Implementing new Topology Mapping Primitives

Guillermo Baltra
Prior Work


• Demonstrated the ability of each primitive to generate significant probing savings. Fewer probes implies potential to:
  – Improve quality of topologies as currently inferred
  – Additional results with same probing budget, e.g. alias resolution.
  – Perform more complete/detailed probing
  – Increase feasible frequency (i.e. speed) of full-topology inferences
Subnet Centric Probing (SCP)

- Adapt the number of probes to the degree of subnetting to avoid wasted probing.
- Discover internal structure of networks.
- Leverage BGP as coarse structure.
(b) Model fidelity: Subnet centric probing captures ≥ 90% of the vertices and edges.
(c) Induced load: Subnet centric uses $\approx 60\%$ of the ground truth load.
Subnet Centric Probing (SCP)

• 3 separable problems via a unified methodology:
  1. Select destinations.
  2. Select sources.
Least Common Prefix (LCP)

- Iteratively pick destinations within prefix that are maximally distant (in subnetting sense)
- 2 numerically consecutive IP addresses more likely to share paths.
- But address “distance” can be misleading: e.g. 18.255.255.100 vs 19.0.0.4 vs. 18.0.0.5
Least Common Prefix (LCP)

- Use knowledge of how networks are provisioned and subnetted.

Easier to believe A and B in different subnets

Than A’ and B’ in different subnets.
Least Common Prefix (LCP)
Least Common Prefix (LCP)

16.0.0.0/9

16.128.0.0/9

16.255.255.255

16.0.0.0

16.128.0.0

16.128.0.0/9

16.0.0.0/9

16.255.255.255/9
Least Common Prefix (LCP)
LCP and Target Determination

Center IP address in address range is selected as target for each smaller network.

16.0.0.0
dashed line at the same level as 16.63.255.254

16.128.0.0

16.191.255.254

16.255.255.255

dashed line at the same level as 16.63.255.254

dashed line at the same level as 16.191.255.254

16.0.0.0
LCP and Target Determination

- 16.192.0.0/10
- 16.255.255.255
- 16.0.0.0
- 16.63.255.254
- 16.223.255.254
- 16.191.255.254
- 16.159.255.254
- 16.128.0.0/10
- 16.128.0.0/10
- 16.192.0.0/10
- 16.159.255.254
- 16.223.255.254
- 16.191.255.254
- 16.63.255.254
Edit Distance Metric

• Pairwise probing from same source.
• Compares full traces.
• Load-balancing artificially distorts ED for some paths.
• Doesn’t take benefit from previous traces to the same prefix.
• In practice: recurse all the way down to /32s.
• Note: this occurs even when using Paris-style traceroute. Paris ensures determinism over per-flow load balanced path to a given destination. SCP uses different destinations as part of its exploration algorithm.
Threshold = 2

Legend:
- Monitor
- Hop
- Destination

Pairwise probing:
- Trace Nr. 1
- Trace Nr. 2

Load-balancer
Threshold = 2

Legend:
- Monitor
- Hop
- Destination

Pairwise probing:
- Yellow: Trace Nr. 1
- Green: Trace Nr. 2

Load-balancer

16.0.0.0

16.255.255.255
Threshold = 2
Edit Distance = 4

Legend:
- Monitor
- Hop
- Destination

Pairwise probing:
- Trace Nr. 1
- Trace Nr. 2

Load-balancer
Threshold = 2

Pairwise probing:

Legend:
- Monitor
- Hop
- Destination

Trace Nr. 1
Trace Nr. 2

Load-balancer
Threshold = 2

Load-balancer

Legend:
- Monitor
- Hop
- Destination

Pairwise probing:
- Trace Nr. 1
- Trace Nr. 2
Threshold = 2
Edit Distance = 3

Legend:
- Monitor
- Hop
- Destination

Pairwise probing:
- Yellow = Trace Nr. 1
- Green = Trace Nr. 2

Load-balancer

Trace Nr. 1
Trace Nr. 2
Threshold = 2

Legend:
- Monitor
- Hop
- Destination

Pairwise probing:
- Yellow: Trace Nr. 1
- Green: Trace Nr. 2

Load-balancer

Trace Nr. 1
Trace Nr. 2

16.0.0.0
16.255.255.255
Threshold = 2

Legend:
- Monitor
- Hop
- Destination

Pairwise probing:
- Yellow: Trace Nr. 1
- Green: Trace Nr. 2

Load-balancer
Threshold = 2
Edit Distance = 3
Threshold $= 2$

and so on...

Legend:
- **Monitor**
- **Hop**
- **Destination**

Pairwise probing:
- **Trace Nr. 1**
- **Trace Nr. 2**

Load-balancer

16.0.0.0

16.255.255.255
New Interface Discovery (NID)

• Focus on destination AS:
  – SCP’s objective is to discover structure within the destination AS. SCP should base its operation on new structure (edges, vertices) discovered in target AS.

• Hops inside destination AS are added into a set to which future traces will be compared.

• Number of new hops discovered inside prefix is compared to a threshold.

• Traces sent to original prefix get all their hops inside the prefix counted as newly discovered.

• Not affected by load balancing as a stopping criterion.

• Benefits from load balancing in terms of vertices and edges learning.
New Interface Discovery (NID)

- **Source Distribution:**
  - By focusing on target AS, we can distribute the source of each probe, as opposed to pairwise probing. Using multiple vantage points as part of SCP naturally helps discover AS ingress points.

- **Number of traces per prefix:**
  - For a pairwise probing, 2 traces need to be sent per prefix. This new method allows to sent as many probes as needed.

- **Maintain State:**
  - SCP’s recursive stopping criterion should consider all traces to a destination prefix, rather than just being pair-wise.

- **Future Improvements:**
  - Consider number of new edges.
Threshold = 1

3 inputs:
1. Threshold
2. Set of VPs
3. Prefixes

Output:
Every trace is stored for later analysis.
VPS

A

B

C

D

Threshold = 1

Monitors/Destinations Pairs Queue:

Discovered Prefix-Vertices List:
VPS
A
B
C
D

Threshold = 1

Monitors/Destinations Pairs Queue:

Discovered Prefix-Vertices List:
Threshold = 1

Discovered Prefix-Vertices List:

Monitors/Destinations Pairs Queue:
B-1 | D-2

Discovered Prefix-Vertices List:
VPS
①
② Parent trace
③
④

Threshold = 1

Monitors/Destinations Pairs Queue:

D-2

Discovered Prefix-Vertices List:

1 – 3 – 4 – 5
Threshold = 1

Monitors/Destinations Pairs Queue:

D-2

Discovered Prefix-Vertices List:

1 – 3 – 4 – 5
VPs

Threshold = 1

Parent trace

Discovered Prefix-Vertices List:

Monitors/Destinations Pairs Queue:

D-2 | A-6 | C-7

Discovered Prefix-Vertices List:

1 – 3 – 4 – 5
Monitors/Destinations Pairs Queue:

A-6 | C-7

Discovered Prefix-Vertices List:

1 - 3 - 4 - 5
Threshold = 1

Monitors/Destinations Pairs Queue:
A-6 | C-7

Discovered Prefix-Vertices List:
1 – 3 – 4 – 5
VPS

Threshold = 1

Monitors/Destinations Pairs Queue:
A-6 | C-7 | etc...

Discovered Prefix-Vertices List:
1 − 3 − 4 − 5
VPS

Threshold = 1

Monitors/Destinations Pairs Queue:
C-7 | etc...

Discovered Prefix-Vertices List:
1 – 3 – 4 – 5 – 6 – 8 – 9
Threshold = 1

Monitors/Destinations Pairs Queue:

C-7 | etc...

Discovered Prefix-Vertices List:

1 – 3 – 4 – 5 – 6 – 8 – 9
VPs

Threshold = 1

Monitors/Destinations Pairs Queue:
C-7 | etc...

Discovered Prefix-Vertices List:
1 – 3 – 4 – 5 – 6 – 8 – 9
Monitors/Destinations Pairs Queue:

e tc...

Discovered Prefix-Vertices List:

1 – 3 – 4 – 5 – 6 – 8 – 9
VPS

Threshold = 1

Monitors/Destinations Pairs Queue:

etc...

Discovered Prefix-Vertices List:

1 – 3 – 4 – 5 – 6 – 8 – 9
NID Pseudo Code

• Input:
  – p/m: Destination prefix / mask
  – M: Set of monitors
  – τ: Threshold of new interfaces

• Output:
  – T: Set of path traces; initially empty

Algorithm 1: NID(p/m, M, τ)

1: I = empty set // global variable
2: T = empty set // global variable
3: SCP(p/m, M, τ)
Algorithm 2: SCP(p/m, M, \( \tau \))

1: \( A = \text{ASN}(p/m) \)
2: \((d_1, d_2) = \text{determine\_targets}(p/m) \) \quad // LCP algorithm
3: \((m_1, m_2) = \text{assign\_monitors}(M, p/m) \) \quad // Random, VPS, IPS, Max.
4: \( t_1 = \text{trace}(m_1, d_1) \)
5: \( t_2 = \text{trace}(m_2, d_2) \)
6: \( T = T \cup (t_1 \cup t_2) \)
7: \( t'_1 = \{\text{interface } i \text{ in } t_1 | \text{ASN}(i) = A\} \)
8: \( t'_2 = \{\text{interface } i \text{ in } t_2 | \text{ASN}(i) = A\} \)
9: \textbf{for } j = 1 \textbf{ to } 2 \textbf{ do}
10: \quad \textbf{if } t'_j \text{ is parent\_trace } \textbf{then}
11: \quad \quad \textbf{if } \text{length}(t'_j) > 0 \textbf{ then}
12: \quad \quad \quad I = I \cup t'_j
13: \quad \quad \text{SCP}(p/(m + 1), M, \tau)
14: \quad \textbf{else:}
15: \quad \quad \textbf{if } |t'_j - I| \geq \tau \textbf{ then}
16: \quad \quad \quad I = I \cup t'_j
17: \quad \quad \text{SCP}(p/(m + 1), M, \tau)
Random Monitor Assignment

• Each destinations gets a monitor assigned randomly.
• Uniform Distribution.
• Random assignment of destinations to vantage points is wasteful.
  – E.g. empirically, the 16 /24’s in a /20 prefix are hit on average by 12 unique VPs.
  – 2 paths from 2 different monitors towards the same prefix might be too similar, and therefore it is inconvenient to assign the monitors consecutively.
Vantage Point Spreading

• New destinations are assigned to VPs not yet used for probing the original BGP prefix
• Uniform distribution of VPs when more destinations than VPs.
• Implementation: build list of all monitors in random order for each prefix. Round robin for assignment.

e.g. Some /16 prefix: Mon 1  Mon 2  Mon 3  ...  Mon N
Vantage Point Spreading

• Issues:
  – Early termination.
  – Doesn’t maximize the benefit of each probe in terms of new vertices learnt.
  – 90% of prefixes send less probes than the number of available monitors.
Prefixes: 1500
Monitors: 60
Method: Random
Zooming in ....

CDF

Prefixes: 1500
Monitors: 60
Method: Random
Vantage Point Spreading

• Proposed Solution: Rank Ordering Methods
  – Maximum Coverage and Ingress Point Spreading.
  – By carefully deciding the order in which monitors are assigned, the amount of new information learnt can be enhanced.
  – Pre-probing or previous data required as bootstrap.
2 consecutive traces might share many hops, thus forcing stop criterion too early. The path might be too short and therefore not maximizing the benefit of that particular probe.

- Long path maximizes the number of vertices learnt.
- Greater chances of finding different ingress routers.
Max Coverage

• IANA: /8 allocations by region.
• Focuses in maximizing vertices outside destination-AS.
• Max hop count first vs. Min hop count first.
• Abstract example:
  – One IP address per each /8 prefix randomly selected.
  – Every monitor probes every destination address.
  – Traces are connected with each other forming the following network...
Each address of each /8 is selected and worked through the process ones, as this example does for 4 /8.

RANK ORDER:

VPs 1 2 3 4 5 6

Destinations

1 /8 2 /8 3 /8 4 /8 5 /8...

254 /8 255 /8
1. Find the shortest path from source to destination for all VPs.
2. Find the longest path from all the shortest paths.

RANK ORDER:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RANK ORDER:

<table>
<thead>
<tr>
<th>VPs</th>
<th>Destinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/8</td>
</tr>
<tr>
<td>2</td>
<td>2/8</td>
</tr>
<tr>
<td>3</td>
<td>3/8</td>
</tr>
<tr>
<td>4</td>
<td>4/8</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>254/8</td>
</tr>
</tbody>
</table>

RANK ORDER:

| 1   | 6   | 3   |
RANK ORDER:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

VPs

1
2
3
4
5
6

Destinations

1/8
2/8
3/8
4/8
5/8
254/8
255/8
RANK ORDER:

<table>
<thead>
<tr>
<th>VP</th>
<th>1</th>
<th>6</th>
<th>3</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
</table>

Destinations:

| 1/8 | 2/8 | 3/8 | 4/8 | 5/8 | ... | 254/8 | 255/8 |
RANK ORDER:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>6</th>
<th>3</th>
<th>5</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
</table>

VPs

1
2
3
4
5
6

Destinations

1/8
2/8
3/8
4/8
5/8
254/8
255/8
MAX

• Input:
  – G: Graph built by pre-probed traces
  – M: Set of monitors
  – P: Set of each /8 prefix

• Output:
  – R: Set of ranked monitor lists per /8 prefix
Algorithm 2: MAX(G, M, P)

1: for each prefix in P do
2: \[ Y = \emptyset \] (Set of shortest paths from monitors to selected destination)
3: \[ \text{dest}_{\text{prefix}} = \text{select one destination in P from G.} \]
4: for each monitor in M do
5: \[ t = \text{shortest\_path(monitor, dest}_{\text{prefix}}, G) \]
6: \[ Y = Y \cup t \]
7: for each monitor in M do
8: \[ m = \text{max}(Y) \]
9: append(m) to \( r_{\text{prefix}} \)
10: delete all hops in m from all traces in Y
11: \[ R = R \cup r_{\text{prefix}} \]
Ingress Point Spreading (IPS)

- Focuses in finding destination prefix ingress routers.
- Builds a list of ranked monitors for each prefix from data gathered by pre-probing.
- Relates ‘monitor – first hop inside /8 prefix – destination’ for all traces in database.
- First monitors in list are chosen from those that have a unique first hop inside /8 prefix towards destination prefix.
- Rest of list is filled with monitors that have a trace towards the expansion of the original prefix, e.g. if the original prefix is /16, the expansion would be /15, /14, ..., /8.
- Needs more data than the Max Coverage method.
IPS: Intuition

By starting to probe with those monitors that have followed different paths into the destination prefix during pre-probing, increases the chances of finding destination prefix ingress router and thus delaying meeting stop criterion.
RANK ORDER:
IPS++

- Avoid IPS early termination.
- Ensure use of all ingress routers, and thus possible paths.
- Number of ingress routers approximated to the closest power of 2.
Suppose current prefix has 3 ingress routers, then the closest power 2 is 4.
Ingress Routers Analysis

Prefixes Fraction vs. Number of Ingress Routers

- /8
- /9
- /10
- /11
- /12
- /13
- /14
- /15
- /16
- /17
- /18
- /19
- /20

outside /8 prefix
Ingress Point Spreading (IPS)

• Input:
  – T: Set of pre-probed traces
  – M: Set of monitors
  – p/m: Set of prefixes / masks

• Output:
  – R: Set of ranked monitor lists per prefix / mask
Algorithm 3: IPS(T, M, p/m)

1: {I} = ⊥ (set of tuples formed by (monitor, first hop in destinations /8 prefix and destination) from each trace)

2: {J} = ⊥ (set of tuples formed by monitor, first hop in /8 prefix from traces with destination inside p/m and mask)

3: for each trace in T do
4:     (src, dst) = source and destination of trace
5:     D = /8 prefix of dst
6:     hop = find first hop in trace inside D
7:     I = I U (src, hop, dst)
8: end for

9: for each prefix in p/m do
10:   r_{prefix} = ⊥ (array of tuples containing monitor and first hop)
11:   for each monitor in M do
12:       flag = False
13:       for mask = m to 8 in steps = -1 do
14:           if flag then
15:               break // continue with next monitor
for each tuple in I that contains monitor do
    D = /mask prefix of dst in tuple
    P = /mask prefix of p
    if D == P then
        append (src, hop, mask) to \( r_{prefix} \)
        flag = True
    end if
end for
end for
delete tuples from \( r_{prefix} \) with repeated monitors and first hops
sort\( (r_{prefix}) \) // by mask from higher to lower
\( J = J \cup r_{prefix} \)
end for
References: