University of Alberta

NETWORK TOPOLOGY INFERENCE WITH END-TO-END UNICAST MEASUREMENTS

by

Amir Malekzadeh

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Spring 2013
Edmonton, Alberta

Abstract

Network tomography is the problem of discovering the delay and loss rate of the internal links of a network, assuming the internal nodes are not cooperating. The first step to solving this problem is finding network topology. Well-known tools such as *traceroute* solve this problem, however they depend on cooperation by the internal nodes.

This thesis studies the problem of topology identification without relying on the cooperation of the internal nodes of the network. Our work is based on a probing method called the *sandwich* method. We suggest a novel probing scheme called TSP which is based on end-to-end unicast delay measurements and combines the ideas of *sandwich* and *traceroute*. We also develop two topology inference algorithm to find the topology of the network. One of the algorithms uses *sandwich* data and the other uses TSP.

Our simulation-based experiments show that TSP improves the topology identification process substantially compared to previous methods.

Acknowledgements

I would like to thank my supervisor, Professor Mike H. MacGregor for his invaluable guidance, encouragement and patience throughout this thesis. I cannot imagine this project done without his incredible support.

I would also like to thank all my friends at University of Alberta for being supportive and making my life easy in graduate school. Lastly, I thank my family whom I can always count on.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

There are many network-based applications and services that need information about the internal condition of the network on which they are operating. Network monitoring, high quality media streaming, peer-to-peer and collaborative environments are examples of services that can benefit from knowledge of network topology, bandwidth, and traffic intensity [1, 2, 3]. High quality media streaming needs the available bandwidth to adjust the transmission mode [4]. Peer-to-peer and collaborative applications can use this information to find peers and communicate with them more effectively [5].

Obtaining the necessary statistical data is not an easy task for the applications. The applications work on the end nodes and a huge amount of important data is only available at the internal nodes. The administrators and owners of the internal nodes e.g., routers, do not have much motivation to share this data with the users of the end nodes [6]. Moreover, it is getting harder to do so as the traffic load through the Internet is getting larger. The routers need to process more packets and there is less resource to spend on unnecessary (from the administrators' point of view) tasks. Also due to security reasons the administrators try to keep the statistical information about the network confidential. For example on a public network e.g., the Internet, routers are often configured not to respond to Internet Control Message Protocol (ICMP) messages to reduce overhead and avoid security issues [7, 8, 9]. These messages are usually used as a source of information about the state of a network node or a packet. Therefore, the applications that use ICMP messages may not work properly with public networks.

Currently most of the well known network topology and behaviour identification tools such as *traceroute* try to get as much help from internal nodes as they can. They send requests and messages to the internal nodes and receive feedback from those nodes. For example, *traceroute* uses ICMP error messages sent by the routers and finds out what routers are on the path between two specific nodes. However, one cannot rely on the cooperation of nodes administered by someone else. Depending on the cooperation of internal nodes, and relying on specific settings and configuration details within them, makes this approach very vulnerable and in many cases limit their use to local networks.

Due to the problems with cooperation-based methods, some researchers have started to work on the problem of inferring internal network structure and statistics without relying on internal nodes [10, 11, 3, 12, 13, 14]. This approach is known as *network tomography*. In this approach we assume that we have access to a number of end nodes in the network, and we need information about paths through network. The first step is to infer the network topology [13]. There are different variations of this problem with different assumptions, such as the number of source and receiver nodes communicating with each other. Various characteristics of the network may be assumed, such as whether or not we can send multicast messages [13, 3].

Network tomography techniques are facing two important challenges. First, how to collect data from the network efficiently and in a way that they do not rely on the internal nodes and do not interfere with their performance. Second, how to infer the network topology and other desired information from the data. To deal with the first challenge there are some probing techniques suggested to sample the network's behaviour. Also depending on the level of our access to the network we can get some information by monitoring some network's devices. For the second part of the problem there are also different suggested approaches such as statistical methods, search algorithms, etc. that try to find out the network's topology.

These solutions have some major limitations. Some of them are useful in only a few cases, and some of them face a trade off between complexity and accuracy. In this work we suggest a new solution to infer network topology. We focus on the case where we have one source and several receivers, and we use unicast end-to-end messages to probe the network. Also we assume the routes in the network are fixed during the time that these messages are sent.

Cotes et al. introduced a method a delay-based unicast probe called the sandwich probe [13] which collects data from the network and they use a complex method to infer logical topology using the data which faces the trade-off between accuracy and running resources. As our first contribution we propose a light inference algorithm which uses the same data but is much less complex to infer topology.

All the tomography methods that we studied produce logical topology and has very limited information about the internal nodes of the network. Our main contribution is using the idea of *traceroute* to get more information about those nodes. *Traceroute* uses a very elegant idea to infer the path between two nodes, and in this work we tried to adapt that idea to the new limitations (i.e., no ICMP). Sandwich probe is also another elegant idea in the field. We introduce a new probing scheme which combines these two ideas that are previously used to infer network topology. We also suggest a constructive methods to infer the network topology using the information we get from this probes.

This new probing scheme gives us more information about the network than the regular sandwich probe. Therefore we can have more accurate topology inference. The regular sandwich probe and most of the previous probing schemes collect end-to-end data about the paths between pairs of

nodes. Our method on the other hand, collects chunks of data about each step along the path. This additional information not only helps with the topology inference, also has the potential to replace *traceroute*.

The rest of the thesis is organized as follows. In Chapter 2 we talk about the tomography problem more accurately and introduce some major works done and directions of the field. In Chapter 3 and Chapter 4 we introduce our ideas to solve the tomography problem with some new features. Then we talk about how the methods are tested, present the results of our experiments and compare the methods in Chapter 5. Finally we conclude the thesis in Chapter 6.

Chapter 2

Background and Related Work

In this chapter the tomography problem is modelled and some previous works are explained in order to introduce the major directions that researchers in this filed are working on. The methods proposed to solve network tomography problems can be broken into two major steps: data collection and topology inference. During the first step some data about the network is collected by sending probe messages or monitoring the network's events. In the second step this data is used to infer the network topology or other characteristics of the network. There are several ideas in these two steps that we discuss separately.

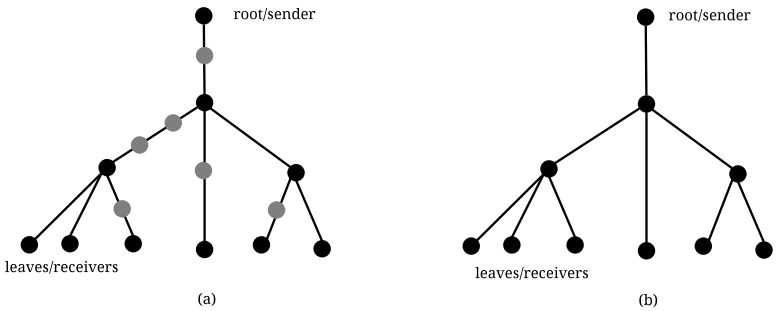
Researchers use different models to model the problem. We use a single model that is close to most of the models and try to explain different algorithms on this same model. In the following we first explain how we model the network and then we get to the algorithms.

2.1 Tree Model

We discuss the problems in which there is a single *source* node and there is a set of *receivers* R. The induced topology of the network is a tree. We name the source also as the *root* and the receivers also as the *leaves*. A tree $\mathcal{T}$ is defined by its set of nodes V and set of direct links L. Among the internal nodes of the tree, i.e., all the nodes except the root and leaves, there are nodes with only one child. Most of the topology inference algorithms do not infer these nodes and only the nodes where some branching happens are important to these algorithms. The topology that is inferred by these algorithms is called the *logical* topology. Figure 2.1 shows the difference between physical and logical topologies.

2.2 Data Collection

The methods proposed to collect tomography data can be classified in different ways. One the most important characteristics of a method is whether or not it relies on cooperation by nodes internal to the network. If the network on which the tomography method is performed is a public network, and we do not have control of the internal nodes (routers, switches, etc.), we can only expect a



(a) shows a tree topology and (b) shows its associated logical topology.

Figure 2.1: Physical and Logical Topologies

limited amount of information from those nodes [7]. On the other hand if we own the network, we can program the internal nodes to collect and report the information we need. Thus, most of the methods which assume cooperative behaviour of internal nodes are useful for administrators of corporate networks.

The second important factor about a method is whether it is passive or active. Passive methods do not interfere with the network's normal traffic – they just act as monitors and try to reach a conclusion using the available characteristics of the traffic. This is only possible if we are able to use the internal nodes to collect data. Active methods send probe messages across the network to elicit information. Active methods can use end-to-end probes so they do not need to rely on help from internal nodes. The downside of active methods is that they impose extra load on the network; these methods try to keep the extra load as low as possible so as not to affect the network's QoS.

The probes sent by active methods can be multicast or unicast messages. Although not all networks support multicast, some methods require multicast messages to collect the data they need [13]. Also there are some methods that try to minimize the size of the data sent with each probe using approaches like network coding [14, 15].

A probe is sent to measure some characteristic(s) of the network. The most popular characteristics are loss rate and delay along a path from one node to another. Some early work that uses delay measurements requires time synchronization among the nodes so the delay can be measured accurately [12]; this may be difficult in some networks [16]. Methods that use loss rate [13] have the advantage that they do not need time synchronization. Their disadvantage is that packet loss rates are very low in a network that is functioning well, so this type of approach is less useful [16].

2.2.1 Sandwich Probe

In 2002, Coates et.al. proposed a method called the *sandwich probe* that measures delay and does not need time synchronization [11]. Assume we have two different receivers in the network, for example n_1 and n_2 in Figure 2.2. The paths from the root to n_1 and n_2 have a common segment, which is the path from the root to n_3. The goal of the sandwich probe is to estimate the delay on this common segment. A single sandwich probe consists of three packets: two small packets named p_1

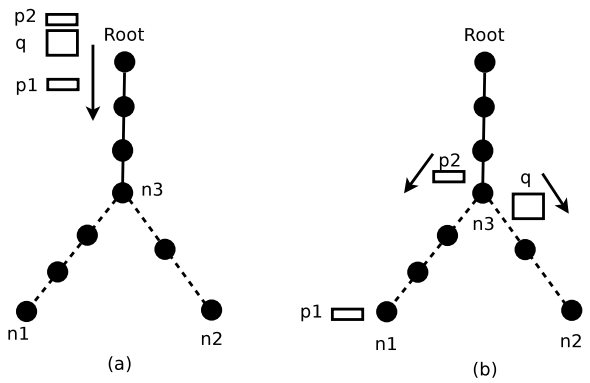


Figure 2.2: Sandwich Probe

and p_2 with the same length, and a large packet named q. As shown in Figure 2.2-a all three packets are sent from the root. The small packets are destined to n_1 and the large one is destined to n_2. First we send p_1. After waiting for a delay of d, we send q, and then we send p_2 right after q. Packet p_2 gets queued behind q in every node from the root to n_3, where their paths diverge (see Figure 2.2-b). As a result, it takes more time for p_2 to be delivered than p_1. Assume the time difference between the delivery of p_1 and p_2 is $d + \Delta d$. Coates et al. [11] show that if we set an appropriate value of d, choose the sizes of packets correctly, and perform enough measurements then the average of Δd is a good estimate of the delay of the path from the root to n_3. For two receivers i and j, we define $x_{i,j}$ as the estimated delay of their common path to the root. One of the important features of the sandwich probe is that the delay differences are measured at a single node, so time synchronization is not required.

2.2.2 Additive Metrics

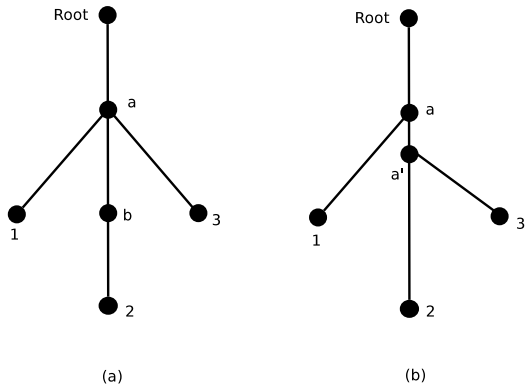
In 2010 Ni et al. suggested a framework to combine information from different probes [17, 3]. For a tree $\mathcal{T}(V, L)$, a function d is an *additive metric* if

a) $0 < d(l) < \infty, \ \ \forall l \in L$

b) $d(i, j) = \sum_{l \in \mathcal{P}(i,j)} d(l), \ \ \forall i, j \in V,$

where $\mathcal{P}(i, j)$ is the path between the pair of nodes i and j. $d(l)$ represents the length of the link l in the tree. For any $i \in R$ let $\rho(i) = d(root, i)$ and for any $i, j \in R$ let $\rho(i, j) = d(root, a_{ij})$ where a_{ij} is the lowest common ancestor of i and j. New additive metrics can be constructed by convex combination of a number of additive metrics, so the information from different probing schemes can be used together. In order to do this, the output of the probes need to get converted to an additive metric. Ni et al. give some examples in their work.

1) *Loss-Based Additive Metric*: Let $0 < \alpha_l < 1$ be the probability that a packet get lost on the link l and $\mathbb{P}(L_i = 1)$ be the probability of a packet get lost on its path to the leaf i, an additive



(a) the actual topology of the network (b) mistaken topology

Figure 2.3: Common Mistake in Previous Methods

metric based on this measure is

$$d_{loss}(l) = -log\ \alpha_l,\ \ \forall l \in L$$

$$\rho_{loss}(i) = -log\ \mathbb{P}(L_i = 1)\ \ i \in R$$

$$\rho_{loss}(i,j) = -log\ \frac{\mathbb{P}(L_i = 1)\mathbb{P}(L_k = 1)}{\mathbb{P}(L_i L_j = 1)}\ \ i,j \in R.$$

2) *Delay-Based Additive Metric*: Let x_l be the measured delay of the link l and x_i be the measured delay for the path from root to the leaf i. The additive metric that is suggested for this measure is

$$d_{delay}(l) = var(x_l),\ \ \forall l \in L$$

$$\rho_{delay}(i) = var(x_i),\ \ i \in R$$

$$\rho_{delay}(i,j) = cov(x_i, x_j),\ \ i,j \in R.$$

3) *Traceroute-Based Additive Metric*: If some cooperation from the routers is possible and there is some result from traceroute messages, another additive metric can be defined using the hop count to each leaf.

One of the main problems of the previous methods is that they collect little information about the internal nodes of the network. Therefore, when the algorithms introduced in Section 2.3 are creating the topology using this information they cannot place the internal nodes accurately. Figure 2.3 shows a very common mistake in these algorithms. A probing scheme that collects more information about the internal nodes can help to prevent such mistakes. In this thesis we suggest such a probing scheme.

2.3 Topology Inference

There are two popular approaches for topology inference. The first is based on maximum-likelihood, and the second is constructive. The constructive algorithms require much less computation than the maximum-likelihood approaches, but usually give less accurate results [16].

2.3.1 Maximum-Likelihood Approach

Vardi was one of the first researchers working on network tomography. He was working on the problem of estimating end-to-end path properties using link measurements. In 1996 he suggested the following equation to model the problem of estimating the number of packets sent between each pair of end nodes [18].

$$\mathbf{Y} = \mathbf{A}\mathbf{X}, \tag{2.1}$$

where $\mathbf{Y}$ is a vector of the measured data, which is the number of packets passing through each direct link during a specific time period. So $\mathbf{Y}$ has an element for each direct link of the network. $\mathbf{X}$ is a vector containing the amount of traffic on each end-to-end path during the time period. $\mathbf{A}$ is a binary routing matrix. Let p be the number of end-to-end paths whose traffic is being estimated and $|L|$ be the number of direct links in the network. $\mathbf{A}$ is a $|L| \times p$ binary matrix in which the element in row i and column j is 1 if and only if the i-th end-to-end path contains the j-th direct link.

Some other researchers used this model for other network tomography problems. In the problem Vardi was working on path characteristics were desired and link characteristics were given. In other problems with the same model, link characteristics (delay, loss, etc.) or the routing matrix may be desired. The measurements are usually path-based or link-based. $\mathbf{Y} = (Y_1, ..., Y_t)'$ is a vector of the measured data , $\mathbf{X} = (X_1, ..., X_s)'$ is the vector of parameters to be estimated and $\mathbf{A}$ is a $t \times s$ binary routing matrix. Usually X_i is a random variable parametrized by some γ_i and $\boldsymbol{\gamma} = (\gamma_1, ..., \gamma_s)'$ is what we need to estimate.

Maximum-likelihood approaches are relatively computationally complex, and they are feasible only for small routing trees. Therefore, there are some methods to deal with the trade-off between complexity and accuracy of these approaches. One of these ideas is to use pseudo-likelihood models. This means dividing the problem described by Eq. 2.1 into several sub-problems, solving each separately, and then combining the solutions to get to the final solution. For the topology inference problem it means finding the logical topology induced by different groups of receivers and then merging the topologies to create the complete logical topology.

Multicast End-to-End Loss

Duffield et al. [13] created a multicast loss probing framework and suggested to use maximum likelihood techniques to find the topology of the network. Assume a Bernoulli loss model for each link l of the network, which means a packet successfully traverses l with the probability α_l. Multicast

probes are sent from the root towards all of the receivers. The result of each probe is described as $X = (X_v)_{v \in V}$ where $X_v = 1$ if the probes have reached v and $X_v = 0$ otherwise. If we run a set of n probes $x = (x^{(1)}, ..., x^{(n)})$ with $x^{(i)} = (X_v^{(i)})_{v \in V}$, the log-likelihood function for this event is defined as

$$\mathcal{L}(\mathcal{T}, \alpha) = log\ P(x|\mathcal{T}, \alpha).$$

For a specific tree $\mathcal{T}$ and a set of probes, $\hat{\alpha}_{\mathcal{T}}$ is a loss probability set that maximizes $\mathcal{L}(\mathcal{T}, \alpha)$.

$$\hat{\alpha}_{\mathcal{T}} = \arg\max_{\alpha} \mathcal{L}(\mathcal{T}, \alpha)$$

Duffield et al. suggested to use an ML classifier to find the topology $\hat{\mathcal{T}}_{ML}$ that maximizes $\mathcal{L}(\mathcal{T}, \hat{\alpha}_{\mathcal{T}})$

$$\hat{\mathcal{T}}_{ML} = \arg\max_{\mathcal{T} \in \mathcal{T}(R)} \mathcal{L}(\mathcal{T}, \hat{\alpha}_{\mathcal{T}}) \tag{2.2}$$

Markov Chain Monte Carlo

Coates et al. suggested a maximum likelihood model based on pairwise similarity function for nodes of the network [19]. For a tree with an end-node set R, there is a $|R| \times |R|$ matrix $\boldsymbol{X}$ of estimated pairwise similarities. Let X_{ij} be a random variable parametrized by γ_{ij} for any pair of end-nodes i, j. A sample $\boldsymbol{x} = \{x_{ij}\}$ of $\boldsymbol{X}$ is measured. Let $p(\boldsymbol{x}|\boldsymbol{\gamma})$ be the probability density function of the random variables, which means $\boldsymbol{X} \sim p(\boldsymbol{x}|\boldsymbol{\gamma})$. $\boldsymbol{X}$ can come from the sandwich probes discussed in section 2.2. Setting $\mathcal{L}(\boldsymbol{x}|\mathcal{T})$ as

$$\mathcal{L}(\boldsymbol{x}|\mathcal{T}) \equiv \sup_{\gamma \in \mathcal{G}(\mathcal{T})} p(\boldsymbol{x}|\boldsymbol{\gamma}),$$

where $\mathcal{G}(\mathcal{T})$ is the set of all possible $\boldsymbol{\gamma}$'s for the tree $\mathcal{T}$, they defined the maximum likelihood tree as

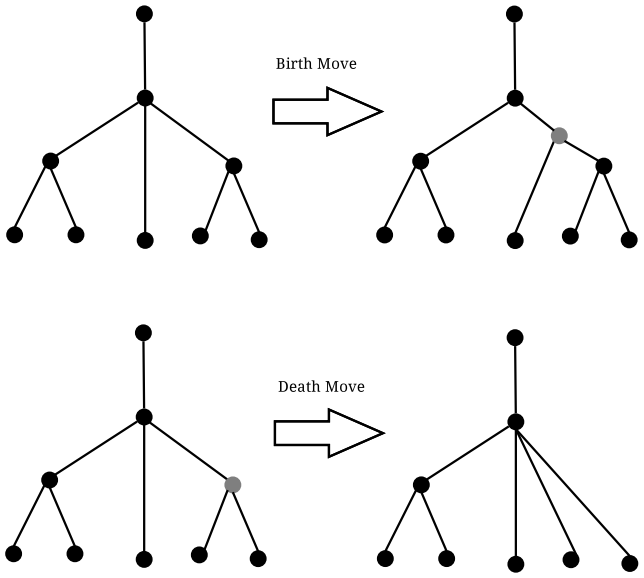
$$\hat{\mathcal{T}}(\boldsymbol{x}) = \arg\max_{\mathcal{T} \in \mathcal{T}(R)}\ log\ \mathcal{L}(\boldsymbol{x}|\mathcal{T}), \tag{2.3}$$

where $\mathcal{T}(R)$ is a forest of all trees with the set of leaves R.

Also we can assume that the random variables X_{ij} are independent so if we let $h_{ij}(x_{ij}|\gamma_{ij} = log\ p(x_{ij}|\gamma_{ij})$ the log-likelihood is

$$log\ p(\boldsymbol{x}|\boldsymbol{\gamma}) = \sum_{i \in R} \sum_{j \in R \setminus \{i\}} h_{ij}(x_{ij}|\gamma_{ij}).$$

If the number of receivers is not very small it is not feasible to search the whole forest $\mathcal{T}(R)$ to find the tree that maximizes Eq. 2.3. Coates et al. suggested to use a random search technique to search the forest more efficiently and they came up with a Markov Chain Monte Carlo (MCMC) algorithm to find an approximate solution for Eq. 2.3. They make a Markov chain with the trees in the forest $\mathcal{T}(R)$. This chain is created using two moves that change a state, i.e., a tree, into another state. These two moves are depicted in Figure 2.4. The first move, which is called the *birth move*, takes a tree, selects a node p with more than two children, inserts a node p' as the new parent of two



In the upper figure the gray node is inserted and in the lower one the gray node is deleted.

Figure 2.4: Markov Chain Moves

current children of p. The *death move* selects a node with two children and deletes it. It is easy to see that these two moves are reversible so the search can come back from mistaken decisions.

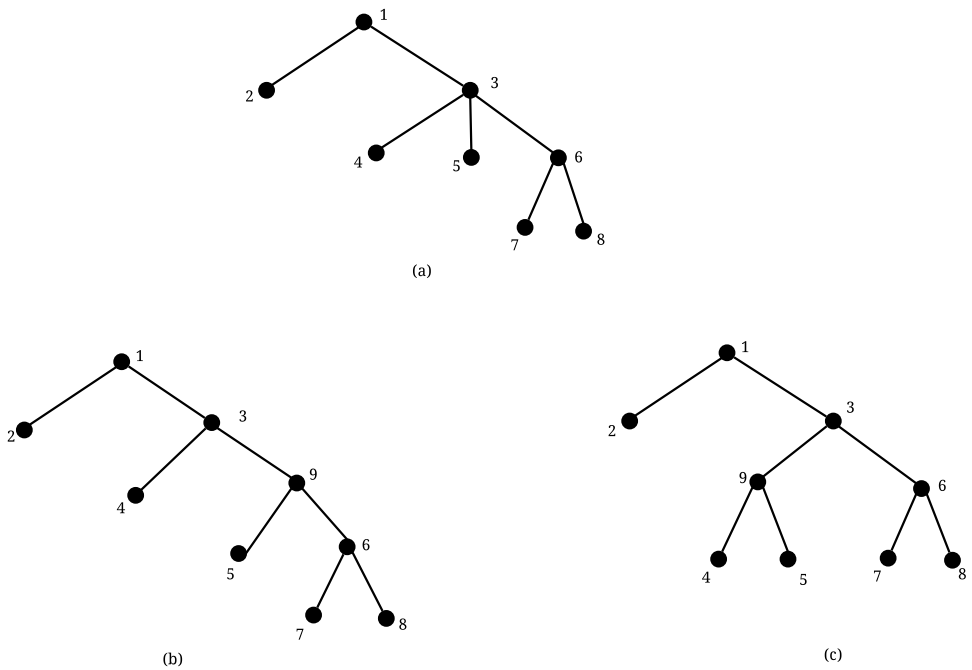
The search keeps making these moves and computes $\mathcal{L}(x|\mathcal{T})$ for each tree it visits. The longer the search is run, the higher chance it has of reaching better trees. The starting point of the search can be any tree or it can be the result of some other methods.

The problem with Eq. 2.3 is that trees with more links can get better scores because they have more degrees of freedom, i.e., they can get overfit. Thus, the search algorithm may end up with a solution with too many links. To resolve this problem Coates et al. tweaked the equation and added a penalty for the number of links:

$$\hat{\mathcal{T}}_\lambda(\boldsymbol{x}) = \underset{\mathcal{T}\in\mathcal{T}(R)}{\arg\max}\ log\ \mathcal{L}(\boldsymbol{x}|\mathcal{T}) - \lambda n(\mathcal{T}) \tag{2.4}$$

where $n(\mathcal{T})$ is the number of direct links, i.e., $|L|$, in the tree $\mathcal{T}$ and $\lambda \geq 0$ is a parameter to control the number of links.

Adding a penalty parameter can solve the overfitting problem, but it creates a new problem. Tweaking λ is an important task and the authors do not suggest an exact method to do so. A probable problem that we faced during testing this method is shown in Figure 2.5. The original tree that the algorithms is tested on is shown in Figure 2.5-a and Figure 2.5-b shows the initial result that MCMC starts with (this tree is given by *ALT* algorithm which will be introduced later in Section 2.3.2). The only change is needed to get from Figure 2.5-b to the correct topology is a *death move* on the node 9. MCMC makes that move but because the penalty is not perfectly set it makes an additional *birth move*, adds the node 9, and ends up with Figure 2.5-c as the final result. And if the penalty is too harsh, obviously the final result will have too few nodes. To conclude performance of MCMC relies



(a) the original tree (b) the initial state of MCMC (c) the final state of MCMC

Figure 2.5: MCMC without a perfect λ

on the precision of λ which makes MCMC hard to use.

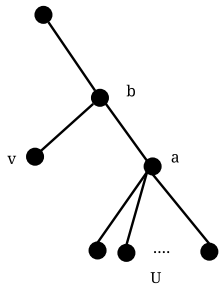
2.3.2 Constructive Approach

Constructive methods gradually build up the network tree. There are several proposed constructive methods; most of them have a similar bottom-up approach:

1. Choose a pair/group of nodes with the highest similarity.
2. Merge the pair/group into a new single node.
3. Update the similarities between the new node and the other nodes.
4. Continue until only one node remains.

The first step to set up such a method is to define a similarity measure for nodes/leaves and this measure has to be monotone if you start from the root and move lower in the tree. Ranasamy et al. [20] suggested such a method in 1999 that uses multicast packet loss measurements. They define the similarity score for a pair of nodes as the probability that a packet is lost on its path from the root to their lowest common ancestor.

In 2002, Duffield et al. [13] extended Ranasamy's algorithm to make it suitable for general, non-binary trees. They defined a function $B(U)$ which is the probability that a packet reaches the lowest common ancestor of a set of nodes U. So a pair of nodes, a and b, have the highest similarity if $B(a, b)$ has the minimum value. They also argue that the minimum of $B(U)$ occurs when U is



$B(U)$ is the probability of a packet reaching a and $B(U \cup v)$ is the same value for b, so $B(U) < B(U \cup v)$

Figure 2.6: Shared Packet Loss

a set of siblings. If U is a set of siblings then for $U' \subseteq U$ we have $B(U') = B(U)$. Figure 2.6 depicts this statement. Their algorithm after finding the pair $U = a, b$ with the minimum $B(U)$, checks all the other leaves u and if $B(U \cup u) = B(U)$ then it adds u to the set U and at the end merges the whole set as a new node. Computing $B(U)$ is not an easy task, so instead they used the joint probability of the packets reaching the leaves to estimate $B(U)$; they also set a threshold for comparing similarities.

Coates et al. [10] proposed another bottom-up method named ALT based on their likelihood approach explained in section 2.3.1.Markov Chain Monte Carlo, which uses unicast measurements to infer binary trees. They defined random variables $X_{ij} \sim p(x_{ij}|\gamma_{ij})$ as pairwise similarities between the nodes. A sample $\boldsymbol{x} = \{x_{ij}\}$ of $\boldsymbol{X}$ is measured (using the sandwich probe). Let $h_{ij}(x_{ij}|\gamma_{ij} = log(p(x_{ij}|\gamma_{ij}))$. The similarity function for a pair of end-nodes is defined as

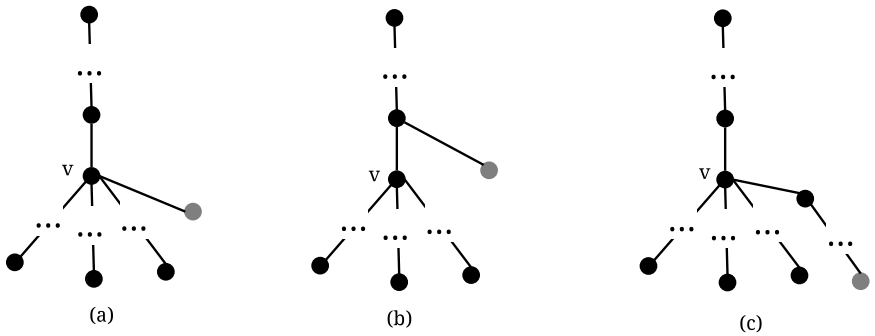
$$\hat{\gamma}_{ij} = \arg\max_{\gamma \in \mathbb{R}} (h_{ij}(x_{ij}|\gamma) + h_{ji}(x_{ji}|\gamma). \tag{2.5}$$

In each round the pair of nodes i and j with the maximum $\hat{\gamma}_{ij}$ are selected and merged into a new node k, which is their parent in the final tree. The next step is to update the similarities. Let R_k be the set of all the leaves that are descendants of k. Initially $R_i = i$ for all the leaves and when i and j are merged into k, $R_k = R_i \cup R_j$. The similarities between the new node k and all the other remaining leaves v is

$$\hat{\gamma}_{kv} = \hat{\gamma}_{vk} = \arg\max_{\gamma \in \mathbb{R}} \sum_{r \in R_k} h_{rv}(x_{rv}|\gamma) + h_{vr}(x_{vr}|\gamma). \tag{2.6}$$

The algorithm keeps performing these steps until only one node remains.

More recently, Ni et al. [3] suggested a dynamic algorithm to add or delete a receiver from the network. Using their method, it is possible to add nodes one by one and construct the whole topology. The leaves are added to the tree in a recursive manner. The recursive insertion algorithm inserts a new node to a sub-tree. Figure 2.7 shows the three different options that this algorithm has when inserting the new node in the node v's sub-tree. The new node is either one of v's siblings, a child of v, or a descendant. If the latter option is chosen the recursive algorithm should run for one of v's children. Some of the probing techniques discussed in Section 2.2 provide depth estimation



(a) the new node is a child of v (b) sibling of v (c) one of v's descendants

Figure 2.7: Node Insertion

for the common ancestors of each pair of leaves, such as the sandwich probe or the additive metrics. The insertion algorithm uses these data to decide among the three options.

The algorithm uses a perset parameter as the shortest possible length of a link called Δ. This parameter is important in deciding between the three cases depicted in Figure 2.7 during the node insertion process. Setting Δ to a proper value is important because if Δ is too large the result tree might end up with too few links and if it is is too small the problem shown in Figure 2.3 might happen. However having a precise estimation of the length of the links can help in the latter case. Similar to MCMC, Ni et al. do not suggest a method to set Δ. We will talk more about this method and Δ in Chapter 3.

2.4 Conclusion

We split topology finding process into two phases, data collection and topology inference and we introduced some major directions in each phase. In data collection we talked about how different characteristics of the network can be sued to get information about the internal topology. The current data collection methods do not give enough information about the internal network topology, so some topology inference algorithms try to use some adjustable parameters to improve their performances.

We categorized the tomography inference algorithms in maximum-likelihood and constructive approaches. Although there have been great improvements in the field, each of these approaches faces different trade-offs and limitations. Maximum-likelihood methods, e.g., MCMC idea by Cotes et al. [11], have to face the overfitting problem. Cotes et al. came up with the idea to add a penalty parameter to their maximization criteria. But tweaking this parameter is a tricky task which is not described how to get done.

Constructive methods are simpler in general. Some of these methods are useful for a limited types of networks, e.g., binary trees, or are less precise than maximum-likelihood methods. Ni et al. suggested a constructive method which outperforms previous constructive methods, but as MCMC

its performance relies on the precision of a preset parameter.

To summarize, the current methods do not collect enough information and use very complex methods that need some preset parameters and it is not clear how these parameters should be set. In these thesis we are trying to improve the data collection methods and suggest a topology inference algorithm to use the data and find the network topology.

Chapter 3

Deciding with Difference Sets (DDS)

In the previous chapter we introduced the tomography problem and discussed some works that are done in this area. As mentioned tomography methods consist of two steps: data collection, and topology inference. One of the data collection methods discussed in Chapter 2 is called sandwich probe which is suggested by Cotes et al. [13]. The sandwich probes gives an estimation for the distance from the source to the common parent of each pair of receivers. Cotes et al. suggested a Maximum-Likelihood approach to infer topology using sandwich data. It is discussed in Chapter 2 that because of the high complexity of this approach it has to deal with the trade-off between accuracy and running resources. In this chapter we introduce a constructive method to find the network topology using the information from sandwich probes. This approach is much less complex than Maximum-Likelihood.

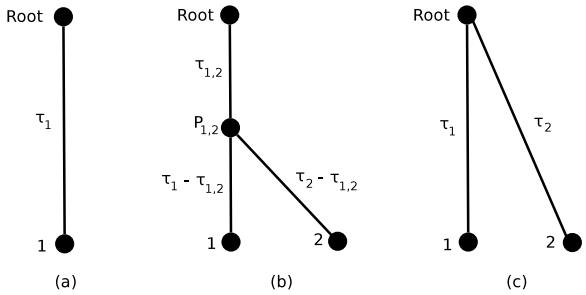
Before we get to the algorithm there are some symbols needed to be explained.

- τ_a: Delay (distance) from the source to the node a.
- $P_{a,b}$: The common parent of destinations a and b.
- $\tau_{a,b}$: Delay (distance) from the source to $P_{a,b}$, or common delay.
- $x_{a,b}$: Estimated value for $\tau_{a,b}$ using sandwich probe.
- x_a: Estimated value for τ_a using sandwich probe. This can be obtained by sending all the three sandwich parts to the same node a.

We first assume that for all leaves a, b we have the exact values of τ_a, τ_b, and $\tau_{a,b}$. We suggest an algorithm to create the tree using this information, then extend our algorithm to handle the real problem in which we just have an estimation for each piece of data.

3.1 Deterministic case

In this section we assume that there is no randomness in our system, so we have exact delay values. We suggest an algorithm to infer the logical topology of a network, given the values of τ_a and $\tau_{a,b}$



(a) Tree with one leaf. (b) $\tau_{1,2} > 0$. (c) $\tau_{1,2} = 0$.

Figure 3.1: Tree after inserting leaves 1 and 2.

for all leaves a, b.

The algorithm starts with a tree containing just the source of the network. This is the root of the tree. We add destination nodes to the tree as leaf nodes, one at a time. This is done in a recursive manner. Here, we explain the steps for the first and second leaves, and then we explain the general recursion step.

3.1.1 Base Case

First Leaf - Inserting the first leaf is trivial. The result is a tree with just one edge, of length τ_1 (Figure 3.1-a).

Second Leaf - We use $\tau_{1,2}$ to tell us where to insert the second leaf. We are working with logical topologies, so there are only two possibilities for a tree with two leaves. If $\tau_{1,2} > 0$ then the result is Figure 3.1-b, otherwise it is Figure 3.1-c.

3.1.2 Recursion Step

We have explained the base case. Now we explain how to add leaf n using recursion. We divide this problem into two cases: a) the root has only one child, b) the root has more than one child. We discuss the first case, and then reduce the second case to the first case.

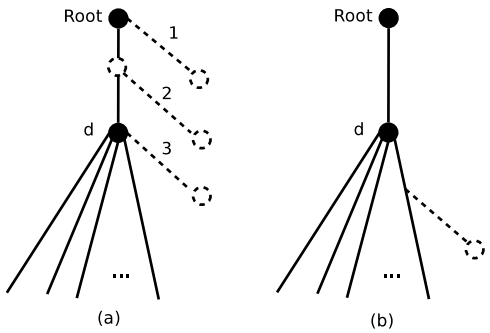
Root has only one child

Figure 3.2 shows the first case. This case also can be divided into two cases, depending on whether or not the following condition holds.

$$\forall a, b < n : \tau_{a,n} = \tau_{b,n} \tag{3.1}$$

Condition 3.1 holds

If Condition 3.1 holds then $P_{a,n}$ is the same node for every existing leaf a. In this case, there are three possible locations for leaf n. These are shown by dotted lines in Figure 3.2-a. The location for leaf n is chosen as follows:



Dotted lines show the possibilities for leaf n.

Figure 3.2: The recursion step.

If $\forall a : \tau_{a,n} = 0$, select location number 1.

If $\forall a : 0 < \tau_{a,n} < \tau_d$, select location number 2.

If $\forall a : \tau_{a,n} = \tau_d$, select location number 3.

In the first case, the delay from the source to $P_{a,n}$ is zero, so $P_{a,n}$ must be the source node. In the second case, the delay from the source to $P_{a,n}$ is less than the delay from the source to d, so $P_{a,n}$ must be higher in the tree than d. In the third case, the delay from the source to $P_{a,n}$ matches the delay from the source to d, so we assume that d is the parent of a and n.

Condition 3.1 does not hold

Here, leaf n *does not* have the same parent as all the other nodes, so it must be somewhere in the sub-tree of node d (Figure 3.2-b). Node d has several children, and each of these children has several leaves in their sub-trees. The measurements used to infer the tree are the delays from pairs of nodes to their common parent, so only the root can have one child; all other internal nodes must have multiple children. If leaf n is in the sub-tree of d and d is not n's parent, the common parent of leaf n and all the other leaves in the sub-tree would be d, except for leaves in the sub-tree of one of d's children. For example in Figure 3.3, d has three children and each child has a sub-tree. Leaf n is in the sub-tree C so $P_{n,a}$ is d for all the leaves a in the sub-trees A and B. Therefore $\tau_{n,a}$ is equal to τ_d for all those leaves. On the other hand, the common parent of leaf n and the leaves in the sub-tree C are deeper than d inside C, so $\tau_{n,a}$ is greater than τ_d for those leaves a. We just need to find the sub-tree C such that for all leaves a in the sub-tree C and all leaves b in any other sub-tree the following condition holds:

$$\tau_{n,a} > \tau_{n,b} \qquad a \in C, b \notin C \tag{3.2}$$

Given this, we can ignore d's other children, consider d as the root and recursively insert leaf n into that sub-tree. Because d has at least two children, the sub-tree has fewer leaves than the parent

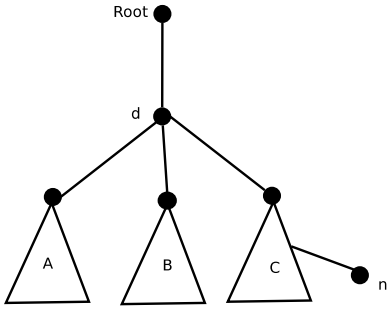


Figure 3.3: Closer look at Figure 3.2-b

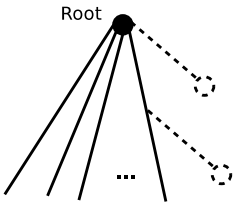


Figure 3.4: Root has more than one child

tree, so the recursive algorithm eventually reaches the base case of a tree with one leaf.

Root has more than one child

Now we discuss the case where the root has more than one child (Figure 3.4). Here, if leaf n branches out at the root, we should have $\forall a < n : \tau_{a,n} = 0$. Otherwise, similar to section 3.1.2-b we need to find the sub-tree of the root whose leaves have higher common delay with leaf n than the other leaves. Then we ignore all the other children of the root and insert leaf n.

3.2 Stochastic case

In reality there is no such thing as the exact delay from a source to a destination. This is a measured, stochastic value, so there will always be some randomness. We have to use the estimated values obtained from several measurements with sandwich probes. The algorithm we propose is a slight variation of the one for the deterministic case.

What is the problem with just re-using the previous algorithm? In Figure 3.5 suppose we are inserting leaf n into the tree and we are at the point where we should decide if the new leaf should be attached at node d, somewhere on the edge between d and its parent, or in one of the sub-trees of d's children. If $\tau_{a,n}$ for all leaves a in d's sub-tree is equal to τ_d, we say n is attached to d. If $\tau_{a,n}$ is less than τ_d, then n is on a branch on the edge between d and its parent. If $\tau_{a,n}$ for all leaves a in the sub-tree of one of d's children is greater than τ_d, leaf n should go in that sub-tree. However, in the stochastic case, it is very unlikely that the values will be exactly equal, ruling out the first

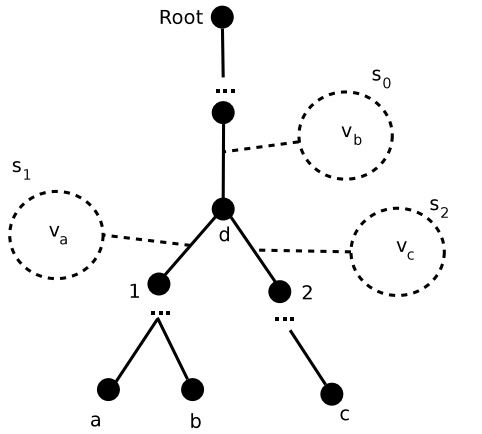


Figure 3.5: The system of delay differences

possibility. For the other two cases, natural variations in the network due to changing loads could be large enough to cause the algorithm to make the wrong decision. In general, we would like to use as much data as we can to reduce errors in tree construction.

During insertion of a new leaf, when we are deciding between the sub-trees of a node d (e.g., node d in Figure 3.5) we build a collection of $C+1$ sets, where C is the number of children of d. We number the sets $s_0, s_1, ..., s_c$, Set s_0 holds information for the edge from d to its parent. Assuming d's children are numbered from c_1 to c_C, the other sets hold information for the edges from d to the child with the corresponding number. That is, there is one set for each child of d.

Each set holds a number of values, where each value is associated with one leaf of the sub-tree rooted at that child. For example, in Figure 3.5, at node d set s_0 is associated with the edge from d to its parent. Sets s_1 and s_2 hold information for the children of d. The values in these sets are calculated as follows.

For each child c_i of d we consider all the leaves a in its sub-tree. If $x_{a,n}$ is greater than x_d we insert the value $x_{a,n} - x_d$ into set s_i (e.g., v_a and v_c in Figure 3.5). Otherwise we insert the value $x_d - x_{a,n}$ into s_0's set (e.g., v_b in Figure 3.5). We assume that each $x_{a,n}$ is normally distributed, so these differences are also normally distributed.

After examining all the leaves, we calculate the average value for each set to select one of the sub-trees. As we noted above, the main problem is the extreme unlikeliness of equality. If leaf n branches out at d, all the average values should be zero, but because we have statistical fluctuations in our measurements, this is very unlikely. To solve this problem we set a threshold and call a value *zero* iff the average of the set is less than its standard deviation. After calculating the averages we check all the sets. If all the averages are zero we attach n to d. If the average of s_0 is not zero we say leaf n should branch out on the edge from d to its parent, and we set the length of the edge from d to its new parent equal to the average value of s_0. If one of the other sets has a positive value, we

select the corresponding child and perform the recursion as before. If the standard deviation is not too high, all the averages except for at most one of them should be zero, according to our definition. If we still have more than one positive value we have to choose the one with the greater ratio of average to variance because, if this ratio is grater, it is more unlikely that the measurement error causes the positive value.

3.3 Analysis

To add a new leaf n to the tree, we start at the root and walk down through the tree. In this process we meet $O(d)$ nodes, where d is the depth of the tree. At each node we add all the leaves in the sub-tree of that nodes into the sets and run some operation of $O(1)$ for each leaf. In conclusion it is $O(dN)$ operation to add a new leaf n, so $O(dN^2)$ to build the whole tree.

During the data collection phase, we send $O(N^2)$ sandwich probes to measure all the $x_{i,j}$ and each packet travels through $O(d)$ nodes. In total $O(dN^2)$ packets are sent.

3.4 Example

In this section we apply the suggested algorithm to a small example of a network to demonstrate the algorithm step-by-step.

The example network has four receivers. Figure 3.6 shows the topology of the network and Table 3.1 contains the exact values of $\tau_{a,b}$ for all leaves a, b. The element in row a and column b shows $x_{a,b}$ for $a \neq b$, and the element in row and column a shows τ_a. These are the values we would have in a deterministic version of the problem. To bring in measurement noise, suppose we have a standard deviation of 0.2 in all our measurements. Table 3.2 shows example measured values corresponding to the exact values in Table 3.1.

Table 3.1: Actual τ values

	1	2	3	4
1	0	2	2	2
2	2	0	6	3
3	2	6	0	3
4	2	3	3	0

Table 3.2: Measured x values

	1	2	3	4
1	4.90	1.90	2.00	2.05
2	1.90	7.85	6.05	2.90
3	2.00	6.05	6.95	3.10
4	2.05	2.90	3.10	5.00

We start with a single node as the root and insert the first leaf into the tree. This gives us the tree in Figure 5.1-a. Inserting the second leaf is also straightforward: $x_{1,2}$ is greater than zero, so we

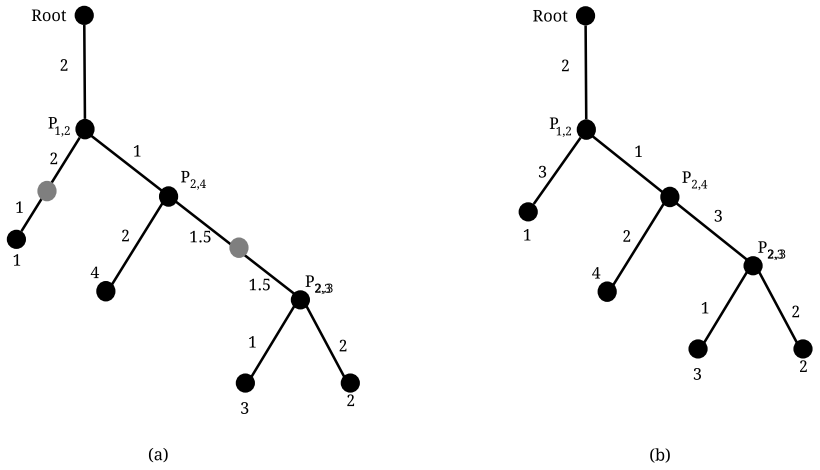


Figure 3.6: The actual tree.

should add a new node $P_{1,2}$ in the middle of the existing edge. The result is shown in Figure 5.1-b. Then we can set the length of the edges using $x_{1,2} = 1.90$, $x_1 = 4.90$, and $x_2 = 7.85$.

Now we add the third leaf. The root has only one child $P_{1,2}$, and condition 3.1 in section 3.1.2 does not hold, so as section 3.1.2-b says we build a collection of three sets at node $P_{1,2}$:

- s_0: Holds information for the edge from $P_{1,2}$ to the root.
- s_1: Holds information for the edge from $P_{1,2}$ to leaf 1.
- s_2: Holds information for the edge from $P_{1,2}$ to leaf 2.

Then we compare $x_{3,a}$ for all the existing leaves a to 1.90, which is the distance of $P_{1,2}$ to the root. For the first leaf, $x_{3,1} > 1.90$, so we insert $x_{3,1} - 1.90 = 0.10$ into s_1. For the second leaf, $x_{3,2} > 1.90$, so we insert $x_{3,2} - 1.90 = 4.15$ into s_2. Only the average of s_2 is greater than its standard deviation, so the third node should be attached to the edge corresponding to s_2, which is the edge from $P_{1,2}$ to leaf 2.

So far, we have the tree in Figure 5.1-c. To add the fourth and last leaf, similarly to the third leaf, we build a collection of three sets at node $P_{1,2}$. With similar reasoning to the case of the third leaf, we find that the fourth leaf should be placed somewhere in the right sub-tree of $P_{1,2}$. As noted in section 3.1.2-b we ignore the rest of the children of $P_{1,2}$, consider $P_{1,2}$ as the root, and recursively add the fourth leaf to the new tree. This time we build a collection of sets at node $P_{2,3}$:

- s_0: Holds information for the edge from $P_{2,3}$ to $P_{1,2}$.
- s_1: Holds information for the edge from $P_{2,3}$ to leaf 2.
- s_2: Holds information for the edge from $P_{2,3}$ to leaf 3.

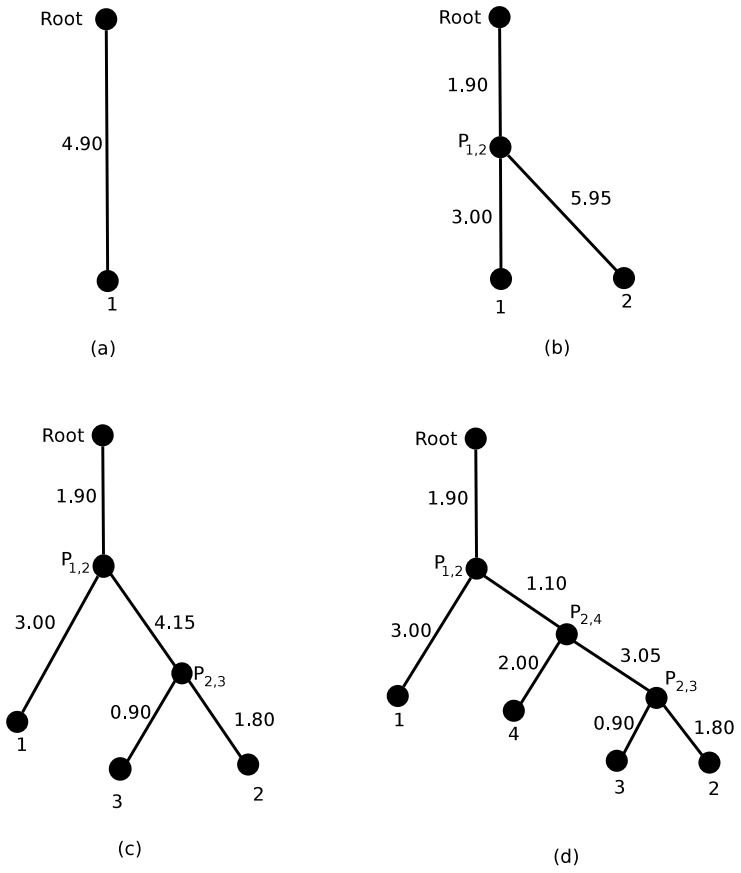


Figure 3.7: Creating the example tree.

Now we compare $x_{4,2}$ and $x_{4,3}$ to 6.05, which is the distance of $P_{2,3}$ to the root. Both are less than 6.05, so we insert $6.05 - x_{4,2} = 3.15$ and $6.05 - x_{4,3} = 2.95$ into set s_0. The only set whose average is greater than its standard deviation is s_0, so the fourth leaf should branch out on the edge from $P_{2,3}$ to its parent. The length of the edge from $P_{2,3}$ to the fourth leaf's parent is equal to the average of s_0. The result is shown in Figure 5.1-d. In this example, even with some noise in the measurements, the tree we have constructed corresponds to the structure of the actual network.

3.5 Conclusion

In this chapter we introduced a constructive topology inference method called DDS which uses sandwich data. In Chapter 2 we pointed a method suggested by Ni et al. [3] which is a similar method, called *Tomo* that inserts the leaves one by one. The main difference between their method and ours is how the thresholds are set. We use the standard deviation of a value as a threshold if the value does not reach its standard deviation we call it a zero. *Tomo* uses a preset threshold Δ as the smallest possible length for a link. Setting the threshold is an important task in this method. If Δ is too large the algorithm will not recognize the links shorter that Δ and may produce a wrong result. So it has to be equal or smaller that the shortest link, and if it is too small then a noisy data may cause extra links created in the resulting tree. Ni et al. [3] show that if the error of each estimation is less than $\frac{\Delta}{4}$ the algorithm will find the correct topology. The main benefit of our method compared to *Tomo* is that it does not have any additional parameter and only uses sandwich data. In Chapter 5 we compared these method to each other as well as to another method which is introduce in the following chapter.

Chapter 4

Traceroute with Sandwich Probe (TSP)

In the previous chapter we suggested a solution to solve the tomography problem which uses the sandwich probing scheme. The probing scheme like the other previous probing schemes give very little information about the internal structure of the network. *Traceroute* is a well-known tool that can collect such information, but it needs the cooperation of the internal nodes so we cannot use it for our problem. But it is built based on a very elegant idea. We tried to alter the idea and apply it to our constraints.

In this chapter we suggest another topology inference method to solve the network tomography problem. This method uses a new probing scheme, called Traceroute with Sandwich Probe (TSP) and gives more information about the network topology. This probing scheme is based on the ideas of the sandwich probe [13] and *traceroute*. A topology inference algorithm is designed to use the information given by the new probing scheme which unlike the previous methods is capable of finding the physical topology of the network rather than the logical topology.

4.1 TSP Probing Scheme

To gain more information about the internal structure of the network without relying on cooperative internal nodes we combine the sandwich probe with the idea of *traceroute* to create a new type of probe. In the following we first explain *traceroute* then introduce TSP.

4.1.1 Traceroute

Traceroute is a network tool that discovers and the path (route) between two nodes in the network and measures the transit delay on each link. It uses a property of Internet Protocol (IP) packets called Time To Live (TTL) which is renamed to hop limit in ipv6. TTL is a parameter in IPv4 header whose purpose is to prevent undeliverable packets from wandering in the network for ever and it sets an upper bound on the time a packet can exist. The sender of a packet sets the TTL to an integer less

than 255 (usually 128 or 64). Then each router that receives the packet reduces this value by one and if it reaches zero the packet gets dropped by the router. This way the network will not get swamped by undeliverable packets.

To discover the details of the path between two nodes a and b, *traceroute* sends several ICMP packets from a to b. The first packet has a Time-To-Live (TTL) value of 1. Thus, it goes only one hop and gets dropped by the first node along the path. This first node returns an ICMP error message to a containing its own IP address. Now node a knows the address of the first node in the path. It can estimate the round trip delay on this first hop by calculating the time between sending its ICMP packet and receiving the error message. Node a repeats this process, incrementing the TTL by 1 until the ICMP packet it sends reaches node b and b sends an ICMP reply message to a. The result is that node a discovers all the nodes along the path to b, as well as the round trip times to each of them.

4.1.2 TSP

As mentioned in Chapter 2 not all routers behave as *traceroute* expects. Some routers are configured not to send ICMP error messages at all, while others send the messages but do not include their IP addresses in them. In our work, we assume that the internal nodes of the network are configured not to reply to ICMP, so we cannot use *traceroute*. But TTL is a part of IP, not ICMP and it is necessary for all the routers to reduce the TTL and drop the packets whose TTL is zero.

We combine the idea of using TTL and create an altered version of the sandwich probe which we introduced in Chapter 2. We know that a sandwich probe estimates the delay of the common path to two nodes. We note that these do not have to be two different nodes. If we send all three packets of the sandwich probe to the same node, and set the TTL of the large packet to a certain value k, the large packet will be dropped after k hops. The delay difference between the two small packets gives the delay along the path, up to the k-th node. Figure 4.1 illustrates an example for $k = 2$. For a particular receiver node i, we define $y_{i,k}$ as the delay from the root to the $k - th$ node along the path to i. Using this version of the sandwich probe, we can perform a process similar to *traceroute*. Instead of ICMP packets, we send a sandwich probe. The time measurements are performed in the end node instead of the starting node. This method gives us one way delays to the nodes along the path, as opposed to the round trip delay times elicited by *traceroute*. The TSP does not give us any information about the addresses of the nodes along the path.

Some hops may have higher delay variance than others, so in order to have reliable measurements, we need to send multiple probes. Therefore if we only have sufficient resources to send a limited number of probes, we try to keep the standard error of our measurements equal, rather than keeping the number of measurements equal for each hop.

Traceroute's output is enough for inferring the topology, but TSP is not because it does not give the IP addresses. In order to infer the topology we need the original sandwich probe and TSP results

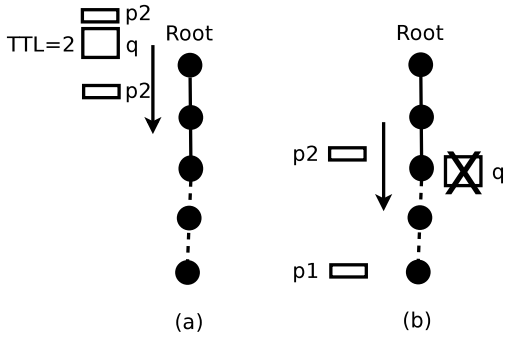


Figure 4.1: TSP Probe

together. The sandwich probe gives delay estimates for the shared segment along the paths to two nodes; we call these estimations $x_{i,j}$ for the nodes i and j.

4.2 Inferring the Topology

Assume we have measured all the values of $x_{i,j}$ and $y_{i,k}$ involving the root and the receivers of interest. We infer the topology of this portion of the network using a constructive method. We start with a tree containing only the root, and try to add leaves to the tree one by one. In the process, we construct the internal topology as well. Assume we have a tree, which is a partial tree of the whole network. Now assume we want to add a new leaf n to the tree (see Figure 4.4). First we find the leaf n' in the tree that maximizes $x_{n',n}$. The paths from the nodes n and n' to the root have a common segment, and $x_{n',n}$ is our estimate of the delay along that segment. We need to find out at which node in the path from n' to the root these two paths separate. The delay from the separation node to the root has to be close to $x_{n',n}$. So we find the number k which minimizes $|x_{n',n} - y_{n',k}|$, which means the k-th node in the path of the root to n' has the closest estimated delay to $x_{n',n}$. In Figure 4.4, $k = 3$. We consider the k-th node as the separation node, and add node n to the tree. As a result the path from the root to the node n is the same as the path to node n' up to the k-th node, and the rest of the path consists of new nodes.

4.3 Analysis

To add a new leaf n to the tree, we have to find the node i that maximizes $x_{i,n}$. This takes $O(N)$ operations, where N is the number of leaves. Then we need to find the closest $y_{i,k}$ to $x_{i,n}$, which takes $O(d)$ operations, where d is the depth of the tree. Thus, the time complexity for building the whole tree is $O(N^2 + Nd)$.

During the data collection phase, we send $O(N^2)$ sandwich probes to measure all the $x_{i,j}$, and we send $O(Nd)$ TSP probes to measure all the $y_{i,k}$. As the time complexity is small, we do not have to measure all the $x_{i,j}$ and $y_{i,k}$ before running the algorithm. We can start the algorithm and do the

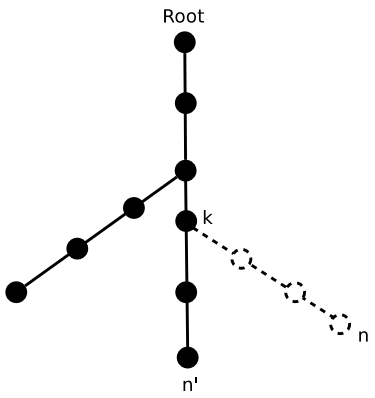


Figure 4.2: Topology Inference Example

measurements when they are needed. This way we can optimize the number of measurements. For example, in Figure 4.4, the delays to the nodes along the path between the root and node k need to be measured only once. In many tomography applications, it is enough to have the logical topology, which means the internal nodes where no branching occurs are not important. In this case we do not need all the $y_{i,k}$. For example, in Figure 4.4 the distances of the dashed nodes are not necessarily required. When we are adding a new node n, and $x_{n',n}$ is the largest $x_{i,n}$ we can use binary search to find the $y_{n',k}$ that is closest to $x_{n',n}$. Therefore, the number of measurements is $O(N^2 + Nlogd)$. This is also the time complexity of building the tree.

4.4 Example

In this section we apply TSP on the same example network we used in Chapter 3 to better compare the two methods. Figure 4.3 shows the network topology we are studying. The input of the inference algorithm includes sandwich probes' output i.e., pairwise common path estimations or $\boldsymbol{X} = (x_{i,j})$ and TSP's output or $\boldsymbol{Y} = (y_{i,k})$. Table 4.1 and Table 4.2 show $\boldsymbol{X}$ and $\boldsymbol{Y}$ calculated from Figure 4.3-a. In Table 4.1 the element in row i and column j shows $x_{i,j}$. In Table 4.2 each row represents the measurements for one leaf. In row i the first element is $y_{i,1}$, the second one is $y_{i,2}$, etc.

Table 4.1: Actual x values of the tree.

	1	2	3	4
1	0	2	2	2
2	2	0	6	3
3	2	6	0	3
4	2	3	3	0

Table 4.2: Actual y values of the tree.

1	2	4	5		
2	2	3	4	6	8
3	2	3	4	6	7
4	2	3	5		

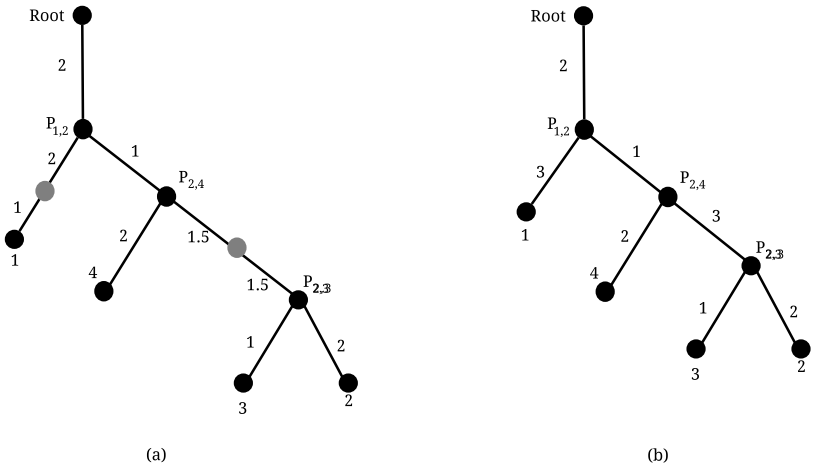


Figure 4.3: The Actual Tree.

Table 4.3 and Table 4.4 show our estimations of Table 4.1 and Table 4.2 that are given by the probing phase.

Table 4.3: Estimated x values of the tree.

	1	2	3	4
1	0.00	1.90	2.00	2.05
2	1.90	0.00	6.05	2.90
3	2.00	6.05	0.00	3.10
4	2.05	2.90	3.10	0.00

Table 4.4: Estimated y values of the tree.

1	1.90	3.8	4.90		
2	1.90	3.05	4.60	5.90	7.85
3	2.00	2.90	4.45	6.10	6.90
4	2.10	3.10	4.90		

We start by adding a single node as the root and the leaves are added one by one to the tree. Inserting the first node gives us the tree in Figure 4.4-a, the number of the nodes and the link delays come from the first row in Table 4.4. The length of the first link is 1.9, which is the first element of the the row. The length of the second link is the difference between the second and the first elements i.e., $3.8 - 1.9 = 1.9$ and the length of the third one is the difference between the fourth and the third elements.

To insert the second leaf we first look into Table 4.3 and find $x_{1,2}$ which is 1.9. Now we should find the node in the path from leaf 1 to the root whose distance to the root is closest to 1.9. This is the first node from the root and is called $P_{1,2}$ in Figure 4.4-b. This means the paths of leaf 1 and 2 separate at this node. Now using the second row of Table 4.4 we create the rest of the path of leaf 2 and we get the tree in Figure 4.4-b. The separation point is $P_{1,2}$ whose distance to the root is 1.9. The distance from next node in the path of the leaf 2 to the root is 3.05, so the length of the next link is $3.05 - 1.9 = 1.15$ and the rest of the path is created with the same process for the first leaf.

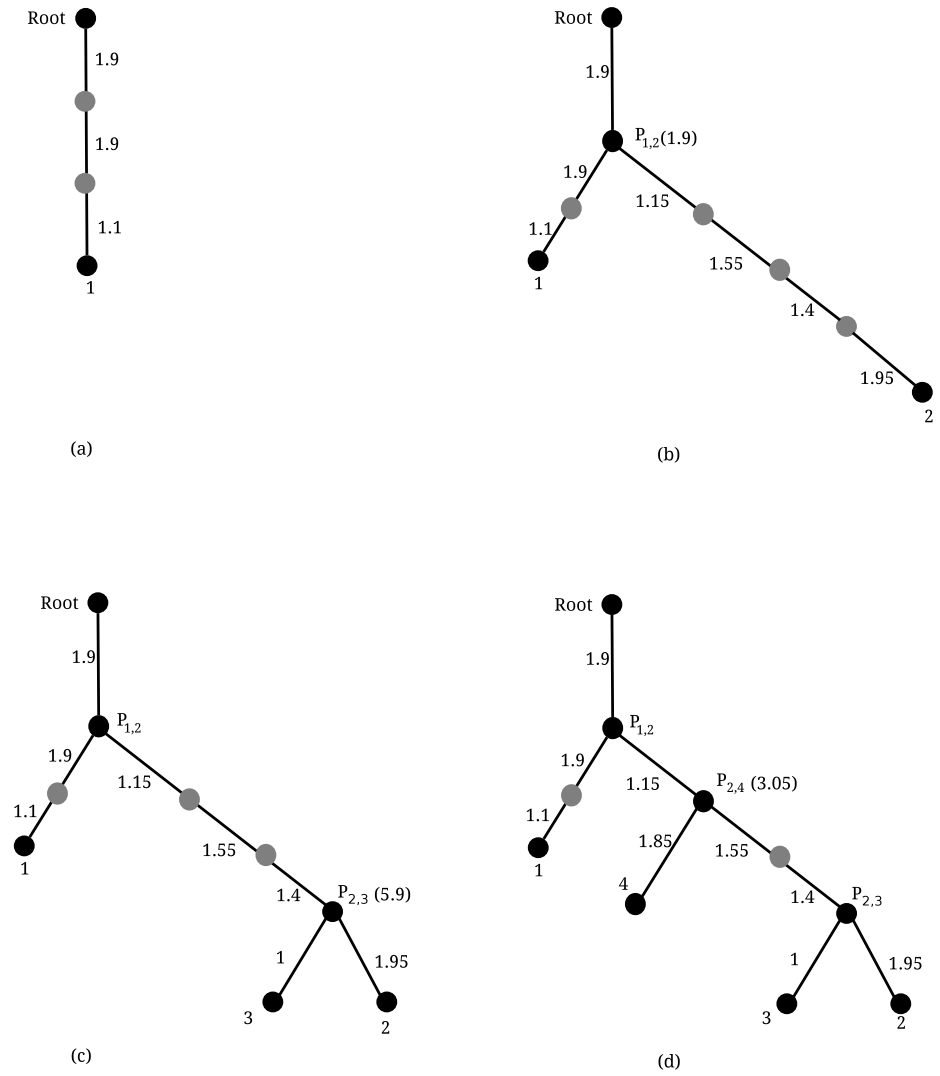


Figure 4.4: Topology Inference Example

For the third leaf, we first have to find the leaf n that maximizes $x_{3,n}$. This is the second leaf. Table 4.3 shows that $x_{3,2} > x_{3,1}$ so the maximum of $x_{3,n}$ is $x_{2,3}$ which is 5.5. The next step is to find the node in the path from the root to the leaf 2 whose distance to the root is closest to 5.5. This node is the parent of the leaf 2 whose distance from to the root is 5.9. This node is called $P_{2,3}$ in Figure 4.4-c and is the separation point of the paths of the leaves 2 and 3. The rest of the path for the leaf 3 contains only one link which can be created using the third row in Table 4.4.

Finally to add the last leaf, the procedure is the same as for the third one. First we find the node n in Table 4.3 that maximizes $x_{4,n}$ which is leaf 3. The node with the closest distance to $x_{4,3}$ is the node called $P_{2,4}$ in Figure 4.4-d; $x_{4,3} = 3.1$ and the root's distance to $P_{2,4}$ is 3.05. According to Table 4.4 the path of the leaf 4 contains one more link and we create that using the fourth row in Table 4.4.

The final result of the algorithm is shown in Figure 4.4-d. The most visible difference between

the result of this method and the result in Chapter 3, given by DDS, is that this method gives the physical topology as in Figure 4.3-a as opposed to DDS which gives the logical topology i.e., Figure 4.3-b.

4.5 Conclusion

We introduced another topology inference method in this chapter named TSP. This method is designed so it can produce physical topologies rather than logical topologies which is a benefit compared to other methods. This capability is achieved by a new probing scheme which combines traceroute and sandwich ideas and collects more information about internal nodes of the network.

This method and other methods we discussed are compared in the following chapter to show how practical they are.

Chapter 5

Experiments

In previous chapters we talked about the tomography problem and suggested some methods to solve it. In this chapter we compare the performance of our methods to each other and to previous methods to find out how useful our methods can be in practice. Also we need to know what are the limitations, benefits, and costs of each method compared to others to be able to choose the right method for each application. We introduced two methods in Chapter 3 and Chapter 4 named DDS and TSP. Ni et al. also proposed a method [3] which was mentioned in Chapter 2. To the best of our knowledge, their algorithm has the best performance among the algorithms that are already suggested for the same class of topology identification problems that we study. Note that there are two variations of this method: *Tomo* and *TRTomo*. We use the first one as the second one uses information from the internal nodes which is not compatible with our problem constraints. These three methods (DDS, TSP, and Tomo) are compared in this chapter.

As mentioned before, tomography methods consist of two steps, data collection and topology inference. We use two different experiment methods: network simulation and model-based simulation. The difference is in the data collection step. For network simulation we used the *OMNET++* framework [21] to create a very detailed network model which covers router queues, propagation delays, etc. to have a realistic environment for our experiments. Network simulation however needs more resources to run and allows fewer experiments than model-based simulation where the delays follow a simple model rather than coming from simulated queues. In the model-based simulation we create randomly generated networks. The simulation is highly abstract and link delays follow a specific random distribution. This abstract model allows us to run many experiments in different situations. We explain each experiment set-up separately and report the results in this chapter. These simulations are used to collect data about test networks and in the next step topology inference algorithms are run separately on the data and the results are compared.

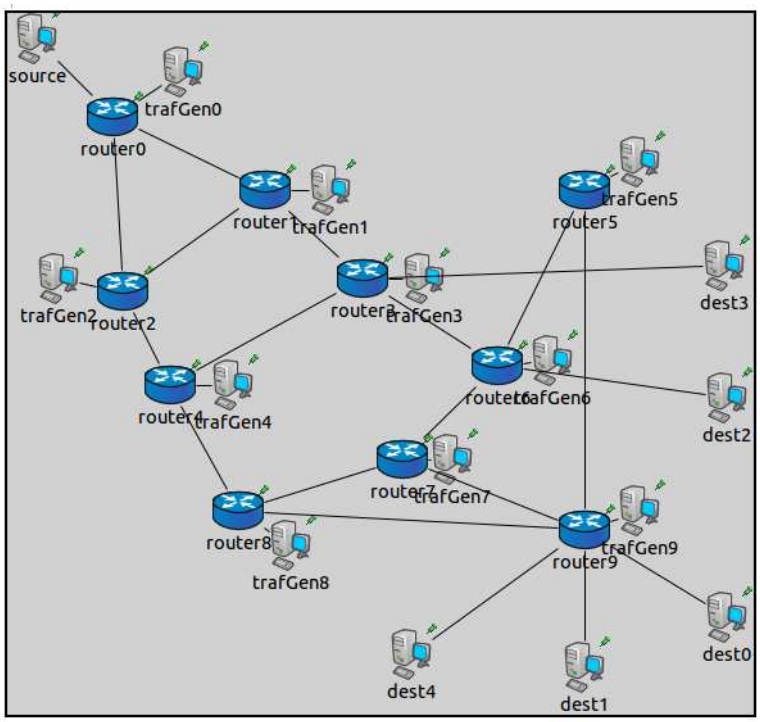


Figure 5.1: An example simulated network.

5.1 Network Simulation

As mentioned above we use *OMNET++* [21] simulation environment to create a realistic network to run our experiments. OMNET++ is an open-source discrete simulation environment written in C++. Its main purpose is network simulation but it was created as a general simulation environment for distributed or parallel systems. We use the *INET* framework [22] which simulates several protocols for wired and wireless networking, such as UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, 802.11. It also supplies models for many useful network elements, e.g., routers, switches, wired and wireless hosts, etc.

5.1.1 Simulated Network Example

Figure 5.1 shows an example simulated network. The network contains four types of nodes: routers, traffic generators, source and destination nodes and network links which are responsible for limiting the bandwidth and latency. The source and destination nodes are the nodes which run the tomography algorithms that are explained in previous chapters. The job of the routers and traffic generators are discussed in more detail in this section.

Routers

The delays that are imposed on the packets while passing through the routers are very important to our tomography methods. Therefore realistic routers are necessary for the simulation. The INET

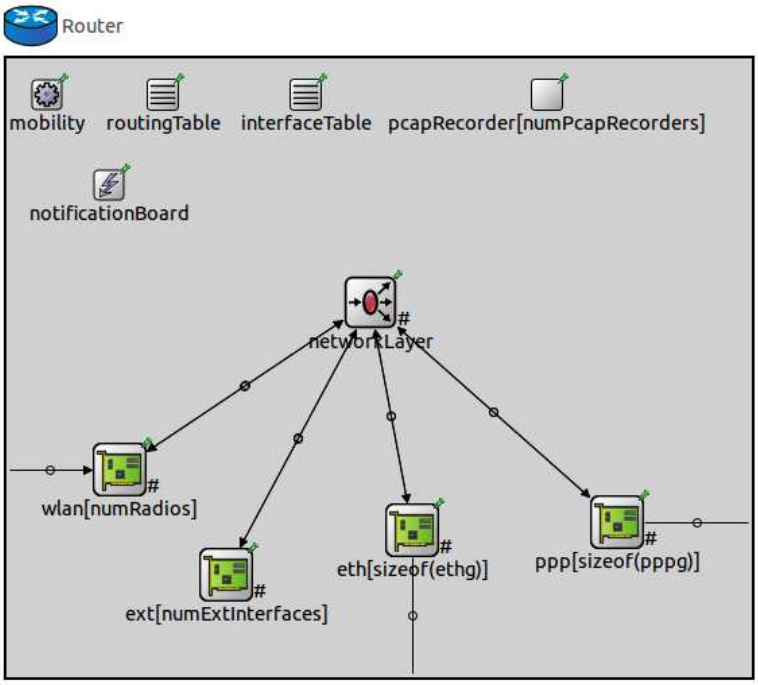


Figure 5.2: Architecture of routers in INET

framework has useful tools for this matter. Figure 5.2 shows the internal modules of an INET router.

Network Layer module handles the IP, ARP, and ICMP protocols. It also performs routing protocols by using the routing table module.

Routing Table handles adding, removing and finding best path for a given destination subnet. This module has interfaces so other modules (i.e., Network Layer) can use it for routing.

The router can have different kinds of **Network Interfaces**: wlan, ethernet, and ppp. The interfaces have internal drop-tail queues to handle arriving and outgoing packets.

Background Traffic

In order to have a realistic environment there has to be realistic background traffic. The data carried as payload by the background traffic is not important but this traffic is the greatest source of randomness in our system so its behaviour should be similar to typical traffic in the Internet. We use a model similar to the model used by the IEEE 802.16 working group for performance simulation of proposed wireless MAC/PHY standards [23]. They use a fairly simple model to generate HTTP/TCP, FTP, voice, and video streaming traffic.

The base of the model is called the *Interrupted Poisson Process (IPP)*, which is a two state process that is run by the traffic generator. This model generates self-similar traffic whose behaviour is close to the traffic on the Internet. A combination of four IPPs (4IPP) has been shown to give good results [24]. Figure 5.3 depicts an IPP; it has an *On* and an *Off* state and when it is on it generates packets at a configurable rate. Transition between the two states is done randomly with

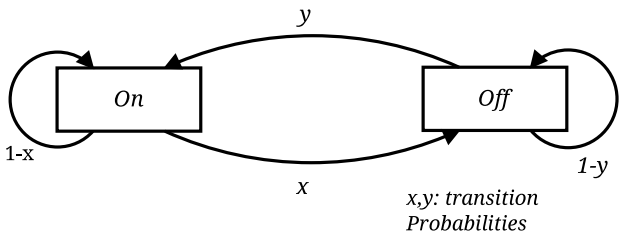


Figure 5.3: IPP Model Framework

the configurable probabilities of x and y. Four IPPs with different parameters are used to generate the traffic. Each IPP's data rate depends on how often it is on and it is configured by x and y parameters. It also depends on how many packets it produces when it is on. In order to achieve a given data rate the sum of packets generated by the four IPPs should be computed.

5.1.2 Test Topologies

The networks that we run the experiments on have to have similar topologies to real networks to show the performance of the methods in the real world. CAIDA has developed a tool called *skitter* [25] to gather information about topology of the Internet. It uses a traceroute-like method and is used to map the AS level topology of the Internet. We took the data measured by running *skitter* daily in 2005 that contains data from over one million nodes. We selected different parts of the large network to run the evaluations.

5.2 Model-based Simulation

There is significant queue processing for the background traffic in the network simulation. As a result, these runs require substantial memory and time, making it difficult to run experiments for large networks and/or run many experiments. We used another form of simulation to cover this problem. Although this is not as realistic as our network simulation, it gives us the opportunity to test our methods on many different cases and larger networks.

In this method instead of using routers and queues and background traffic, we suppose each link in the network has a random delay with a certain mean value. With this assumption there is no need to simulate the queueing of the background traffic. We created 500 random trees each with 30 leaves. Each link in the network has a random delay with a mean value between 0.2ms and 20ms (as observed in our network simulations). In the probing phase when probing a path, we take samples of the link delays along the path and use the sum of the samples. For each path we perform a number of probes and use the average of the results for the topology inference algorithm.

5.3 Evaluation Parameters

Ni et al. [3] suggested two parameters to compare the performance of topology inference methods. These parameters are:

- **Correctness ratio,** which is the ratio of the correctly inferred internal nodes of the network to the total number of the internal nodes. An internal node is considered correctly inferred if there is a node in the inferred topology with the same set of leaves in its sub-tree. A higher correctness ratio means the resulting tree is closer to the actual topology. If this ratio is equal to 1, it means the inferred topology is completely correct.

- **Node ratio,** which is the ratio of the number of the internal nodes in the actual network to the number of internal nodes in the inferred topology. A higher difference between this ratio and 1 means the inference algorithm is less accurate in the number of internal nodes. If the ratio in more (or less) than 1 it means the resulting tree has more (or fewer) internal nodes than the actual network.

5.4 Results

The results of network and model-based simulations are presented separately in this section. Note that the TSP method builds physical topologies as opposed to Tomo and DDS which build logical topologies. In order to compare their correctness and node ratio we have to convert TSP's results to corresponding logical topologies then calculate correctness and node ratios.

5.4.1 Network Simulation Results

We built ten different test networks in the simulation environment and run the three tomography methods on them. For each network we sent 500 probe messages and compared each method's result to the correct topology and computed its correctness and node ratio. As mentioned in Chapter 3 Tomo needs an additional parameter called Δ which is the length of the shortest link. We derived Δ on a test network and used the same value for all the networks.

Table 5.1 shows the results for ten test networks. In terms of the correctness ratio TSP has far better performance than the others in most cases. DDS also outperforms Tomo in this parameter. This can mean Tomo is very dependant on how we set Δ. TSP's node ratio is also better than the other methods on average and in most cases but it has a poor result on the last large network. DDS has the worst performance in this parameter. Among these ten sample networks TSP has inferred two topologies completely correctly and the other two methods could not find any topology completely correctly.

In order to show what kinds of mistakes are made by each method, we compare the results of sample network number 4. Figure 5.4 shows the correct logical topology of this network and

Table 5.1: Network Simulation Results

Sample No.	# of Receivers	Correctness Ratio			Node Ratio		
		TSP	DDS	Tomo	TSP	DDS	Tomo
1	10	1.00	0.86	0.71	0.86	1.29	0.86
2	16	0.78	0.89	0.56	0.89	1.33	0.78
3	16	0.89	0.44	0.22	0.89	1.44	0.78
4	16	0.89	0.67	0.67	0.89	1.44	1.33
5	16	1.00	0.78	0.67	1.00	1.33	0.67
6	16	1.00	0.60	0.30	1.00	1.40	0.90
7	16	0.75	0.87	0.50	0.87	1.12	0.87
8	16	0.73	0.27	0.27	0.73	1.00	0.91
9	20	0.33	0.66	0.58	1.08	1.42	1.16
10	20	0.42	0.33	0.33	0.5	1.42	1.25
Average Error		0.22	0.36	0.52	0.14	0.32	0.20

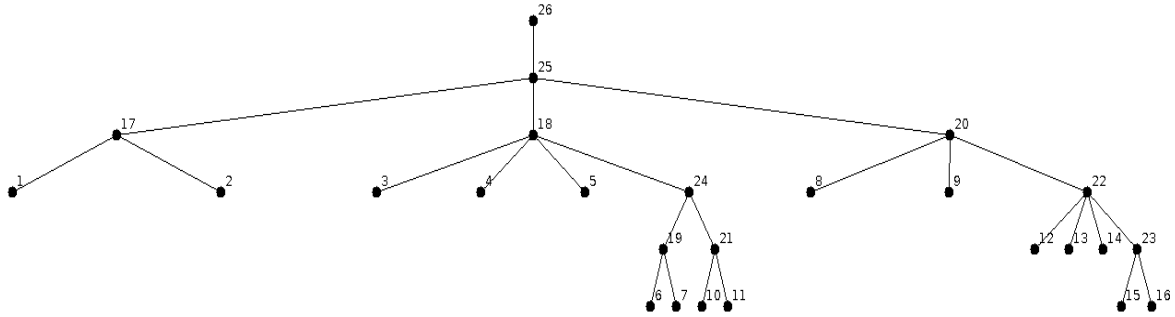


Figure 5.4: Correct Logical Topology of Sample Network No. 4

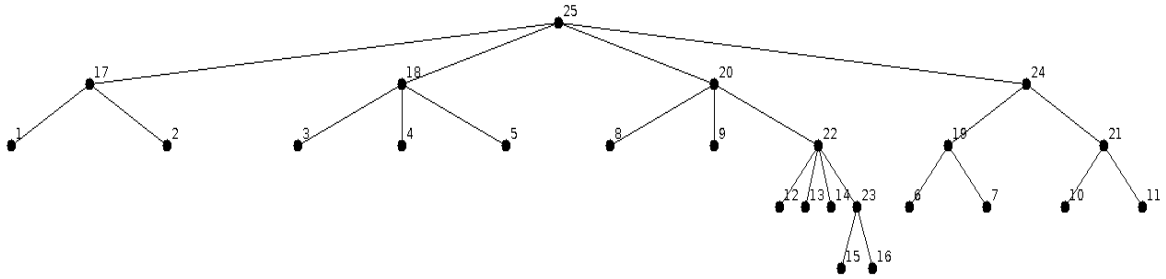


Figure 5.5: Logical Topology of Sample Network No. 4 Built by TSP

Figure 5.5 shows the logical topology returned by the TSP method. The major problem with TSP's result is that node 24 is misplaced. This is happened because when the method inserts nodes 6, it cannot find the correct common parent of nodes 6 and 3 (or 4 or 5) and inserts the node in the wrong place. But after that the nodes that are near node 6 are placed correctly relative to node 6.

Figure 5.6 depicts Tomo's result and shows why its node ratio is higher than 1. In this case Tomo cannot eliminate small measurement errors and adds a few extra links to the network. This is the problem we discussed in Chapter 2 that occurs when a measurement error makes the inference algorithm add a small extra link. DDS's result is very similar to Tomo's and has only one more extra

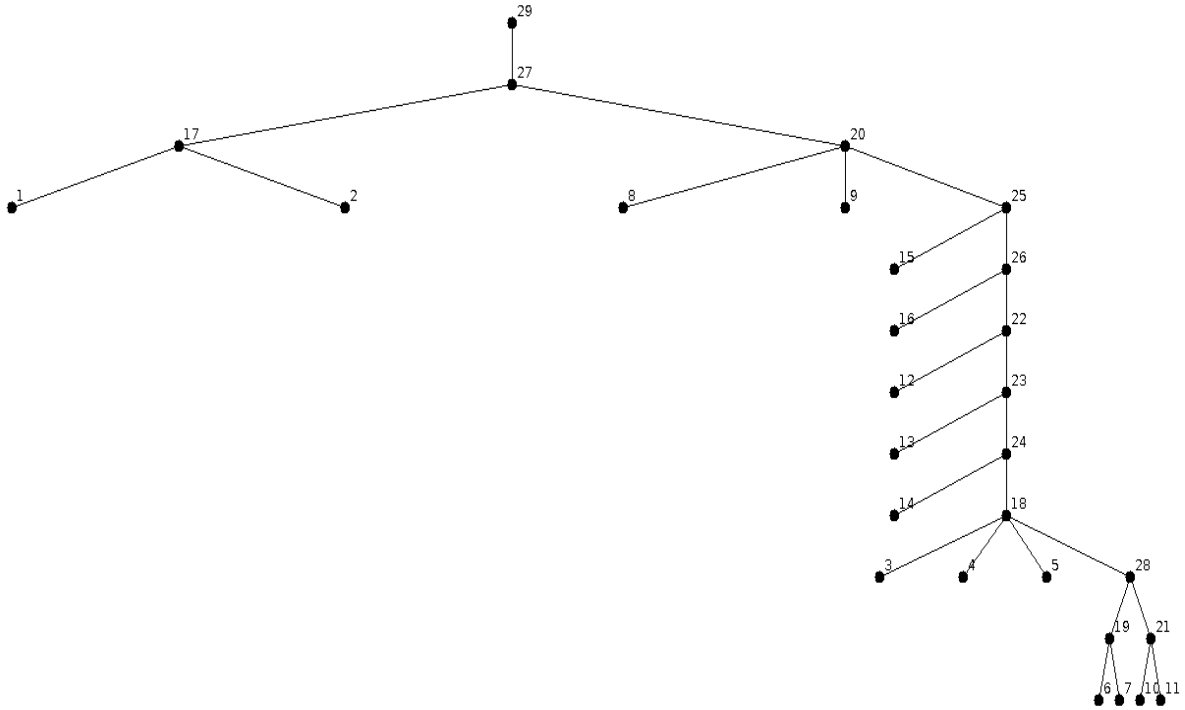


Figure 5.6: Logical Topology of Sample Network No. 4 Built by Tomo

link so we do not present that topology here but it means that DDS and Tomo suffer from similar problems.

5.4.2 Model-based Simulation Results

Figure 5.7 compares the correctness ratio of TSP, DDS, and Tomo considering the number of probes sent. As the number of probes grows, TSP outperforms Tomo by up to 20% higher correctness ratio. DDS shows a poor performance compared to the other two methods and is not able to benefit much from additional data when we send more probe messages which means the standard deviation does not help to detect the measurement errors as much as Δ does.

Figure 5.8 depicts the node ratio of the two methods. As you can see TSP has a weaker node ratio when we send fewer probes, but as more probes are sent the ratio approaches 1. Tomo on the other hand reaches the perfect node ratio quickly but after that its node ratio gets worse. Comparing the two charts we can see that Tomo improves its correctness ratio with more probes but it is not able to improve the node ratio as much. Again DDS has a poor result in this parameter.

5.4.3 Accuracy Limitation

We would very much like to understand whether increasing the number of probes leads us to a perfectly correct topology every time. Unfortunately the answer is no. We found a problem during

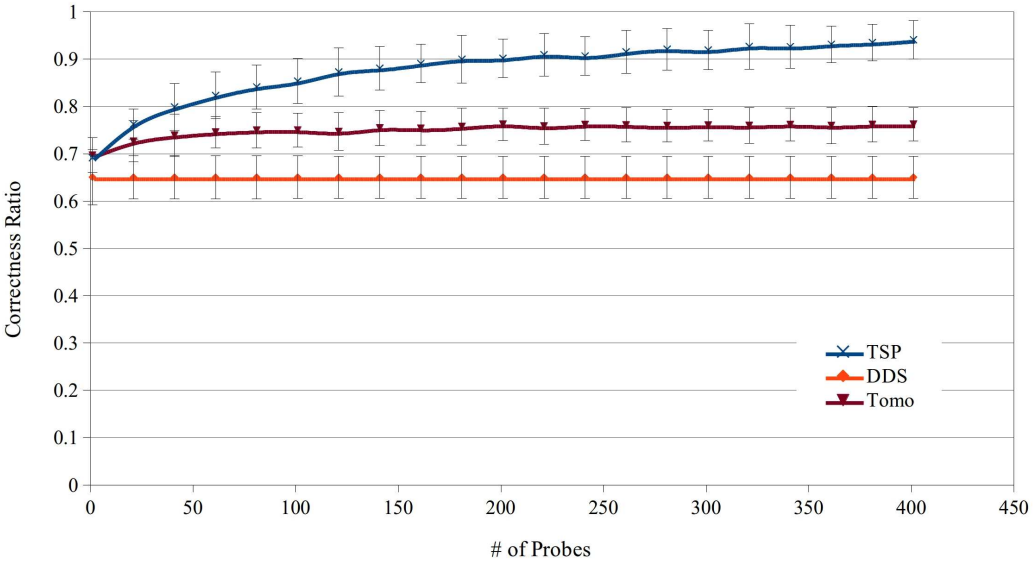


Figure 5.7: Correctness Ratio

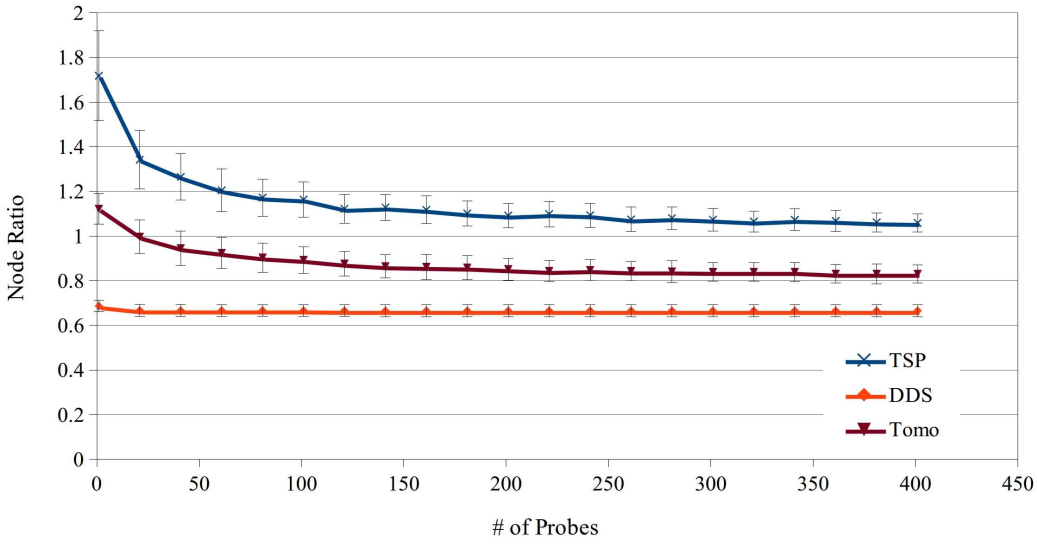


Figure 5.8: Node Ratio

our network simulations which also exists in the Internet. All of the methods that we use are based on an assumption that the delay we measure is monotonic along the path and strictly increasing. However the way the routes handle routing, processing, and dropping messages is more complex

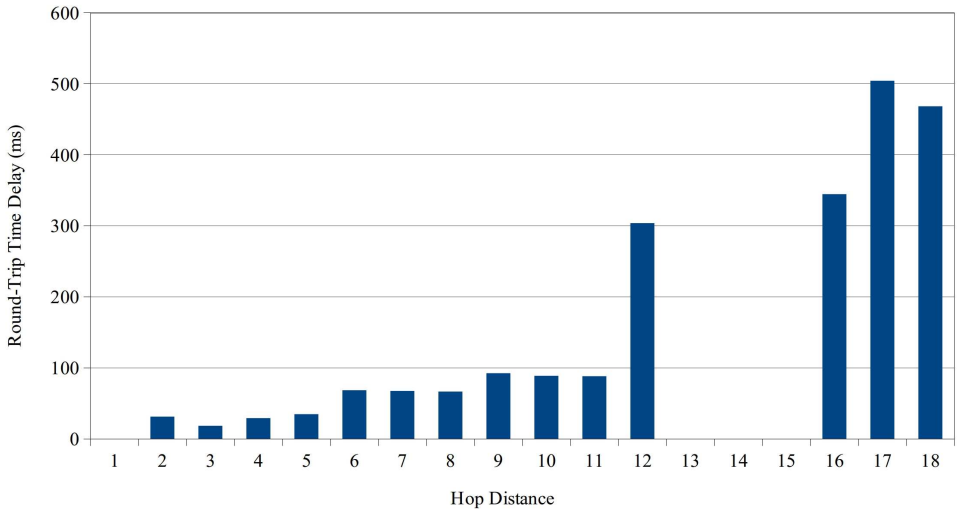


Figure 5.9: RTT delays along a sample 18-hop path.

than our models and that assumption is not always true.

This problem can be seen in Figure 5.9. The figure shows Round-Trip Time (RTT) delays measured using traceroute along a sample path in the Internet. Each hop's delay is shown separately and the hops without a delay are the ones which do not respond to ICMP queries. It is visible that although the delay is growing overall along the path, it is not monotonic and decreases at some hops.

This non-monotonicity causes errors in our methods and our results depend on how often it occurs in the network. We can make our measurements more accurate by increasing the number of probes but this non-monotonicity is in the network's nature, so to face this problem we have to alter the topology inference algorithms.

5.5 Conclusion

We developed two experimental systems to test our tomography methods. The first system is a detailed simulation with a fairly realistic behaviour and the second one has a simple model for propagation delays and is less realistic but is much lighter than the first system and can handle larger networks and a larger number of experiments.

The results of the simulations show that TSP outperforms the other methods in most cases. Comparing DDS and Tomo in different experimental systems gives different results. The reason can be Tomo's dependence on Δ. Setting Δ in model-based simulation is easier than network simulation and Tomo shows better performance with more accurate Δ. The other difference between the two systems is the effect of non-monotonicity we discussed in Section 5.4.3. This problem is not seen in model-based simulation as opposed to network simulation.

We compare computational complexity of TSP, DDS, and Tomo as well as their traffic overhead

in Table 5.2. The table shows that TSP builds the topology more efficiently than the other two methods, but they all run in polynomial time. TSP sends more probe messages that the other two methods thus uses more bandwidth and imposes more traffic on the network. If the network is very busy this can be problematic especially if the probing process lasts too long, because then the network's routing paths might change.

Table 5.2: Comparison of Computational complexities and traffic overhead.

N is the number of receivers and d is the depth of the network.

Method	Computational Complexity	Traffic Overhead
TSP	$O(N^2 + dN)$	$O(N^2 + Nlogd)$
DDS	$O(dN^2)$	$O(dN^2)$
Tomo	$O(dN^2)$	$O(dN^2)$

Beside its better performance, TSP is capable of building physical topologies and this is another advantage compared to other methods.

Chapter 6

Conclusions and Future Work

In this thesis we addressed the network tomography problem, which is the problem of finding information about a network's internal nodes and links. This information is necessary for some network monitoring, peer-to-peer, and collaborative tasks. An important part of the problem, which we worked on, is finding the topology.

The basis of our work is a probing method called the sandwich probe which is suggested by Cotes et al. [13]. We developed two tomography methods called DDS and TSP. DDS is a topology inference algorithm that finds network's topology using sandwich results and TSP is a new probing scheme. Sandwich probes give us limited data about network's internal nodes. Combining the idea of sandwich probe and traceroute we introduced a new probing scheme called TSP that collects step by step data along network paths. We also developed a compatible topology inference algorithm for TSP probing.

In order to test our methods we developed two different simulation systems. The first system is a detailed network simulation that simulates low layer protocols, queueing at the routers, and background traffic. This system is fairly realistic and shows the problems that happen in the real world, but it needs a large amount of process resources to execute. The second system is simulation of a simple model for propagation delays. In this system we assume each link in a network adds a random delay to all passing packets. This simple system can handle larger number of sample networks and larger networks than the first system.

We compared performance of our methods with each other and another method called Tomo which is suggested by Ni et al. [3]. The comparison shows that the TSP method has a better performance in most cases and its results are closer to the correct topologies. In terms of traffic overhead TSP sends more messages than the other two methods in each round of probing which can be a problem if the network is busy. Also TSP has the capability of building physical topologies, but the other two methods do not.

This work can be expanded in a few directions, and one of the most important ones is to look into the problem mentioned in Chapter 5 which is caused by non-monotonicity of measured delays along a network path. Solving this problem may cause significant improvements in our results.

Another possible direction is to use other capabilities of the TSP method. As mentioned in Chapter 4 the TSP method does part of the job of traceroute. We may be able to find more ways to benefit from this possibility. Our methods can also be developed more to handle multiple sources rather than a single source. Rabbat et al. have introduced a method that uses a single source method to find the topology of a multi source system [26].

Also we can optimize the traffic overhead of the methods and reduce the number of probes sent. All the data collected by the probes is not useful, for example in the TSP method during new node insertion we only use data about a part of the node's path. There are other ways to optimize the probing process that are studied by other researchers. Gu et al. have suggested one way [27] that we can try to apply to our methods.

Bibliography

[1] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware 2001*, pp. 329–350, Springer, 2001.

[2] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "Promise: peer-to-peer media streaming using collectcast," in *Proceedings of the eleventh ACM international conference on Multimedia*, pp. 45–54, ACM, 2003.

[3] J. Ni, H. Xie, S. Tatikonda, and Y. Yang, "Efficient and dynamic routing topology inference from end-to-end measurements," *IEEE/ACM Transactions on Networking (TON)*, vol. 18, no. 1, pp. 123–135, 2010.

[4] M. Liška and P. Holub, "Couniverse: Framework for building self-organizing collaborative environments using extreme-bandwidth media applications," in *Euro-Par 2008 Workshops-Parallel Processing*, pp. 339–351, Springer, 2009.

[5] P. Holub, H. Rudová, and M. Liška, "Data transfer planning with tree placement for collaborative environments," *Constraints*, vol. 16, no. 3, pp. 283–316, 2011.

[6] R. Cáceres, N. Duffield, J. Horowitz, and D. Towsley, "Multicast-based inference of network-internal loss characteristics," *Information Theory, IEEE Transactions on*, vol. 45, no. 7, pp. 2462–2480, 1999.

[7] B. Yao, R. Viswanathan, F. Chang, and D. Waddington, "Topology inference in the presence of anonymous routers," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 1, pp. 353–363, IEEE, 2003.

[8] M. Baltatu, A. Lioy, F. Maino, and D. Mazzocchi, "Security issues in control, management and routing protocols," *Computer Networks*, vol. 34, no. 6, pp. 881–894, 2000.

[9] M. H. Gunes and K. Sarac, "Resolving anonymous routers in internet topology measurement studies," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pp. 1076–1084, IEEE, 2008.

[10] R. Castro, M. Coates, and R. Nowak, "Likelihood based hierarchical clustering," *Signal Processing, IEEE Transactions on*, vol. 52, no. 8, pp. 2308–2321, 2004.

[11] M. Coates, R. Castro, R. Nowak, M. Gadhiok, R. King, and Y. Tsang, "Maximum likelihood network topology identification from edge-based unicast measurements," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 1, pp. 11–20, 2002.

[12] N. Duffield, J. Horowitz, and F. Lo Prestis, "Adaptive multicast topology inference," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1636–1645, IEEE, 2001.

[13] N. Duffield, J. Horowitz, F. Lo Presti, and D. Towsley, "Multicast topology inference from measured end-to-end loss," *Information Theory, IEEE Transactions on*, vol. 48, no. 1, pp. 26–45, 2002.

[14] M. Gjoka, C. Fragouli, P. Sattari, and A. Markopoulou, "Loss tomography in general topologies with network coding," in *Global Telecommunications Conference, 2007. GLOBECOM'07. IEEE*, pp. 381–386, IEEE, 2007.

[15] C. Fragouli and A. Markopoulou, "A network coding approach to network monitoring," in *43rd Allerton Conference on Communication, Control, and Computing, Monticello, IL*, pp. 28–30, IEEE, 2005.

[16] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu, "Network tomography: recent developments," *Statistical Science*, pp. 499–517, 2004.

[17] J. Ni and S. Tatikonda, "Network tomography based on additive metrics," *Information Theory, IEEE Transactions on*, vol. 57, no. 12, pp. 7798–7809, 2011.

[18] Y. Vardi, "Network tomography: Estimating source-destination traffic intensities from link data," *Journal of the American Statistical Association*, vol. 91, no. 433, pp. 365–377, 1996.

[19] M. Coates and R. Nowak, "Sequential monte carlo inference of internal delays in nonstationary data networks," *Signal Processing, IEEE Transactions on*, vol. 50, no. 2, pp. 366–376, 2002.

[20] S. Ratnasamy and S. McCanne, "Inference of multicast routing trees and bottleneck bandwidths using end-to-end measurements," in *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, pp. 353–360, IEEE, 1999.

[21] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, p. 60, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[22] A. Varga *et al.*, "Inet framework for omnet++ 4.0," *http://inet.omnetpp.org/*, retrieved on May 2013.

[23] C. Baugh and J. Huang, "Ieee 802.16 task group 3, 802.16. 3c-01/30r1: Traffic model for 802.16 tg3 mac," *PHY Simulations*.

[24] A. T. Andersen and B. F. Nielsen, "A markovian approach for modeling packet traffic with long-range dependence," *Selected Areas in Communications, IEEE Journal on*, vol. 16, no. 5, pp. 719–732, 1998.

[25] "The CAIDA USCD macroscopic topology dataset - 2005," *http://www.caida.org/tools/measurements/skitter*, retrieved on May 2012.

[26] M. Rabbat, R. Nowak, and M. Coates, "Multiple source, multiple destination network tomography," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 3, pp. 1628–1639, IEEE, 2004.

[27] Y. Gu, G. Jiang, V. Singh, and Y. Zhang, "Optimal probing for unicast network delay tomography," in *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9, IEEE, 2010.