The Theory of Network Tracing

H. B. Acharya and Mohamed G. Gouda

The University of Texas at Austin
acharya@cs.utexas.edu

May 2009

Why Trace a Network?

- Knowing the topology of a network in the Internet can be useful:
  - Route aggregation (e.g., CIDR)
  - Security against address spoofing
  - Security against Denial of Service attacks

- Question:
  Is it possible to trace the topology of a network in the Internet?
Tracing a Network

- Use “Traceroute” to compute the sequence of nodes, i.e. trace, between any two given nodes in the network.

- Example: Using Traceroute to trace a network N, one may get the two traces (a d e b) and (a d e c) and infer the following topology of N:

![Diagram of network topology]

The Trouble with Traceroute

- In a computed trace, some nodes appear with an anonymous identifier “*i”, rather than with their unique identifiers (i.e. their IP addresses).

- Example:
  Using Traceroute to trace a network N, one may get the two traces (a d *1 b) and (a *2 e c) and infer that the topology of N is any one of ten candidate topologies.
Previous Work

- [Yao et al - 03], [Jin et al - 06], [Gunes et al - 08] have focused on the problem of inferring the topology of a traced network from its set of traces, computed by Traceroute.

- They “choose” to infer the network topology with the smallest number of anonymous nodes.

- Two problems:
  - this choice is arbitrary, and
  - the problem becomes NP-Complete.

Our Approach

- Attempt to identify a set of “reasonable conditions” that need to be satisfied by the computed trace set so that the inferred network topology is unique.

- The result of our attempts is a mixed bag of positive and negative results.
Network N

- N is a connected, undirected graph where each node has a unique identifier: a, b, ... For example, network N₀:

```
     a
    / \  
   b   c   d
```

- Each node in N is:
  - either regular a, b, .., or irregular a/*, b/*, ..

- Each node in N is:
  - either terminal “square”, or nonterminal “circle”

---

Trace t Generable from N

- t is a sequence of node identifiers:
  - t represents a simple path in N
  - The first and last identifiers in t are the unique identifiers of terminal nodes in N
  - Each regular node “a” in N appears as “a” in t
  - Each irregular node “a/*” in N appears as either “a” or “*i” in t

---
Traces Generable from $N_0$

- Network $N_0$:

  \[
  \begin{array}{c}
  \text{a} \\
  \text{b} * \text{c} * \text{d} \\
  \text{c} * \text{d} \\
  \text{d} * \text{c} \\
  \end{array}
  \]

- $t_1 = (b \ c \ d)$ valid trace
- $t_2 = (d \ c \ b)$ same as $t_1$
- $t_3 = (a \ *2 \ *3 \ a)$ not valid
- $t_4 = (*5 \ c \ d)$ not valid
- $t_5 = (d \ a)$ not valid
- $t_6 = (a \ *1 \ c)$ not valid

Trace Set $T$ Generable from $N$

- $T$ is a nonempty set of traces, each of which is generable from $N$.

- $T$ satisfies four conditions that we describe, one by one, next

- These conditions are intended to simplify the construction of $N$ from a given $T$. 
Condition 1

- For every pair of terminal nodes “a” and “b” in N, T has a trace whose first and last node identifiers are “a” and “b”.
- Motivation:
  - $T_1 = \{(a \ b), (b \ c \ d)\}$ is generable from network $N_0$.
  - As $T_1$ does not satisfy Condition 1, it is also generable from network $N_1$:

```
\begin{center}
\begin{tikzpicture}
    \node (a) [shape=rectangle] at (0,0) {a};
    \node (b) [shape=rectangle] at (1,-1) {b};
    \node (c) [shape=circle] at (2,0) {c};
    \node (d) [shape=rectangle] at (3,-1) {d};
    \path (a) edge (b);
    \path (b) edge (c);
    \path (c) edge (d);
\end{tikzpicture}
\end{center}
```

Condition 2

- Each edge in N appears in at least one trace in T
- Motivation:
  - $T_2 = \{(a \ c \ b), (b \ c \ d), (d \ c \ a)\}$ is generable from $N_0$
  - As $T_2$ does not satisfy Condition 2, it is also generable from network $N_2$:

```
\begin{center}
\begin{tikzpicture}
    \node (a) [shape=rectangle] at (0,0) {a};
    \node (b) [shape=rectangle] at (1,-1) {b};
    \node (c) [shape=circle] at (2,0) {c};
    \node (d) [shape=rectangle] at (3,-1) {d};
    \path (a) edge (c);
    \path (c) edge (d);
\end{tikzpicture}
\end{center}
```
Condition 3

- Every node in N appears by its unique identifier in at least one trace in T.
- Motivation:
  - $T_3 = \{(a \ b), (a \ast 1 \ d), (a \ b \ast 2 \ d), (b \ast 3 \ d)\}$ is generable from $N_0$
  - As $T_3$ does not satisfy Condition 3, it is also generable from network $N_3$:

![Diagram](chart1)

Condition 4

- For all pairs of nodes (x, y), the same nodes appear between x and y in all traces
- Motivation:
  - $T_4 = \{(a \ c \ d), (a \ast 1 \* 2 \ d), (a \ast 3 \ b), (b \ast 4 \ d)\}$ is generable from $N_0$
  - As $T_4$ does not satisfy Condition 4, it is also generable from network $N_4$:

![Diagram](chart2)
Network Tracing Problem

- Design an algorithm that takes a trace set $T$, generable from a network, and computes network $N$ such that
  - $T$ is generable from $N$ and
  - $T$ is not generable from any other network
- This problem has trivial solution if $T$ has no *’s
- Does the problem have solution if $T$ has only one *?

Theorem 1

- There is no algorithm that takes a trace set $T$, with one anonymous identifier, that is generable from a network, and computes a network $N$ such that
  - $T$ is generable from $N$, and
  - $T$ is not generable from any other network.
- Proof: Show a trace set $T$, with one anonymous identifier, that is generable from two distinct networks $N’$ and $N’’$. 
Proof of Theorem 1

- \( T = \{(a \ b), (a \ast 1 \ d), (a \ f), (b \ c \ d), (b \ f), (d \ e \ f) \} \)

Theorem 2

- There is an algorithm that takes \( T \), generable from a tree network, and computes a tree network \( N \) such that
  - \( T \) is generable from \( N \), and
  - \( T \) is not generable from any other tree network.

- Proof: We exhibit the tree construction algorithm.
Tree Construction Algorithm

- \( T = \{ (a *1 b), (a e c), (a *2 *3 *4 d), \\
(b *5 c), (b *6 *7 f d), (c *8 d) \} \)

is generable from tree N:

- Each leaf in N is a terminal node in T, e.g. a, b, and d.
- A terminal node in T may not be leaf, e.g. c.

Step 1: Detecting Leaves

- \( T = \{ (a *1 b), (a e c), (a *2 *3 *4 d), \\
(b *5 c), (b *6 *7 f d), (c *8 d) \} \)

- A terminal y is not a leaf iff T has three traces satisfying 
  \(|(x..y)| + |(y..z)| = |(x..z)|\)
- From \(|(a e c)| + |(c *8 d)| = 2+2 = 4 = |(a *2 *3 *4 d)|\), we conclude that c is not a leaf.
Step 2: Parents of Leaves

- $T = \{ (a *1 b), (a e c), (a *2 *3 *4 d), (b *5 c), (b *6 *7 f d), (c *8 d) \}$

- From leaf $a$ and $(a e c)$ in $T$, conclude that $e$ is the parent of $a$ in $N$. Replace $*1$ and $*2$ by $e$.

- From leaf $b$, $(a *1 b)$, and $e$ being the parent of $a$, conclude that $e$ is the parent of $b$ in $N$. Replace $*5$ and $*6$ by $e$.

- From leaf $d$ and $(b *6 *7 f d)$ in $T$, conclude that $f$ is the parent of $d$ in $N$. Replace $*4$ and $*8$ by $d$.

Step 3: Prune Leaves

- $T = \{ (a e b), (a e c), (a e *3 f d), (b e c), (b e *7 f d), (c f d) \}$

- Remove all leaves from traces in $T$.

- Remove all singleton traces from $T$.

- $T = \{ (e c), (e *3 f), (e *7 f), (c f) \}$

- Repeat Steps 1-3 on the smaller $T$ until $T$ becomes empty.
Tracing Ring Networks

- Theorem 3:
  There is an algorithm to trace odd rings.

- Theorem 4:
  There is no algorithm to trace even rings.

Next, we describe the algorithm, of Theorem 3, to trace odd rings.

Tracing Odd Rings

- Step 1: Place first two terminals
- Step 2: Place other terminals
- Step 3: Place nonterminals
Step 1 : Place First Two Terminals

- Example: \( T = \{(a \ b \ c), (a \ *1 \ d), (d \ e \ *2 \ *3 \ c)\} \)

- \( T \) has 5 node identifiers: a,b,c,d,e.
  Of these, 3 are terminals: a,c,d.

- We place the terminal nodes a,c using trace (a b c):

```
  c -- a
```

Step 2 : Place Other Terminals

- Example: \( T = \{(a \ b \ c), (a \ *1 \ d), (d \ e \ *2 \ *3 \ c)\} \)

```
  c -- a -- d
```

- Place the remaining terminal nodes (i.e. d) using its traces with a and c: (a *1 d) and (d e *2 *3 c)
Step 3: Place Nonterminals

- Example: \( T = \{(a \ b \ c), \ (a \ *1 \ d), \ (d \ e \ *2 \ *3 \ c)\} \)

- Place node b using its trace \((a \ b \ c)\)
- Place node e using its trace \((d \ e \ *2 \ *3 \ c)\)

Tracing Mostly Regular Networks

- A network, where each node has at most one irregular neighbor, is called *mostly regular*.

- Theorem 5:
  There is an algorithm to trace mostly regular even rings.

- Theorem 6:
  There is no algorithm to trace mostly regular networks.
Weak Network Tracing Problem

- Design an algorithm that takes a trace set T and computes a “small” set of networks \(\{N_1, \ldots, N_k\}\) such that
  - T is generable from each \(N_i\) and
  - T is not generable from any other network

- “small” means that k is a constant that does not depend on \(n\), the number of node identifiers in T.

- If \(k\) is allowed to be a function of \(n\), the weak network tracing problem is trivially solvable.

Impossibility of Weak Network Tracing

- Theorem 7:
  There is no algorithm to weakly trace every network.
Conclusions

- To trace any network $N$ in the Internet, a tracer has only two options:
  
  -- Either partition network $N$ into a set of trees, trace each tree (by itself) to compute a subnetwork of $N$, and assemble the computed subnetworks to obtain network $N$

  -- Or compute many networks, each of which can be network $N$, and use additional information to determine which of the computed networks is $N$