Flow-based Cost Query

`draft-gao-alto-fcs-05`

Presenter: Jensen Zhang

Mar 19, 2018@IETF 101

Updates: Overview

- Many updates from -04 (Dec 13, 2017, IETF 100 Interim) to -05 (Mar 5, 2017, IETF 101)
  - Improve the clarity of the document by explicitly stating the problems.
  - Make terms used in this document clear.
  - Move Section 6 “Advanced Flow-based Query” out of this document.
  - Change “ALTO Address Type Conflicts Registry” to “ALTO Address Type Compatibility Registry”.

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

Design Option	Partial Mesh Src-Dst Pairs (Current Option)	Extensible Header Space Set (Another Option)

Example	`[{"srcs": [addr1],`
	`"dsts": [addr3, addr4]},`
	`{"srcs": [addr2],`
	`"dsts": [addr3, addr5]}]`
		`{"f1": {"ipv4:destination": v1,`
		`"ethernet:vlan-id": v2},`
		`"f2": {"ipv4:destination": v3,`
		`"ipv4:source": v4},`
		`"f3": {"ipv4:destination": v5,`
		`"ipv4:source": v6,`
		`"ethernet:vlan-id": v7}}`

Compatibility	Response can be compatible	Incompatible
Request Size	Can be reduced	Cannot be reduced
Extensibility	Can introduce new endpoint address types	Can introduce new flow attributes
Flexibility	Cannot request non-endpoint flow attributes	Can support arbitrary flow attributes
Complexity	Validation is simple (Only need to check source and destination)	Validation is complex (Need to check every shown attributes)

Comparison between two design options

New Registered Address Types

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

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.

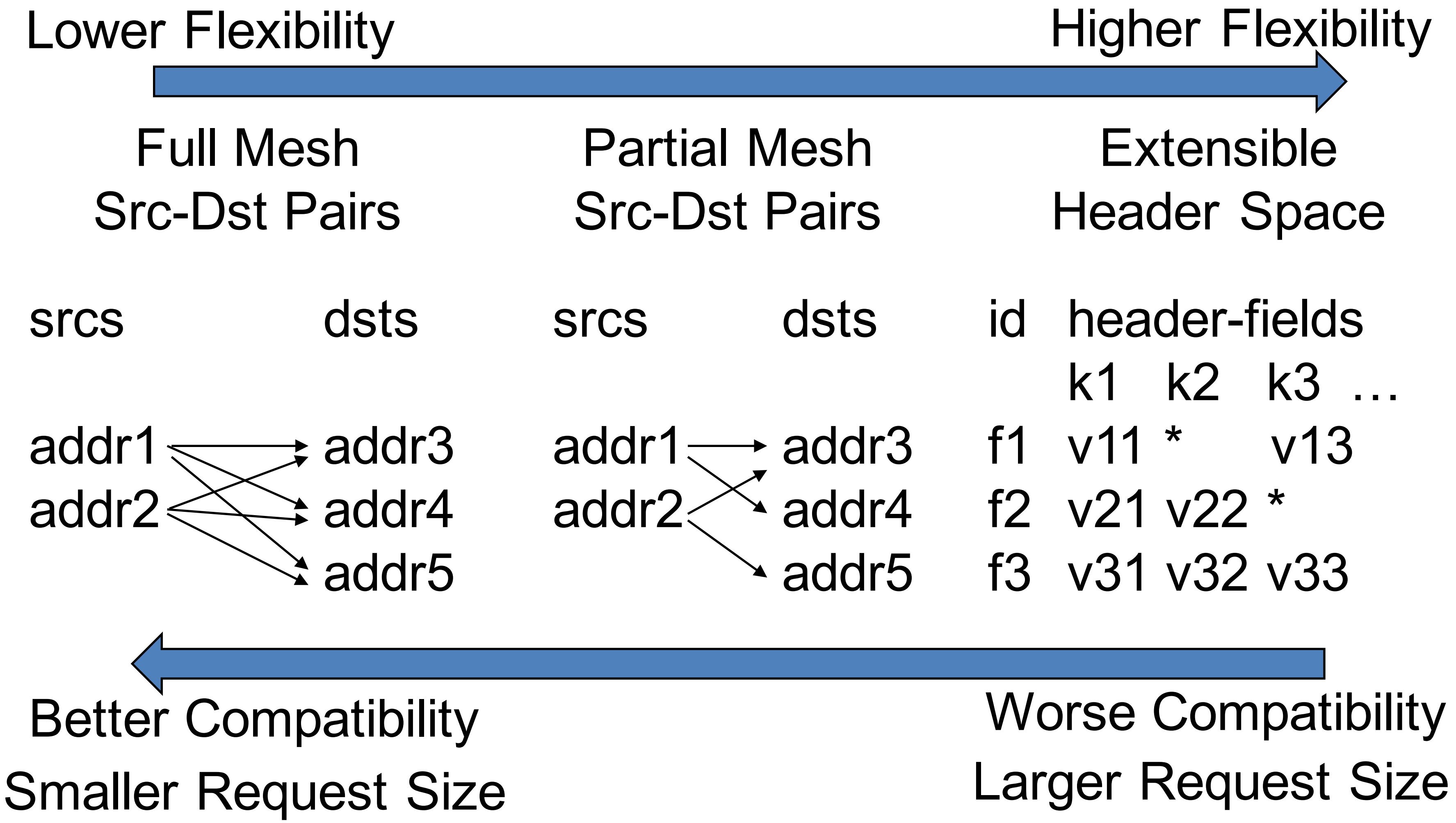
Next Steps

- Request for reviews/comments
- WG item?

Backup Slides

Flexible Shape of Query Space

- Different flexibilities of the query space



Lower Flexibility

Higher Flexibility

Full Mesh
Src-Dst Pairs

srcs	dsts
addr1	addr3
addr2	addr4
	addr5

Partial Mesh
Src-Dst Pairs

srcs	dsts
addr1	addr3
addr2	addr4
	addr5

Extensible
Header Space

id	header-fields			
	k1	k2	k3	…
f1	v11	*	v13	
f2	v21	v22	*	
f3	v31	v32	v33	

Better Compatibility
Smaller Request Size

Worse Compatibility
Larger Request Size

Flexible Shape of Query Space

- Full Mesh Src-Dst Pairs (Base ALTO Protocol)

  -
    ```
    {"srcs": [addr1, addr2]
     "dsts": [addr3, addr4, addr5]}
    ```
- Partial Mesh Src-Dst Pairs (Section 5 of FCS)
  - Advantage:
    - The response can be **compatible** with the base ALTO protocol
    - The size of request **can be reduced** by using multiple smaller full meshes
  - Drawback: Non-endpoint attributes cannot be supported

  -
    ```
    [{"srcs": [addr1],
      "dsts": [addr3, addr4]},
     {"srcs": [addr2],
      "dsts": [addr3, addr5]}]
    ```

Question: Can we achieve a unified query model?

- Extensible Header Space (Section 6 of FCS)
  - Advantage: non-endpoint attributes can be supported
  - Drawback: The response is **incompatible**; the size of request **cannot be reduced**

  -
    ```
    {"f1": {"ipv4:destination": v11, "ethernet:vlan-id": v13},
     "f2": {"ipv4:destination": v21, "ipv4:source": v22},
     "f3": {"ipv4:destination": v31, "ipv4:source": v32,
            "ethernet:vlan-id": v33}}
    ```

Expressive Query Entry Encoding

- Expressive Endpoint Address
  - *"An endpoint is an application or host that is capable of communicating (sending and/or receiving messages) on a network."* (RFC7285 Sec 2.1)
  - Encode 5-tuples to endpoint addresses
  - New AddressTypes for ALTO Address Type Registry
    - Use address type identifier to express **protocol semantics**
    - Different address types can use the **same address encoding** with **different semantics** (e.g. "tcp" and "udp")
- Extensible Flow Description
  - ALTO Header Field Registry
    - Current registry is a subset of **OpenFlow match fields**
    - Follow the **TLV dependencies** defined in OpenFlow
    - Allow to register new header fields

The Key Remaining Issue

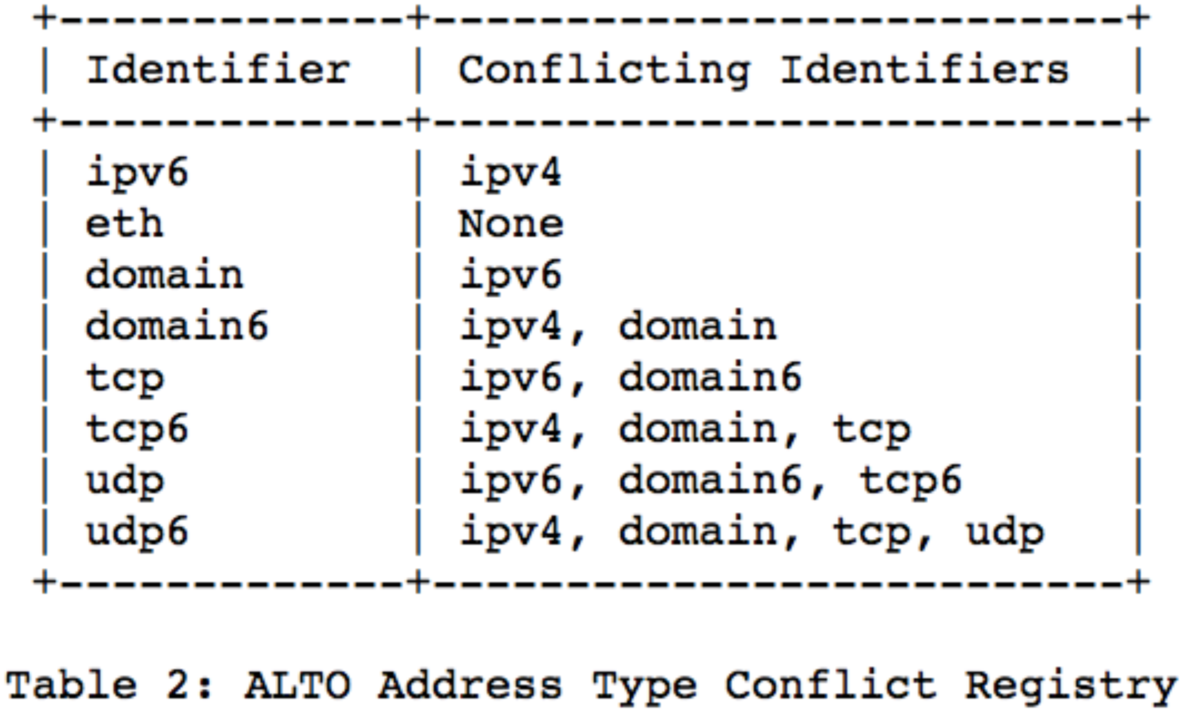
- Validation requirement
  - **Client: I want to query the cost of flow A**
  - **Server: the descriptor of flow A is invalid**
  - *"If the ALTO server does not define a cost value from a source endpoint to a particular destination endpoint, it MAY be omitted from the response"* (RFC7285 Sec 11.5.1.6)
  - General Problem from Client: **Which flows are available from this server?**
- Case1: Endpoint Conflict

  -
    ```
    {"srcs": ["tcp:203.0.113.45:54321"]
     "dsts": ["udp:8.8.8.8:8080"]}
    ```
- Case2: Invalid Flow Descriptor

  -
    ```
    {"flow1": {"ipv4:source": "203.0.113.45",
               "tcp:source": 54321,
               "udp:destination": 8080}}
    ```

Endpoint Conflict

- Declare conflicts of each address type
  - The conflicting identifier list of the future registered address types could be **longer and longer**
  - Some network with special technologies (e.g. NAT) may **avoid some conflicts**



Identifier	Conflicting Identifiers
ipv6	ipv4
eth	None
domain	ipv6
domain6	ipv4, domain
tcp	ipv6, domain6
tcp6	ipv4, domain, tcp
udp	ipv6, domain6, tcp6
udp6	ipv4, domain, tcp, udp

Table 2: ALTO Address Type Conflict Registry

Invalid Flow Descriptor

- Different cases of invalid flow descriptor
  - Missing required header fields
    - Validation: Declare “required” header fields list in “capabilities”
  - Conflicting header fields/values
    - Validation: Apply the TLV format validation defined in OpenFlow
  - Unsupported header fields
    - Validation: Check “required” and “optional” header fields list
- Limitation of a single “required” list
  - Server: Each flow MUST contain “ipv4 source and destination” **OR** “ipv6 source and destination”
  - A single “required” header fields list cannot express such a validator
  - Introduce “or-required”:

    -
      ```
      {"or-required":
      [["ipv4:source", "ipv4:destination"],
       ["ipv6:source": "ipv6:destination"]]}
      ```

Next Steps

- Move “Address Type Registry” and “Address Type Conflict Registry” to a new draft?
  - Consider other drafts (e.g. cellular addresses) have the same requirement
- Request for reviews/comments
- WG item?