Flow-based Cost Query

draft-gao-alto-fcs-06

J. Jensen Zhang
Kai Gao
Junzhuo Wang
Qiao Xiang
Y. Richard Yang

July 16, 2018@IETF 102
Updates: Overview

• Changes since -05 revision:
  • Major updates
    • Added *flow-specific information announcement* in the flow-based filter.
    • Introduced transmission-type="multicast" as a supported flow-specific information announcement.
  • Minor updates
    • Revised examples to better illustrate new features.
    • Renamed the address type "Domain Name" to "Internet Domain Name" to distinguish it with the "Domain Name" in the draft-ietf-alto-unified-props-new.
New Requirements

- Some novel network traffic optimization frameworks may involve multicast flows to improve the throughput. Previous revision (-05 and before) can only support the query for unicast flows.
- Unicast flows query may conduct inaccurate result in some cases:

  - Additional Requirement: The ALTO server SHOULD allow the ALTO client to specify different data transmission types (unicast/multicast) for transmissions in the query space.

```plaintext
If request availbw for unicast flows:
f1 (s1 -> d1): [ane1, ane2]  
f2 (s1 -> d2): [ane1, ane3]  
an1.availbw = 100  
an2.availbw = 100  
an3.availbw = 100  
f1.bw + f2.bw <= 100

If request availbw for multicast flows:
f0 (s1 -> p1): [ane1]  
f1 (p1 -> d1): [ane2]  
f2 (p1 -> d2): [ane3]  
an1.availbw = 100  
an2.availbw = 100  
an3.availbw = 100  
f0.bw, f1.bw, f2.bw <= 100
```
Flow-Specific Announcement

- This document proposes a general solution to query flows:
  - Extended Endpoint Address (ipv4/ipv6 -> eth, tcp, udp, ...)
  - Flow-based Filter (cross-product -> arbitrary end-to-end pairs)
  - Flow-specific Announcement (non-endpoint attributes)

- Extended Input Parameters:

```java
object {
  ... 
  [EndpointFilter endpoints;]
  [ExtEndpointFilter endpoint-flows<1..*>;]
} ReqEndpointCostMap;

object {
  [JSONObject flow-spec-announce;]
} ExtEndpointFilter : EndpiontFilter;
```

- "flow-spec-announce" is used to specify all non-endpoint attributes for flows.
How Does it Address Multicast Query

- First-step solution: Introduce "transmission-type" in "flow-spec-announce".

  ```json
  { "srcs": ["tcp6:203.0.113.45:54321"],
  "dsts": ["tcp6:group1.example.com:21"],
  "flow-spec-announce": {
    "transmission-type": "multicast" } }
  ```

- The ALTO server will interpret the destination addresses as multicast groups, and then expose all flows to the multicast targets:

  ```json
  "endpoint-cost-map": {
    "tcp6:203.0.113.45:54321": {
      "tcp6:group1.example.com:21": [ "ane:S2", "ane:D3" ],
    "tcp6:group1.example.com:21": {
      "tcp6:[fe80::40e:9594:da3d:34b]:21": [ "ane:G1" ],
      "tcp6:[fe80::826:daff:feb8:1bb]:21": [ "ane:G2" ] }
  }
  ```
Open Discussions

• Response for different flows between same endpoints.
  – Extend the response format without changing the request format
• Difficulty to implement it in real systems.
Next Steps

- Check more use cases for flow-based query.
- Implement it and show some experimental results by next meeting.
- WG item?
Backup Slides
Review Flow-based Query Design

General Requirements on ALTO for the Unified Interface:
- **More flexible input**: Target of FCS
- **More flexible output**: Target of Path Vector, Unified Property, Multi-Cost (RFC8189), Cost Calendar

Requirements on the Input Flexibility:
- **#1** More flexible shape of query space
- **#2** More expressive encoding of query entry

Basic Proposal of FCS:
- Arbitrary end-to-end query
- Expressive endpoint address
- Extensible flow description and arbitrary flow query
Remaining Issues

• Q1: How to achieve a unified query model?
  – We have two design options for the query model:
    • Partial mesh src-dst pairs
    • Extensible header space set

• Q2: How to resolve the flow attribute conflicts?
  – A flow definition may be invalid: A TCP socket source address cannot establish a valid connection with a UDP socket destination address.
  – Allow the server to notify this invalidity to the client as early as possible.
## Q1: Unified Query Model

<table>
<thead>
<tr>
<th>Design Option</th>
<th>Partial Mesh Src-Dst Pairs (Current Option)</th>
<th>Extensible Header Space Set (Another Option)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>[&quot;srcs&quot;: [addr1], &quot;dsts&quot;: [addr3, addr4]}, {&quot;srcs&quot;: [addr2], &quot;dsts&quot;: [addr3, addr5]}]</td>
<td>{&quot;f1&quot;: {&quot;ipv4:destination&quot;: v1, &quot;ethernet:vlan-id&quot;: v2}, &quot;f2&quot;: {&quot;ipv4:destination&quot;: v3, &quot;ipv4:source&quot;: v4}, &quot;f3&quot;: {&quot;ipv4:destination&quot;: v5, &quot;ipv4:source&quot;: v6, &quot;ethernet:vlan-id&quot;: v7}}</td>
</tr>
<tr>
<td>Compatibility</td>
<td>Response can be compatible</td>
<td>Incompatible</td>
</tr>
<tr>
<td>Request Size</td>
<td>Can be reduced</td>
<td>Cannot be reduced</td>
</tr>
<tr>
<td>Extensibility</td>
<td>Can introduce new endpoint address types</td>
<td>Can introduce new flow attributes</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Cannot request non-endpoint flow attributes</td>
<td>Can support arbitrary flow attributes</td>
</tr>
<tr>
<td>Complexity</td>
<td>Validation is simple (Only need to check source and destination)</td>
<td>Validation is complex (Need to check every shown attributes)</td>
</tr>
</tbody>
</table>

**Comparison between two design options**
# New Registered Address Types

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Encoding</th>
<th>Semantics</th>
<th>Potential Use Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>eth</td>
<td>MAC Address (EUI-48 or EUI-64)</td>
<td>The ethernet address</td>
<td>Layer2 flows between inter DCNs</td>
</tr>
<tr>
<td>domain</td>
<td>Domain Name (RFC2181)</td>
<td>Can be resolved by an A record</td>
<td>CDN</td>
</tr>
<tr>
<td>domain6</td>
<td>Domain Name (RFC2181)</td>
<td>Can be resolved by an AAAA record</td>
<td></td>
</tr>
<tr>
<td>tcp</td>
<td>IPv4 Socket Address</td>
<td>The client/server address of a tcp socket with an IPv4 address</td>
<td></td>
</tr>
<tr>
<td>udp</td>
<td>IPv4 Socket Address</td>
<td>The client/server address of a udp socket with an IPv4 address</td>
<td>Flow-level scheduling</td>
</tr>
<tr>
<td>tcp6</td>
<td>IPv6 Socket Address</td>
<td>The client/server address of a tcp socket with an IPv6 address</td>
<td></td>
</tr>
<tr>
<td>udp6</td>
<td>IPv6 Socket Address</td>
<td>The client/server address of a udp socket with an IPv6 address</td>
<td></td>
</tr>
</tbody>
</table>
Q2: Flow Attribute Conflicts

- **Original Design:**
  - Declare *conflicts* of new address type with each existing address types.
  - For example: tcp and udp
  - Some network with special technologies (e.g. NAT) may avoid some conflicts. So a server can declare the capability disagree with the conflicts defined in the registry.

- **Key observation:** Most of address types conflict with others.

- **Current Design:**
  - Declare *compatibility* instead of conflicts.
  - If the address type combination of a src-dst pair is not defined in the compatibility registry, it SHOULD be regarded as invalid.
  - A server can extend compatible address type combinations into its own capability.