Universidade do Minho
Escola de Engenharia

Fábio Rafael Azevedo Costa

Internet Tomography: Network topology discovery and network performance evaluation

Outubro de 2013

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Fábio Rafael Azevedo Costa

Internet Tomography: Network topology discovery and network performance evaluation

Dissertação de Mestrado
Mestrado em Redes e Serviços de Comunicação

Trabalho realizado sob orientação de

Professor Stefano Giordano
Professor Alexandre Santos

Outubro de 2013

Acknowledgements

First of all, I would like to thank the Erasmus program and University of Pisa for accepting me as a mobility student. It was a wonderful year, full of new friends and new experiences.

Secondly I would like to thank my family and my friends for all of their support and love during this year.

I would also like to thank the Telecommunication Networks Research Group of the Dept. of Information Engineering of the University of Pisa for receiving me and all of the people in the lab for their help and contribution to my professional growth and the very good time we spent together, with a special thank to Valerio Dei for all of his help on my studies and for showing me some of the good things in Tuscany.

Thanks to my professors and tutors prof. Stefano Giordano, prof. Davide Adami, prof. Michele Pagano, prof. Gregorio Procissi and prof. Alexandre Santos for their help and support.

I also would like to thank University of Minho and all the professors that contributed to my professional growth.

And finally, I would like to give a special thank to José Teixeira, for all the support, help, contribution for my personal and professional growth and for all the good moments we spent together.

Abstract

Due to the security threats and complexity of network services, such as video conferencing, internet telephony or online gaming, which require high QoS guarantees, the need for monitoring and evaluating network performance, in order to promptly detect and face security threats and malfunctions, is crucial to the correct operation of networks and network–based services. As the internet evolves in size and diversity, these tasks become difficult and demanding. Moreover, administrative limitations can restrict the position and the scope of the links to be monitored, while legislation imposes limitations on the information that can be collected and exported for monitoring purposes and almost all organization can't monitor or have knowledge or evaluate the performance of the entire network. They only can do this to part of the network, which corresponds to their own network.

In this thesis, we propose the use of tomographic techniques for network topology discovery and performance evaluation. Network tomography studies the internal characteristics of the network using end-to-end probes, ie, it does not need the cooperation of the internal nodes of the network and can be successfully adopted in almost all scenarios. Thus, it is possible to have knowledge of the network characteristics out of the administrative borders.

In this thesis we propose a new approach to Probe Packet Sandwich, where we use TTL-limited probes to infer the delay of a path hop-by-hop. We have shown that this approach is more effective than existing ones.

This work was developed under the ERASMUS student mobility program, in the Telecommunication Networks Research Group, Dept. of Information Engineering, University of Pisa.

Keywords: Internet Tomography, Network Topology Discovery, TTL-Limited Probes, Packet Sandwich, Link Failure Detection

Resumo

Devido às ameaças de segurança e complexidade dos serviços de rede, tais como videoconferência, telefonia via Internet ou jogos on-line, que exigem altas garantias de QoS, a necessidade de monotorização e avaliação de desempenho da rede, a fim de detectar prontamente e enfrentar as ameaças de segurança e mau funcionamento, é crucial para o correto funcionamento das redes e serviços baseados em rede. À medida que a Internet evolui em tamanho e diversidade, essas tarefas tornam-se difíceis e exigentes. Além disso, as limitações administrativas podem restringir a posição e o alcance dos links a serem monitorizados, enquanto a legislação impõe limitações sobre as informações que podem ser coletadas e exportadas para fins de monotorização e quase todas as organizações não podem controlar ou ter conhecimento ou avaliar o desempenho de toda a rede. Eles só podem fazer isso a parte da rede, o que corresponde à sua própria rede.

Neste trabalho, nós propomos o uso de técnicas tomográficas para a descoberta da topologia da rede e avaliação de desempenho. A tomografia de rede estuda as características internas da rede usando medições fim-a-fim, ou seja, não necessita da ajuda dos nós internos da rede, podendo ser adoptadas com sucesso em quase todos os cenários. Desta maneira é possível obter conhecimento das características da rede para além dos limites administrativos.

Neste tranalho propomos uma nova abordagem do Packet Sandwich Probe, onde utilizamos pacotes TTL-Limited para inferir o delay de um path hop-by-hop. Nós mostramos que esta abordagem é mais eficaz que outras já existentes.

Este trabalho foi desenvolvido no âmbito do programa de mobilidade de estudantes ERASMUS, no Grupo de Investigação em Redes de Telecomunicações, Departamento de Engenharia de Informação da Universidade de Pisa.

Palavras-Chave: Tomografia de Rede, Descoberta da Topologia de Rede, TTL-Limited Probes, Packet Sandwich, Detecção de Falhas de Link

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Due to the security threats and complexity of network services, such as video conferencing, internet telephony or online gaming, which require high QoS guarantees, the need for monitoring, management and evaluation of the network performance, in order to promptly detect and face security threats and malfunctions, is crucial to the correct operation of networks and network–based services. As the internet evolves in size and diversity, this tasks become difficult and demanding. Moreover, administrative limitations can restrict the position and the scope of the links to be monitored, while legislation imposes limitations on the information that can be collected and exported for monitoring purposes and almost all organization can't monitor, manage, have knowledge or evaluate the performance of the entire network. They only can do this to part of the network, which corresponds to their own network.

In this work, the first main topic to be addressed is network topology discovery: such definition includes all techniques that allow obtaining internal knowledge of internal network. This knowledge may be useful in various fields such troubleshooting, SLA verification, topology aware (multicast apps), network management, network monitoring, routing decisions or network performance evaluation.

Many prior studies have focused on Internet topology discovery based on ICMP commands, such as ping and traceroute, SNMP querys, which were able to reconstruct also layer-2 topology, and OSPF listening. However, these methods require cooperation of the internal nodes, which is not always available due to security requirements (firewall, ICMP rate limiter). Nevertheless, this limitation can be resolved through network tomography.

Network tomography covers a large variety of techniques which try to obtain information about the state of the network (be it either its topology or the state of congestion of its internal links) by applying statistical inference to measurements which are performed by the end nodes. The main advantage of such an approach is that it does not require cooperation from internal nodes, thus being able to cross the administrative borders of the networks.

Our objectives in this work is the use of network tomography techniques, more specifically in network delay tomography for not only network topology discovery, but also for discovering network link capacities. Then we use these results and loss tomographic techniques for detecting link failure on the network.

(For any problem with notation, please consult Appendix A)

1.2 Thesis Layout

This thesis is structured into five chapters: the present Chapter 1 is a brief introduction of the proposed work, its motivation and objectives; then, on Chapter 2 we address the problem of network topology discovery. We start for the related work on this field and then we present our proposal to solve the problem of network topology discovery. On the third chapter, we address the problem of link failure detection: firstly we present a proposal which requires the internal help of the internal nodes of the network. Then we present a proposal which doesn't require internal help of network nodes, however, both require the knowledge of the network topology. On the first proposal we assume the knowledge of the network. The second proposal requires the results obtained in the second chapter. On Chapter 4, we present the tools in order to get the results and in the next section, section Chapter 5, we present the results obtained. Finally, on Chapter 6, we present some conclusions and future work.

Chapter 2

Network Topology Discovery

2.1 Prior work

There are many different techniques and tools for network topology reconstruction. A large number of prior studies have focused on Internet topology discovery based on ICMP commands, such as ping and traceroute, e.g. [2–8]. In addition, ICMP measurements are RTT measures which are not feasible due to asymmetric paths. Other studies focused on SNMP querys, e.g. [9–11], which are able to reconstruct also layer 2 network topology. And others studies based on OSPF listening [12]. However, these kind of methods requires cooperation of the internal nodes of the network, which is not always available due to security requirements (such as firewalls or query blockers).

More recent research have focused on tomographic inference of router-level topology using end-to-end measurements of packet delay or loss. The main advantage of such technique is that it doesn't need the cooperation of internal nodes. Network tomography can be divided in two types, network loss tomography, which is based on end-to-end packet loss, and network delay tomography. Since the loss rate of actual links are very low, loss tomography techniques are not well suited for network topology reconstruction because several measures have to be made in order to obtain relevant data.

Initial work on network tomography methodologies focused on the use of multicast measurements [13–17]. Multicast traffic introduces a well structured correlation in the end-to-end behavior observed by the receivers that share the same multicast session. This correlation allows to infer the performance characteristics as packet loss rates and packet delay variance [18], and allows to estimate the delay distributions on each individual link [19]. Also it has a low impact on the internet traffic. However, multicast is not supported in all the networks due to scalability limitations, and internal performance observed by multicast packets may differ significantly from that observed by unicast packets [20]. More recent studies focused on unicast probing as an alternative to multicast, e.g. [18, 20–25]. These techniques are able to reconstruct the topology, based on the hop count [26] or delay distribution and clustering algorithms [25]. However, these algorithms lack on adicional information and/or they are incomplete since they only reveal branching nodes (they represent the node where the path from sender to receiver splits).

Andrea di Pietro presents, in his PhD thesis [27], a novel approach to network topology discovery based on the choice among a set of possible topological hypotheses and on the packet sandwich probe [25]. In his thesis, he states that link capacities usually belong to a restricted set of standard values (Ethernet, Optical Carrier, . . .), and defining a finite space of possible topological hypotheses, the most likely hypothesis can be chosen. Thus, it is possible to not only discover the spanning trees associated to each sender probe but also to discover the link capacities [28, 29]. In his work, he also presents a merge algorithm capable of merging all reconstructed spanning trees [30]. However, despite of the very good results on topology reconstruction, his work have some limitation and strong assumptions, that are described below (Section 2.4.2), which can lead to wrong reconstructed topologies or a complete fail of the algorithms.

In this chapter we will focus on Andrea's approach but we will aim to overcome the limitations of his work and to reduce the assumptions in order to obtain a more complete and reliable topology reconstruction.

2.2 The sandwich probing scheme

The sandwich probing scheme, described in [25], is delay-based and only measures delay differences. This way, there is no need for clock synchronization. The packet sandwich probe is a particular packet train that consists of three packets, two small packets ($p1$ and $p2$) separated by one much larger packet (q). The working principle of the sandwich probe is explained in Figure 2.1: the small packets ($p1$ and $p2$) are destined to the same receiver, while the larger packet (q) is destined to other receiver. The idea behind the packet sandwich probe is that the second small packet queues behind the larger packet, inducing extra separation between the two small packets on the shared path. Each measurement (d) is generated by the interarrival time of two small packets at a single receiver and it is related with the shared path of the receivers, i.e., the greater the length of the shared path, the greater will be the time difference between the small packets (d).

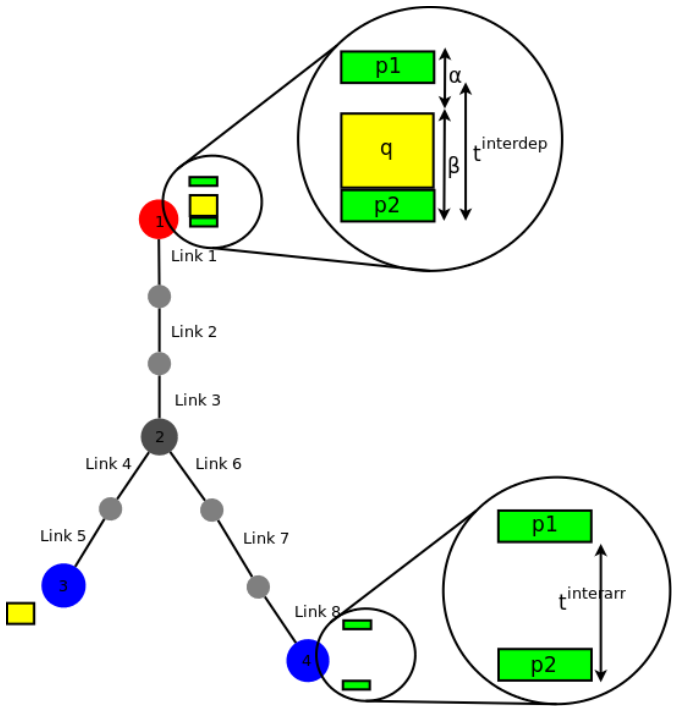

FIGURE 2.1: Packet Sandwich Probe working principle.

As shown in Appendix B, in the Equation B.9, the interarrival time the two small packets ($t_{1,2}^{interarr}$) can be calculated through the queueing time of those packets (Q^1 and Q^2).

Let us consider the case where there is no cross-traffic on the network. The packet $p1$ experiences no queueing delay ($Q^1 = 0$), however, the second small packet ($p2$) will always experience some queueing delay on the shared path by queueing behind the large packet q. Let us now consider the queueing delay experienced by the packet $p2$ on the i-th node of the shared path, i.e, on the i-th link of the shared path (Q_i^2). When the i-th node starts the transmission of the packet q on the i-th link, the node $i-1$ starts the transmission of the packet $p2$ on the link

$i-1$[1]. So this second small packet will reach the i-th node namely after $\frac{L^p}{C_{i-1}}$ seconds (plus the propagation delay of that link ($P_{d_{i-1}}$) and the processing delay of the packet on the i-th node ($P^2_{r_i}$)), it will be queued and it will wait to packet q to finish its transmission (and we can not forget that the packet q also had the transmission delay on the link $i-1$ ($P_{d_{i-1}}$) and the processing delay at the i-th node ($P^q_{r_i}$)). Thus, the queuing time at the i-th node can be expressed by the following equation:

$$Q^2_i = \left(P_{d_{i-1}} + P^q_{r_i} + \frac{L^q}{C_i}\right) - \left(P_{d_{i-1}} + P^2_{r_i} + \frac{L^p}{C_{i-1}}\right) \tag{2.1}$$

Since the shared path is the same and the processing delays can be discarted, as stated in Appendix B, we can simplify the Equation 2.1 to the following equation:

$$Q^2_i = \frac{L^q}{C_i} - \frac{L^p}{C_{i-1}} \tag{2.2}$$

Note that this value is a time value and cannot be negative, thus:

$$\begin{aligned} & Q^2_i \geq 0 \\ \Leftrightarrow\ & \frac{L^q}{C_i} - \frac{L^p}{C_{i-1}} \geq 0 \\ \Leftrightarrow\ & \frac{L^q}{L^p} \geq \frac{C_i}{C_{i-1}} \end{aligned} \tag{2.3}$$

In practice, we set L^q to be the MTU of the network, which is usually 1500 bytes, and with $L^p = 56$ bytes, we have the following relation: $\frac{C_i}{C_{i-1}} \leq 25$, which is typically satisfied in actual networks, as also stated in [25].

By summing up all the queueing times of the packet $p2$ we can simplify the Equation B.9, resulting the following equation:

[1]Note that this is not completely true. As stated in Appendix B, one packet is subjected by several delays. When a node finished to transmit one packet, it starts to transmit another one, but the first packet will experience some propagation delay on the link and some processing delay as soon it reaches the node.

$$\begin{aligned}
t_{1,2}^{interarr} &= t_{1,2}^{interdep} + Q^2 - \cancel{Q^1} \\
&= t_{1,2}^{interdep} - Q_1^2 + \frac{L^q}{C_1} + \frac{L^q}{C_2} - \frac{L^p}{C_1} + \ldots + \frac{L^q}{C_N} - \frac{L^p}{C_{N-1}} \\
&= t_{1,2}^{interdep} - \frac{L^q}{C_1} + \sum_{i=1}^{N-1} \frac{L^q - L^p}{C_i} + \frac{L^q}{C_N}
\end{aligned} \tag{2.4}$$

Since the $t_{1,2}^{interdep}$ contains the queueing time of the second small packet on the source node and since it is already added on the queuing time of the second small packet, we must remove it.

Let us have a look to the α parameter on Figure 2.1. According to [25], there should be a time difference between the transmission of the packet $p1$ and the packet q so that the packet q and the packet $p2$ don't queue behind the packet $p1$ (by queueing behind we mean queueing on the same node (same queue)), and the α value represent that time difference. If the packet q queues behind the packet $p1$, the measured sample can significantly differ from its theoretical value [31]. We will talk further about this value in Section 2.3, but for now we can easily notice that there is a minimum value for the α, which is the transmission time of the packet $p1$ on the source node. Thus, this value can be expressed as shown in the Equation 2.5.

$$\alpha = \frac{L^p}{C_1} + \delta \tag{2.5}$$

As we can also see from Figure 2.1, the δ value can be expressed as $\delta = t^{interdep} - \beta$, where β is equal to the sum of the transmission times of the packets q and $p2$. So, we can rewrite the equation Equation 2.4, resulting:

$$\begin{aligned}
t_{1,2}^{interarr} &= \delta + \beta - \frac{L^q}{C_1} + \sum_{i=1}^{N-1} \frac{L^q - L^p}{C_i} + \frac{L^q}{C_N} \\
&= \delta + \cancel{\frac{L^q}{C_1}} + \frac{L^p}{C_1} - \cancel{\frac{L^q}{C_1}} + \sum_{i=1}^{N-1} \frac{L^q - L^p}{C_i} + \frac{L^q}{C_N} \\
&= \delta + \frac{L^q}{C_1} + \sum_{i=2}^{N-1} \frac{L^q - L^p}{C_i} + \frac{L^q}{C_N}
\end{aligned} \tag{2.6}$$

If $\delta = 0$, then we have the same situation as in [28], i.e., the packets are sent back-to-back and the formula for the interarrival time between the two small packet are the same in [28]. Nevertheless, on this work we prefer to use the following formula:

$$\begin{aligned} t_{1,2}^{interarr} &= \delta + \frac{L^q}{C_1} + \sum_{i=2}^{N-1} \frac{L^q - L^p}{C_i} + \frac{L^q}{C_N} \\ &= \alpha - \frac{L^p}{C_1} + \frac{L^q}{C_1} + \sum_{i=2}^{N-1} \frac{L^q - L^p}{C_i} + \frac{L^q}{C_N} \\ &= \alpha + \sum_{i=1}^{N-1} \frac{L^q - L^p}{C_i} + \frac{L^q}{C_N} \end{aligned} \tag{2.7}$$

The cross-traffic induces substantial variation in the measurements, with queuing delays before and after the branching node, disrupting the measured spacing from its theoretical value. However, according to the paper [25], we can assume that the cross-traffic has a zero-mean effect on the measurements, provided that α value is sufficiently large.

In the case of heavy loaded networks, beyond the cross-traffic induces variation in the measurements, it may also be losses of probe packets. In this case, the measurement is discarted and a new measurement is made. In [25] states that all the measurements provide informative measurement of the delay difference, except when one or more packets are dropped, which make perfect sense, since if the packet q is dropped we can also have a measurement, but it is not good measurement, i.e., the packet q could be dropped anywhere on his path, but if it was dropped on the shared path, the measurement is wrong; and, in the case of one of the small packets is dropped, we don't have measurement at all. In that paper also states that the measure delay difference is made locally (on the receiver), however, in this work, this delay difference will be made on the sender node once the confirmation of the reception of the probe packets is required. So, once an acknowledgement must be sent to the sender node, the arrival time can be in it, except for the packet q (the arrival time of the packet q is irrelevant).

2.3 Alpha measurement

As stated in the previous section, the α value represents the time difference so that the packet q and packet $p2$ don't queue behind the packet $p1$. Is also stated that the cross-traffic induces substantial variation in the measurements, however we can assume a zero-mean effect provided that α value is sufficiently large. Here arises the question: *How do we get the value of α?* We know that the packet q must not queue behind $p1$, so, in the extreme, we can send the packet q only when the confirmation of the packet $p1$ is received. But it is an extreme case, and having this time as the α value, it would increase significantly the probing time.

Let us consider the worst possible case, which is when there is no shared path (we know that the packet q induces extra separation between the small packets, so the smaller the length of the shared path, the higher the probability of packets queueing together). Let us assume also that the packet travel back-to-back, which means that $\alpha = 0$ and there is no cross-traffic. Once there is no shared path, the queueing time of the packet $p2$ won't contain the transmission time of the packet q. However, since $\alpha = 0$, the queueing time of the packet $p2$ contain and will be the transmission time of the packet $p1$. So there is a minimum value for α, which is the the transmission time of the first small packet. Let us consider the delay experienced by the packet $p2$ on the i-th node of the path. When the i-th node starts the transmission of the packet $p1$ on the i-th link, the node $i - 1$ starts the transmission of the packet $p2$ on the link $i-1$ (still, this is the worst case scenario). The queuing time at this node can be expressed by the Equation 2.2 except that instead of L^q we will use L^p as expressed on the following equation:

$$Q_i^2 = \frac{L^p}{C_i} - \frac{L^p}{C_{i-1}} \tag{2.8}$$

From Equation 2.8, we can easily notice that, if the capacities of the i-th link and the link $i-1$ are equal, the queueing time is equal to 0; if it is bigger, the queueing time will not be negative, but will be 0; otherwise, there will be some queueing delay. Note that, if the capacity of the i-th link is bigger than the capacity of the link $i - 1$, the packet $p1$ will have some advance from the second small packet, which will result that in the following node (node $i+1$) will start the transmission of the packet $p1$ before the node $i - 1$ finishes the transmission of the packet p2.

The α value can be, therefore, calculated by the interarrival time of these packets. In this work, in order to have a more reliable value for α, we will add 20 % of the calculated value. This small amount of time added to α was given intuitively.

This measurements can also be affected by the noise traffic, though. So, in order to get a good estimate for the α value, we will use the noise reduction algorithm presented in [31]. This algorithm is very important for this work; it not only will be used to get a more reliable estimate for the α value, but also it will be used to select the best measurements of the interarrival times of the sandwich probing. This algorithm has as input all the arrival times of the packets $p1$ and $p2$ (this is the main reason for the acknowledgements of these packets do include their arrival times) and selects the best measurements, i.e., the measurements less affected by the noise traffic.

Notice that, by only using the noise reduction algorithm, we can obtain the measurements less affected by the noise traffic, which are the best measurements. However, by using the α parameter, the best measurements are even better measurements, and the number of best measurements is higher.

2.4 The decision theoretic approach

2.4.1 Andrea's approach

The decision theoretic approach used in this work is completely based on the decision theoretic approach presented in [28]. This paper states that in the real networks, the link capacities usually belong to a restricted set of standard values (Optical Carier (OC), Ethernet, ...) so, the set of capacities $\{C_1,C_2,\dots\}$ is discrete. By taking advantage of the TTL field of the packets is also possible to have an estimate of the depth of the shared path. It is simple to notice that the depth of the shared path of one receiver pair (i, j) will be never bigger than the minimum TTL of the paths from the source (s) to the receivers i and j: $N \leq min(\#\Lambda_{s\to i}, \#\Lambda_{s\to j})$. Thus, given the set of all possible links capacities $(\mathcal{S})$ and an upper bound of the depth of the shared path (N value), it is possible to off-line pre-evaluate all the possible values of the metric d by using Equation 2.7. Any possible combination of D elements of $\mathcal{S}$ will be referred as *Link Capacities Combination* (LCC).

Given estimates $\hat{d}$ of the metric obtained by averaging a given number of interarrival time samples (Equation 2.9), the fundamental decision problem is to select the correct LCC that originated it.

$$\hat{d} = \frac{\sum_{i=1}^{M} d_i}{M}, \text{ M is the number of interarrival samples} \tag{2.9}$$

As stated in [28], different combinations or combinations that only differ for the elements order can produce the same value of the metric. Thus, topology reconstruction based on LCCs decisions can be affected by a certain degree of ambiguity. However, in the paper states that the incidence of this ambiguity is limited for realistic network sizes and capacity sets. In addiction, It is possible to find out the whole combination of end-to-end capacities by sending the three packets to the same receiver i ($d_{i,i}$).

According to the paper, we can model the sample mean $\hat{d}$ as a Gaussian random variable by invoking the Central Limit Theorem. Thus, the link capacity combination decision can be made according to the Maximum Likelihood criterion, which collapses into the minimum distance criterion. The selected link combination $\bar{C}$ will be calculated through the following equation:

$$\bar{\mathcal{C}} = arg\ min_{C \in LCC} \mid \hat{d} - \gamma(\mathcal{C}) \mid \tag{2.10}$$

where $\gamma(C)$ is the function that evaluates the metric d of a link capacity combination by using Equation 2.7.

Once all decisions are made for each pair of receivers, they can be elaborated to infer the network topology. Such a task will be elaborated by the algorithm presented in [28], which is the following algorithm (Algorithm 1):

Algorithm 1: Spannig tree reconstruction

```
1  Initializate set $\mathcal{L}$ containing the leaves of the spanning tree ($\mathcal{T}$) ;
2  foreach leaf $i \in \mathcal{L}$ do
3      define $\Lambda_{root \to i} = \mathcal{B}_{i,i}$
4  end
5  while $\mathcal{L}$ is not empty do
6      Choose the set $\mathcal{B}_{i,j}$, with $i, j \in \mathcal{L}$, composed by the maximum
       number of links;
7      Find the set M $\subset$ $\mathcal{L}$ such that $\mathcal{B}_{m,n} = \mathcal{B}_{i,j}$ $\forall_{n.m} \in M$;
8      Remove from $\mathcal{L}$ all the nodes that belong to M;
9      Add to $\mathcal{L}$ a new node k;
10     foreach node $h \in \mathcal{L}$ do
11         if $h \neq k$ then
12             define $\mathcal{B}_{k,h} = \mathcal{B}_{m,h}$ with $m \in M$
13         end
14     end
15     define $\Lambda_{root \to k} = \mathcal{B}_{i,j}$;
16     foreach $m \in M$ do
17         define $\Lambda_{k \to m} = \Lambda_{root \to m} \setminus \Lambda_{root \to k}$
18     end
19 end
```

Is still possible to obtain better topology reconstruction by taking advantage of LCCs topological relations presented in [29]. The main idea presented in this paper is that, if the measurements are not completely wrong, if the pre-evaluated interarrival time for one pair of receivers (i, j) is less or equal to the pre-evaluated interarrival time for other pair of receivers (i, k) then is possible to conclude that the link combination for the pair (i, j) is a subset of the link combination of the pair (i, k). It can be formalized as follows:

$$\begin{aligned} \gamma_{i,j} \leq \gamma_{i,k} \Rightarrow LCC_{i,j} \subseteq LCC_{i,k} \\ \gamma_{i,j} \geq \gamma_{i,k} \Rightarrow LCC_{i,j} \supseteq LCC_{i,k} \end{aligned} \tag{2.11}$$

Constraints from previous decisions can prevent errors that may occur due to the presence of ambiguous LCCs. However, if an error is made on the first iterations, it will propagate to the rest of the inferred topology.

2.4.2 Limitations

Despite of the very good results obtained by the approach proposed by Andrea's, it still some limitations and it makes strong assumptions. As we already stated before, one of the limitations of Andrea's work is that the LCC reconstructed suffer a certain degree of ambiguity due to link ordering. This problem can happen if a branching node is not directly connected to a probe or other branching node. This problem doesn't lead to wrong network topologies but it can lead to topologies with wrong capacity links.

Other problem of his work, and this problem can lead to erroneous reconstructions, is that it doesn't know the exactly depth of the shared path (it knows for the pairs constituted with the same receiver, but not to the others constituted by different receivers), it only knows the maximum depth of the shared path, which coincides with the minimum TLL of paths of both receivers. Note that the chosen LCC can have less or even more links than the actual shared path. Thus, if the algorithm works, it will lead to a wrong network topology. Having knowledge of the exact depth of the shared path can lead to a more accurate LCC decisions.

Assuming that the capacity of a link i is less than 25 times the capacity of previous link ($\frac{C_i}{C_{i-1}} \leq 25$) is a very strong assumption, even that in [28] states that it is satisfied in actual networks. In a topology, if there is a link that doesn't satisfy this condition, by Andrea's approach, the LCC's that contains this link and the LCC's that doesn't will have the same metric, and the algorithm may fail or reconstruct a wrong topology.

These three problems are the main limitation in Andrea's work. However there are still problems due to the reconstruction be a router-level reconstruction. In other words, layer 2 paths will be reconstructed as one link. Something similar will happen in the case of VPN's, i.e., the link reconstructed can represent several physical links.

2.4.3 The new approach

In order to obtain a more precision of the network topology reconstruction, we have focused on the link ordering limitation. As we know, this limitation results due to the LCCs that only differ for the element order and the lack of branching points on the paths.

If the internal help of the network nodes was available, we could solve this problem with existing tools, such as *traceroute*[32]. However, as a security measure, many of the internal nodes have the icmp replys blocked. So we can not rely on internal help of the network nodes and so, we can not use the traceroute tool.

However, we decided to use the underlying idea of the traceroute, which is make several measurements with different TTL. On the small packets we can not do this, otherwise we will not obtain a measurement, but we can manipulate the TLL of the large packet.

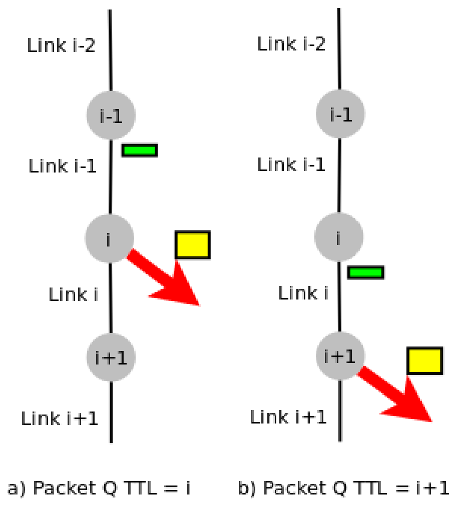

FIGURE 2.2: Packet q TTL manipulation.

Let us consider the case where there is no cross-traffic on the network. As shown in Figure 2.2 , if the TTL of the packet q is set to $i-1$ then it will be dropped at the node i (note that the sender node is first node on the path), and the shared path will be until the link $i-1$ (included); if it is set to i then it will be dropped at the node $i+1$, and the shared path will be until the link i (included). So, by controlling the TTL of the packet q we can control where the packet q will be dropped and so, the depth of shared path of the probing packets until the branching point. The interarrival time of the small packets is completely related to the length of the shared path, as shown in Equation 2.7. An increase or decrease in the shared path will result also in an increase or a decrease of the interarrival time of the small packets, respectively.

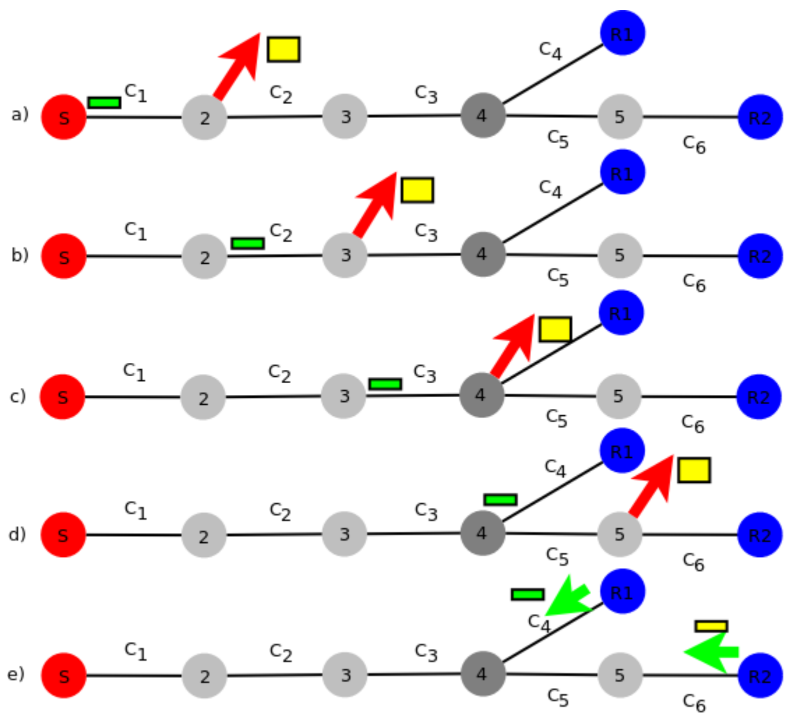

FIGURE 2.3: Packet q TTL incrementation.

Let us consider the example shown in Figure 2.3 assuming there is no cross-traffic. In the case a), the TTL of the packet q was set to 1, and as shown, the packet is dropped in the second node. The delay experienced by the packet in this path will be just the delay experienced in the sender node which will be $Q^2 = Q_1^2 = \frac{L^q}{C_1}$, which will result in the interarrival time shown in Equation 2.12a.

In the case b), the TTL of the packet q was set to 2. The packet is dropped in the third node and the delay experienced will be the sum of the delay in the first and in the second node: $Q^2 = Q_1^2 + Q_2^2 = \frac{L^q}{C_1} + \frac{L^q}{C_2} - \frac{L^p}{C_1}$, resulting in an interarrival time shown in Equation 2.12b.

For the case c), following the same reasoning line we will obtain an interarrival time shown in Equation 2.12c.

By looking to Figure 2.3, we can easily conclude that the shared path between nodes R_1 and R_2 are links 1, 2 and 3. Therefor, we also can conclude that there will be no queuing delay on forth node. By setting the TTL to 4, the packet will be dropped on the fifth node, which is not on the shared path, so the queueing time will be equal to the queueing time calculated for a TTL set to 3 and consequently the interarrival times will also be equal, as shown in Equation 2.12d. The same happens in the case e), however, in this case, the packet q reaches is destination and a acknowledgement is sent back to the sender.

$$\begin{align}
t_{1,2}^{interarr}(1) &= \frac{L^q}{C_1} + \alpha \tag{2.12a}\\
t_{1,2}^{interarr}(2) &= \frac{L^q - L^p}{C_1} + \frac{L^q}{C_2} + \alpha \tag{2.12b}\\
t_{1,2}^{interarr}(3) &= \frac{L^q - L^p}{C_1} + \frac{L^q - L^p}{C_2} + \frac{L^q}{C_3} + \alpha \tag{2.12c}\\
t_{1,2}^{interarr}(4) &= \frac{L^q - L^p}{C_1} + \frac{L^q - L^p}{C_2} + \frac{L^q}{C_3} + \alpha \tag{2.12d}\\
t_{1,2}^{interarr}(5) &= \frac{L^q - L^p}{C_1} + \frac{L^q - L^p}{C_2} + \frac{L^q}{C_3} + \alpha \tag{2.12e}
\end{align}$$

With all equations in Equation 2.12 we can deduce a new formula for the interarrival time, expressed as follows:

$$t_{1,2}^{interarr}(ttl) = \begin{cases} \alpha + \sum_{i=1}^{ttl-1} \frac{L^q - L^p}{C_i} + \frac{L^q}{C_{ttl}} & \text{if } ttl \leq N \\ \alpha + \sum_{i=1}^{N-1} \frac{L^q - L^p}{C_i} + \frac{L^q}{C_N} & \text{if } ttl > N \end{cases} \tag{2.13}$$

Notice the Equation 2.12a, all the values are known, except for C_1, which is the value we want to calculate (the capacity of the link). So, in a perfect world with no cross-traffic we could simply calculate this value by resolving the equation. Considering there is noise on the network, we have to use the Equation 2.10. To obtain the capacity of the second link, we have two variables (C_1 and C_2), but C_1 can be previously calculated and thus, we can calculate C_2 and so on. The point of all of this is that we can calculate the capacities of one path, one at a time, thus solving the not only problem of link ordering but also, we can obtain the real value of the shared path (not the estimate). Note that when ttl after reaching the value of the depth of the shared path (N), the formula for $t_{1,2}^{interarr}(ttl)$ is always the same, i.e., the value will be always equal until packet q reaches the destination.

Also notice that making the measurements to each TTL will increase the number of measurements to each receiver, which will may have impact on the network traffic.

However, since this approach obtain better results than the approach presented by Andrea, this increase compensates.

Let us consider the Figure 2.2 where the capacity of the link i is greater than 25 times the capacity of the previous link (link $i-1$) (which we will represent by $L25$), i.e., $\frac{C_i}{C_{i-1}} > \frac{L^q}{L^p} \Leftrightarrow \frac{C_i}{C_{i-1}} > 25$. So, we have that $t_{1,2}^{interarr}(i-1) = t_{1,2}^{interarr}(i)$, so the capacity of the link i can not be determined, we only know what is its minimum capacity. On the other hand, we know how much is its minimum capacity, which is 25 times the capacity of the previous link. Depending on the set of possible capabilities, the possibilities for this link become very few.

However, having that $t_{1,2}^{interarr}(i-1) = t_{1,2}^{interarr}(i)$, the i-th node can be confused with a branching node. Which can difficult the reconstruction of the shared path. Let us consider the various locations of the link $L25$ on a shared path, as shown in Figure 2.4. The yellow link represents the link $L25$.

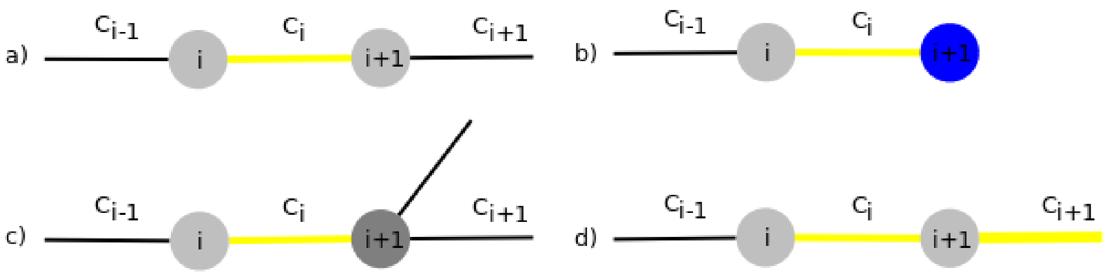

FIGURE 2.4: Different topologies cases.

The case a) shows the case where, in shared path, there is a link $L25$ followed by a link that is not a link $L25$. Following the Equation 2.13, the value of interarrival time will increase to $TTL = i-1$, included. When $TTL = i$, the value of the interarrival time will remain the same and when $TTL = i+1$, the value will increase again. Thus, it is not possible to the node i be a branching point, since the value of the interarrival time will no longer increase again after the branching node.

The b) case shows a particular case. Note that we are talking about what happens in the shared path, and the yellow link is the last link of the path. This means that the entire path was shared. This only happens when one intends to calculate the interarrival time of a probe pair constituted only by one receiver, in our notation, $t_{i,i}^{interarr}$. In these cases there are no branching points, so it is not possible to confuse any node contained in the path with a branching node.

The case d) presents a further special case where there are two links whose capacities are 25 times bigger than the previous (2 links $L25$ consecutive). This just makes the capabilities of these links even more restricted, becoming easier to get their capacities. However, both the node i and node $i+1$ can be confused with branching nodes. Everything will depend on the following link.

The case c) shows the case where the branching node is the node where the yellow link is connected. There is no queuing delay in the branching node, as the link i is a link $L25$, at node i also won't exist queuing delay. I.e., after the node i, included, will not exist more queuing delay. And the branching point can be the node i or the node $i+1$. This problem can be exacerbated considering the cases d) and c) together, which would result in three hypotheses for branching node.

Following this line of reasoning we could say that until any node after the not increase of the value of the interarrival time can be a branching node. Therefore, it is necessary to define when can exist a branching node. A node m may be a branching node if it satisfies the following condition:

$$t_{i,j}^{interarr}(m) == t_{i,j}^{interarr}(TTL\ max\ j)$$

and also satisfies the following conditions in the following order:

$$\begin{array}{ll}
1) & \Phi\ isEmpty \\
& 1.1)\quad \mathcal{B}_{j,j_m} \neq \mathcal{B}_{i,i_m}, \text{ e é único} \\
& 1.2)\quad \dfrac{C_m}{C_{m-1}} \leq \dfrac{L^q}{L^p}, \text{ e é único} \\
& 1.3)\quad \dfrac{C_m}{C_{m-1}} > \dfrac{L^q}{L^p} \\
2) & (m-1 \in \Phi\ and\ \dfrac{C_{m-1}}{C_{m-2}} > \dfrac{L^q}{L^p})
\end{array}$$

where Φ is the set of possible branching nodes in a shared path of a pair of receivers (i,j). Having several possibilities for branching nodes into a shared path means having several possibilities of the link capacities in a shared path, which also means having several possibilities for the spanning tree of one sender node. However, it is not possible to know the correct spanning tree without more information.

As shown in [28], $\mathcal{B}_{i,j}$ is reconstructed by Equation 2.10. Although it is possible to calculate $\mathcal{B}_{i,j}$ for each ttl only with this expression, we will not do so. In this work we present a new algorithm for choosing the paths and shared paths. This algorithm is divided into two parts: the first one where the selection is made for each of the paths for each receiver (only $\mathcal{B}_{i,i}$ values are used), shown by Algorithm 2; and another part where the choice of shared paths for each pair of receivers is made, presented by Algorithm 3.

Algorithm 2: Decision algorithm part one

```
input  : $t^{arr}_{p1}$ ; $t^{arr}_{p2}$ ; $\mathcal{S}$ ; timeout ; map ttl-receiver
output: $\mathcal{B}$
1  Initializate set $L$ containing the leaves of the spanning tree ($\mathcal{T}$) ;
2  foreach $i,j \in L$ do
3      $\mathcal{B}_{i,j} = \emptyset$
4  end
5  foreach $i \in L$ do
6      ttl max = $TTL_{max}$ (i) ;
7      $d_{prev} = 0$ ;
8      for $ttl = 0$ until $ttl = $ ttl max, not included do
9          set $d$ = noise_reduction ($t^{arr}_{p1}$ , $t^{arr}_{p2}$, timeout) ;
10         $\hat{d}$ = mean $d$ ;
11         if $\hat{d} - d_{prev} > min\_value(\mathcal{B}_{i,i})$ then
12             $\bar{\mathcal{C}} = arg\ min_{\mathcal{C} \in LCC} \mid \hat{d} - \gamma(\mathcal{C}) \mid$, such that, for k = 0 until
               k = ttl, not included, the k-th element of $\mathcal{B}_{i,j}$ is equal to
               the k-th element of $\mathcal{C}$ ;
13             $\mathcal{B}_{i,j} = \bar{\mathcal{C}}$ ;
14         else
15             $\bar{\mathcal{C}}$ = s $\in \mathcal{S}$, such that, s > $\frac{L^q}{L^p} \times$ last element of $\mathcal{B}_{i,j}$ ;
16             $\mathcal{B}_{i,j}[ttl] = \bar{\mathcal{C}}$ ;
17         end
18         $d_{prev} = \hat{d}$
19     end
20 end
```

Since the cross-traffic can induce variations in measurements, even though there are links $L25$ or even after the branching point, the values of the interarrival time may not be the same, although they are very close to each other. So we need a function that, by introducing the capacity of one link, it returns the minimum queuing delay expected by the addition of a new link to the shared path, provided that the new link is not a link $L25$. For example, given a certain capacity *Capacl*, this function should choose the larger capacity of the set S, so that the capacity chosen is not 25 times the capacity introduced and returns the minimum queuing delay of packet $p2$ using Equation 2.2. If the difference between the $\hat{d}$ of the current ttl and $\hat{d}$ of the previous ttl is very close to the returned value by the function, then it is possible to conclude that there is one more link in the shared path. Otherwise, it may mean that the following link is a link $L25$, or there are no more links in the shared path.

Algorithm 3: Decision algorithm part two

```
foreach $i,j \in L$, that that, $i \neq j$ do
    ttl max = $TTL_{max}$ (j) ;
    $d_{prev} = 0$ ;
    Initializate set $\Phi$ as empty ;
    for $ttl = 0$ until $ttl = ttl\ max$, not included do
        set $d$ = noise_reduction ($t^{arr}_{p1}$ , $t^{arr}_{p2}$, timeout) ;
        $\hat{d}$ = mean d ;
        if $\hat{d}$ - $d_{prev}$ > min_value($\mathcal{B}_{i,i}$) then
            if ttl-th element of $\mathcal{B}_{i,i} \neq$ ttl-th element of $\mathcal{B}_{j,j}$ then
                Algorithm fail
            else
                $\mathcal{B}_{i,j}[ttl] = \mathcal{B}_{i,i}[ttl]$ ;
            end
        else
            if $\Phi$ is empty then
                if $\mathcal{B}_{i,i}[ttl] \neq \mathcal{B}_{j,j}[ttl]$ or ($\mathcal{B}_{i,i}[ttl] = \mathcal{B}_{j,j}[ttl]$ and $\frac{\mathcal{B}_{i,i}[ttl]}{\mathcal{B}_{i,i}[ttl-1]} \leq \frac{L^q}{L^p}$) then
                    continue = false ;
                else
                    if $\frac{\mathcal{B}_{i,i}[ttl]}{\mathcal{B}_{i,i}[ttl-1]} > \frac{L^q}{L^p}$ then
                        $\mathcal{B}_{i,j}[ttl] = \mathcal{B}_{i,i}[ttl]$ ;
                        add $\infty$ to $\mathcal{B}_{i,j}[ttl]$ ;
                        add ttl to $\Phi$ ;
                    else

                    end
                end
            else
                if $\Phi$ contains $ttl - 1$ and $\frac{\mathcal{B}_{i,i}[ttl-1]}{\mathcal{B}_{i,i}[ttl-2]} > \frac{L^q}{L^p}$ then
                    $\mathcal{B}_{i,j}[ttl] = \mathcal{B}_{i,i}[ttl]$ ;
                    add $\infty$ to $\mathcal{B}_{i,j}[ttl]$ ;
                    add ttl to $\Phi$ ;
                else
                    continue = false ;
                end
            end
        end
    end
end
```

If the algorithm concludes that there is one more link on the shared path, where a pair (i,i), then this choice is made by the Equation 2.10, but with some restrictions, namely the new combination will be calculated taking into account the capabilities of the first $ttl-1$ links are equal to the capacities of the links previously calculated. If a pair (i,j) with i $\neq$ j, then the capacity of the ttl-th link of the shared path will be equal to the capacity of ttl-th link of the path (i,i) which in turn must be equal to the capacity of the ttl-th link of the path (j,j). If different, then something went wrong. It may have been the capacity of these links that have been misplaced or it may have been a difference of the current $\hat{d}$ and previous one which was a relatively high value.

Regarding the reconstruction algorithm, it works correctly if all $\mathcal{B}_{i,j}$ are correct. Therefore the algorithm used will be the same.

Thus, our work can exclude the assumption of the no existence of links $L25$ in the network. However, we may be not able to accurately identify the capacity of that link, but the possibilities will be very few.

2.5 The merge algorithm

2.5.1 Overview

The merging algorithm proposed in [30], does not require further probing traffic and is able to reveal all the nodes of the network (not only the branching nodes). This is an algorithm to be applied in network scenarios where each probe is either a sender probe or a receiver probe.

This algorithm works in two phases: at first, it scans the path connecting each sender-receiver pair and assigns the same label to the nodes representing the same node on different trees. After that, a tree merging operation based on the value of the labels is performed. Let $I_n^{i \to j}$ be n-th node on the network path connecting i and j, and let $N_{i,j}$ be the total number of nodes composing such a path; the merging algorithm works as follows:

Algorithm 4: Spanning tree merging algorithm

```
input  : The set of all spanning trees
output: The merged graph $\mathcal{G}$
1  foreach pair $(i,j)$ do
2      for $n = 1,2,\ldots,N_{i,j}$ do
3          if neither node $I_n^{i\to j}$ nor node $I_{N_{i,j}+1-n}^{j\to i}$ are labbeled then
4              both nodes are assigned with same label ;
5          else
6              if one of the nodes $I_n^{i\to j}$ and $I_{N_{i,j}+1-n}^{j\to i}$ has already been
               labeled then
7                  the node that wasn't been labeled is assigned is the
                   same label.
8              end
9          end
10     end
11 end
12 Merge tree as follows:
13     – the nodes set of the resulting graph is the set-theoretic union of
       the nodes sets of the input graphs ;
14     – all the edges in the input graph are retained in the resulting graph ;
15 Prune the edges of the resulting graph by leaving one single edge for
   each pair of connected nodes ;
```

2.5.2 Limitations

Note that the node $I_n^{i\to j}$ and the node $I_{N_{i,j}+1-n}^{j\to i}$ cannot be the same node. In the cases where the routing is not symmetric the paths i to j and j to i may differ. So, in [30], it is assumed that the network under test implements symmetric routing.

Although we tried to consider no such assumption, we were not able to find a solution for this problem. One solution that we found was merge only the path that was equal for both probes, however this could lead to a complete wrong topology. In our work, the topology will be reconstructed even if there are asymmetric routing, however, we need to discovery is the path is symmetric or asymmetric. To be a symmetric path, the following conditions must be satisfied:

$$1)\ \#\mathcal{B}_i i \ = \ \#\mathcal{B}_j j$$
$$2)\ \#\mathcal{B}_i i_n \ = \ \#\mathcal{B}_j j_{\#\mathcal{B}_j j+1-n},\ \forall n \in 1,2..\#\mathcal{B}_i i$$

If these conditions are satisfied, then we can assume that the path is symmetric. But note that two paths can have the same depth and the same link capacity order and yet not be the same path. So false positives may occur.

Other problem of the merge algorithm, also stated in [30], is that is not always true that all nodes corresponding to the same physical device are assigned the same label. In some specific topologies, such as the one shown if figure Figure 2.5 the label assignment algorithm may fail as it may assign different labels to nodes which actually correspond to the same node. In this case, multiple instances of this nodes, which will be referred to as multiple label nodes, will be present in the reconstructed global topology. However, as shown in [30], the actual impact of such nodes is quite low also in the case of very meshed topologies.

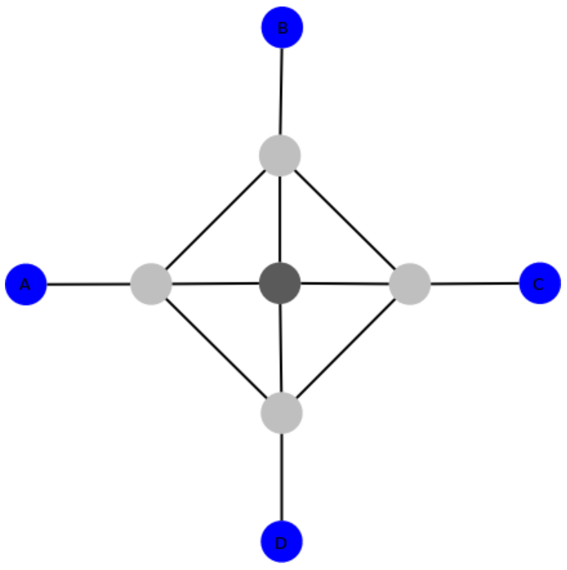

FIGURE 2.5: Topology which causes the failure of the spanning tree merge algorithm: the darker node is the multiple label node.

This problem could be simply resolved if we were able to somehow get an ID from the node, such as the IP. However, in this work we assume there is no cooperation from the network internal nodes, and we weren't able to find any solution to get the nodes ID. So we weren't able to find a solution for this problem.

Chapter 3

Link Failure Detection

3.1 Prior Work

Initial work on network tomography methodologies focused on the use of multicast measurements [13–17]. Multicast traffic introduces a well structured correlation in the end-to-end behavior observed by the receivers that share the same multicast session. This correlation allows to infer the performance characteristics as packet loss rates, packet delay variance and the delay distributions on each individual link [18, 19].

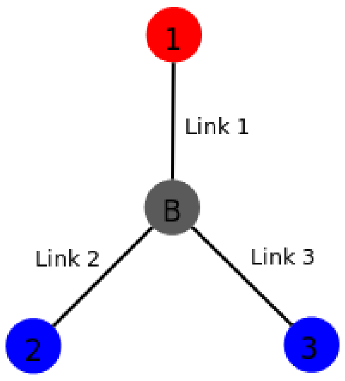

FIGURE 3.1: Simple Tree Topology.

To illustrate the idea behind multicast based loss inference, consider the simple tree in Figure 3.1 with the source (the red node) sending multicast packets to the two leafs nodes 2 and 3 (blue nodes). If a multicast probe is sent by the source node to both receivers, node 2 and 3, but the probe arrives only at node 3, and not at node 2, then it is possible to immediately infer that the loss occurred on link 2. By sending many multicast probes from source node to receivers nodes 2 and 3, it is possible to infer the loss rates on the two links, link 2 and link 3. Furthermore it it also possible to infer the loss rate on link 1 [33]. However, multicast is not supported in all the networks due to scalability limitations. Hence, new tomographic methods emerged as an alternative to multicast base on unicast probes.

Papers [34, 35] use only unicast end-to-end flows for the simpler goal of identifying the congested links, i.e. identify if the link loss rate or delay exceeds some value

Other studies focused on Boolean Tomography. This it is a class of network tomography in which links can have two possible states: "good" or "bad". When all links from one path are "good" then the path will also be "good", but if there is at least one link that is "bad", then the path will be "bad". And tries to identify the smallest set of "bad" links that explains the end-to-end unreachabilities. However, paper [36] shows that a basic "Boolean Tomography" has several limitations and performs poorly in identifying multiple link failures. So, it presents *Tomo* that constitutes an extension of the Boolean tomography approach with multiple probing sources and destinations. The problem of multiple sources and multiple destinations is an instance of the *Minimum Hitting Set* problem, which optimization version in NP-Hard. However, the paper states that has been shown that a greedy heuristic approximates the solution to Min Set Cover within an approximation ratio of $\log \mu$, where μ is the set of elements from which the hypothesis set can be chosen. *Tomo* a greedy heuristic with the aim of solving the *Minimum Hitting Set* given the reachability matrix ($\mathcal{R}$).

Nevertheless, in paper [37] states that the gist behind *Tomo* is that a few congested links are responsible for many congested paths. This algorithm favors links that participate in more congested paths. It also states that the Boolean Inference problem is ill-posed, i.e., given any network graph and the outcome (set of "bad" paths), there may be possible solutions for "bad" links. In this paper, the authors introduce an algorithm that solves the Probability Computation problem than those required by Boolean Inference and more challenging network conditions.

3.2 Link failure detection with internal help

In this part of the work we went out a bit of the main focus of this chapter, which is the detection of link failures without cooperation of the internal nodes, and we assume that we have total control over the internal nodes. More precisely, in this part we aim to solve the problem of link failure detection between two probe nodes (a sender node and a receiver node) using the *Tomo* algorithm presented in [36].

According to [36], *Tomo* constitutes an extension of the Boolean tomography approach with multiple probing sources and destinations. It is a greedy heuristic with the aim of solving the *Minimum Hitting Set* given reachability matrix ($\mathcal{R}$). The reachability matrix $\mathcal{R}$ reflects the status of each path. The status of one path $\mathcal{P}_{ij}$ is 0 if is down or 1 if is up, in other words, the status of one path is 0 if the packet from i to j was dropped (lost) and 1 if it reached the destination. So, $\mathcal{R}_{ij} = 1$ if the path from i to j is good, and $\mathcal{R}_{ij} = 0$ otherwise. Considering $f(l)$ denote the status of the link l, if $\mathcal{R}_{ij} = 1$ then $f(l) = 1, \ \forall \ l \ \in \ \mathcal{P}_{ij}$. If $\mathcal{R}_{ij} = 0$ then $\exists \ l \ \in \ \mathcal{P}_{ij} \ f(l) \ = \ 0$.

Algorithm 5 – *Tomo* proceeds iteratively as follows: first the set of the failure sets $\mathcal{F}$ is initialized containing all the broken paths and all of them are still unexplained, so the set of unexplained $\mathcal{F}_u$ is initialized (it will the same than $\mathcal{F}$ initially). Also, we initialize the candidate set $\mathcal{U}$, containing all the links from all broken paths, and then remove the links that are contained in the working paths, since they cannot be down. In each iteration, it is computed for each link $l \ \in \ \mathcal{U}$, the number of unexplained failure sets that l intersects with (called the "score" of the link l). Then the link or links with the highest score are added to the hypothesis set.

This problem is a single-source, single destination problem. If we consider the "normal routing" we only have one path to measure and we can only obtain the status of each link in the path. However, we have complete control over the topology, which means that we can change the routing. Taking this into account, we can calculate all the paths from the sender to the receiver. Thus, we can obtain the status of more paths. However, all the paths from sender probe to the destination probe may not include all the links of the topology, which means that we will not obtain the status of these links.

Algorithm 5: Tomo

```
input : $\mathcal{R}$
output: $H$
1  Initializate $\mathcal{F} = \emptyset$ The set of failure sets ;
2  foreach $\mathcal{R}_{ij} = 0$ do
3      $\mathcal{F} = \mathcal{F} + \mathcal{P}_{ij}$ {Add the failure set due to each broken path} ;
4  end
5  Initializate $H = \emptyset$ {Hypothesis set of failed links} ;
6  Initializate $\mathcal{F}_u = \mathcal{F}$ {Set of unexplained failure sets} ;
7  $\mathcal{U} = \cup \mathcal{P}_{ij}\ \forall \mathcal{P}_{ij} \in \mathcal{F}$ {The candidate set} ;
8  Remove from $\mathcal{U}$ every link $l$ on a working path ;
9  while $\mathcal{F}_u \neq \emptyset$ AND $\mathcal{U} \neq \emptyset$ do
10     foreach link $l \in \mathcal{U}$ do
11         $C(l)$ = set of failure sets in $\mathcal{F}_u$ containing $l$ ;
12         score($l$) = $|C(l)|$ {The number of unexplained failure sets that
           $l$ intersects with} ;
13     end
14     $\mathcal{F}_m = \{l_m | l_m = argmax_{l \in \mathcal{F}}$ score($l$)$\}$ {The set of links with the
       maximum score} ;
15     foreach link $l_m \in \mathcal{F}_m$ do
16         $H = H \cup \{l_m\}$ {Add $l_m$ to hypothesis set} ;
17         $\mathcal{F}_u = \mathcal{F}_u - C(l_m)$ {All failure sets in $C(l_m)$ are now
           explained. Remove from $\mathcal{F}_u$} ;
18         $\mathcal{U} = \mathcal{U} - \{l_m\}$
19     end
20 end
```

For calculating all the paths from the sender to the node we will use an algorithm based on the Breadth-first search algorithm [38], which is the Algorithm 6, described in Appendix C.

However, we do not need to obtain the reachability of all paths calculated since some of them may be linearly dependent from the others. Thus, we only need obtain the reachabilities only the paths that are linearly independent. In order to obtain the linearly independent matrix, we modified the function *rref* [39] from *MatLab*. This function produces the reduced row echelon form of a matrix using Gauss Jordan elimination with partial pivoting, the number of rows with all zeros are the linearly dependent vectors. So, our modification to the function was only saving the initial index of each vector, and, in the end, eliminate the rows that originated all zeros rows. We present our modification in Appendix D: at blue are the lines that we added to the file *rref.m*.

The reachabilities can be measured by only send icmp ”ping” packet’s to the destination, if a reply is received in a certain period of time, then the path is up, otherwise will be set as down. After obtaining the linearly independent matrix and the reachabilities matrix $\mathcal{R}$ we can apply the tomo algorithm to infer which link or links were more likely to be down.

3.3 Link failure detection with no internal help

Returning to our main objective, we want now to detect link failure without internal cooperation. Assuming that the reconstruction algorithm reconstructed the network topology correctly, then we can obtain the routing matrix $\mathcal{P}$ constituted by all paths from the sender probe to the receiver probes of all probes. Note that we are in a multiple source, multiple destination case, which *Tomo* was developed to resolve.

Addressing this problem similarly to the problem in the previous section, we can obtain the linearly independent matrix of the routing matrix $\mathcal{P}$, and get the reachability in the same way as proposed in the previous section. By applying the *Tomo* algorithm we can infer the link or links more likely to be down.

However, in the previous case, is possible to not obtain the status for all links, due to the paths from the source to the destination doesn’t cross that link. But, in this case, the topology was reconstructed by the probes that will verify the reachability. The reconstruction algorithm can not be able to reconstruct all links of the topology (it is impossible to reconstruct a link if the packets does’t cross them), however, for getting the reachabilities, all the links reconstructed will be used.

Chapter 4

Implementation and Tests

In this chapter we will talk about the implementation of algorithms described in previous chapters and the tools used to obtain results. We decided to completely separate these two parts, i.e., measurements are obtained and stored in files. Later, these files are used as input for the algorithms. Thus, it is possible to obtain measures of any tool and use the same programs to generate results.

Firstly we will focus on the test preparation, i.e., the tools that we will use to obtain measurements, the topologies, the routing adopted and the traffic generator. Then we will focus on how we make the measurements. Finally, we will focus on the tools developed that implements the algorithms proposed in the previous chapters.

4.1 Test Preparation

Obtaining reliable measurements in order to successfully test our algorithms is a major concern. The way to get the most realistic measurements would be to make measurements on real equipment in real networks. However it would be very expensive (if we have to buy or rent the equipment) and/or it would take a lot of time just to prepare a test scenario. An alternative would be the use of emulation or simulation tools.

4.1.1 Measurement Tools

4.1.1.1 CORE

The first tool that was addressed was the Common Open Research Emulator (CORE)[40, 41]. CORE is an emulator developed by Network Technology research group that is part of the Boeing Research and Technology division. As an emulator, it builds a representation of a real computer network that runs in real time and the live-running emulation can be connected to physical networks and routers. It provides an environment for running real applications and protocols, taking advantage of visualization provided by the Linux or FreeBSD operating systems. The architecture consists of a GUI for easily drawing topologies, a services layer that instantiates lightweight virtual machines, and an API for tying them together [41].

The main problem of this tool is that it only emulates layers 3 and above. We could use this tool for Link Failure Detection, since the link failure just verifies the reachability of the probes. However, it cannot be used for network topology discovery, since our work rely on queueing time, which is layer 1 and 2.

However, it is possible to join CORE with a simulation tool for layers 1 and 2, such as, EMANE or ns-3, but this junction is not very simple to do, so this tool was discarted.

4.1.1.2 GNS3

Other possible choice is Graphical Network Simulator [42]. No official document regarding GNS3 was found, however, GNS3 is a simulator that uses emulators to run the operating systems in the devices as in real networks. It uses the Cisco IOS emulator *Dynamips*, *VirtualBox* that runs desktop and server operating systems as well as Juniper JunOS and *Qemu*, which runs Cisco ASA, PIX and IPS.

As stated in [43], one of the main limitations of GNS3 is obtaining the proper devices to be used with the simulator. Without these devices, operating system GNS3 cannot perform any significant tasks. Other limitation is the high consumption of processing resources. Although we were unable to find a reference value for the number of nodes in a topology in execution at the same time, from our experience while searching this tool and by consulting the GNS forum, the number of nodes running simultaneously is about 15 nodes, which is very low.

This option was also discarted.

4.1.1.3 NS-3

The network simulator 3 [44, 45] is a discrete-event network simulator, targeted primarily for research and educational use. In comparison with other discrete-event network simulators, ns-3 is distinguished by the following high level design goals: C++ and Python emphasis; Callback-driven events and connections; Flexible core with helper layer; Emphasis on emulation.

We choose this simulator to obtain the measurements because script development in C++ is quite appealing. And it was mentioned several times while We were searching for tools. Also, from [45], it seemed to be a good bet for obtaining the measurements.

4.1.2 Network Topology

After resolving the problem of which tool to use to get the measurements, other problem rises, which topologies using for testing. We will build some toy topologies to check whether the algorithms are well implemented, but we need some tool to generate topologies, in order to test the algorithms in different topologies to evaluate their performance.

NS-3 Topology Generator In the site of ns-3, they present a tool [46] with a GUI which the user can draw one network topology and this tool is capable of generate the ns-3 *C++* or *Python* code of that topology.

BRITE NS-3 has also a integration Module with BRITE topology generator [47]. BRITE topology generator framework was built with flexibility and extensibility in mind and is able to quickly and efficiently generate large topologies in 4 different Mmodels: WAXMAN, BA, BA-2 or GLP.

Inet-3.0 Other solution is generate topologies with inet-3.0 [48]. Inet-3.0 is other topology generator, however, this generator can only generate topologies with a minimum number of nodes of 3037, which is a very, very large network. I will not need so big networks, so this tool is excluded.

Once we need to control the capacity of the links of the topology, we decided to generate topologies with BRITE, but not inside ns3 context, i.e., we will generate the topology network, save the topology in a file, change the capacities randomly given a set of possible links and then we will import that topology to ns3.

4.1.3 Routing

The routing protocol is very important for the merge algorithm, the major problem of this algorithm is the occurrence of asymmetric paths. This asymmetric paths occur due to the variety of ”best” routes to forwards the packet. In our first approach, we wanted to give as input to the ns-3 script, the routing algorithm to be applied. Thus we could evaluate the impact that the routing algorithm has on the reconstruction algorithm. More precisely, we wanted to choose RIP and OSPF. However, these routing algorithms are not implemented on ns3. The DCE quagga support [49] allows the use of these routing protocols. However, we were unable to join the DCE quagga support with the ns3.

Ns-3 has several modules for routing but a large part of them are mobile routing protocols. The routing module that most closely resembles the ospf protocol is the Global Routing, this routing executes a Dijkstra Shortest Path First (SPF) algorithm on the topology for each node, but it does not take into account the capacities of the links.

4.1.4 Location of the probes

Supposing the topology is a real network topology, we cannot place probes in an arbitrary location of the network, e.g., assuming that the topology is the topology of an ISP, we cannot place probes at the core of the ISP, the probes should be connected in the access network, which, generally, is one the periphery of the network.

However, we prefer to give the location of the probes as input of the ns-3 script.

4.1.5 Traffic Generator

In order to create some noise for a more realistic values of the measurements, we must join a traffic generator in the network. NS-3 has a traffic generator, the *OnOffApplication*. This application, given the destination node and the data rate, generates traffic until it is stopped. In order to create the minimum traffic possible and have the more realistic values for the measurements, a noisy host will be created on the same network of the probing node, this way, all the path from the sender to destination for each receiver probe will have traffic, but only that path. However, since for each sender probe, there is multiple receiver nodes and the measurements are made to each receiver probe but not at the same time. So, multiple *OnOffApplication* should be added on the sender noisy node (node in the same network than sender probe), and for each receiver probe start and stop the corresponding *OnOffApplication*. Other way to resolve this was to create an application similar to *OnOffApplication* but at any time of the execution it would be possible to change the source, the destination and the data rate of the generator.

4.2 Network Topology Discover Script

The network topology discover script is the script responsible for generate values that will be used to make the topology reconstruction. This script has as input the BRITE topology (as stated before, this topology will be manually done or can generated by BRITE and then, the links will be modified to a given set of possible capacities. Even the topologies manually done must obtain the BRITE layout), the location of the probes and the percentage load of noise that will be on the measurements should also be given as input. Also, the timeout and the number measurements can be set when starting the script.

In any ns-3 script made by us, the first thing to be done is build the topology and assign the probes to their locations. Then the noisy hosts (host that will generate noise) will be assigned to the same nodes than the probes. After everything connected, routing set, the generator will calculate the bottlenecks for each path, in order to set the required noise percentage on the network. The noise on the path will be the percentage of the bottleneck. This is accomplished by the Ping Pair application and the Ping Pair algorithm, which are described below.

After the bottlenecks are calculated, then the alphas for each path should be calculated. The calculation of the alphas is accomplished by Path Capacity Application and algorithm that also are described below.

After the alphas are calculated, we can start the measurements on the probes. This is accomplished by the Network Topology Discovery Application. One application is installed in each probe on the topology.

4.2.1 Network Topology Discover Application

This application is the application responsible for making the measurements and obtain the arrival times of packet $p1$ and $p2$. But before starting to make the measurements, this application requires the probe pairs (destinations pairs), the alpha values to the receivers and a path to store the measured values.

The user can set the number of measurements that each sender will make for each ttl when he starts the script. If none is give, then the application will make 10 measurements.

We will not use any transport protocol, we made the probe sandwich header that uses the IP protocol 200.

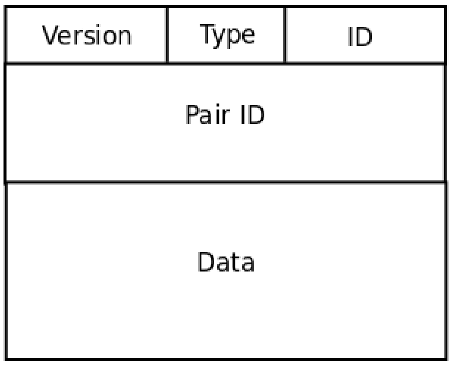

FIGURE 4.1: Packet Sandwich Header.

The Packet Sandwich Header is a very small header, the width represents 8 bits, i.e., Version + Type + ID = 8 bits, and PairID = 16 bits. The Data value depends on the Type and ID of the packet. The Type can be REQ (1) or ACK (2), if a node received a REQ, then he must send a ACK back to the receiver with the same ID and with the arrival time of the packet. The arrival time is a integer with 8 bytes. The ID can be P1 (0), Q (1) or P2 (2). The PairID represents the number of the packet train sent to the same receiver to the same ttl. The Data value, if the packet is REQ, then data will have 64 (minimum size of ethernet packets) - 20 (size of the ip header) - 15 (size of the ethernet header) - 3 (size of the packet sandwich header) = 26 bytes, if this packet is a packet P1 or P2. Otherwise this packet will have 1462 bytes. If the packet is an ACK, then Data will have 8 bytes that represents the arrival time of the packet if it is a P1 or P2 packet, otherwise Data will have 0 bytes.

Each sender probe will make the measurements for each probe pair at a time. When it finished the measurements for a probe pair, then it will begin the measurements on the following probe pair. When all the measurements are done, it will store the measurements and the script is over.

4.3 Link Failure Detection

4.3.1 Link Failure Detection with help

The Link Failure Detection with help script is the script responsible for, given the brite topology, the source and the destination, to calculate all the paths from the sender to the receiver and present the result as a matrix. Through that matrix, the linearly independent matrix is calculated containing only the linearly independent paths. Also the timeout and the number of measurements (tries for reaching the node) can be specified when starting the script.

The applications, on the source and the destination nodes are installed and then a random link or set of links will be calculated to be shutdown in the network.

In this particular case, we can not use any routing protocol, otherwise, when sending the packet to verify the reachability will always be the same. So, in this particular case, we will use static routing. For each path to be measured, the routing table of the involved internal nodes will be modified so that the packet cross the path that we are analyzing.

After doing this and getting the reachability for all paths, then the node will store the values in a file, ending the script.

4.3.2 Link Failure Detection with no help

In this case, we must use the routing protocol. And in this approach, the probes must be located in the exact same nodes than they were when the reconstruction was made. In this script, besides the brite topology and the location of the nodes, the matrix with the paths linearly independent must be introduced. In each path, the sender and the destination must be identified, otherwise, it wouldn't be possible to know who should make measurements. The number of measurements (tries) and timeout can be set at the input.

Then a random link or set of links will be calculated to be shutdown in the network. And then, the applications are installed in the nodes and the reachabilities can be calculated. Since we are using the ns-3 Global Routing, the routing table remains the same even after the link or links go down. So, all paths that contain the link or links down, their reachability will be 0. Otherwise, the routing could resolve the problem and all the paths could be all good.

When all probes finished their measurements, they will store the files with the measurements and the script is over.

4.3.3 Link Failure Detection Application

The link failure detection application consists in only send a packet to the destination. If a ack is received inside the timeout period, then the reachability is 1, and it starts the measurements for the next path. Otherwise, it will resend the packet until the number of tries, if this value is exceeded, the reachability will be 0.

4.4 Other Scripts

4.4.1 Path Capacity Script

This script is not a main objective for our work, but since the application must be made then script just to evaluate the path capacity can be made. Also it will permit to pre-evaluate the alpha values (and we can pass the alpha values to network topology discover script).

This script has as the same input as the network topology discover script. And after installing the path capacity applications, the measurements can start. When all the probes finish the measurements, they will store the measurements in files and the script is over.

4.4.1.1 Path Capacity Application

The path capacity application consists in send two small packet of the packet sandwich back-to-back to a single receiver and obtain their arrival time.

4.4.2 Path Pair Script

This script also is not a main objective for our work, but since the application must be made then script just to evaluate the bottleneck capacity, then the script can also be made. However, in this script, the source and the destination node is a required input. This application is well defined in [50].

And after installing the ping pair application on the sender node, the measurements can start. When it finishes the measurements, it will store the measurements in files and the script is over.

4.4.2.1 Path Pair Application

The ping pair application consists only in sending two back-to-back icmp packets to the receiver and store the times obtained.

4.5 Algorithms implementation

For the implementation of the algorithms we decided to choose the language *C++*. Such language was chosen because: first, some of the algorithms implemented would be reused for obtaining measurements, which restricted my choices to only two languages *C++* and *Python*); secondly, because we were much more comfortable with the chosen language. Though we were not very ”strong” programming in the *C++* language, we had obtained some knowledge in programming language *C*, in *Python* we had no knowledge.

Three main programs were made: the reconstructor, the link failure detector with internal help and the link failure detector with no internal help.

4.5.1 Reconstructor

This program is responsible for the reconstruction of the topology. It is divided into three parts: the decision of the links, the reconstruction of spanning trees, and finally, the merging of spanning trees. As the third part does not always work, due to asymmetric routing, we decided to record the results of the reconstruction of spanning trees at the end of the second part and read these files at the beginning of the third part.

The reconstruction of the spanning trees only includes the Reconstruction Algorithm (Algorithm 1) and saves the spanning trees in binary files. This task is accomplished through the Boost library [51]. The third part is the Merge Spanning Tree Algorithm (Algorithm 4) and if this is successfully applied, then another algorithm is responsible for the correct identification of the various nodes and links of all spanning trees, according to the merged topology. The purpose of this algorithm is to get only an ID for a link or node for all spanning trees. At the end of this part both spanning trees and merged topology are stored in files.

The first part contains two algorithms, the Noise Reduction Algorithm and LCC Decision Algorithm (Algorithm 2 and 3). The Noise Reduction Algorithm was implemented according to his definition, without any included special libraries.

Our first approach to pre-evaluate offline all the possible LCC was to build a database. The database chosen was SQLite [52] however other databases could be chosen such as MySQL. Other solutions was considered like a database in XML with a search algorithm RAPIDXML. This approach was latter discarted due to the increased time to reconstruct a topology. This pre-evaluation is made in real-time at the same time that the algorithm is making the decision. But note that, for a very large set of possible links this can consume a lot of RAM.

4.5.2 Link Failure Detector with help

This program is the program responsible for detecting link failures. It only includes the Tomo Algorithm (Algorithm 5). Given the set of reachabilities, this program will return the link or links that are more likely to be down. In order to produce a graphical representation of the solution, the brite file that was used for making the measurements is required as input.

4.5.3 Link Failure Detector with no help

This program is the program responsible for detecting link failures where there is no cooperation from the internal nodes. It is implemented in a similar way to *Link Failure Detector with help*.

4.5.4 Others

4.5.4.1 DataBaseConstructor

This program was made initially when we first used the database to make the reconstruction of the topology. However, the idea of using a database was discarted and so did this program.

This program is able to build a database given the set of possible links and the maximum number of hops of the shared path.

4.5.4.2 ProbeLocation

This program was initially create to calculate where the probes should be placed in order to use the minimum number of nodes and obtain the best reconstruction possible. However, as stated before, supposing a real topology, this nodes cannot be placed in an arbitrary location. Usually this nodes are in the periphery of the network. Since the nodes in the periphery usually are the nodes with less adjacencies this algorithm chooses the links that have the minimum number of adjacencies. In particular, it will always choose a node that have only one adjacency. In no maximum number of probes to be assigned is set at the input, this algorithm will return the location of $N/2$ probes, where N is the number of nodes in the topology. If there are many nodes with only one adjacency, this value may increase.

4.5.4.3 PathCapacity

Since a script for ns-3 was made to pre-evaluate the α value of the topology, the program to evaluate the α value also must be made. This algorithm only includes the Noise Reduction Algorithm to choose the best measurements and evaluates the α values by simply evaluating the interarrival time of packet $p1$ and $p2$.

4.5.4.4 PingPair

Like the program PathCapacity, since the PingPair script was made, also the PingPair program was made. It also uses the Noise Reduction Algorithm to choose the best measurements, but it uses an algorithm [50] to determinate the bottleneck capacity.

4.5.4.5 Paths

Paths is a program that contains two functions: one function that from all spanning trees in a topology reconstruction, it returns the all-path matrix; and other that returns the linearly independent matrix given the all-path matrix. This second functions is completely based on the function of the matlab presented in Appendix D.

The returned value of this program is used as input to the Link Failure Detection with No Help ns-3 Script. This value will determine which path are needed to calculate the reachability matrix.

4.5.4.6 RemakeBrite

As the name indicates, this program will remake a brite file. Given the set of possible link capacities, this program will randomly choose a link from this set and will assign it to the brite link.

4.5.5 Results Visualization

For display of the results, we found a list[1] of free software tools capable of graphically display the spanning trees and merged topology. Although no specific test has been done in each of the tools, we choose Graphviz [53] because it seemed to be the most simple and easy to use (you only need to ”write” the graphic in DOT language and run the program. this case was used neato, which is the program to draw not oriented graphs).

[1]This list can be consulted in:
http://www.dmoz.org/Science/Math/Combinatorics/Software/Graph_Drawing

Chapter 5

Results

5.1 An Example

Let us consider the Toy Topology on the in Figure 5.1. This is an example of a topology that we will use for test our algorithms. By invoking the program that chooses the location for the probes, it will return (0,3,4,7,9,10,13,14) since these are the nodes that contain only one adjancy. When we add the probes to the topology, we will also create a link between the probe and the node choosen. This link will have a capacity of 100Mbps. The topology will be like as shown in Figure 5.2

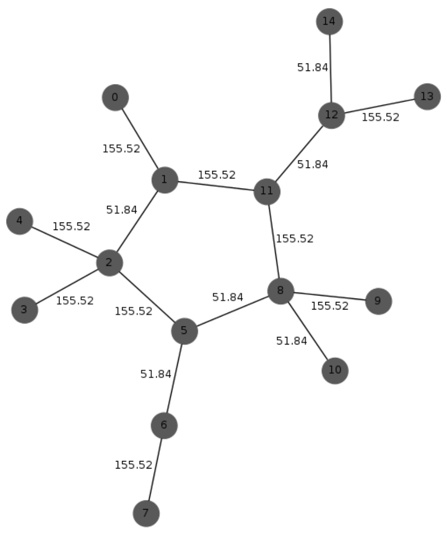

FIGURE 5.1: Toy Topology.

Figure 5.2 shows the node probes P0, P3, P4, P7, P9, P10, P13 and P14 connected in the nodes 0, 3, 4, 7, 9, 10, 13 and 14 respectively. Notice that in the program the probes P0, P3, P4, P7, P9, P10, P13 and P14 will be identified as 0, 1, 2, 3, 4, 5, 6 and 7 respectively.

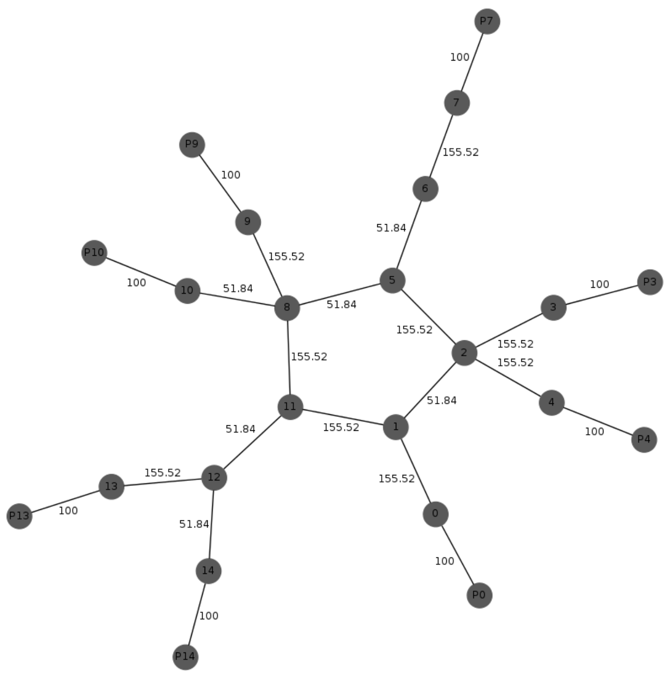

FIGURE 5.2: Toy Topology with 8 probes connected.

5.2 Network Topology Discovery

Using the toy topology in Figure 5.1, an example of a spanning tree obtained (in this case for probe 6) is shown in Figure 5.3.

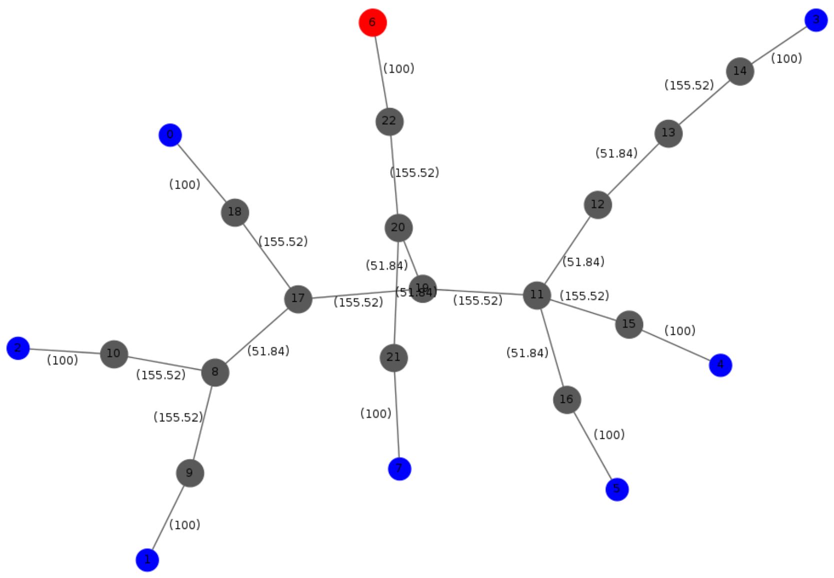

FIGURE 5.3: Spanning Tree Probe 6: The red node represents the sender node, the blue nodes represents the receiver nodes and the gray links and the gray nodes are the links inferred.

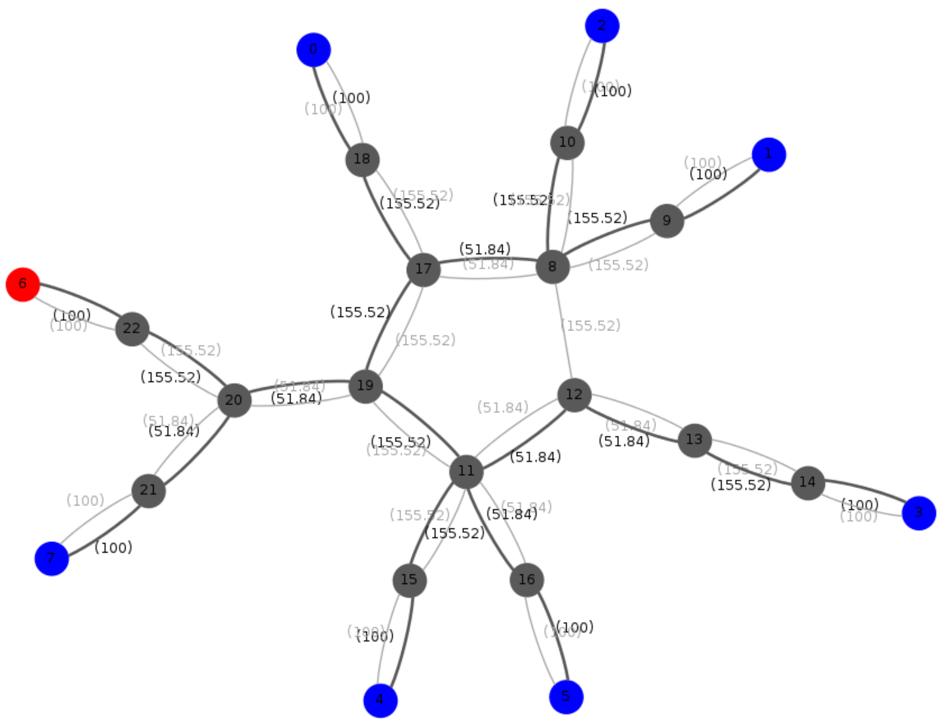

FIGURE 5.4: Spanning Tree Probe 6: Comparasion with the real topology. The darker and thicker nodes and links are the inferred topology, while the lighter and thinner nodes and links are from the real topology.

In the Figure 5.4 we show a comparasion between the reconstructed spanning tree and the actual network. As we can see in the figure, not only the spanning tree was correctly reconstructed but also all the capacities were correctly identified. And as shown in the figure, only one link was not recontructed by this probe.

Note that while Figure 5.3 was one of the output of the reconstruction algorithm, Figure 5.4 was manualy made ir order to the comparation between the the spanning tree and the actual network to be easier.

By combining all the spanning trees calculated, we can get the Merged Topology as shown in Figure 5.5, the comparation with the real topology is shown in Figure 5.6.

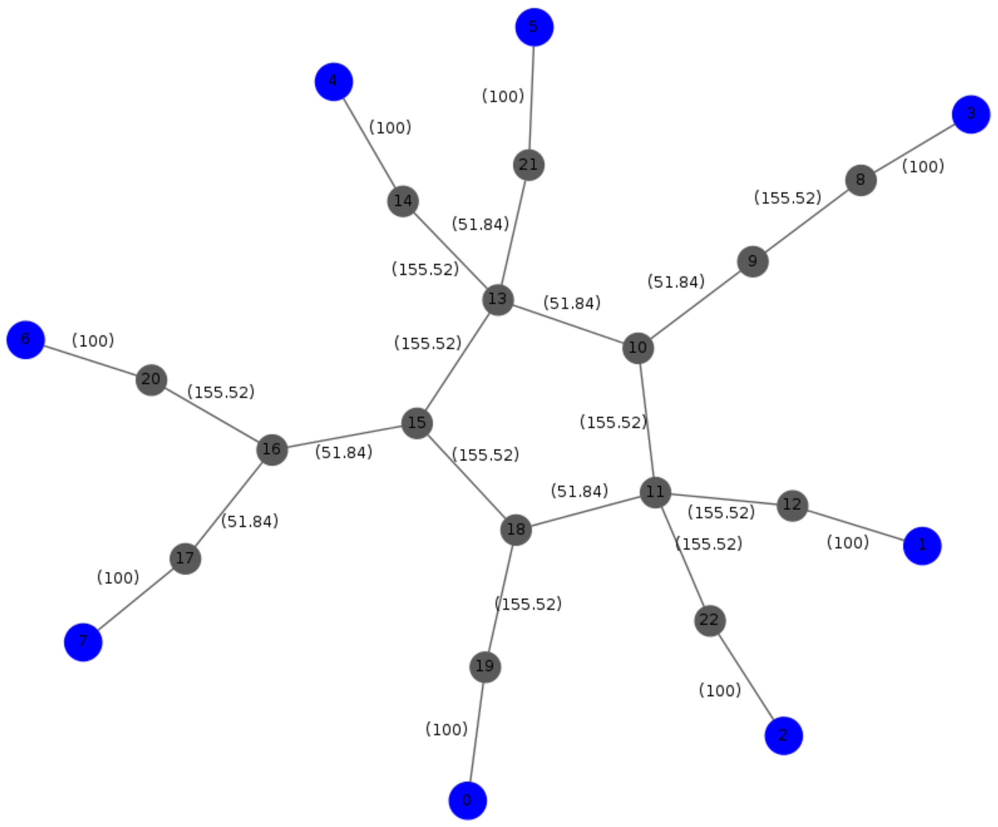

FIGURE 5.5: Merged Topology.

As we can see in Figure 5.6, for the toy topology, we were able discovery all the topology (100 % reconstruction) and discovery the capacity of each link in the topology.

5.2.1 Comparasion with Andreas Results

Also, the Andrea's algorithm works and reconstructs sucefully the topology. However, there can be some problems due to link ordering as we can see from nodes

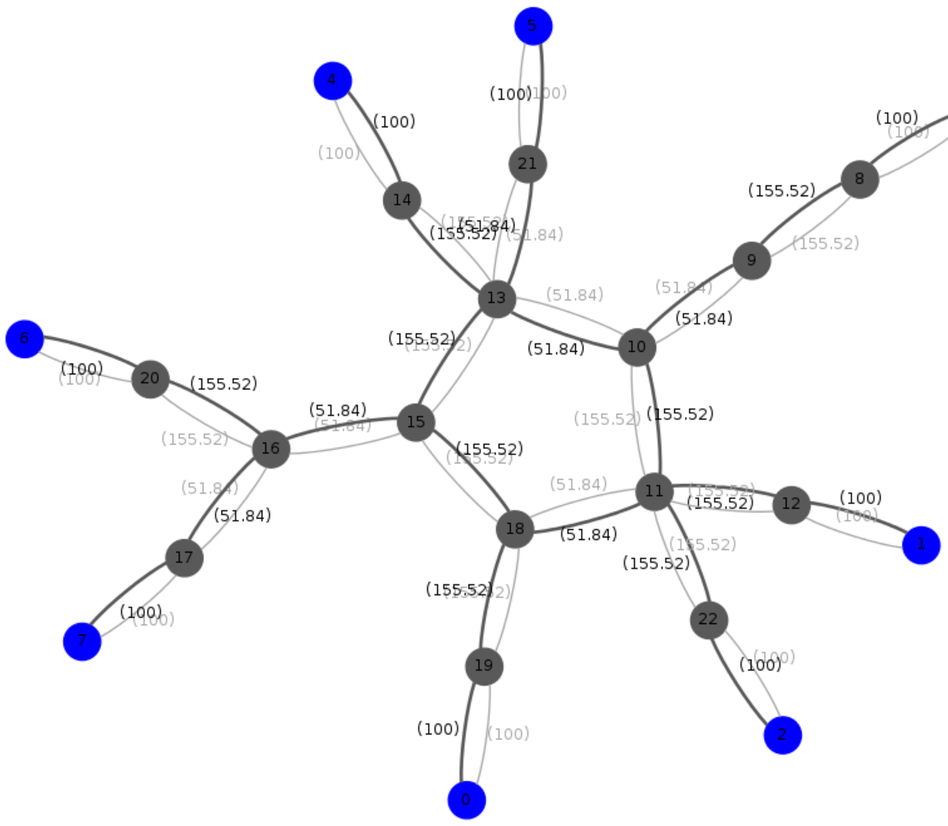

FIGURE 5.6: Merged Topology: Comparasion with the real topology.

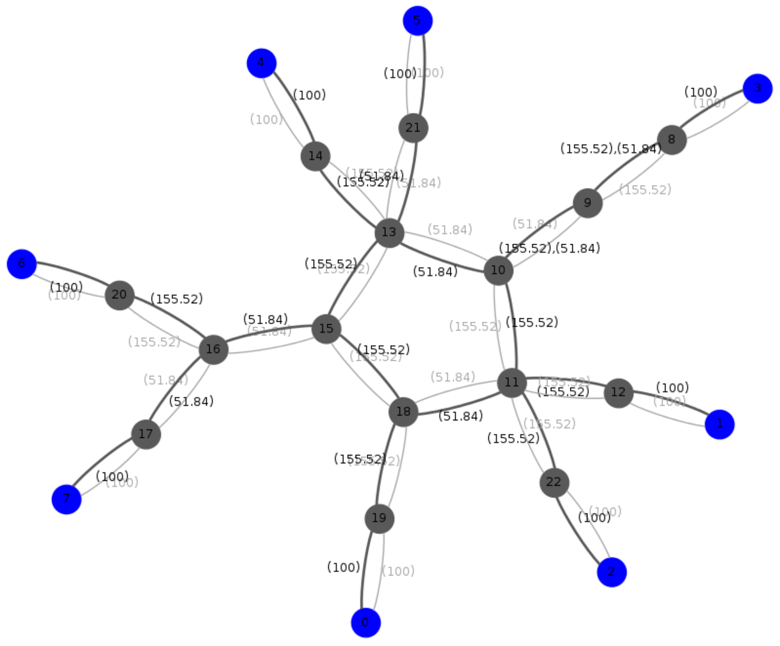

FIGURE 5.7: Merged Topology Andrea's: Comparasion with the real topology.

10 to 8. As shown in Figure 5.7, links that connects nodes 10 and 9 and nodes 9 and 8 have more than one hypothesis for link capacity. These links can a capacity of 155.52 Mb or 51.54 Mb, but is unknown which capacity relates to each link.

5.2.2 Other

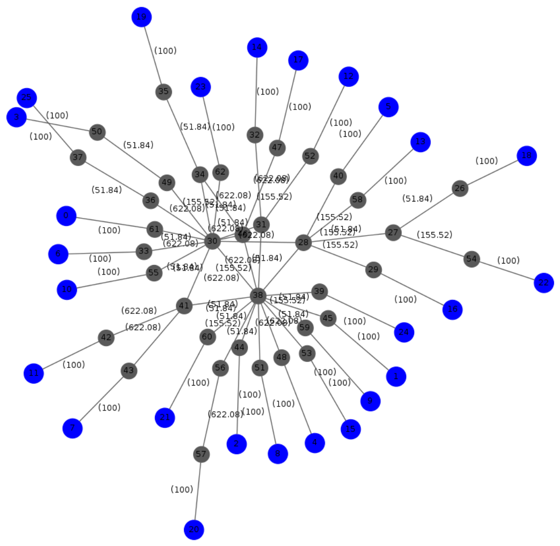

FIGURE 5.8: BRITE generated topology: Model GLP (40 nodes, 47 links).

Figure 5.8 represents a topology generated by BRITE in model GLP with 40 nodes. This topology was also completly reconstructed and we were able to discover all link capacities.

We also made some test in relativey "large topologies". One example is in Appendix E. This topology was manually made and it tries to simuate a ISP topology, highly redundant. The algorithm was able to infer 60 of the 78 nodes and 81 of 191 links with 38 probes.

Appendix F is other example, however this states a complete fail of the merge algorithm. Note that for each assymetric path, two complete path are added to the topology.

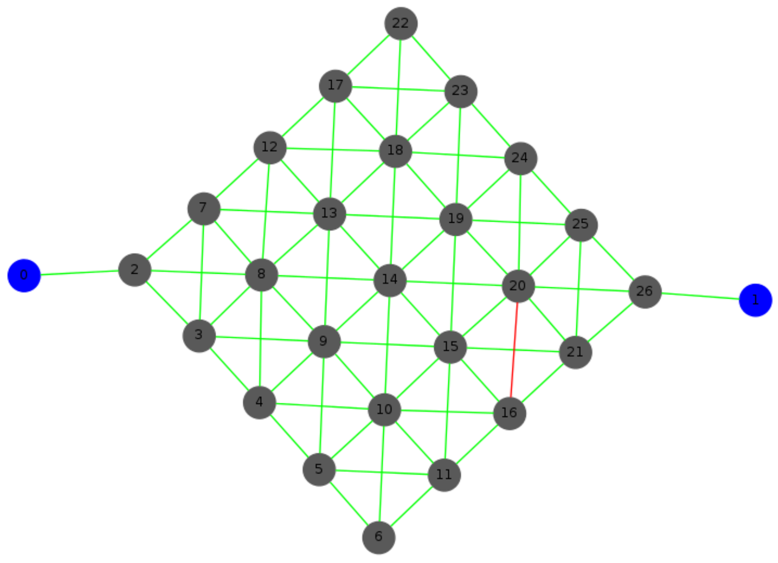

FIGURE 5.9: Link Failure Detection example: The green links mean good links, i.e, links that are up (working), red links mean donw link (not working), gray links mean links its state is unknown.

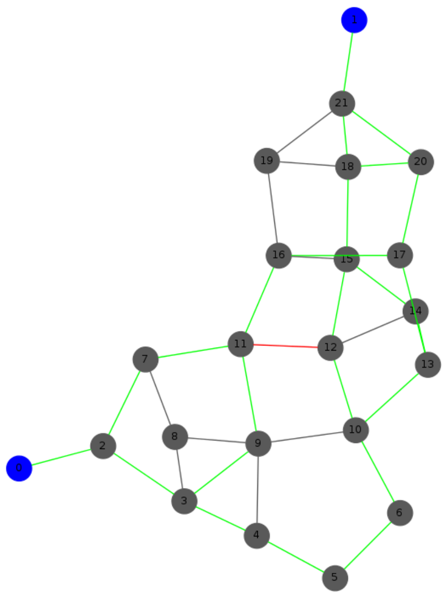

FIGURE 5.10: Link Failure Detection example 2.

5.3 Link Failure Detection

5.3.1 With Help

Figure 5.9 and Figure 5.11 shows two examples of Link Failure Detection with internal help. The gray nodes represent the network topology; the blue nodes represent the the probing nodes: the probe node represented as '0' is the sender probe and the probe node represented as '1' is receiver probe. The green links represents the good links, i.e., working links, the red links represents the links that are not working and the gray links represents links which state is unknown (no traffic crossed these links). Note that the network graph considered in this case is a oriented graph, which means that not all links may reach the destination, like the topology Figure 5.11, and in some topologies, may have no path from one sender to the receiver.

5.3.2 With No Help

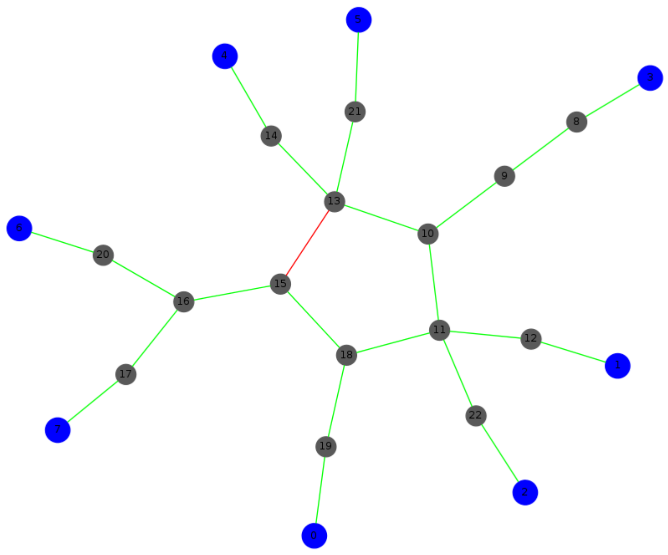

Figure 5.11: Link Failure Detection From the Reconstructed Topology.

In the case of Link Failure Detection without internal help, unlike Link Failure Detection with internal help, the state of all links must be known (all links reconstructed) once there will be traffic crossing all links. The gray nodes are the inferred nodes and the blue nodes are the probe nodes. The green links are the good links and the red links are the failed links.

Figure 5.11 shows a perfect identification of the failed link. However, in some cases, the identification is not perfect, i.e., the links identified as failed sometimes include the actual failed link and some working links (false positives). If the actual failed link was the link from node 10 to node 9, the links identified as failed would be links from 10 to 9; from 9 to 8 and from 8 to 3.

Chapter 6

Conclusion

6.1 Conclusion

In this thesis, we presented a novel approach to Probe Packet Sandwich, by limiting the TTL of the large packet. We show that by controlling the TTL of the large packet of the packet train, we could control the the shared path, and consequently the interarrival time of the small packets. This way the reconstruction of the path could be hop-by-hop, which resolves the problem of link ordering and it is possible to know exactly the depth of the shared path. Also, we show that with this approach, there is no need to still assume that there is no links whose capacity is bigger 25 times or more the capacity of the previous link. We show that we are always able to find these links, the problems that can emerge is that, there can be more than one hypothesis for the capacity of that link, or when this link is the last link of the shared path. In such cases, there will be uncertainty for the location of the branching node, consequently, the depth of the shared path.

We showed that our approach performs better than the Andrea's approach (due that our approach is capable of resolving the link ordering problem). We also show that the spanning trees are almost always correctly constructed. However the merging algorithm is not always applied without any error. Due to the routing algorithm, in very large topologies many paths can be asymmetric.

Then, given the result of a reconstructed topology, we were always able to find links that failed on the network. However, in some results, working links are also identified as failed. This happens when a portion of the network is composed by consecutive nodes with two links connected and one of the links between them is down.

6.2 Future work

There are still many improvements that we can make in this work. We could change the algorithm for link failure detection, despite the good results that we had, Tomo is not a very good algorithm. It is ill-posed as state in [37]. It favors the links that participate in more congested paths, which means that good links may be identified as failed links and some actual failed links not identified as failed link.

Discovery if it is possible to discover the % assigned to each service in a DiffServ network, or infer a congested link or the link delay also could be added.

Appendix A

Notation

General	
P_r	Processing delay
Q	Queueing delay
T	Transmission delay
P_d	Propagation delay
D	Distance between two nodes
P_s	Propagation speed
L	Length of the packet
C	Transmission rate (link Capacity)
t^{dep}	Departure time (at the sender node)
t^{arr}	Arrival time (at the receiver node)
$t^{interdep}$	Interdeparture time (at the sender node)
$t^{interarr}$	Interarrival time between two packets
#	carnality (number of elements)

Packet Sandwich	
$p1$	Packet 1 (first small packet of the sandwich probe)
$p2$	Packet 2 (second small packet of the sandwich probe)
q	Packet q (large packet of the sandwich probe)
L^p	Size of the packets p1 and p2
L^q	Size of the packets q
d	Interarrival time between packets p1 and p2
δ	Interdeparture time between packets p1 and q
α	Transmission time of p1 on the first link

β	Transmissions times of q and p2 on the first link
$\hat{d}$	Mean value of the d values
$\mathcal{B}_{i,j}$	The set of real links capacities in the shared path of receivers i and j
$\mathcal{B}_{i,j_n}$	The n-th link capacity of the shared path of receivers i and j
$\mathcal{C}i,j$	The set of links capacities in the shared path of receivers i and j
$\mathcal{S}$	The set of possible links capacities
$L25$	Represents a link that its capacity is greater than 25 times the capacity of the previous link
LCC	Link Capacities Combination
$\Lambda_{i\to j}$	Path (link capacities) from node i to node j
$\mathcal{L}$	The set containing the leaves (receiver probes) of the spanning tree
$\mathcal{T}$	The spanning tree
$\bar{\mathcal{C}}$	The selected LCC
γ	Function that evaluates the metric d of a LCC
Φ	The set of possible branching nodes for a shared path

Link Failure Detection

$\mathcal{E}$	Set of all links
$\mathcal{P}$	Routing matrix (set of all paths)
$\mathcal{P}_{id}$	Routing matrix linearly independent
$\mathcal{R}$	Reachabilities matrix

Appendix B

Interarrival Time Between Two Packets

As shown in [1] a packet is subjected to several delays since it is created in the source node until it is fully received and processed in the destination node. The most important delays are the processing delay (P_r), the queueing delay (Q), the transmission delay (T) and the propagation delay (P_d). The Figure B.1 explain very well these delays and where they occur.

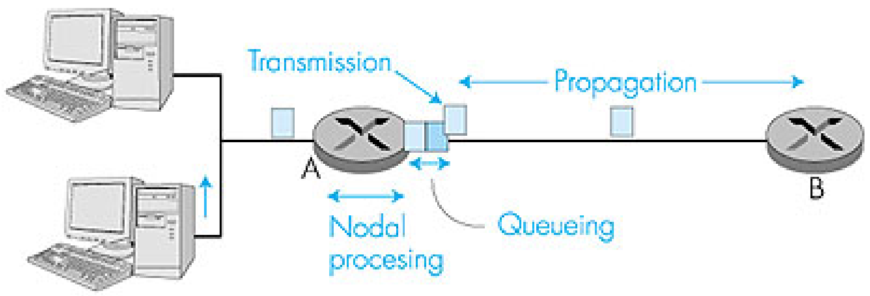

FIGURE B.1: Network delays (figure also from [1]).

The time between the packet is created in the source node until it is fully received and processed in the destination node, i.e., the packet delivery time (P_t) can be calculated as the sum of the delays described above as shown in Equation B.1.

$$P_t = \sum_{i=1}^{N} \left(T_i + P_{d_i} \right) + \sum_{i=1}^{M-1} Q_i + \sum_{i=2}^{M} P_{r_i} \tag{B.1}$$

where (N) is the number of links comprising the path between the source and the destination and (M) is the number of nodes contained in the path (source and destination nodes included).

Since the path between the source and the destination, both included, has $M-1$ links, Equation B.1 can be simplified to:

$$\begin{aligned} P_t &= \sum_{i=1}^{N} \Big(T_i + P_{d_i} + Q_i + P_{r_i} \Big) \\ &= T + P_d + Q + Pr \end{aligned} \tag{B.2}$$

The propagation delay is the quotient between the length of the cable or the distance between the nodes (D), if it is a wireless connection, and the propagation speed (P_s). The propagation speed depends on the physical medium of the link. In the case of an optical fiber, this speed will be the speed of light in glass which is typically around 180,000 to 200,000 km/s as shown in [54]. The transmission delay is not related with the length of the cable or the distance between the nodes. This delay is directly proportional with the length of the packet (L) and inversely proportional with the transmission rate of the link (R). The equations are shown in the Equation B.3 and Equation B.4 respectively.

$$P_d = \frac{D}{P_s} \tag{B.3}$$

$$T = \frac{L}{R} \tag{B.4}$$

Thus, we can rewrite the Equation B.1 to the folowing equation:

$$P_t = \sum_{i=1}^{N} \Big(\frac{L}{R_i} + \frac{D_i}{P_{s_i}} + Q_i + P_{r_i} \Big) \tag{B.5}$$

Let's now consider the following scenario: one node (source node) sends two identical packets to other node (receiver node) but but separated in time that we will call $(t^{interdep})$. The time difference between the two packets that we will call by $(t^{interarr})$ is the difference between the arrival time of the second packet (t_2^{arr}) and the arrival time of the first packet (t_1^{arr}) as shown in Figure B.2 and demonstrated in Equation B.6.

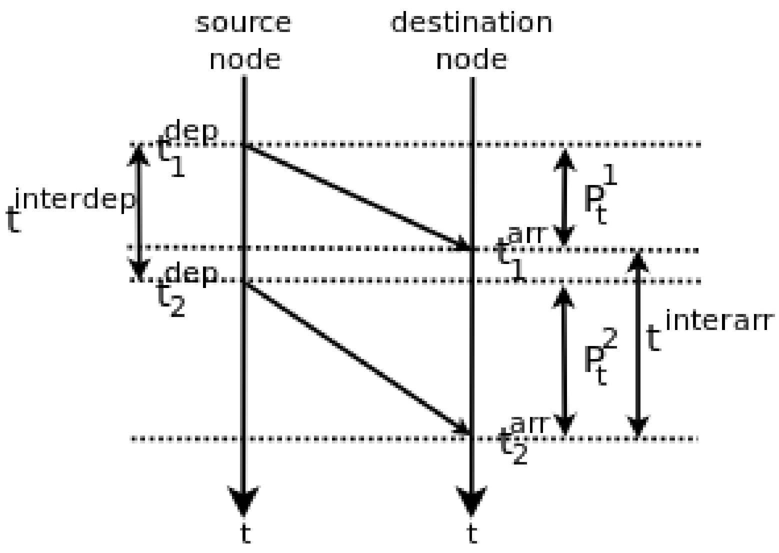

FIGURE B.2: Packet delivery between source and destination nodes.

$$t_{1,2}^{interarr} = t_2^{arr} - t_1^{arr} \tag{B.6}$$

We also know that the arrival time of one packet is the instant of time that the packet is sent from the source node (t^{dep}) plus the packet delivery time (P_t):

$$t^{arr} = t^{dep} + P_t \tag{B.7}$$

From Equation B.7 and Equation B.5 we can expand the Equation B.6 resulting the following equations:

$$\begin{aligned} t_{1,2}^{interarr} &= (t_2^{dep} + P_t^2) - (t_1^{dep} + P_t^1) \\ &= t_{1,2}^{interdep} + \sum_{i=1}^{N}\Big(\frac{L^2}{R_i^2} + \frac{D_i^2}{P_{s_i}^2} + Q_i^2 + P_{r_i}^2\Big) - \sum_{i=1}^{N}\Big(\frac{L^1}{R_i^1} + \frac{D_i^1}{P_{s_i}^1} + Q_i^1 + P_{r_i}^1\Big) \end{aligned} \tag{B.8}$$

Now, taking into account that the packets are identical, we know that the length of the packets are the same. By assuming that the two packets traverse the same path, i.e., exactly the same links and the same nodes we know that the transmission rate and the distance/length of the link are the same for both packets. Since the packets are sent close in time, the medium of the link is the same and subjected to the same conditions so we know that the propagation speed are equal. Given that the packets are identical, we can say the processing time of the packets are the same despite of not being entirely true. The processing time also depends on work load of the CPU of the routers but since that this values are in the range of pico/nanoseconds, they can be discarted.

Thus, we can simplify the previous equation (Equation B.8) resulting the following equation:

$$\begin{aligned} t_{1,2}^{interarr} &= t_{1,2}^{interdep} + \sum_{i=1}^{N}\Big(\frac{L^2}{R_i^2} + \frac{D_i^2}{P_{s_i}^2} + Q_i^2 + P_{r_i}^2\Big) - \sum_{i=1}^{N}\Big(\frac{L^1}{R_i^1} + \frac{D_i^1}{P_{s_i}^1} + Q_i^1 + P_{r_i}^1\Big) \\ &= t_{1,2}^{interdep} + \sum_{i=1}^{N}\big(Q_i^2\big) - \sum_{i=1}^{N}\big(Q_i^1\big) \\ &= t_{1,2}^{interdep} + Q^2 - Q^1 \end{aligned} \tag{B.9}$$

Appendix C

All paths src-des calculation through Breath Algorithm

Algorithm 6: Breath-first search Algorithm

```
input  : $\mathcal{G}$ ; $s$ ; $d$
output: $\mathcal{P}_{sd}$
if $s$ equal $d$ then
    Return $\emptyset$
end
Initialize queue $\mathcal{Q} = \emptyset$ ;
enqueue $\{s\}$ in $\mathcal{Q}$ ;
while $\mathcal{Q}$ not empty do
    t ← dequeue $\mathcal{Q}$ {Set Containing the path} ;
    adjacents = adjacent_nodes(last(t)) {Set composing by the adjacent
    nodes};
    foreach $adj \in adjacents$ do
        if $adj = d$ then
            $p$ = t ;
            $p \cup \{adj\}$ ;
            $\mathcal{P}_{sd} \cup \{p\}$ ;
        else
            if $adj \notin t$ then
                $p$ = t ;
                $p \cup \{adj\}$ ;
                enqueue $p$ in $\mathcal{Q}$ ;
            end
        end
    end
end
```

Appendix D

Linear Independent Matrix Function

lim.m

```
function [R,A,jb] = lim(A,tol)

Z=A;
[m,n] = size(A);
lin_order=1:m;

% Does it appear that elements of A are ratios of small integers?
[num, den] = rat(A);
rats = isequal(A,num./den);

% Compute the default tolerance if none was provided.
if (nargin < 2), tol = max(m,n)*eps(class(A))*norm(A,'inf'); end

% Loop over the entire matrix.
i = 1;
j = 1;
jb = [];
while (i <= m) && (j <= n)
   % Find value and index of largest element in the remainder of
   % column j.
   [p,k] = max(abs(A(i:m,j))); k = k+i-1;
   if (p <= tol)
      % The column is negligible, zero it out.
      A(i:m,j) = zeros(m-i+1,1);
      j = j + 1;
   else
      % Remember column index
      jb = [jb j];
      % Swap i-th and k-th rows.
      lin_order([i k])=lin_order([k i]);
      A([i k],j:n) = A([k i],j:n);
      % Divide the pivot row by the pivot element.
      A(i,j:n) = A(i,j:n)/A(i,j);
      % Subtract multiples of the pivot row from all the other
      % rows.
      for k = [1:i-1 i+1:m]
         A(k,j:n) = A(k,j:n) - A(k,j)*A(i,j:n);
      end
      i = i + 1;
      j = j + 1;
   end
end

% Return "rational" numbers if appropriate.
if rats
    [num,den] = rat(A);
    A=num./den;
end

for i=1:length(jb)
    R(i,:) = Z(lin_order(i),:);
end

end
```

Appendix E

A large Topology

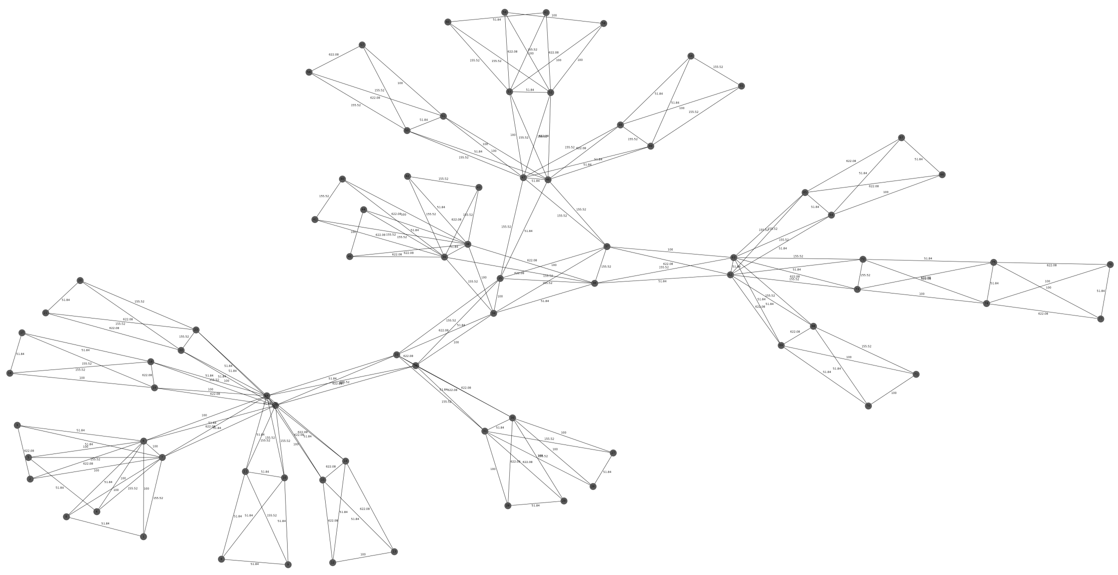

Figure E.1: Large topology.

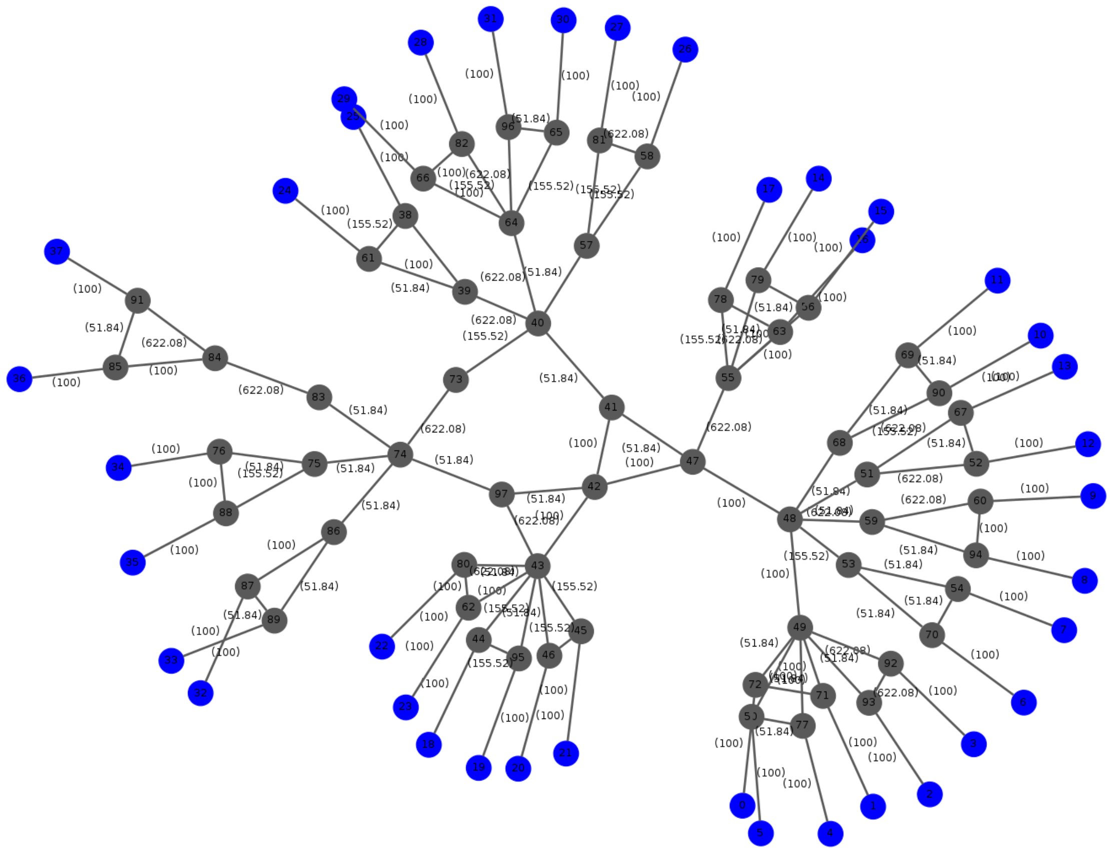

FIGURE E.2: The merged Large Topology: the reconstruction and merge algorithms were able to find 60/78 nodes and 81/191 Links with 38 probes.

Appendix F

Merge Algorithm Fail

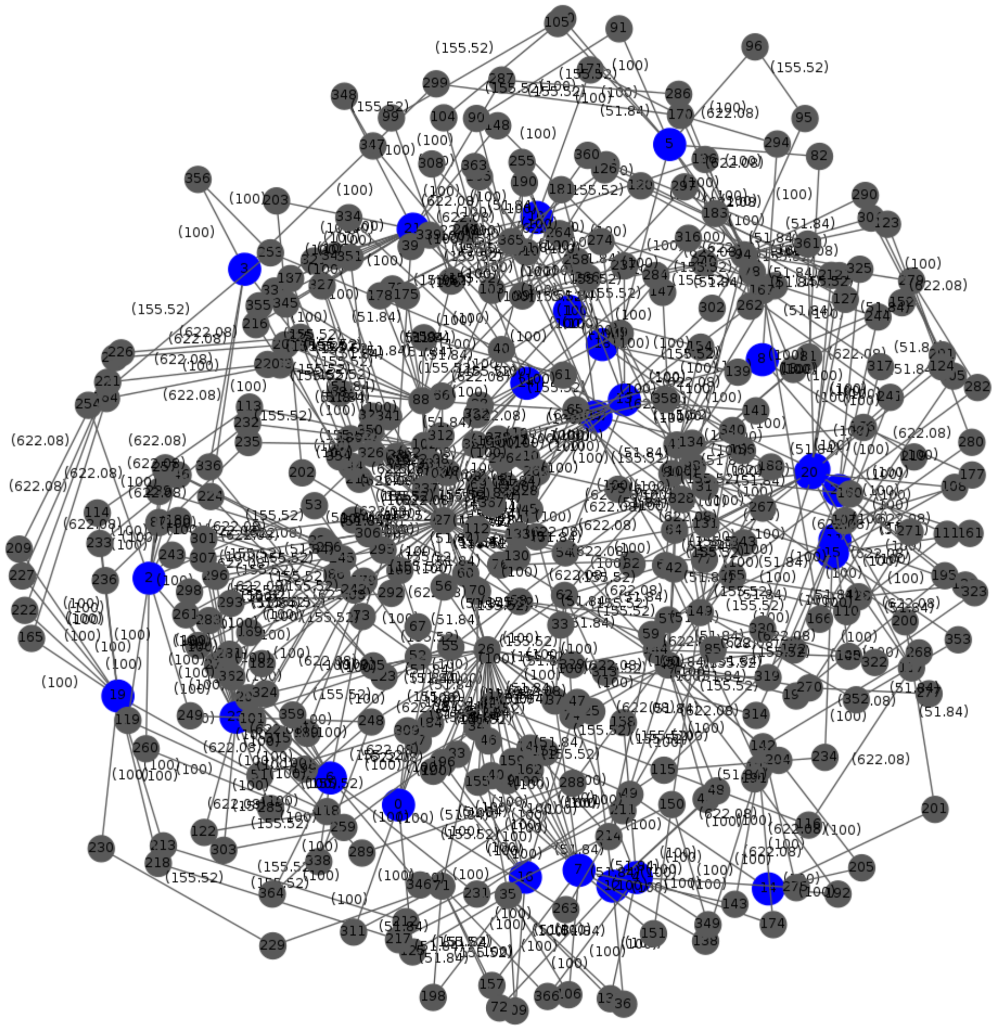

FIGURE F.1: Merge topology fail: In 300 paths, 99 were assymetric.

Bibliography

[1] Delay and loss in packet-switched networks, October 2013. URL http://netlab.ulusofona.pt/rc/book/1-introduction/1_06/index.htm.

[2] Ramesh Govindan and Hongsuda Tangmunarunkit. Heuristics for internet map discovery. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1371–1380. IEEE, 2000.

[3] Benoit Donnet, Philippe Raoult, Timur Friedman, and Mark Crovella. Efficient algorithms for large-scale topology discovery. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 327–338. ACM, 2005.

[4] Benoit Donnet, Timur Friedman, and Mark Crovella. Improved algorithms for network topology discovery. In *Passive and Active Network Measurement*, pages 149–162. Springer, 2005.

[5] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring isp topologies with rocketfuel. *ACM SIGCOMM Computer Communication Review*, 32(4): 133–145, 2002.

[6] Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann, Matús Mihal'ak, and L Shankar Ram. Network discovery and verification. *Selected Areas in Communications, IEEE Journal on*, 24(12): 2168–2181, 2006.

[7] Benoit Donnet. Internet topology discovery. In *Data Traffic Monitoring and Analysis*, pages 44–81. Springer, 2013.

[8] Thomas Bourgeau and Timur Friedman. Toward fast and efficient ip-level network topology capture. In *Proceedings of the 2012 ACM conference on CoNEXT student workshop*, pages 5–6. ACM, 2012.

[9] Yuri Breitbart, Minos Garofalakis, Ben Jai, Cliff Martin, Rajeev Rastogi, and Avi Silberschatz. Topology discovery in heterogeneous ip networks: the netinventory system. *IEEE/ACM Transactions on Networking (TON)*, 12(3): 401–414, 2004.

[10] Suman Pandey, Mi-Jung Choi, Sung-Joo Lee, and James W Hong. Ip network topology discovery using snmp. In *Information Networking, 2009. ICOIN 2009. International Conference on*, pages 1–5. IEEE, 2009.

[11] Bruce Lowekamp, David O'Hallaron, and Thomas Gross. Topology discovery for large ethernet networks. In *ACM SIGCOMM Computer Communication Review*, volume 31, pages 237–248. ACM, 2001.

[12] Aman Shaikh, Mukul Goyal, Albert Greenberg, Raju Rajan, and KK Ramakrishnan. An ospf topology server: Design and evaluation. *Selected Areas in Communications, IEEE Journal on*, 20(4):746–755, 2002.

[13] Ramón Cáceres, Nick G Duffield, Joseph Horowitz, and Donald F Towsley. Multicast-based inference of network-internal loss characteristics. *Information Theory, IEEE Transactions on*, 45(7):2462–2480, 1999.

[14] Tian Bu, Nick Duffield, Francesco Lo Presti, and Don Towsley. Network tomography on general topologies. In *ACM SIGMETRICS Performance Evaluation Review*, volume 30, pages 21–30. ACM, 2002.

[15] Nick G Duffield, Joseph 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*, volume 3, pages 1636–1645. IEEE, 2001.

[16] Nick G Duffield and Francesco Lo Presti. Network tomography from measured end-to-end delay covariance. *IEEE/ACM Transactions on Networking (TON)*, 12(6):978–992, 2004.

[17] Nick G Duffield, Joseph Horowitz, F Lo Presti, and Don Towsley. Multicast topology inference from measured end-to-end loss. *Information Theory, IEEE Transactions on*, 48(1):26–45, 2002.

[18] Nick G Duffield, Joseph Horowitz, F Lo Presti, and D Towsley. Network delay tomography from end-to-end unicast measurements. In *Evolutionary Trends of the Internet*, pages 576–595. Springer, 2001.

[19] AHeroIII Coates, Alfred O Hero III, Robert Nowak, and Bin Yu. Internet tomography. *Signal Processing Magazine, IEEE*, 19(3):47–65, 2002.

[20] Nick Duffield, Francesco Lo Presti, Vern Paxson, and Don Towsley. Network loss tomography using striped unicast probes. *Networking, IEEE/ACM Transactions on*, 14(4):697–710, 2006.

[21] Yolanda Tsang, Mehmet Yildiz, Paul Barford, and Robert Nowak. Network radar: tomography from round trip time measurements. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 175–180. ACM, 2004.

[22] Brian Eriksson, Gautam Dasarathy, Paul Barford, and Robert Nowak. Toward the practical use of network tomography for internet topology discovery. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[23] Brian Eriksson, Gautam Dasarathy, Paul Barford, and Robert Nowak. Efficient network tomography for internet topology discovery. *IEEE/ACM Transactions on Networking (TON)*, 20(3):931–943, 2012.

[24] Jian Ni and Sekhar Tatikonda. Network tomography based on additive metrics. *Information Theory, IEEE Transactions on*, 57(12):7798–7809, 2011.

[25] Mark Coates, Rui Castro, Robert Nowak, Manik Gadhiok, Ryan King, and Yolanda Tsang. Maximum likelihood network topology identification from edge-based unicast measurements. *ACM SIGMETRICS Performance Evaluation Review*, 30(1):11–20, 2002.

[26] Vanniarajan Chellappan and Kamala Krithivasan. Network (tree) topology inference based on prüfer sequence. In *Communications (NCC), 2010 National Conference on*, pages 1–5. IEEE, 2010.

[27] Andrea di Pietro. *Architectures and algorithms for packet processing and network monitoring.* PhD thesis, University of Pisa, 2011.

[28] Andrea Di Pietro, Domenico Ficara, Stefano Giordano, Francesco Oppedisano, and Gregorio Procissi. Network topology discovery based on a finite set of hypotheses. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–5. IEEE, 2008.

[29] Gianni Antichi, Andrea Di Pietro, Domenico Ficara, Stefano Giordano, Gregorio Procissi, and Fabio Vitucci. Network topology discovery through self-constrained decisions. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1–6. IEEE, 2009.

[30] Andrea Di Pietro, Domenico Ficara, Stefano Giordano, Francesco Oppedisano, Gregorio Procissi, and Fabio Vitucci. Merging spanning trees in tomographic network topology discovery. In *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–5. IEEE, 2009.

[31] Andrea Di Pietro, Domenico Ficara, Stefano Giordano, Francesco Oppedisano, and Gregorio Procissi. Noise reduction techniques for network topology discovery. In *Personal, Indoor and Mobile Radio Communications, 2007. PIMRC 2007. IEEE 18th International Symposium on*, pages 1–5. IEEE, 2007.

[32] Tracerotue manual, October 2013. URL http://www.zytek.com/traceroute.man.html.

[33] D Ghita. *Practical Network Tomography.* PhD thesis, PhD thesis, Ecole Polytechnique Federale de Lausanne, 2012.

[34] Venkata N Padmanabhan, Lili Qiu, and Helen J Wang. Server-based inference of internet link lossiness. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 1, pages 145–155. IEEE, 2003.

[35] Nick Duffield. Network tomography of binary network performance characteristics. *Information Theory, IEEE Transactions on*, 52(12):5373–5388, 2006.

[36] Amogh Dhamdhere, Renata Teixeira, Constantine Dovrolis, and Christophe Diot. Netdiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data. In *Proceedings of the 2007 ACM CoNEXT conference*, page 18. ACM, 2007.

[37] Denisa Ghita, Can Karakus, Katerina Argyraki, and Patrick Thiran. Shifting network tomography toward a practical goal. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, page 24. ACM, 2011.

[38] Breadth-first search algorithm, October 2013. URL http://en.wikipedia.org/wiki/Breadth-first_search.

[39] Matlab rref function documentation, October 2013. URL http://www.mathworks.com/help/matlab/ref/rref.html.

[40] Common open research emulator (core), October 2013. URL http://cs.itd.nrl.navy.mil/work/core/.

[41] Jeff Ahrenholz. Comparison of core network emulation platforms. In *MILITARY COMMUNICATIONS CONFERENCE, 2010-MILCOM 2010*, pages 166–171. IEEE, 2010.

[42] Graphical network simulator (gns3), October 2013. URL http://www.gns3.net/.

[43] Woratat Makasiranondh, S Paul Maj, and David Veal. Pedagogical evaluation of simulation tools usage in network technology education. *Engineering and Technology*, 8:321–326, 2010.

[44] Network simulator 3, October 2013. URL http://www.nsnam.org/.

[45] Thomas R Henderson, Mathieu Lacage, George F Riley, C Dowell, and JB Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 2008.

[46] Ns-3 topology generator, October 2013. URL http://www.nsnam.org/wiki/index.php/Topology_Generator.

[47] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: An approach to universal topology generation. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, pages 346–353. IEEE, 2001.

[48] Jared Winick and Sugih Jamin. Inet-3.0: Internet topology generator. Technical report, Technical Report CSE-TR-456-02, University of Michigan, 2002.

[49] Dce quagga support, October 2013. URL http://www.nsnam.org/docs/dce/manual-quagga/html/getting-started.html.

[50] Andrea Di Pietro, Domenico Ficara, Stefano Giordano, Francesco Oppedisano, and Gregorio Procissi. Pingpair: a lightweight tool for measurement

noise free path capacity estimation. In *Communications, 2008. ICC'08. IEEE International Conference on*, pages 1–5. IEEE, 2008.

[51] Boost c++ libraries, October 2013. URL http://www.boost.org/.

[52] Sqlite, October 2013. URL http://www.sqlite.org/.

[53] Graphviz - graph visualization software, October 2013. URL http://www.graphviz.org/.

[54] Optical fiber cable (propagation speed and delay), October 2013. URL http://en.wikipedia.org/wiki/Optical_fiber_cable.