Flow-based Cost Query

draft-gao-alto-fcs-01

Kai Gao\textsuperscript{1} \quad J. Jensen Zhang\textsuperscript{2} \quad J. Austin Wang\textsuperscript{2}

Qiao Xiang\textsuperscript{3} \quad Y. Richard Yang\textsuperscript{3}

\textsuperscript{1} Tsinghua University \quad \textsuperscript{2} Tongji University \quad \textsuperscript{3} Yale University

March 31@IETF 98
Flow-based Design in a Nutshell

Cost Services: Cost Map (Non-Query Service), Filtered Cost Map, ECS (Query Service)

Motivations:

- Flow correlation (CoFlow...)
  ➜ Extend the query scheme
  ➜ Augment the request or introduce new media-type for the request
- Fine-grained routing (OpenFlow, ECMP, MPLS...)
  ➜ Effect both the request and response
  ➜ Why not introduce a new resource (service)? (incomplete)
  ➜ Why not introduce a unified resource (service)? (complete)

Previous work:

- draft-wang-alto-ecs-flow: augment the syntax of TypedEndpointAddress -> EndpointURI
- draft-gao-alto-fcs: Introduce “application/alto-flowcost+json”, “application/alto-flowparams+json"

Major update since -00:

- Claim draft-wang-alto-ecs-flow-01 as the basic flow-based query design
- Claim draft-gao-alto-fcs-00 as the advanced flow-based query design
Key Issues

- **#1 How to encode a flow**
  - `<src, dst>` *(downward compatible)*
  - `{attribute -> value}` *(novel specification)*

- **#2 How to declare the capabilities**
  - "Boolean flow-query-support;"?
  - "JSONString support-attributes<1..*>;"?
  - TLV dependencies?

- **#3 How to encode a query scheme**
  - CommodityFilter? FlowNameFilter? FlowSpecFilter?

- **#4 How to deal with multipath**
  - Provide statistics? Exploration? Warning?
#1 Flow Expression Encoding

Basic Flow Encoding

- Commodity-based
  - `<src, dst>`
- Endpoint URI
  - `<protocol>:<address|name>[:<port>]`

Flow expression:

```
{
  "src": "tcp:192.168.1.2:80",
  "dst": "tcp:192.168.1.3:51234"
}
```

Advanced Flow Encoding

- Flow ID
  - Same format as a PIDName
    [RFC7285#Section 10.1]
- Typed header field
  - `<protocol-name>::<field-name>`

Flow expression:

```
"ssh-flow": {
  "ipv4:src": "192.168.1.2",
  "ipv4:dst": "192.168.1.3",
  "tcp:dst": "22",
  "eth:vlan-id": "20"
}
```
#2 Capabilities and #3 Query Schemes

Object {
    JSONString cost-type-names<1..*>;
    [JSONBool cost-constraints;]
    [JSONBool flow-based-filter;]
    [JSONString protocols<1..*>;]
} FlowFilteredCostMapCapabilities;

{ // ECS IRD Example
    “cost-type-names”: [“pv-ane”],
    “flow-based-filter”: true,
    “protocols”: [“ipv4”, “tcp”, “udp”]
}

{ // ECS Request Example
    “cost-type”: {“cost-mode”: “path-vector”,
                  “cost-metric”: “ane”},
    “endpoint-flows”: [{
        “src”: “tcp:10.0.0.1:8080”,
        “dst”: “tcp:10.0.0.2:51234”}
    ]
}

Object {
    JSONString cost-type-names<1..*>;
    TypedHeaderField required<1..*>;
    [TypedHeaderField optional<1..*>;]
    [JSONBool cost-constraints;]
} FlowCostMapCapabilities;

{ // FCS IRD Example
    “cost-type-names”: [“pv-ane”],
    “required”: [“ipv4:src”, “ipv4:dst”],
    “optional”: [“tcp:src”, “tcp:dst”]
}

{ // FCS Request Example
    “cost-type”: ..., 
    “flows”: {
        “test-l4-flow”: {
            “ipv4:src”: “10.0.0.1”, “ipv4:dst”: “10.0.0.2”,
            “tcp:src”: “8080”, “tcp:dst”: “51234”
        }
    }
}


#4 Multipath Issue

Notice that it is not a flow-based-specific issue. It exists for both flow-based query and non-flow-based query

```
// Statistics (Recommended)

“flow-cost-map”: {
    “test-l3-flow”: {“min”: 20, “max”: 40, “avg”: 30, “var”: 50}, ...
} // How to deal with the path vector?

// List all the potential paths

“flow-cost-map”: {
    “test-l3-flow”: [20, 40], ... // Means two different paths matching the same flow spec
} // How to work with multi-cost extension together?

// Warning

“flow-cost-map”: {
    “test-l3-flow”: “MP”, ...
} // The client may waste a query (this result is useless for the client)
```
Other Considerations

Basic Flow-based Error Handling

object-map {
  EndpointURI -> DstErrors;
} EndpointCostErrorMap;

object-map {
  EndpointURI -> EndpointFilterError;
  [JSONString conflicts<2..2>;]
  [JSONString unsupported;]
} DstErrors;

object {
  [JSONString conflicts<2..2>;]
  [JSONString unsupported;]
} EndpointFilterError;

Advanced Flow-based Error Handling

object-map {
  FlowId -> FlowCostError;
} FlowCostErrorMap;

object {
  [TypedHeaderField conflicts<2..*>;]
  [TypedHeaderField missing<2..*>;]
  [TypedHeaderField unsupported<1..*>;]
} FlowFilterError;
Open Discussions

- #0 Who is better to define flows?
  - Client-defined: specify the flow definition in the request
  - Server-defined: maybe in a prop-map, provided to the client for querying
- #1 New cost service or unified property service?
- #2 Simple constraints or general query language?
- #3 Endpoint aggregation or flow aggregation?
Open discussion: possible to use property map to implement flow-based query?

- Property Map to define the supported header fields and TLV dependencies
  - Declare the supported header fields for each endpoint?
- Property Map to define the supported flows
  - List all supported flows? (Too complex. A huge map)
- Property Map to provide the flow costs
  - Depends on the flow definitions
#2 General Query Across Resources

- Property Query Constraints
  - `{ “properties”: [“ipv4:src”, “tcp:src”], “constraints”: [“[1] eq 8080”] }

- Resource Dependency and Resource Query Joint
  - “flow-cost-prop-map” uses “flow-spec-prop-map”
  - The client can send a joint query:

```json
{ // A Joint Query Example
  “flow-spec-prop-map”: {
    “properties”: [“ipv4:src”, “tcp:src”],
    “constraints”: [“[0] eq 10.0.0.1”,
                    “[1] eq 8080”]
  },
  “flow-cost-prop-map”: {
    “entities”: “flow-spec-prop-map.cost-map.keys”,
    “properties”: [“cost”]
  }
}
```

```json
{ // A Joint Query for Path Vector
  “pv-cost-map”: {
    “cost-type”: {
      “cost-mode”: “path-vector”,
      “cost-metric”: “ane”
    },
    “pid-flows”: [{“src”: “PID1”, “dst”: “PID2”}]
  },
  “nep-map”: {
    “entities”: “union(pv-cost-map.cost-map.values)”,
    “properties”: [“availbw”],
    “query-id”: “pv-cost-map.meta.vtag.query-id”
  }
}
```

Remove the State
#3 Flow Aggregation

- PID is an approach to achieve the endpoint aggregation
- Define PFID to achieve the aggregation of flows?

```
"network-map": {
  "PID1": ["10.0.1.0/24"],
  "PID2": ["10.0.2.0/24"],
  ...
}

"pid-flows": [
  {"src": "PID1", "dst": "PID2"},
  ...
]

"flows": {
  "PFID1": {
    "ipv4:src": "10.0.1.0/24",
    "ipv4:dst": "10.0.2.0/24",
    "eth:vlan-id": "10"
  },
  ...
}
```
Future Work

Status:

- We are implementing the prototype in OpenDaylight

Next Step:

- Considering to merge with Path Vector?
- Try to use Unified Property Map?
Thank you!
Backup Slides