Topology Inference Based on End-to-End Measurement

Yuan Le
2009/09/23

Why topology inference?
- To transmit packages from the source to the destination
- To manage the network and increase the performance
- To get a profit in a Peer-to-Peer network
- To find a better server for downloading
- To identify the position of another client

Two Primary Approach

Traceroute-like Measurement
- Use tools based on feedback messages from the internal nodes
- Take a long time to identify a path between a pair of hosts

Network Tomography
- Do not require extra cooperation from the internal nodes
- Utilize the correlations among the observed losses and delays of probing packages.

Traceroute-like Measurement

- Use tools based on feedback messages from the internal nodes
- Take a long time to identify a path between a pair of hosts

Traceroute-like Measurement
- Do not require extra cooperation from the internal nodes
- Utilize the correlations among the observed losses and delays of probing packages.

Traceroute-like Measurement

TRACEROUTE-LIKE
- Do not require extra cooperation from the internal nodes
- Utilize the correlations among the observed losses and delays of probing packages.

NETWORK TOMOGRAPHY
- Do not require extra cooperation from the internal nodes
- Utilize the correlations among the observed losses and delays of probing packages.

Solution for redundancy traceroutes
- Select the traceroute target that can give us more information about the routers
  - Random Probe: The server select the target randomly from its unmeasured hosts for a given host (simplest)
  - Longest Path Probe: A longer path may contain more hops, therefore, may contain more undiscovered links. The host select the farthest unmeasured host
  - Max - Delta Probe: Denote the Euclidean distance between host a and b is $E(a,b)$, and the length of current shortest path between a and b is $D(a,b)$, the host select the target has the maximum $\Delta(a,b) = D(a,b) - E(a,b)$

Traceroute-like Measurement

- Use tools based on feedback messages from the internal nodes
- Take a long time to identify a path between a pair of hosts

Traceroute-like Measurement
- Do not require extra cooperation from the internal nodes
- Utilize the correlations among the observed losses and delays of probing packages.

Solution for redundancy traceroutes
- Select the traceroute target that can give us more information about the routers
  - Random Probe: The server select the target randomly from its unmeasured hosts for a given host (simplest)
  - Longest Path Probe: A longer path may contain more hops, therefore, may contain more undiscovered links. The host select the farthest unmeasured host
  - Max - Delta Probe: Denote the Euclidean distance between host a and b is $E(a,b)$, and the length of current shortest path between a and b is $D(a,b)$, the host select the target has the maximum $\Delta(a,b) = D(a,b) - E(a,b)$
Traceroute-like Measurement

- Traceroute-like
  - Router-level path
  - Internet Control Message Protocol (ICMP)
  - Send a series of IP datagram with increasing TTL
  - Problem: High package redundancy
  - Problem: Anonymous router
    - Do not return the ICMP error messages
    - Return ICMP messages only when load is light
    - Discard ICMP messages
    - All we can know is there is an anonymous router

Solution for the anonymous routers

- Treat each occurrence of an anonymous router as a unique one. However, huge inflation and inaccuracy.
- Merge the anonymous router to minimize the number. However, expensive in time complexity. $O(N(N+n_i+n_i^2/2))$

Solution for the anonymous routers

- To reduce the time complexity, relax the consistency constraint, allow some inconsistent merging.
  - Isomap Merging Algorithm
    - Merge anonymous routers based on their multidimensional Euclidean distance.
    - $O(N + n_i + n_i^2)$ time complexity
  - Neighbor Matching Algorithm

Isomap Merging Algorithm

- Isomap
  - To estimate point coordinates in a multidimensional space with an incomplete distance matrix as input.
  - Steps
    - Constructs a neighborhood graph on top of the points.
    - Computes pairwise shortest-path distances in the graph.
    - Applies MultiDimensional Scaling to the complete distance matrix

Isomap Merging Algorithm

- Merging Algorithm
  - Initial Pruning
  - Construction of distance matrix
  - Coordinate estimation
  - Router merging
To reduce the time complexity, relax the consistency constraint, allow some inconsistent merging.

- Isomap Merging Algorithm
- Neighbor Matching Algorithm
  - Merge the pairs of anonymous routers which share at least one neighbor (known router or host), and do not appear in the same traceroute path.
  - O(n^2) time complexity

Focus on the logical topology instead of physical topology.

Use tree topology to infer the topology from one source to a set of destinations.

The procedure of inferring the topology is to construct a tree with a root node as the source host and a set of leaf nodes as the destination.

Based on some assumptions.
  - The link states are independent from link to link
  - The links are stationary

Basic Idea
  - Suppose a logical path connects a source node S and a destination node D, through the internal node I, then sometimes the outcome variable from S to D can be expressed by the outcome variable from S to I and I to D.

Examples of such kinds of measurement
- Link Loss
  - L(S,D) = L(S,I) * L(I,D), L(,) = 0, 1
- Link Utilization
  - U(S,D) = U(S,I) * U(I,D), U(,) = 0, 1
- Link Delay
  - T(S,D) = T(S,I) + T(I,D), T(,) ≥ 0

Basic Idea (2)
- For a given source node S, and two destination node D1 and D2, which share the link from S to a internal node I, if we define the distance between two node based on the outcome variable we showed before, then the distance from S to I, denote as shared path length, can be computed with the outcome variable from S to D1 and S to D2.
- Examples
  - Link Loss or Utility
    - Denote d(A,B) = Prob[L(A,B) = 1], then d(S,I) = d(S,D1)d(S,D2)/d(S,D1,D2)
  - Link Delay
    - Denote d(A,B) = variance of T(A,B), then d(S,I) = covariance of T(S,D1) and T(S,D2)
**Tomography Measurement**

- Problems
  - Not always additive
  - The joint distributions of the outcome variables are not given
  - How to construct the tree by using these distance and share path length
  - How to update the tree when a new node joins

---

**Additive Metrics**

- Let $T(s,D) = (V,E)$ be a logical routing tree with the source node $s$ and destination nodes $D$, we say $d$ is an additive metric on $T(s,D)$ if:
  - $0 < d(e), \forall e \in E$
  - $D(i,j) = \sum d(e), \forall e \in P(i,j), \forall i,j \in V$

---

**Tomography Measurement**

- Examples using Additive Metrics
  - Loss-based additive metric
    - $d_i(e) = -\log \alpha_e, \forall e \in E, \alpha_e$ is the success rate of edge $e$
    - $d_i(s,D) = -\log \Pr[\text{Li}(s,D) = 1]$
    - $d_i(s,D_1,D_2) = d_i(s,D_1) + d_i(s,D_2) - d_i(s,D_1D_2)$
  - Delay-based additive metric
    - $d_i(e) = \text{var}(T_e), \forall e \in E$
    - $d_i(s,D) = \text{var}(T_D)$
    - $d_i(s,D_1,D_2) = \text{cov}(T_{D_1}, T_{D_2})$

---

**Tomography Measurement**

- Problems
  - Not always additive
  - The joint distributions of the outcome variables are not given
  - How to construct the tree by using these distance and share path length
  - How to update the tree when a new node joins

---

**Tomography Measurement**

- Assume source node $s$ send $n$ probes to a set of destination nodes $D$, for any probed node $i$ in $D$, let $T_i(t)$ be the measured delay of the $t$-th probe from $s$ to $D_i$, with $T_i(t) = \infty$ means $D_i$ does not receive the $t$-th probe. Then,
  - $L_i(s,D) = \begin{cases} 1, & T_i(t) < \infty \\ 0, & T_i(t) = \infty \end{cases}$
  - $U_i(s,D) = \begin{cases} 1, & T_i(t) - \min[T_i(t)] < \varepsilon \\ 0, & T_i(t) - \min[T_i(t)] > \varepsilon \end{cases}$

---

**Tomography Measurement**

- Problems
  - Not always additive
  - The joint distributions of the outcome variables are not given
  - How to construct the tree by using these distance and share path length
  - How to update the tree when a new node joins
**Rooted Neighbor Joining Algorithm**

- **Input:** source s, Destinations D, \( d_s(D) \) and \( d_s(D_i, D_j) \), \( \Delta > 0 \)
- **V = {s} \cup D, E = \phi \)
- Loop:
  - Find \( D_i, D_j \) in D with the largest \( d_s(D) \). Create a logical node \( D_f \) as the parent of \( D_i \) and \( D_j \). Then,
  - \( D = D \setminus \{D_i, D_j\} \)
  - \( V = V \cup \{D_f\} \)
  - \( E = E \cup \{(D_f, D_i), (D_f, D_j)\} \)
  - \( +\ d_s(D_f, D_i) = d_s(D_i) - d_s(D_i, D_j) \)
  - \( +\ d_s(D_f, D_j) = d_s(D_j) - d_s(D_i, D_j) \)
  - For every \( D_k \) in D that \( d(D_i, D_j) - d(D_i, D_k) \leq \Delta/2 \), do
    - \( D = D \setminus D_k \)
    - \( E = E \cup \{(D_f, D_k)\} \)
    - \( +\ d_s(D_f, D_k) = d_s(D_k) - d_s(D_i, D_j) \)
  - For each \( D_k \) in D, compute:
    - \( d_s(D_k, D_f) = \frac{1}{2} \left[ d_s(D_k, D_i) + d_s(D_k, D_j) \right] \)
    - \( D = D \cup D_k \)
    - \( +\ d_s(D_f, D_k) = d_s(D_k) \)
  - If \( |D| = 1 \) then for \( D_k \) in D, \( E = E \cup \{(s, D_k)\} \)
  - Else, goto loop

**Problems**

- Not always additive
- The joint distributions of the outcome variables are not given
- How to construct the tree by using these distance and share path length
- How to update the tree when a new node joins

**Dynamic Tree Topology**

**To add a new destination node j to the routing tree \( T = (V, E) \) via an existing node \( k, \Delta \) is the estimated minimum link length. Denote \( f(k) \) as the parent of \( k \) on the original tree.**

**Add_node(T, k, j, \Delta)**

- \( f(k) \) is a leaf node
  - Create a node \( p \) as the parent of \( k \) and \( j \)
  - \( V = V \cup \{p, j\} \)
  - \( E = E \cup \{(f(k), p), (k, p), (p, j)\} \setminus \{(f(k), k)\} \)
- \( f(k) \) has l children \( c_1 \ldots c_l \)
  - Select a destination node \( d_i \) descended from \( c_i \)
  - Measure \( d_s(s, d_1, d_2) \) and \( d_s(j, d_i) \) for \( i = 1 \ldots l \)
  - Find \( d_i \) with the largest \( d_j(j, d_i) \)
  - **Case (a):** \( d_j(s, d_1, d_2) - d_j(j, d_i) \geq \Delta/2 \)
    - (j will be a sibling of k on the new tree.)
    - Create a node \( p \) as the parent of \( k \) and \( j \)
    - \( V = V \cup \{p, j\} \)
    - \( E = E \cup \{(f(k), p), (k, p), (p, j)\} \setminus \{(f(k), k)\} \)
  - **Case (b):** \( |d_j(s, d_1, d_2) - d_j(j, d_i)| < \Delta/2 \)
    - (j will be a child of k on the new tree.)
    - \( V = V \cup \{p, j\} \)
    - \( E = E \cup \{(k, j)\} \)
- **Case (c):** \( d_j(s, d_1, d_2) - d_j(j, d_i) \geq \Delta/2 \)
  - (j will be a sibling or descendant of \( c_i \) on the new tree.)
  - Execute \( \text{add_node}(T, c_i, j, \Delta) \)
**Remove_node** (T, j)

If k is a leaf node:
V = V \ {j}, E = E \ (f(j), j)).

If f(j) has only one child c left:
V = V \ f(j),
E = E \ [(f(f(j)), f(j)), (f(j), c)) \cup (f(f(j)), c)].