG to specify ALTO protocol
- Use YANG to provide ALTO services
  - NETCONF
  - RESTCONF

Embedding in RESTCONF

- Already defined "RPCs"
  - Defined HTTP methods and patterns using HTTP GET, POST, PUT, PATCH …
- Additional RPCs
  - endpoint-cost-service through POST

- Already defined data instances
- ALTO data instances

Main issue: Existing RESTCONF design does not allow filtered maps

Impossibility to Encode Filtered Maps or Endpoint Prop

- The only way to specify query parameters in the RESTCONF GET method is to encode the parameters in the GET URI. The relevant encoding structure then is:

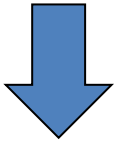
```
GET /<restconf>/<path>?<query>
 ^       ^         ^       ^
 |       |         |       |
method  entry  resource  query
 M       M        O       O

M=mandatory, O=optional
<text> replaced by client with real values
```

Defines an encoding to identify a **single** node (referred to as the target resource) in the data tree.

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, but “select” does not have content match ability.

Potential Extension to RESTCONF

```
select-expr = path '(' select-expr / '*' ')' /
              path ';' select-expr /
              path
path = api-identifier [ '/' path ]
```



```
select-expr = path '(' select-expr / '*' ')' /
              path ';' select-expr /
              path
path = (api-identifier | list-instance)  [ '/' path ]
list-instance = api-identifier "=" key-value ["," key-value]*
key-value = string
```

Filtered Cost Map Service using Extension

```
GET /restconf/data/resources/cost-maps \
    /cost-map=INPUT_COST_MAP_RES_ID \
    ?select=resource-id;tag;meta; \
    map=INPUT_SRC_PID_1(src; \
        dst-costs=INPUT_DST_PID_1;...;dst-costs=INPUT_DST_PID_q); \
    map=INPUT_SRC_PID_2(src; \
        dst-costs=INPUT_DST_PID_1;...;dst-costs=INPUT_DST_PID_q); \
    ...\
    map=INPUT_SRC_PID_p(src; \
        dst-costs=INPUT_DST_PID_1;...;dst-costs=INPUT_DST_PID_q) \
    &content=all HTTP/1.1
Host: alto.example.com
Accept: application/yang.data+json,application/yang.errors+json
```

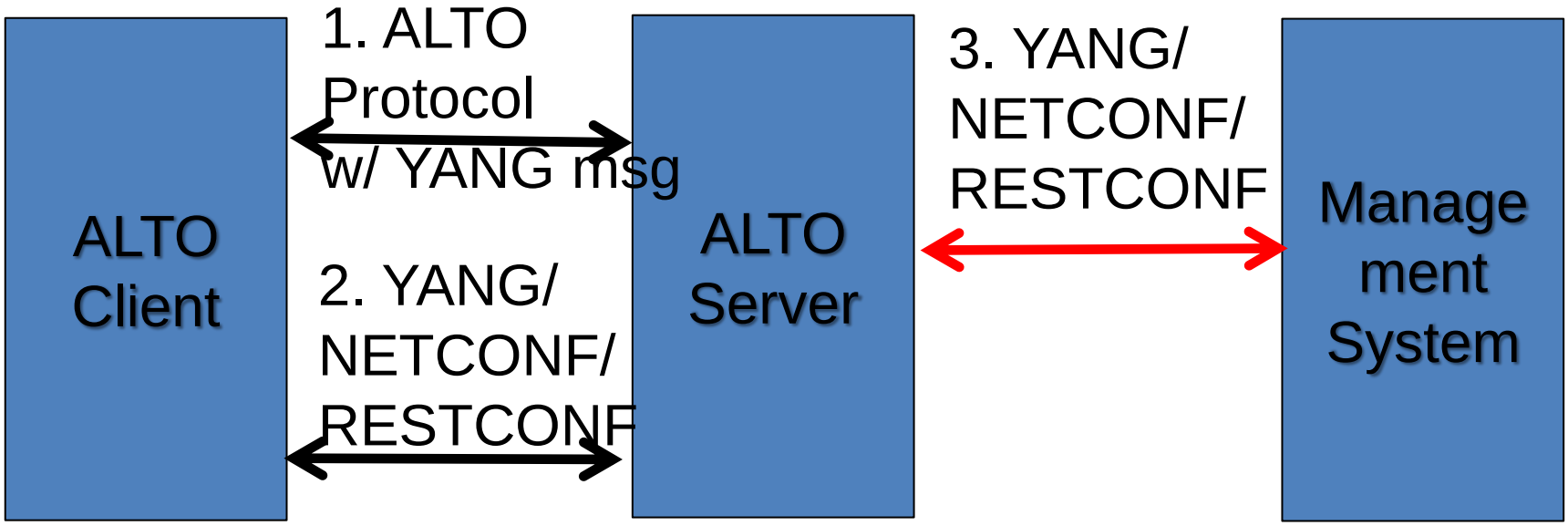
Summary on Proving ALTO Services using RESTCONF

- It is more challenging to provide semantically equivalent ALTO services in RESTCONF: need extensions to RESTCONF, and the extension is not totally generic (only on list instance)

- Need a new mechanism (e.g., POST query using say XQuery might be more systematic) but need to be developed.

WG Discussion

- Do we specify a YANG model for ALTO, from which we
  - Design parallel YANG encoded ALTO msg, and/or
  - Use NETCONF/RESTCONF to provision ALTO server
- Do we explore the possibility to embedding ALTO services in NETCONF/RESTCONF?



Backup Slides

Motivation

- Substantial recent development in using the YANG modeling language in the networking community (e.g., ODL)

- Our basic goal: investigate if we can use YANG to provide the aforementioned definitions.
  - ALTO base protocol (RFC7285) uses an adaptation of the C-style struct notation to define ALTO JSON objects, and states (Section 8.2) that *"Note that, despite the notation, no standard, machine-readable interface definition or schema is provided in this document. Extension documents may describe these as necessary."*

- Additional goal: investigate how we can apply YANG into an ALTO system.

Example: Network Map Service YANG RPC

```
+---x network-map-service
     |  +--ro output
     |     +--ro network-map-service
     |        +--ro meta
     |        |  +--ro vtag
     |        |     +--ro resource-id    resource-id
     |        |     +--ro tag            tag-string
     |        +--ro network-map* [pid]
     |           +--ro pid                        pid-name
     |           +--ro endpoint-address-group* [address-type]
     |              +--ro address-type       endpoint-address-type
     |              +--ro endpoint-prefix*   endpoint-prefix
```

Handling Resource Storage with Multiple Versions

- Design option 1:
  - key resources by version tag, i.e. (resource-id, tag) tuple
  - Issues:
    - Redundancy among multiple versions
    - Need to store most updated version tag of each resource separately, e.g. in the IRD
- Design option 2:
  - The only full resource stored is the most updated version, i.e. resources can be keyed by resource-id alone.
  - Store diffs of past versions if history versions are needed.

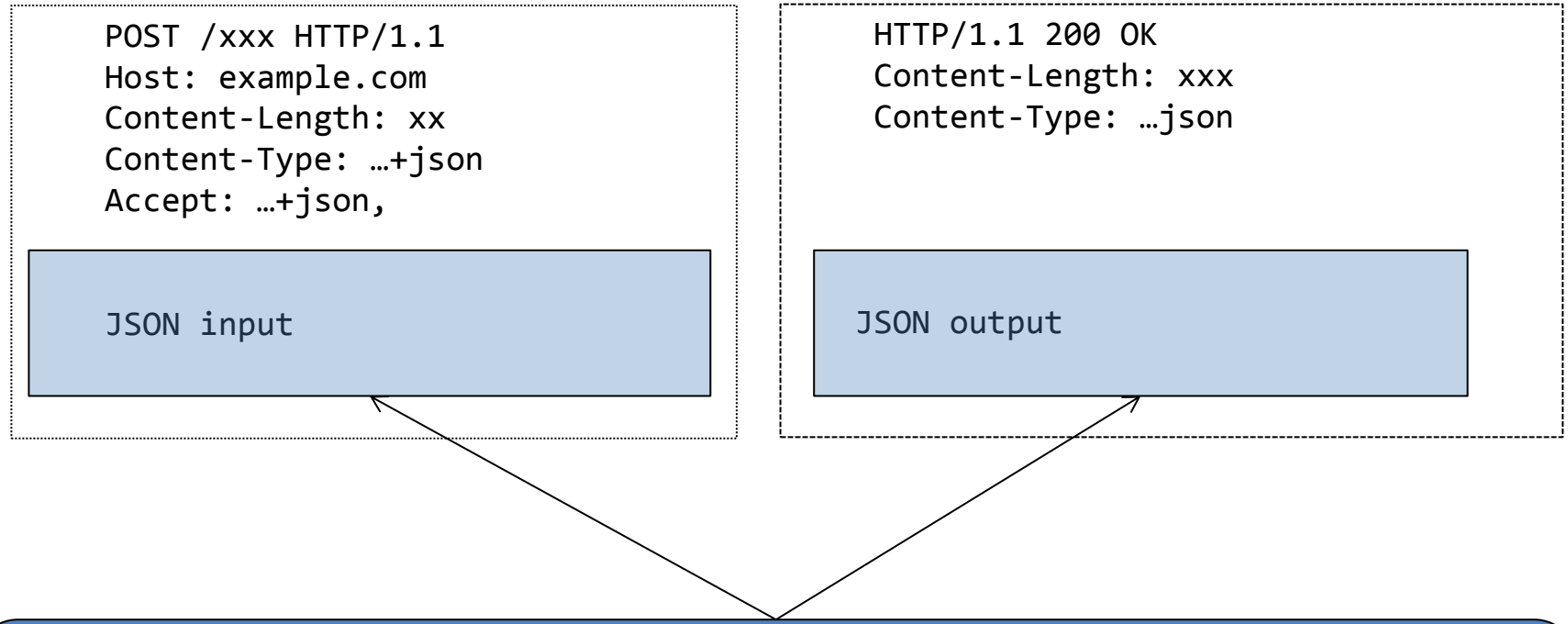
NETCONF Server Infrastructure

Standard Operations (<get>, <edit-config>,etc.) and Custom RPCs

NETCONF Datastore: **YANG Model**
conceptual place where the data to store and access information

Actual Data:
Database, files, flash memory locations, etc.

General Question: YANG Spec for JSON Messages



What is the condition that YANG can specify these JSON input/output, efficiently? One can consider this as a kind of reverse engineering.

What do we mean by “to Specify?”

- JSON generated by YANG can be understood by native JSON parser.
- JSON object generated from native protocol can be validated by YANG.

- Ordering issue:
  - A JSON object is a set of key-value pairs, and a major flexibility of JSON is that the key-value pairs are unordered. On the other hand, YANG is based on XML. Hence YANG encoding is ordered due to the inherent order of XML encoding.
  - Example:
    - The following two JSON objects are equivalent:
      - `{ "foo" : { "bar" : 123, "baz" : 456 } }`
      - `{ "foo" : { "baz" : 456, "bar" : 123 } }`
    - However, the following two XML documents are NOT the same:
      - `<foo> <bar>123</bar> <baz>456</baz> </foo>`
      - `<foo> <baz>456</baz> <bar>123</bar> </foo>`
- To avoid trivial compatibility issues, we assume a preprocessor that will reorder the keys in a JSON object to try to match a YANG model.

YANG Non-encodable JSON Message: Non-Object

- A JSON message containing standalone non-object JSON values (string, number, array, null, or Boolean) cannot be modeled by YANG, e.g.,
  - “abc”
  - [1, 2, 3]

YANG Non-encodable JSON Message: Non-YANG Arrays

```
JSON-array := yang-array | non-yang-array

yang-array     := object-array | primitive-array
object-array   := every element is an object
primitive-array:= every element is a primitive
primitive      := boolean | number | string

non-yang-array   := nested-array | object-mix-array
nested-array     := an array that contains as
                    its element another array
object-mix-array := an element is an object
                    while another element is primitive
```

Non-encodable JSON Message: Non-YANG Arrays

- Examples of non-yang arrays:
  - Object-mix-array:

```
{ "foo" : [ { "bar": "baz" }, 123] }
```

  - Nested-array:

```
{ "foo" : [ 1, 2, 3, [4, 5] ] }
```

- Examples of yang-arrays:
  - Primitive-array:

```
{ "foo" : [123, "abc", 24, 5] }
```

```
YANG encoding:
  leaf-list foo {
    type union {
      type string;
      type int32;
    }
  }
```

  - Object-array:

```
{ "foo" : [
  { "bar": "abc", "baz": "xyz" },
  { "baz": "xyz" }
] }
```

```
YANG encoding:
  list foo {
    config false;
    leaf bar { type string; }
    leaf baz { type string; }
  }
```

NETCONF <get>

- Since the focus of the current ALTO services is to read state data, we use the <get> RPC.

- Two filtering types in <get>
  - Subtree filtering
  - XPATH filtering (optional; modified standard XPATH processing to include full path)

Outline

- Overview
- Use YANG to specify ALTO protocol
- Use YANG to provide ALTO services
  - NETCONF
    - Network maps

Network Maps Yang Model

```
+--ro network-maps
|  +--ro network-map* [resource-id]
|     +--ro resource-id    alto:resource-id
|     +--ro tag            alto:tag-string
|     +--ro map* [pid]
|        +--ro pid                        alto:pid-name
|        +--ro endpoint-address-group* [address-type]
|           +--ro address-type       endpoint-address-type
|           +--ro endpoint-prefix*   endpoint-prefix
```

Full Network Map using Subtree Filtering

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

Explicit selection nodes to identify output.

Required content match node.

Full Network Map Reply Example

```
<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-netconf">
      <network-maps>
        <network-map>
          <resource-id>myNetMap1</resource-id>
          <tag>da65eca2tus10ce8b0740a1938e3f8eb1d4785</tag>
          SEE NEXT SLIDE
        </network-map>
      </network-maps>
    </resources>
  </data>
</rpc-reply>
```

Full Network Map Reply Example (Cont'd)

```
<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>
```

Full Network Map using XPATH

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

Filtered Network Map using Subtree

```
<rpc message-id=SEQ-NUM
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <resources xmlns="urn:ietf:params:xml:ns:yang:alto-service-netconf">
        <network-maps>
          <network-map>
            <resource-id>INPUT-NETWORK-MAP-RESOURCE-ID</resource-id>
            <tag/>
            <map>
              <pid>INPUT-PID-1</pid>
              <endpoint-address-group/>
            </map>
            <map>
              <pid>INPUT-PID-2</pid>
              <endpoint-address-group/>
            </map>
            ...
            <map>
              <pid>INPUT-PID-k</pid>
              <endpoint-address-group/>
            </map>
          </network-map>
        </network-maps>
      </resources>
    </filter>
  </get>
</rpc>
```

Filtered Network Map using XPATH

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

             | /t:resources/t:network-maps/
               t:network-map[t:resource-id=INPUT-NETWORK-MAP-RESOURCE-ID]/tag

             | /t:resources/t:network-maps/
               t:network-map[t:resource-id=INPUT-NETWORK-MAP-RESOURCE-ID]/
               map[pid=INPUT-PID-1 or pid=INPUT-PID-2 or ... or pid=INPUT-PID-k]" />
  </get>
</rpc>
```

Need XPATH union to get all output

Outline

- Overview
- Use YANG to specify ALTO protocol
- Use YANG to provide ALTO services
  - NETCONF
    - Network maps
    - Cost maps

Full Cost Map using Subtree

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

Full Cost Map Reply Example

```
<rpc-reply message-id=SEQ-NUM
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <resources>
      <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>PID1</dst>
              <cost>1</cost>
            </dst-costs>
            <dst-costs>
              <dst>PID2</dst>
              <cost>5</cost>
            </dst-costs>
            <dst-costs>
              <dst>PID3</dst>
              <cost>10</cost>
            </dst-costs>
          </map>
          <map>
            <src>PID2</src>
            <dst-costs>
              <dst>PID1</dst>
              <cost>5</cost>
            </dst-costs>
            <dst-costs>
              <dst>PID2</dst>
              <cost>1</cost>
            </dst-costs>
            <dst-costs>
              <dst>PID3</dst>
              <cost>15</cost>
            </dst-costs>
          </map>
          <map>
            <src>PID3</src>
            <dst-costs>
              <dst>PID1</dst>
              <cost>20</cost>
            </dst-costs>
            <dst-costs>
              <dst>PID2</dst>
              <cost>15</cost>
            </dst-costs>
          </map>
        </cost-map>
      </cost-maps>
    </resources>
  </data>
</rpc-reply>
```

Full Cost Map using XPATH

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

Outline

- Overview
- Use YANG to specify ALTO protocol
- Use YANG to provide ALTO services
  - NETCONF
    - Network maps
    - Cost maps
    - Extracting IRD info from maps

Extracting IRD Info (Available Resources)

```
+--ro network-maps
|  +--ro network-map* [resource-id]
|     +--ro resource-id    alto:resource-id
|     +--ro description
|     +--ro tag            alto:tag-string
|     +--ro map* [pid]
|        +--ro pid                          alto:pid-name
|        +--ro endpoint-address-group* [address-type]
|           +--ro address-type       endpoint-address-type
|           +--ro endpoint-prefix*   endpoint-prefix
```

Extracting Available Resources through Subtree

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

Full Network Map Service using RESTCONF

```
+--ro network-maps
|  +--ro network-map* [resource-id]
|     +--ro resource-id    alto:resource-id
|     +--ro tag            alto:tag-string
|     +--ro map* [pid]
|        +--ro pid                          alto:pid-name
|        +--ro endpoint-address-group* [address-type]
|           +--ro address-type       endpoint-address-type
|           +--ro endpoint-prefix*   endpoint-prefix
```

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

Full Network Map Response

```
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-restconf:network-map" : {
    "resource-id" : "myNetMap1",
    "tag" : "da65eca2tus10ce8b0740a1938e",
    "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"]
          }
        ]
      }
    ]
  }
}
```

Full Cost Map Service using RESTCONF

```
+--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
```

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

Full Cost Map Response

```
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-restconf:cost-map" : {
    "resource-id" : "myCostMap1",
    "tag" : "tus10ce8b0740a1938eda78bcf",
    "meta": {
      "dependent-vtags": {
        "resource-id": "myNetMap1",
        "tag": " da65eca2tus10ce8b0740a1938e"
      },
      "cost-type": {
        "cost-mode": "numerical",
        "cost-metric": "routingcost"
      }
    },
    "map": [
      {
        "src": "PID1",
        "dst-costs": [
          { "dst": "PID1", "cost": "1" },
          { "dst": "PID2", "cost": "5" },
          { "dst": "PID3", "cost": "10" }
        ]
      },
      {
        "src": "PID2",
        "dst-costs": [
          { "dst": "PID1", "cost": "5" },
          { "dst": "PID2", "cost": "1" },
          { "dst": "PID3", "cost": "15" }
        ]
      },
      {
        "src": "PID3",
        "dst-costs": [
          { "dst": "PID1", "cost": "20" },
          { "dst": "PID2", "cost": "15" }
        ]
      }
    ]
  }
}
```

Impossibility to Encode Filtered Maps or Endpoint Prop

- Filtered maps need the ability to select multiple nodes from the data tree
- Recall: `GET /<restconf>/<path>?<query>`
  - <path> can only select one single node (the “target resource”)
  - The “select” query parameter does not have content match ability
  - Ramification: cannot select multiple nodes based on content
- Hence it is impossible to provide Filtered Maps and Endpoint Property Services using a standard RESTCONF query.

Filtered Network Map Service using Extension

```
GET /restconf/data/resources/network-maps/ \
    network-map=INPUT_NETWORK_MAP_RES_ID \
    ?select=resource-id;tag; \
    map=INPUT_PID_1;map=INPUT_PID_2;...;map=INPUT_PID_k \
    &content=all HTTP/1.1
Host: alto.example.com
Accept: application/yang.data+json,application/yang.errors+json
```

Extracting Available Resources through select

```
GET /restconf/data/resources
  ?select=network-maps/network-map(resource-id;description;tag);
  cost-maps/cost-map(resource-id;description;tag;meta);
  endpoint-property-map/meta
  &content=all HTTP/1.1
Host: alto.example.com
Accept: application/yang.data+json,application/yang.errors+json
```