ALTO WG
Internet-Draft
Intended status: Informational
Expires: October 23, 2016

J. Zhang
Tongji University
K. Gao
Tsinghua University
Y. Yang
Yale University

April 21, 2016

Experiences of Implementing ALTO in OpenDaylight
draft-zhang-alto-opendaylight-impl-01

Abstract

This text introduces some experiences of implementing ALTO in OpenDaylight (ODL). The main key issues about design and implementation are discussed. Some of these issues have been figured out in the current implementation, the others have not. This text also gives some possible designs to discuss.

Status of This Memo

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

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

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

This Internet-Draft will expire on October 23, 2016.

Table of Contents

1. Introduction

ODL is one of the most popular Software Defined Networking (SDN) controller. We have implemented an ALTO server in ODL. However, some issues are very important to the design and implementation of ALTO server. In this document, we present some experiences of implementing ALTO in ODL, and discuss some key issues about the design and implementation.

1.1. Terminology

- ECS: Endpoint Cost Service
- ODL: OpenDaylight, an implementation of SDN controller
- SSE: Server-Sent Event
- MD-SAL: Model-Driven Service Abstraction Layer

1.2. Changes Since Version -00

- Restated fine-grained ECS problem in Section 2 and refined the experience of implementing ECS in ODL in Section 3. The section about "Customized Routing Cost" has been removed because of it is not a specific problem in ODL.
- Introduced details about the experience of implementing auto-map in Section 4.2.
- Updated overview of current implementation in Section 5.1 and introduced the solution for extensibility problem.
- Moved the cross platform problem to Section 6.

2. Key Design Issues

To implement ALTO in OpenDaylight, we identify a set of design and implementation issues:

- T-ALTO-MDSAL: How to use MD-SAL to implement ALTO?

  The core of OpenDaylight is MD-SAL, which provides mechanisms to describe, store, and access state in ODL data store. To achieve a relatively native design, we should use MD-SAL. At the same time, ALTO has defined its own data types such as Endpoint, PID, Vtag, Network Map, Cost Map. Hence, a first, basic design issue is how to represent the basic ALTO data in ODL data store.

- T-CrossPlatform: How to support cross platform?

  Balancing the preceding consideration, although we focus on implementing ALTO in ODL, we should also consider porting to other SDN controllers such as ONOS. Hence, we target a loose coupling architecture, to achieve an extensible, cross-platform design as much as possible.

- T-ECS: How to implement ECS?

Going from syntax to semantics, we first consider ECS, which is a basic service in ALTO. One may consider the map services as aggregation services on top of ECS. Hence, a key implementation design is how to compute the cost between two endpoints in ODL.

Comparing with traditional network, there are several differences in the SDN scenario. The central controller can collect the topology and statistics information of network easily. But some problems like fine-grained path and reactive mode have to be solved.

- T-AutoMap: How to allow a network operator (ALTO server administrator) to define automatically generated network maps?

  One possibility to define a network map is to allow the network operator to upload a static file defining the PIDs of the network map. Although this approach is modular, it is inconvenient. See Section 16 in [RFC7285]. Conceptually, a network map defines a partition of endpoints according to the properties of the endpoints. A mechanism (e.g., a description language) which allows a network operator to define the grouping conditions and then the ALTO server automatically to compute the partition can provide substantial value. After computing a network map, the ALTO server should also be able to compute the corresponding cost map, for each given cost metric. Since network state can be dynamic, we need to update network maps and cost maps when network state changes.

- T-Push: How to push updates to ALTO clients?

  Client would like to receive update information as soon as possible. See Internet draft [DRAFT-SSE].

3. Design and Implement ECS

There are two key issues when we try to implement ECS in ODL:

- How to get an exact forwarding path between two Endpoints.

- How to provide the reasonable costs computation in one query.

We have not yet implemented the functionality of ECS completely because of some challenges. Some of these challenges are caused by the limitation of ODL, but some are general problems in the SDN scenario.

Developers may face several challenges when implementing ECS in ODL. The following are the main challenges we faced:

About computing routing path:

- Challenge 1: There will be multiple fine-grained paths between two Endpoints in the SDN scenario.
- Challenge 2: The routing path may be re-active and have not been applied when the ECS query arrives.
- Challenge 3: SDN controllers like ODL support multiple applications to do the path computation.

About computing cost:

- Challenge 4: How to evaluate the precise cost of a known flow.
- Challenge 5: How to evaluate the reasonable costs when there are shared links.

In the following several subsections, we will talk about details of these challenges and our solutions. Some challenges have not been solved, and we discuss the reasons and give some proposals in Section 6.

3.1. Current Solution to Compute the Routing Path

Currently, our implementation of routing path computation in ODL contains two components: Host Tracker and Forwarding Rules Manager (FRM) Checker. And this implementation can only work with OpenFlow-enabled networks.

The Host Tracker will handle the ARP packets in the network and maintain the information of end hosts. It will store the bindings between MAC addresses and IP addresses. Because our ALTO server works on OpenFlow based networks, we need to know an OpenFlow match to decide a path. The OpenFlow match can be L2 or L3 in the real network. But the ECS query message only provides L3 information (IP address) of endpoints. So we implement a component like Host Tracker to maintain the map from L3 to L2.

ODL provides an FRM to manage the flow tables of real OpenFlow switches connected to the ODL controller. For pro-active paths, we can look up FRM to compute them. FRM Checker is such a component which provides API to compute pro-active paths by accepting L2 or L3 matches.

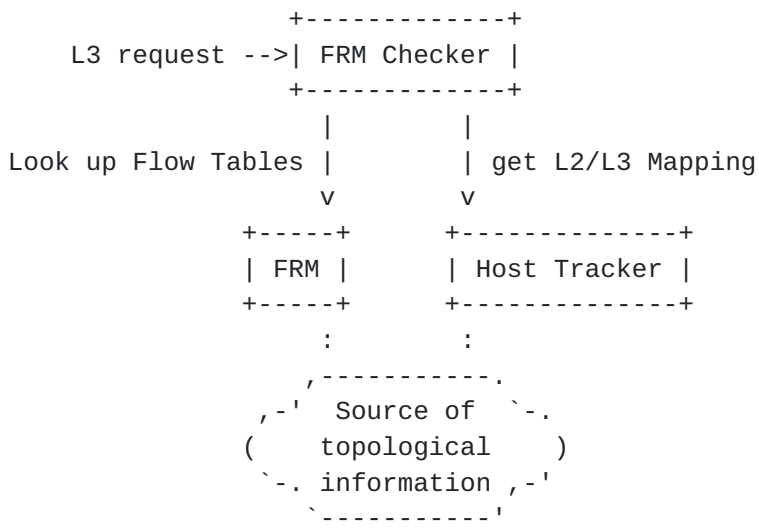

Figure 1: Overview of Routing Path Computation.

The overview of path computation module can be presented in Figure 1. And the algorithm of looking up FRM is presented in Figure 2.

```
while (currentSwitchId != dstSwitchId) {
    r <- loopupFlowTable(switchId, match);
    if (!r) {
        forceComputeRoutingPath(switchId, match);
        r <- loopupFlowTable(switchId, match);
    }
    currentSwitchId = getNextSwitchId(r);
}
```

Figure 2: Algorithm about lookupFRM().

3.2. Multi-Path in ECS

In the actual environment of network, there may be more than one routing path from the source IP to the destination IP. The cost between two Endpoints is decided by the actual routing path, but we may not get the actual routing path from the pair of the source IP and the destination IP. One reason is related to Challenge (2), and the subsection will talk about the details. The other reason is that the ALTO server cannot get enough information from the input of ECS.

For example, assume there are two hosts in the network, labeled as H1 and H2. And there are three switches in the links between H1 and H2. The topology is described as Figure 3. When H1 send data to the TCP port 22 of H2, the packet will be forwarded along the path "H1 - S1 - S3 - S2 - H2". But when H1 send HTTP request to H2, the packet will be forwarded along the path "H1 - S1 - S2 - H2".

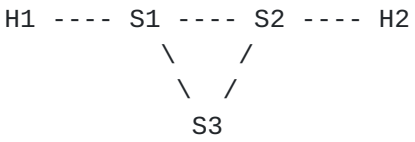

Figure 3: Multi-Path in Network.

In this case, the ALTO server will get two paths when looking up FRM to compute the routing path. Since the ALTO server does not know which type of packet will be sent by H1, it cannot decide which path is the actual one.

This problem is caused by the limitation of ALTO protocol and the features of OpenFlow. One of the most important features in OpenFlow-enabled networks is to support fine-grained path which makes the controlling of paths more flexible. But the original ALTO protocol is not expressive in this scenario. A possible solution is proposed in [DRAFT-ECS-FLOW]. The implementation of this solution is in progress.

3.3. Reactive Mode

We find this is a common problem in OpenFlow-enabled network. Once the network is working on the reactive mode, we may not know the real path only by checking devices information. There may be some routing paths which are still not active. Only when the special packets are sent to the special destination, the rule will be called to insert the Flow Table. So we may not get the routing path by looking up FRM.

We do not have a good solution to handle it. Although several modules in ODL provide some routing services to compute the path (such as l2switch), we still cannot know which module will be active.

We have tried to extend the input and output format of ECS. But it is not enough to solve this challenge.

3.4. Precise Cost Computation

Cost computation is often based on network statistics. In traditional network, we can setup some agents to monitor the network statistics in real time. But in SDN scenario, collecting the network statistics is easier. OpenFlow switches will store these statistics information in the Meter Tables (assume the switches support OpenFlow 1.3). And the ODL controller can look up these information directly without executing any measuring tasks.

The ECS module SHOULD evaluate the path cost as precisely as possible. However, OpenFlow switches can only collect their own statistics. If we want to get the statistics between endpoints, we have to make them aggregate. It MAY NOT be precise. If we want to make the evaluation more precise, we may have to do some real measurements in the network.

3.5. Available Bandwidth with Shared Links

Some cost metrics requested by clients may be shared by different flows, such as 'bandwidth'.

For example, a client sends an ECS request to get the available bandwidths between a list of source IPs and a list of destination IPs. The following example is a very common case:

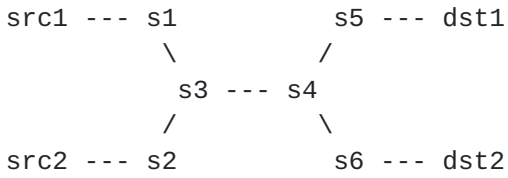

Figure 4: Bandwidth with Links Shared.

In the case described in Figure 4, "s3 - s4" is a link shared by all flows between [src1, src2] and [dst1, dst2]. If the client would like to select two pairs from (srci, dsti), their paths must share bandwidth in the link "s3 - s4". So the ALTO server cannot compute the available bandwidth of each flow individually.

An possible solution is to divide maximum bandwidth and available bandwidth into different 'cost-mode'. But it is still helpless to compute available bandwidth.

Another solution is to introduce Routing State Abstraction ([DRAFT-RSA]). The details will be discussed in Section 6.

3.6. A Comprehensive Architecture

The following is a comprehensive architecture to figure out our design:

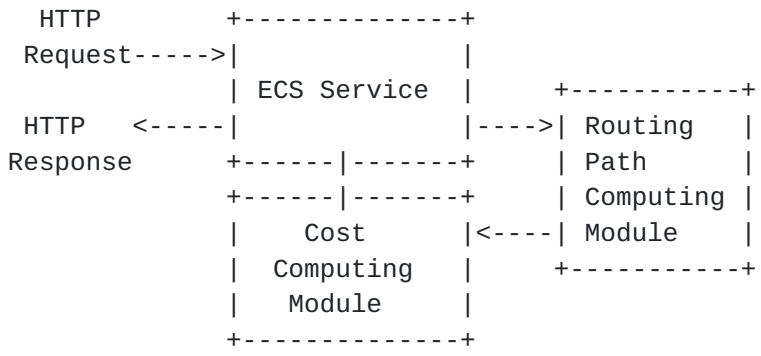

Figure 5: A Comprehensive Architecture of ECS.

4. Design and Implement Dynamic Maps

The ALTO server should be able to handle dynamic network. For example, when some nodes or links in the network topology change, the ALTO server must regenerate Network Maps and recompute Cost Map.

According to our experiences of implementing ALTO in ODL, there may be also several challenges about handling dynamic network. We will indicate these challenges and our solutions below. Some challenges have been solved, and we will introduce our solution. But some challenges still remain to be dealt with. We will also discuss them and the possible solutions in Section 6.

4.1. Challenges about handling dynamic network

The key challenges about dealing with dynamic network are indicated below:

- How to regenerate Network Maps:

  Network Maps are dependent on the network topology. The ALTO server should update Network Maps when the topology changes. For example, when a new host H1 is added to the network, the ALTO server should assign a PID for H1 in one Network Map. The challenge is that different Network Maps may have different rules to decide PID, but it is difficult to describe these rules. So it is hard to regenerate Network Maps automatically.

- When and How to recompute Cost Map:

  Every Cost Map depends on one Network Map. When the dependent Network Map is regenerated, the related Cost Map also need to be updated. Generally speaking, the ALTO server should recompute the cost for the PID which is updated. But sometimes, the update of

PID does not effect the cost. The ALTO server should decide when and how to recompute Cost Map.

- How to handle updates incrementally and quickly:

  According to [DRAFT-SSE], the ALTO server may provide a service which allows user to require incremental updates using SSE. But the ALTO server must have the capability to listen, compute and maintain the incremental updates. The challenge is how to provide incremental updates service correctly and efficiently.

4.2. Current Solution about Dynamic Network

4.2.1. Basic Service to Handle Dynamic Network

To handle the dynamic network, finding the updates of network is the basic capability. The update about hosts is the most basic type of updates.

As the description in Section 5.1.1, the ALTO server introduces a module named 'hosttracker' to find new hosts in the network. For example, once a new host H1 is added to the network, ALTO server will get the address of H1, and record it to the default Network Map.

4.2.2. Solution to Regenerate Network Maps

Our goal is to provide easy-to-use, yet complete specification and algorithms to allow administrators to define grouping of network nodes. We have designed an anchor-based Auto-Map service, which can generate Network Maps from network topology automatically. This service uses the nearest-neighbor algorithm to generate the Network Maps.

Administrators can modify a JSON format configuration file to configure the auto-map service. An example configuration file is presented by Figure 6.

nearest-network-map-config.json

```
{
  "net-map-id": "nearest-network-map",
  "net-map-grp-alg": "nearest-alg",
  "net-map-grp-para": {
    "metric": "hopcount",
    "anchors": {
      "pid1": ["sw1", "sw2"],
      "pid2": ["sw3"],
      "pid3": ["sw4", "sw5"]
    }
  }
}
```

Figure 6: An Example of Network Auto-Map Configuration File.

4.2.3. Solution to Recompute the Cost Map

Auto-Map service also provides a generic method to define the cost computation between two PIDs. The basic idea is to compute inter-PID cost from inter-endpoint costs: Given PIDS Pa and Pb, there will be |Pa| x |Pb| inter-endpoint costs. We provide multiple definitions (median, x-percentile, avg) as the cost from Pa to Pb, and allow multiple algorithms to do the computation (total enumeration, random sampling).

Administrators can also setup a JSON format configuration file to configure the related arguments. An example configuration file is presented by Figure 7.

cost-map-config.json

```
{
  "cost-map-id": "cmap1",
  "uses": [ "my-nn-auto-network-map" ],
  "cost-type": {
    "cost-mode": "numerical",
    "cost-metric": "hopcount"
  },
  "cost-map-group-metric": "avg",
  "cost-map-group-alg": {
    "alg": "random-sampling",
    "count" : 10000
  }
}
```

Figure 7: An Example of Cost Auto-Map Configuration File.

4.2.4. Solution to Handle Incremental Updates

We are implementing ALTO incremental updates using SSE in ODL. The following is a very simple design:

Figure 8: A Simple Architecture of the Update Service.

The update service is a top module to handle HTTP request from the client. The "DAG for Data" module computes JSON patches and store them to maintain all data changes from listener.

5. Achieve MD-SAL and Cross Platform Design

5.1. Overview of Current ALTO Server in ODL

5.1.1. Architecture

ALTO server provides two types of user interfaces -- one for application developers and the other for network managers. The developer interface provides a HTTP server to handle request/response defined in [RFC7285]. And the manager interface is a command-line interface, which provides commands to operate (add/delete/change) the data in data store.

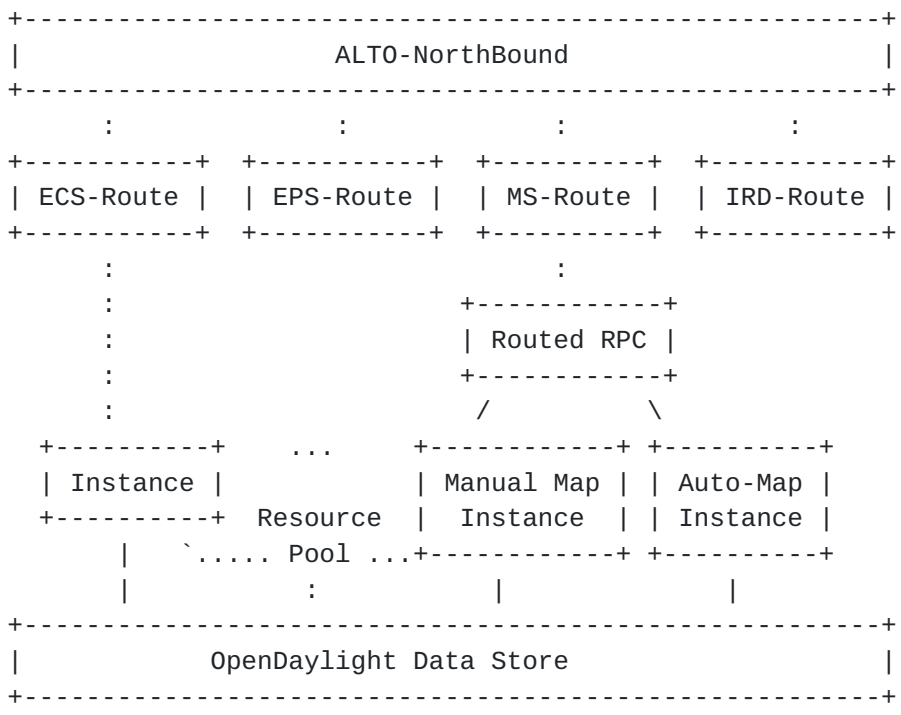

Figure 9: ALTO Architecture Overview.

As depicted in Figure 9, the services in this server are model-driven, and the foundation of these services is the data store in ODL. The models in this ALTO server define two major things: data types and the interfaces of RPCs (See [DRAFT-ALTO-YANG]. As it can be seen from the figure, there are five conceptual components whose names and functionalities are introduced in the following sections.

5.1.2. Components

The following is an introduction about the main components in this ALTO server.

- Northbound:

  The most important functionality of the northbound is to forward the incoming requests to the corresponding route. It has also defined the base URL for ALTO resources. Other connection-related operations can be taken here, such as authentication.

- Route:

  Route, short for northbound route, is where the ALTO protocol is processed. Upon a request arrival, it must check the media types, parse the request body (if any) to customized input formats and forward to the corresponding instance. When the output is

returned, the route must transform it into RFC-compatible format, set the correct media type and forward it to the northbound.

- Instance:

  Instance implements the ALTO services. Different instances can use different information sources and have different internal storage and logic.

- Datastore:

  Datastore is where the related data, including network statistics and user configurations, are stored. The OpenDaylight has already provided a tree-like datastore based on the YANG model.

- Resource Pool:

  The resource pool is where the instances SHOULD be registered. It is essential to support the standard service models and the standard northbound routes, and to provide information to the IRD.

5.1.3. Extensibility

In the practice of implementing ALTO server, we find extensibility is very important. ALTO needs extensibility because of two aspects.

The one is the protocol extension. There are more and more ALTO protocol extensions, and some of them have been used in the practice. ALTO server SHOULD provide a easy way to enable additional services for protocol extensions. In the design of Figure 9, we can add new route modules for the additional services easily.

The other one is the different implementations of services. A better practice is to allow different implementations for the services with the same interface. The architecture of Figure 9 allow different service instances to share the same route modules. It is enough extensible.

5.2. Implementation of Models

Programming in ODL is model-driven since Lithium release. So we should define the data types and RPCs by defining the YANG model. But when we try to use the YANG model defined in [DRAFT-ALTO-YANG] to implement the ALTO server in ODL, several problems occur, making some services not work.

In the following, we present the problems about the YANG model and our corresponding solutions.

5.2.1. The definition of 'cost'.

Outputs of the Cost Map and ECS both require a data type named 'cost', which stands for the cost between a source and a destination.

Section A.1 of [DRAFT-ALTO-YANG] defines 'cost' as following:

```
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.)";
  }
}
```

In this definition, 'cost' is declared as the 'anyxml' statement, which is used to represent an unknown chunk of XML (see [RFC6020]). It is because that 'cost' is defined as a JSONValue in [RFC7285], which could be any valid types in JSON.

But when we tried to implement the 'cost' type with its definition in the Lithium Release of ODL, we found that 'anyxml' was not implemented by the YANG parser as we expected.

Actually, there are two problems needed to be solved:

1. The Cost Map and ECS need different definitions of 'cost' type to generate different JAVA classes in ODL.

2. The 'cost' type could be different built-in types in different Cost Maps or outputs of ECS.

For the first problem, the 'augment' statement in YANG model could solve it.

For the second problem, however, we cannot use the 'anyxml' statement because JAVA is not dynamically typed. In order to support different built-in types, we use 'string' to define 'cost' type. But ALTO server must parse the value of 'cost' by itself.

Following is the current YANG model for the 'cost' type:

```
module alto-cost-default {
  namespace "urn:opendaylight:alto:costdefault";
  prefix "alto-cost-default";

  import alto-service {prefix alto-restconf;}

  augment "/alto-restconf:endpoint-cost-service/alto-restconf:
    output/alto-restconf:endpoint-cost-service/alto-restconf:
    endpoint-cost-map/alto-restconf:dst-costs" {
    leaf cost-default {
      type string;
    }
  }

  augment "/alto-restconf:resources/alto-restconf:cost-maps
    /alto-restconf:cost-map/alto-restconf:map/ alto-restconf:
    dst-costs" {
    leaf cost-default {
      type string;
    }
  }
}
```

5.2.2. The definition of 'constraint'

'Constraint' is an optional capability in [RFC7285]. The definition provided by [DRAFT-ALTO-YANG] is presented as follows:

```
typedef constraint {
  type string {
    pattern "(gt|ge|lt|le|eq) [0-9]+";
  }
  ...
}
```

This definition cannot support float 'cost' type. And we give the following definition to replace with it.

```
typedef constraint {
  type string {
    pattern "(gt|ge|lt|le|eq) [0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?";
  }
  ...
}
```

6. Discussions

6.1. ECS Extension

To address some issues in Section 3, we need to extend the data format of ECS. For example, ODL must know the TCP port of the destination to compute the actual routing path. So the client must indicate this information in the JSON of request.

6.2. Network State Abstraction

In some cases, the client send an ECS request to get the available bandwidths of some flows, which have shared links. The traditional method cannot give reasonable bandwidths for each flow. A possible solution to solve this issue is to introduce Routing State Abstraction.

6.3. A Loose Coupling Design to Support the Cross Platform

The current architecture of the ALTO server couples with the implementation of ODL. A loose coupling architecture design is expected. It will be very helpful to support the cross platform.

According to the discussion in Section 3.1, however, some services cannot decouple with ODL completely, such as ECS.

7. IANA Considerations

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

8. Security Considerations

This document does not introduce any privacy or security issue not already present in the ALTO protocol.

9. Acknowledgments

The authors thank discussions with Xin (Tony) Wang and reviews by Dan Peng and Qiao Xiang.

10. References

10.1. Informative References

[DRAFT-ALTO-YANG]
Shi, X. and Y. Yang, "A YANG Data Model for Base ALTO Data", 2015, <https://datatracker.ietf.org/doc/draft-shi-alto-yang-model/>.

[DRAFT-ECS-FLOW]
Wang, J. and Q. Xiang, "ALTO Extension: Endpoint Cost Service for Flows", 2015, <https://datatracker.ietf.org/doc/draft-wang-alto-ecs-flows/>.

[DRAFT-RSA]
Gao, K., Wang, X., Yang, Y., and G. Chen, "ALTO Extension: A Routing State Abstraction Service Using Declarative Equivalence", 2015, <https://datatracker.ietf.org/doc/draft-gao-alto-routing-state-abstraction/>.

[DRAFT-SSE]
Roome, W. and Y. Yang, "ALTO Incremental Updates Using Server-Sent Events (SSE)", 2015, <https://datatracker.ietf.org/doc/draft-ietf-alto-incr-update-sse/>.

10.2. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997, <http://xml.resource.org/public/rfc/html/rfc2119.html>.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", Oct 2010, <http://xml.resource.org/public/rfc/html/rfc6020.html>.

[RFC7285] Alimi, R., Penno, R., Yang, Y., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", 2014, <http://xml.resource.org/public/rfc/html/rfc7285.html>.

Authors' Addresses

J. (Jensen) Zhang
Tongji University
4800 Cao'an Road
Shanghai 201804
China

Email: jingxuan.n.zhang@gmail.com

Kai Gao
Tsinghua University
30 Shuangqinglu Street
Beijing 100084
China

Email: gaok12@mails.tsinghua.edu.cn

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

Email: yry@cs.yale.edu