Routing State Abstraction
Algorithms for Compressing Path Vectors

draft-gao-alto-routing-state-abstraction-08

Jensen
K. Gao\textsuperscript{3}  X. Wang\textsuperscript{2,4}  Q Xiang\textsuperscript{2,4}  C. Gu\textsuperscript{4}  Y. R. Yang\textsuperscript{2,4}  G. Chen\textsuperscript{1}

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

Mar 19, 2018 @ IETF 101
What is RSA?

- Routing State Abstraction is a set of algorithms to provide the information for a set of correlated flows, encoded as the ALTO path vector extension.

How is RSA related to ALTO WG items?

- RSA provides a concrete implementation of the path vector extension.
- RSA can be used to 1) **compress** and 2) **improve the privacy of** an existing path vector response, **without loss of information**.
Changes

Since -06

▶ Improve the clarity of the algorithms
  ▶ Split the algorithms into small pieces
  ▶ Add an example for each piece
  ▶ Include the algorithms to interact with the path vector extension (encoding/decoding)
▶ Remove some extensions (i.e., client side bandwidth constraints) that are specific to the algorithms

Since -07

▶ Simplify the descriptions of the algorithms
▶ Extend the examples to include intermediate state
▶ Improve the wording
Table of Contents

Core Algorithms
▶ Equivalent Aggregation
▶ Redundant Constraint Identification
▶ Equivalent Decomposition

Interaction with the Path Vector Extension
▶ Decoding from PV
▶ Encoding to PV
Example

PID1 +-----+ +-----+ PID2
  |     |     |     |
  |     |     |     |
  |     |     |     |
  +-----+     +-----+     +-----+ PID2
  \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     \|     

| Link | Description | eh1: [ eh2: [ane:l1, ane:l5, ane:l2]]
|------|-------------|---------------------------
| l1   | sw1 <= sw5  | eh3: [ eh4: [ane:l3, ane:l5, ane:l4]]
| l2   | sw2 <= sw6  | abstract network element property map:
| l3   | sw3 <= sw5  | ane:l1 : 100 Mbps, 1
| l4   | sw4 <= sw6  | ane:l2 : 100 Mbps, 2
| l5   | sw5 <= sw6  | ane:l3 : 100 Mbps, 1
+------|            | ane:l4 : 100 Mbps, 1
                    | ane:l5 : 100 Mbps, 1
Equivalent Aggregation

Merge the links which have the same set of source-destination pairs.

To guarantee “no loss of information”: properties of the resultant link are calculated by “summing” the properties using UPDATE function.

<table>
<thead>
<tr>
<th>metric</th>
<th>UPDATE(x, y)</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>hopcount</td>
<td>x + y</td>
<td>0</td>
</tr>
<tr>
<td>routingcost</td>
<td>x + y</td>
<td>0</td>
</tr>
<tr>
<td>bandwidth</td>
<td>min(x, y)</td>
<td>+infinity</td>
</tr>
<tr>
<td>loss rate</td>
<td>1 - (1 - x) * (1 - y)</td>
<td>0</td>
</tr>
</tbody>
</table>
Example 1

Original:

set of pairs:
  ane:l1 : { eh1->eh2 }
  ane:l2 : { eh1->eh2 }
  ane:l3 : { eh3->eh4 }
  ane:l4 : { eh3->eh4 }
  ane:l5 : { eh1->eh2, eh3->eh4 }

properties:
  ane:l1 : 100 Mbps, 1
  ane:l2 : 100 Mbps, 2
  ane:l3 : 100 Mbps, 1
  ane:l4 : 100 Mbps, 1
  ane:l5 : 100 Mbps, 1

Merge ane:l1 and ane:l2 as ane:a, merge ane:l3 and ane:l4 as ane:b.

set of pairs:
  ane:a : { eh1->eh2 } (same as ane:l1 and ane:l2)
  ane:b : { eh3->eh4 } (same as ane:l3 and ane:l4)
  ane:l5 : { eh1->eh2, eh3->eh4 }

properties:
  ane:a : 100 Mbps, 3 (100 = min(100, 100), 3 = 1 + 2)
  ane:b : 100 Mbps, 2 (100 = min(100, 100), 2 = 1 + 1)
  ane:l5 : 100 Mbps, 1
Redundant Constraint Identification

Each link represents a linear bandwidth constraint. IS_REDUNDANT is an algorithm to find all redundant bandwidth constraints.

(A direct use case) Consider the bandwidth-only requests, if a constraint is redundant, the corresponding link can be removed too.

To guarantee “no loss of information”: bandwidth-only requests
Example 2

bw(eh1->eh2) \leq 100 \text{ Mbps (ane:a)}
bw(eh3->eh4) \leq 100 \text{ Mbps (ane:b)}
bw(eh1->eh2) + bw(eh3->eh4) \leq 100 \text{ Mbps (ane:l5)}

The first two constraints are \textit{redundant}.

bw(eh1->eh2) + bw(eh3->eh4) \leq 100 \text{ Mbps (ane:l5)}

The corresponding PV result:

set of pairs:
ane:l5 : \{ eh1-> eh2, eh3->eh4 \}

properties:
ane:l5 : 100 \text{ Mbps}
Limitations

Before:

set of pairs:
ane:a : { eh1->eh2 }
ane:b : { eh3->eh4 }
ane:l5 : { eh1->eh2, eh3->eh4 }

properties:
ane:a : 100 Mbps, 3 <- redundant
ane:b : 100 Mbps, 2 <- redundant
ane:l5 : 100 Mbps, 1

After removing links with redundant constraints (ane:a and ane:b):

set of pairs:
ane:l5 : { eh1->eh2, eh3->eh4 }

properties:
ane:l5 : 100 Mbps, 1

Routing cost information is “lost”.

Equivalent Decomposition

In general cases links with redundant constraints cannot be removed, but can be decomposed (which can be further aggregated).

Decomposition: split the set of pairs on a link, and treat the link as multiple links traversed by different subsets of pairs.

To guarantee “no loss of information”:

- Let \( P \) be the original set of pairs, \( P_i \) be the set of pairs of the \( i \)-th subset. The subsets should be disjoint \((P_i \cap P_j = \emptyset \text{ if } i \neq j)\) and complete \((\cup P_i = P)\).
- The properties of each decomposed link are the same as the properties of the original link.
Example 3

Before (bw for ane:l5 is changed for demonstration purpose):

set of pairs:
- ane:a : { eh1->eh2 }
- ane:b : { eh3->eh4 }
- ane:l5 : { eh1->eh2, eh3->eh4 }

properties:
- ane:a : 100 Mbps, 3
- ane:b : 100 Mbps, 2
- ane:l5 : 200 Mbps, 1 <- redundant

After decomposing ane:l5 to ane:c and ane:d:

set of pairs:
- ane:a : { eh1->eh2 }
- ane:b : { eh3->eh4 }
- ane:c : { eh1->eh2 }
- ane:d : { eh3->eh4 }

properties:
- ane:a : 100 Mbps, 3
- ane:b : 100 Mbps, 2
- ane:c : 200 Mbps, 1 (same as ane:l5)
- ane:d : 200 Mbps, 1 (same as ane:l5)
Equivalent aggregation and decomposition are discussed in our IWQoS paper\(^1\) but this document uses a new algorithm for decomposition (included in an extended version). Various algorithms exist to find redundant constraints in a set of constraints. The one mentioned in the document is first proposed by Telgen\(^2\) (Benefits: \texttt{simple} and \texttt{multiprocessing-friendly}).

Since we always aggregate after decomposing a link. The algorithm actually combines the aggregation step to reduce the overhead of storing temporary results.

“Perfect” decomposition which minimizes the number of links is \texttt{NP-hard} (binary matrix factorization\(^3\)) so the one proposed in the document is actually a greedy algorithm.

\textbf{Should such information (or part of it) be included in the draft?}


Decoding & Encoding

The transformation between the internal link-oriented data structure and the PV format. Please refer to the draft for more details.

path vectors (PV):
  eh1: [ eh2: [ane:l1, ane:l5, ane:l2]]
  eh3: [ eh4: [ane:l3, ane:l5, ane:l4]]

set of pairs (P):
  ane:l1 : { eh1->eh2 }
  ane:l2 : { eh1->eh2 }
  ane:l3 : { eh3->eh4 }
  ane:l4 : { eh3->eh4 }
  ane:l4 : { eh1->eh2, eh3->eh4 }

P = DECODE(PV)
PV = ENCODE(P)

Only extract the PV part so it is compatible with other extensions like multi-cost.
Summary

Current status:

▶ Role: supplement of the PV extension with referenced implementations.
▶ Better quality: cleaner descriptions and more examples.
▶ Target: Informational track (will be updated in the next revision)

Next steps:

▶ Adopt this document as a WG draft?
▶ Call for reviews from the WG
Q & A

Join the Discussion at alto@ietf.org!

Questions and Comments are Welcome!