Routing State Abstraction
Using Declarative Equivalence

draft-gao-alto-routing-state-abstraction-01

G. Chen\textsuperscript{1} K. Gao\textsuperscript{3} X. Wang\textsuperscript{2} Y. R. Yang\textsuperscript{4}

\textsuperscript{1}Huawei \textsuperscript{2}Tongji University \textsuperscript{3}Tsinghua University \textsuperscript{4}Yale University

October 26, 2015@ ALTO Interim Meeting
Motivation

A general objective of ALTO is to provide generic network state to applications for better traffic optimization.

It is important that ALTO provide abstract network state:
- Protect information privacy
- Improve scalability

network raw state \(\rightarrow\) ALTO abstract state \(\rightarrow\) client
The current ALTO standard can provide
- any network information for a **single flow**
- **flow-irrelevant** network information for **multiple flows**, such as *hopcount*
- statistical network information based on the **Law of Large Numbers** for **multiple flows**, such as the average RTT between PIDs
Motivation

The current ALTO standard can provide

- any network information for a single flow
- flow–irrelevant network information for multiple flows, such as hopcount
- statistical network information based on the Law of Large Numbers for multiple flows, such as the average RTT between PIDs

Generally speaking, where the decisions for each flow are independent
However, many applications require multi-flow coordination

- Map-Reduce scheduling in data centers
- Traffic engineering in an ISP network
- ...

Limitations (cont.)
However, many applications require multi-flow coordination

- Map-Reduce scheduling in data centers
- Traffic engineering in an ISP network
- ...

Path vector can solve this by providing network state with common network elements for all the flows

- network element: link/AS/...
- network state: properties/statistics/...
Motivation

A Path-Vector Example

Figure: Example Topology

```
"PID1": {
    "PID2": ["ne15", "ne56", "ne67", "ne27"],
    "PID4": ["ne15", "ne57", "ne47"]
},
"PID2": {
    "PID1": ["ne27", "ne57", "ne15"],
    "PID3": ["ne35", "ne57", "ne27"]
},
"PID3": {
    "PID2": ["ne35", "ne57", "ne27"],
    "PID4": ["ne35", "ne57", "ne47"]
},
"PID4": {
    "PID1": ["ne47", "ne75", "ne15"],
    "PID3": ["ne47", "ne57", "ne35"]
}
```
Motivation

Key Question

How to compute abstract network state
Motivation

Key Question

How to compute **abstract** network state

- Return **dynamic** network state
How to compute **abstract** network state

- Return **dynamic** network state
- Return **minimal** network state
Motivation

How to compute abstract network state

- Return dynamic network state
- Return minimal network state
- Return equivalent network state
Motivation

Equivalence

A Generic Definition:

*The abstract network state $A$ for a user request is equivalent to the raw network state $R$, if and only if the user can make the same optimized decision with $A$ as with $R$.***
The RSADE, **Routing State Abstraction using Declarative Equivalence**, is proposed to provide such a network state abstraction service for a certain family of optimization: utilizing the objective function.

**Objective Function**

*An expression containing variables and mathematical constants.*

**Variable**

*Just like a math variable but usually has a specific physical significance.*
Optimal Equivalence

If the object function has the same solution using A and R, A and R are considered equivalent.
RSADE

Criteria of Equivalence

Optimal Equivalence

*If the object function has the same solution using A and R, A and R are considered *equivalent*.

Range Equivalence

*If the values of all linear combinations of variables, computed with A and R, have the same range, A and R are considered *equivalent*.*
Flow descriptor

*Specify the relevant flows.*

- Legacy: use EndpointFilter
- New: use FlowFilter*

Equivalence Condition

*Describe how the network can effect the decision making.*

- In RSADE, we limit this to *linear inequalities per link.*
Abstract Network State

- Path vector
  Return path vectors and let application construct the constraints.

- Constraints*
  Construct the constraints for the application using the format defined in *equivalence conditions* and return them.
How to specify FlowFilter

- A list of flows
- Consider possible OpenFlow use case: use tuples instead of destinations alone
- Each flow can be described as a \((src, dst)\) combination

\[
\text{FlowFilter} \ := \ \text{flow-list}
\]

\[
\text{flow-list} \ := \ \text{flow-spec}, \ [\text{flow-list}]
\]

\[
\text{flow-spec} \ := \ \text{generic-match-condition}
\]
RSADE

Extensions for FlowFilter

- Extension 1: more advanced endpoint address descriptors
  - draft-wang-alto-ecs-flows-00
- Extension 2: more fields in the flow specification
  - Examples: web-proxy, qos-group, etc.
Equivalence Condition

How to specify Equivalence Conditions

- Two kinds of inequalities
  - network-irrelevant: the constraint is application-specific
  - network-relevant: the constraint uses properties in the network

- Consider the structure of a network-relevant inequality
  - Math constants (provided by application)
  - Variables (provided by application)
  - Link properties (provided by network)
  - The routing information (provided by network)

\[ R[1] \times \text{flow1} + R[2] \times \text{flow2} \leq 0.8 \times \text{bandwidth} \]

- Optional: provide the objective function
equiv-cond := variable-list X0 link-constraint-list
variable-list := variable-name[, variable-list]
X0 := simple-constraint[, simple-constraint]
simple-constraint := simple-expr CMP-OP simple-expr
simple-expr := constant * variable-name[ + simple-expr]
link-constraints-list := link-constraint[, link-constraint-list]
link-constraint := link-expr CMP-OP link-expr
link-expr := constant | attribute-name | variable-name
| constant * link-expr
| attribute-name * link-expr
| link-expr + link-expr

See draft-gao-alto-routing-state-abstraction-01 for details.
Assume each link is 100Mbps and apply

- **Flow descriptor:**
  flows eh1→eh2(blue) and eh3→eh2(red)
- **Equivalence condition:**
  \[ R[1] * \text{flow1} + R[2] * \text{flow2} \leq \text{bandwidth} \]
We get

ne15: \( 1 \times \text{flow}_1 + 0 \times \text{flow}_2 \leq 100\text{M} \)
ne56: \( 1 \times \text{flow}_1 + 0 \times \text{flow}_2 \leq 100\text{M} \)
ne67: \( 1 \times \text{flow}_1 + 0 \times \text{flow}_2 \leq 100\text{M} \)
ne27: \( 1 \times \text{flow}_1 + 1 \times \text{flow}_2 \leq 100\text{M} \)
ne57: \( 0 \times \text{flow}_1 + 1 \times \text{flow}_2 \leq 100\text{M} \)
ne35: \( 0 \times \text{flow}_1 + 1 \times \text{flow}_2 \leq 100\text{M} \)
In order to satisfy the **minimal** and **equivalent** criteria, we have defined the following terms:

**[Equivalence]** Two constraint sets $S_1 : \{ \bar{x} | A_1 \bar{x} \leq \bar{b}_1 \}$ and $S_2 : \{ \bar{x} | A_2 \bar{x} \leq \bar{b}_2 \}$ of a network function are equivalent if and only if they limit the decision variables in the same way: $X_0 \cap S_1 = X_0 \cap S_2$.

**[Redundant]** A constraint $s$ is redundant to a constraint set $S$ if and only if $s \in S$ and the two sets $S$ and $S \backslash \{s\}$ are equivalent.

**[Minimal Constraint Set]** A constraint set $S$ is minimal if and only if $\forall s \in S$, $s$ is not redundant.
The minimal constraint set is

\[ \text{ne27: } 1 \times \text{flow1} + 1 \times \text{flow2} \leq 100M \]

And the corresponding path vector response is

\[ \text{eh1} \rightarrow \text{eh2}: [ \text{ne27} ], \]
\[ \text{eh3} \rightarrow \text{eh2}: [ \text{ne27} ] \]
Change the bandwidth of ne57 to 70Mbps, we have

- **ne15**: $1 \times \text{flow1} + 0 \times \text{flow2} \leq 100M$
- **ne56**: $1 \times \text{flow1} + 0 \times \text{flow2} \leq 100M$
- **ne67**: $1 \times \text{flow1} + 0 \times \text{flow2} \leq 100M$
- **ne27**: $1 \times \text{flow1} + 1 \times \text{flow2} \leq 100M$
- **ne57**: $0 \times \text{flow1} + 1 \times \text{flow2} \leq 70M$
- **ne35**: $0 \times \text{flow1} + 1 \times \text{flow2} \leq 100M
In this case, the minimal constraint set is

ne27: 1 * flow1 + 1 * flow2 <= 100M
ne57: 0 * flow1 + 1 * flow2 <= 70M

And the corresponding path vector response is

eh1 -> eh2: [ ne27 ],
eh3 -> eh2: [ ne27, ne57]
The **path vector** form of the second response in the example is demonstrated below:

"endpoint-cost-map": {
    "eh1": [ "eh2" : [ "ane1" ] ],
    "eh3": [ "eh2" : [ "ane1", "ane2" ] ]
},

"network-elements": {
    "ane1": { "bandwidth": "100 Mbps" },
    "ane2": { "bandwidth": "70 Mbps" }
}
The **constraint** form of the second response in the example is demonstrated below:

"flow-constraints": [
    "flow1 + flow2 <= 100000000",
    "flow2 <= 70000000"
]
Thank you